# The Quicksort algorithm and related topics

**Vasileios Iliopoulos**

A thesis submitted for the degree of
Doctor of Philosophy

Department of Mathematical Sciences
University of Essex

June 2013

# Abstract

Sorting algorithms have attracted a great deal of attention and study, as they have numerous applications to Mathematics, Computer Science and related fields. In this thesis, we first deal with the mathematical analysis of the Quicksort algorithm and its variants. Specifically, we study the time complexity of the algorithm and we provide a complete demonstration of the variance of the number of comparisons required, a known result but one whose detailed proof is not easy to read out of the literature. We also examine variants of Quicksort, where multiple pivots are chosen for the partitioning of the array.

The rest of this work is dedicated to the analysis of finding the true order by further pairwise comparisons when a partial order compatible with the true order is given in advance. We discuss a number of cases where the partially ordered sets arise at random. To this end, we employ results from Graph and Information Theory. Finally, we obtain an alternative bound on the number of linear extensions when the partially ordered set arises from a random graph, and discuss the possible application of Shellsort in merging chains.

# Acknowledgements

I would like to thank Dr. David B. Penman for his meticulous advise and guidance, as I pursued this work. Our fruitful conversations helped me to clearly express my ideas in this thesis. Thanks are also due to Dr. Gerald Williams for his comments and suggestions during board meetings.

I dedicate this work to my family. Without their support, I wouldn’t be able to carry out and accomplish my research.

# Contents

# Chapter 1

# Preface

The first Chapter serves as an introduction to this work, where in a simple manner, useful notions of algorithmic analysis in general are presented, along with definitions that will be used throughout the thesis. We present an introduction to Quicksort and this Chapter ends with an outline and a summary of the main contributions of this thesis.

## 1.1 Preliminaries

An algorithm is a valuable tool for solving computational problems. It is a well defined procedure that takes some value or a set of values as input and is guaranteed to produce an answer in finite time. See e.g. [15]. However the time taken may be impractically long.

As a useful introduction, we present a simple and intuitive algorithm known as Euclid's algorithm which is still used today to determine the greatest common divisor of two integers. Its definition follows [45]:

**Definition 1.1.1.** *Given two integers $A \geq C$ greater than unity, we want to find their greatest common divisor, g.c.d.(A, C).*

1. *If C divides A, then the algorithm terminates with C as the greatest common divisor.*
2. *If $A \bmod C$ is equal to unity, the numbers are either prime or relatively prime and the algorithm terminates. Otherwise set $A \leftarrow C$, $C \leftarrow A \bmod C$ and return to step* $1$.

By $A \bmod C$, we denote the remainder of the division of $A$ by $C$, namely

$$A \bmod C := A - \left( C \cdot \left\lfloor \frac{A}{C} \right\rfloor \right),$$

which is between $0$ and $C - 1$. Here the floor function $\lfloor x \rfloor$ of a real number $x$ is the largest integer less than or equal to $x$. We observe that this algorithm operates recursively by successive divisions, until obtaining remainder equal to $0$ or to $1$. Since the remainder strictly reduces at each stage, the process is finite and eventually terminates.

Two main issues in relation to an algorithm are its running time or time complexity, which is the amount of time necessary to solve the problem, and its space complexity, which is the amount of memory needed for the execution of the algorithm in a computer. In case of Euclid's algorithm, $3$ locations of memory are required for storing the integer numbers $A$, $C$ and $A \bmod C$. In this thesis, we will mainly be concerned with time complexity.

The aim is usually to relate the time complexity to some measure of the size of the instance of the problem we are considering. Very often we are particularly concerned with what happens when the size $n$ is large – tending to infinity. The notations $O$, $\Omega$, $\Theta$ and $o$, $\omega$ are useful in this context.

## 1.2 Definitions

In this section, we present primary definitions to the analysis presented in the thesis. These definitions come from [15], [26].

**Definition 1.2.1.** *Let $f(n)$ and $g(n)$ be two functions with $n \in \mathbf{N}$. We say that $f(n) = O\big(g(n)\big)$ if and only if there exists a constant $c > 0$ and $n_0 \in \mathbf{N}$, such that $|f(n)| \leq c \cdot |g(n)|$, $\forall n \geq n_0$.*

**Definition 1.2.2.** *Also, we say that $f(n) = \Omega\big(g(n)\big)$ if and only if there exists a constant $c > 0$ and $n_0 \in \mathbf{N}$, such that $|f(n)| \geq c \cdot |g(n)|$, $\forall n \geq n_0$.*

**Definition 1.2.3.** *Furthermore, we say that $f(n) = \Theta\big(g(n)\big)$ if and only if there exist positive constants $c_1$, $c_2$ and $n_0 \in \mathbf{N}$, such that $c_1 \cdot |g(n)| \leq |f(n)| \leq c_2 \cdot |g(n)|$, $\forall n \geq n_0$. Equivalently, we can state that if $f(n) = O\big(g(n)\big)$ and $f(n) = \Omega\big(g(n)\big)$, then $f(n) = \Theta\big(g(n)\big)$.*

We will also use the notations $o$, $\omega$. They provide the same kind of limiting bounds with the respective upper case notations. The difference is that for two functions $f(n)$ and $g(n)$, the upper case notation holds when it does exist some positive constant $c$. Whereas, the respective lower case notation is true for every positive constant $c$ [15]. In other words, $o$ is stronger statement than $O$, since $f(n) = o\big(g(n)\big)$ implies that $f$ is dominated by $g$. Equivalently, this can be stated as

$$f(n) = o\big(g(n)\big) \iff \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0,$$

provided that $g(n)$ is non-zero.

The relation $f(n) = \omega\big(g(n)\big)$ implies that $f$ dominates $g$, i.e.

$$f(n) = \omega\big(g(n)\big) \Longleftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty.$$

The relation $f(n) \sim g(n)$ denotes the fact that $f(n)$ and $g(n)$ are asymptotically equivalent, i.e.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1.$$

Further, best, worst and average case performance denote the resource usage, e.g. amount of memory in computer or running time, of a given algorithm at least, at most and on average, respectively. In other words, these terms describe the behaviour of an algorithm under optimal circumstances (e.g. best–case scenario), worst circumstances and on average [15]. In this work, the study will be concentrated on the average and worst case performance of Quicksort and its variants.

In our analysis, we will frequently come across with harmonic numbers, whose definition we now present.

**Definition 1.2.4.** *The sum*

$$H_n^{(k)} := \sum_{i=1}^{n} \frac{1}{i^k} = \frac{1}{1^k} + \frac{1}{2^k} + \ldots + \frac{1}{n^k}$$

*is defined to be the generalised $n_{th}$ harmonic number of order $k$. When $k = 1$, the sum denotes the $n_{th}$ harmonic number, which we simply write $H_n$. We define also $H_0 := 0$.*

There are numerous interesting properties of harmonic numbers, which are not yet fully investigated and understood. Harmonic series have links with Stirling numbers [26] and arise frequently in the analysis of algorithms. For $n$ large, it is well-known that [44],

$$H_n = \log_e(n) + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} + O\left(\frac{1}{n^6}\right),$$

where $\gamma = 0.57721\ldots$ is the Euler–Mascheroni constant. We will use this often, especially in the form $H_n = \log_e(n) + \gamma + o(1)$.

Note that throughout this thesis, we shall adopt the convention of writing explicitly the base of logarithms. For example, the natural logarithm of $n$ is denoted by $\log_e(n)$, instead of $\ln(n)$. Also, the end of a proof will be denoted by the symbol $\blacksquare$.

## 1.3 Introduction to Quicksort

Sorting an array of items is clearly a fundamental problem, directly linked to efficient searching with numerous applications. The problem is that given an array of keys, we want to rearrange these in non-decreasing order. Note that the order may be numerical, alphabetical or any other transitive relation defined on the keys [46]. In this work, the analysis deals with numerical order, where the keys are decimal numbers and we particularly focus on Quicksort algorithm and variants of it. Quicksort was invented by C. A. R. Hoare [29, 30]. Here is the detailed definition.

**Definition 1.3.1.**

*The steps taken by the Quicksort algorithm are:*

1. *Choose an element from the array, called pivot.*

2. *Rearrange the array by comparing every element to the pivot, so all elements smaller than or equal to the pivot come before the pivot and all elements greater than or equal to the pivot come after the pivot.*
3. *Recursively apply steps* $1$ *and* $2$ *to the subarray of the elements smaller than or equal to the pivot and to the subarray of the elements greater than or equal to the pivot.*

Note that the original problem is divided into smaller ones, with (initially) two subarrays, the keys smaller than the pivot, and those bigger than it. Then recursively these are divided into smaller subarrays by further pivoting, until we get trivially sorted subarrays, which contain one or no elements. Given an array of $n$ distinct keys $A = \{a_1, a_2, \ldots, a_n\}$ that we want to quick sort, with all the $n!$ permutations equally likely, the aim is to finding the unique permutation out of all the $n!$ possible, such that the keys are in increasing order. The essence of Quicksort is the partition operation, where by a series of pairwise comparisons, the pivot is brought to its final place, with smaller elements on its left and greater elements to the right. Elements equal to pivot can be on either or both sides.

As we shall see, there are numerous partitioning schemes, and while the details of them are not central to this thesis, we should describe the basic ideas. A straightforward and natural way (see e.g. [46]) uses two pointers – a left pointer, initially at the left end of the array and a right pointer, initially at the right end of the array. We pick the leftmost element of the array as pivot and the right pointer scans from the right end of the array for a key less than the pivot. If it finds such a key, the pivot is swapped with that key. Then, the left pointer is increased by one and starts its scan, searching for a key greater than the pivot: if such a key is found, again the pivot is exchanged with it. When the pointers are crossed, the pivot by repeated exchanges will “float” to its final position and the keys which are on its left are smaller and keys on its right are greater. The data movement of this scheme is quite large, since the pivot is swapped with the other elements.

A different partitioning scheme, described in [30] is the following. Two pointers $i$ (the left pointer, initially $1$) and $j$ (the right pointer, initially $n$) are set and a key is arbitrarily chosen as pivot. The left pointer goes to the right until a key is found which is greater than the pivot. If one is found, its scan is stopped and the right pointer scans to the left until a key less than the pivot is found. If such a key is found, the right pointer stops and those two keys are exchanged. After the exchange, both pointers are stepped down one position and the lower one starts its scan. When pointers are crossed, i.e. when $i \geq j$, the final exchange places the pivot in its final position, completing the partitioning. The number of comparisons required to partition an array of $n$ keys is at least $n - 1$ and the expected number of exchanges is $\frac{n}{6} + \frac{5}{6n}$.

A third partitioning routine, called Lomuto’s partition, is mentioned in [6] – this involves exactly $n - 1$ comparisons, which is clearly best possible, but the downside is the increased number of exchanges. The expected number of key exchanges of this scheme is $\frac{n-1}{2}$, [48].

We now consider the worst case and best case, analysis of Quicksort. Suppose we want to sort the following array, $\{a_1 < a_2 < \ldots < a_n\}$ and we are very unlucky and our initial choice of pivot is the largest element $a_n$. Then of course we only divide and conquer in a rather trivial sense: every element is below the pivot, and it has taken us $n - 1$ comparisons with $a_n$ to get here. Suppose we now try again and are unlucky again, choosing $a_{n-1}$ as pivot this time. Again the algorithm performs $n-2$ comparisons and we are left with everything less than $a_{n-1}$. If we keep being unlucky in our choices of pivot, and keep choosing the largest element of what is left, after $i$ recursive calls the running time of the algorithm will be equal to $(n-1)+(n-2)+\ldots+(n-i)$ comparisons, so the overall number of comparisons made is

$$1+2+\ldots+(n-1)=\frac{n\cdot(n-1)}{2}.$$

Thus Quicksort needs quadratic time to sort already sorted or reverse-sorted arrays if the choice of pivots is unfortunate.

If instead we always made good choices, choosing each pivot to be roughly in the middle of the array we are considering at present, then in the first round we make $n-1$ comparisons, then in the two subarrays of size about $n/2$ we make about $n/2$ comparisons, then in each of the four subarrays of size about $n/4$ we make $n/4$ comparisons, and so on. So we make about $n$ comparisons in total in each round. The number of rounds will be roughly $\log_2(n)$ as we are splitting the arrays into roughly equally-sized subarrays at each stage, and it will take $\log_2(n)$ recursions of this to get down to trivially sorted arrays.

Thus, in this good case we will need $O\big(n\log_2(n)\big)$ comparisons. This is of course a rather informal argument, but does illustrate that the time complexity can be much smaller than the quadratic run-time in the worst case. This is already raising the question of what the typical time complexity will be: we address this in the next Chapter.

We briefly discuss the space complexity of the algorithm. There are $n$ memory locations occupied by the keys. Moreover, the algorithm, due to its recursive nature, needs additional space for the storage of subarrays. The subarrays' boundaries are saved on to a stack, which is a data structure providing temporary storage. At the end of the partition routine, the pivot is placed in its final position between two subarrays (one of them possibly empty). Recursively, the algorithm is applied to the smaller subarray and the other one is pushed on to stack. Since, in best and average case of Quicksort, we have $O\big(\log_2(n)\big)$ recursive calls, the required stack space is $O\big(\log_2(n)\big)$ locations in memory. However, in worst case the stack may require $O(n)$ locations, if the algorithm is applied to the larger subarray and the smaller one is saved to the stack [63].

This discussion makes it clear that the pivot selection plays a vital role in the performance of the algorithm. Many authors have proposed various techniques to remedy this situation and to avoid worst case behaviour, see [15], [30], [46], [62], [63] and [68]. These include the random shuffling of the array prior to initialisation of the algorithm, choosing as pivot the median of the array, or the median of a random sample of keys.

Scowen in his paper [62], suggested choosing as pivot the middle element of the array: his variant is dubbed "Quickersort". Using this rule for the choice of partitioning element, the aim is the splitting of the array into two halves of equal size. Thus, in case where the array is nearly sorted, quadratic time is avoided but if the chosen pivot is the minimum or maximum key, the algorithm's running time attains its worst case and this variant does not offer any more than choosing the pivot randomly. Singleton [68] suggested a better estimate of the median, by selecting as pivot the median of leftmost, rightmost and middle keys of the input array. Hoare [30] suggested the pivot may be chosen as the median of a random sample from the keys to be sorted, but he didn't analyse this approach.

One point is that Quicksort is not always very fast at sorting small arrays. Knuth [46] presented and analysed a partitioning scheme, which takes $n+1$ instead of $n-1$ comparisons and the sorting of small subarrays (usually from about 9 to 15 elements) is implemented using insertion sort, since the recursive structure of Quicksort is better suited to large arrays. Insertion sort is a simple sorting algorithm, which gradually 'constructs' a sorted array from left to right, in the following manner. The first two elements are compared and exchanged, in case that are not in order. Then, the third element is compared with the element on its left. If it is greater, it is left at its initial location, otherwise is compared with the first element and accordingly is inserted to its position in the sorted array of 3 elements. This process is iteratively applied to the remaining elements, until the array is sorted. See as well in Cormen *et al.* [15], for a description of the algorithm.

## 1.4 Outline and contributions of thesis

This thesis consists of seven Chapters and one Appendix. After the first, introductory Chapter, the rest of the thesis is organised as follows:

In **Chapter 2**, we consider the first and second moments of the number of comparisons made when pivots are chosen randomly. The result for the mean is known and easy: the result for the variance is known, but less easy to find a full proof of in the literature. We supply one. We briefly discuss the skewness of the number of comparisons and we study the asymptotic behaviour of the algorithm.

In **Chapter 3**, we analyse the idea of choosing the pivot as a centered statistic of a random sample of the keys to be sorted and we obtain the average number of comparisons required by these variants, showing that the running time can be greatly improved. Moreover, we present relevant definitions of entropy. Not much of this is original, but some details about why various differential equations that arise in the analysis have the solutions they do (i.e. details about roots of indicial polynomials) are not in literature.

In **Chapter 4**, we analyse extensions of Quicksort, where multiple pivots are used for the partitioning of the array. The main contributions in this Chapter are in sections **4.1** and **4.2**. The results in the former section were published in the paper [34], where the expected costs related to the time complexity and the second moment of the number of comparisons are computed. The latter section contains the analysis of the generalisation of the algorithm. We study the general recurrence model, giving the expected cost of the variant, provided that the cost during partitioning is linear, with respect to the number of keys. We also present the application of Vandermonde matrices for the computation of the constants involved to the cost of these variants.

In **Chapter 5**, various cases of partially ordered sets are discussed and the number of comparisons needed for the complete sorting is studied. The 'information–theoretic lower bound' is always $\omega(n)$ in these cases and we show that the time needed for the sorting of partial orders is $O\big(n\log_2(n)\big)$. The main contribution of this Chapter is the derivation of the asymptotic number of comparisons needed, for the sorting of various partially ordered sets. The basic ideas used here are due to, amongst others, Cardinal *et al.* [13], Fredman [25], Kahn and Kim [40], Kislitsyn [41], but the working out of the detailed consequences for these partial orders seems to be new.

In **Chapter 6**, we consider random graph orders, where the 'information–theoretic lower bound' is of the same order of magnitude as the number of keys being sorted. We derive a new bound on the number of linear extensions using entropy arguments, though it is not at present competitive with an older bound in the literature [4].

In **Chapter 7**, we conclude the thesis, presenting future research directions. At this final Chapter, we derive another bound of the number of comparisons required to sort a random interval order and we discuss the merging of linearly ordered sets.

In **Appendix A**, we present the MAPLE calculations, regarding the derivation of the variance of the number of comparisons of dual pivot Quicksort, analysed in subsection **4.1.1**.

# Chapter 2

# Random selection of pivot

In this Chapter, the mathematical analysis of Quicksort is presented, under the assumption that the pivots are uniformly selected at random. Specifically, the major expected costs regarding the time complexity of the algorithm and the second moment are computed. The derivation of the average costs is unified under a general recurrence relation, demonstrating the amenability of the algorithm to a complete mathematical treatment. We also address the asymptotic analysis of the algorithm and we close this Chapter considering the presence of equal keys.

## 2.1 Expected number of comparisons

This discussion of lucky and unlucky choices of pivot suggests the idea of selecting the pivot at random, as randomisation often helps to improve running time in algorithms with bad worst-case, but good average-case complexity [69]. For example, we could choose the pivots randomly for a discrete uniform distribution on the array we are looking at each stage. Recall that the uniform distribution on a finite set assigns equal probability to each element of it.

**Definition 2.1.1.** *$C_n$ is the random variable giving the number of comparisons in Quicksort of $n$ distinct elements when all the $n!$ permutations of the keys are equiprobable.*

It is clear that for $n = 0$ or $n = 1$, $C_0 = C_1 = 0$ as there is nothing to sort. These are the initial or "seed" values of the recurrence relation for the number of comparisons, given in the following Lemma.

**Lemma 2.1.2.** *The random number of comparisons $C_n$ for the sorting of an array consisting of $n \geq 2$ keys, is given by*

$$C_n = C_{U_n - 1} + C^{\star}_{n-U_n} + n - 1,$$

*where $U_n$ follows the uniform distribution over the set $\{1, 2, \dots, n\}$ and $C^{\star}_{n-U_n}$ is identically distributed to $C_{U_n-1}$ and independent of it conditional on $U_n$.*

**Proof.** The choice of $U_n$ as pivot, and comparing the other $n - 1$ elements with it, splits the array into two subarrays. There is one subarray of all $U_n - 1$ elements smaller than the pivot and another one of all $n - U_n$ elements greater than the pivot. Obviously these two subarrays are disjoint. Then recursively two pivots are randomly selected from the two subarrays, until the array is sorted, and so we get the equation. ∎

This allows us to find that the expected complexity of Quicksort applied to $n$ keys is:

$$\begin{aligned}\mathbb{E}(C_n) &= \mathbb{E}(C_{U_n-1} + C^{\star}_{n-U_n} + n - 1)\\ &= \mathbb{E}(C_{U_n-1}) + \mathbb{E}(C^{\star}_{n-U_n}) + n - 1.\end{aligned}$$

Using conditional expectation and noting that $U_n = k$ has probability $1/n$, we get, writing $a_k$ for $\mathbb{E}(C_k)$, that

$$a_n = \sum_{k=1}^{n} \frac{1}{n}(a_{k-1} + a_{n-k}) + n - 1 \Longrightarrow a_n = \frac{2}{n}\sum_{k=0}^{n-1} a_k + n - 1.$$

We have to solve this recurrence relation, in order to obtain a closed form for the expected number of comparisons. The following result is well-known (e.g. see in [15], [63]):

**Theorem 2.1.3.** *The expected number $a_n$ of comparisons for Quicksort with uniform selection of pivots is $a_n = 2(n+1)H_n - 4n$.*

**Proof.** We multiply both sides of the formula for $a_n$ by $n$, getting

$$na_n = 2\sum_{k=0}^{n-1} a_k + n(n-1)$$

and similarly, multiplying by $n-1$,

$$(n-1)a_{n-1} = 2\sum_{k=0}^{n-2} a_k + (n-2)(n-1).$$

Subtracting $(n-1)a_{n-1}$ from $na_n$ in order to eliminate the sum – see [46], we obtain

$$\begin{aligned} na_n - (n-1)a_{n-1} &= 2a_{n-1} + 2(n-1) \\ &\Longrightarrow na_n = (n+1)a_{n-1} + 2(n-1) \\ &\Longrightarrow \frac{a_n}{n+1} = \frac{a_{n-1}}{n} + \frac{2(n-1)}{n(n+1)}. \end{aligned}$$

“Unfolding” the recurrence we get

$$\begin{aligned}\frac{a_n}{n+1} &= 2\sum_{j=2}^{n}\frac{(j-1)}{j(j+1)} = 2\sum_{j=2}^{n}\left(\frac{2}{j+1}-\frac{1}{j}\right)\\ &= 4H_n + \frac{4}{n+1} - 4 - 2H_n\\ &= 2H_n + \frac{4}{n+1} - 4.\end{aligned}$$

Finally,

$$a_n = 2(n+1)H_n - 4n. \qquad \blacksquare$$

We now show a slick way of solving the recurrence about the expected number of comparisons using generating functions. This approach is also noted in various places, e.g. [44], [55], [63]. We again start from

$$a_n = \frac{2}{n}\sum_{j=0}^{n-1} a_j + n - 1.$$

We multiply through by $n$ to clear fractions, getting

$$\begin{aligned} na_n &= n(n-1) + 2\sum_{j=0}^{n-1} a_j\\ \implies \sum_{n=1}^{\infty} na_n x^{n-1} &= \sum_{n=1}^{\infty} n(n-1)x^{n-1} + 2\sum_{n=1}^{\infty}\sum_{j=0}^{n-1} a_j x^{n-1}\\ \implies \sum_{n=1}^{\infty} na_n x^{n-1} &= x\sum_{n=2}^{\infty} n(n-1)x^{n-2} + 2\sum_{n=1}^{\infty}\sum_{j=0}^{n-1} a_j x^{n-1}.\end{aligned}$$

Letting $f(x) = \sum_{n=0}^{\infty} a_n x^n$,

$$f'(x) = x\sum_{n=2}^{\infty} n(n-1)x^{n-2} + 2\sum_{j=0}^{\infty}\sum_{n=j+1}^{\infty} a_j x^{n-1},$$

changing round the order of summation. To evaluate $\sum_{n=2}^{\infty} n(n-1)x^{n-2}$, simply note this is the 2nd derivative of $\sum_{n=0}^{\infty} x^n$. Since the latter sum is equal to $1/(1-x)$, for $|x| < 1$, its second derivative is easily checked to be $2(1-x)^{-3}$. Multiplying the sum by $x$ now gives

$$f'(x) = 2x(1-x)^{-3} + 2\sum_{j=0}^{\infty} a_j \sum_{n=j+1}^{\infty} x^{n-1}.$$

We evaluate the last double-sum. The inner sum is of course $x^j/(1-x)$ being a geometric series. Thus we get

$$\begin{aligned} f'(x) &= 2x(1-x)^{-3} + \frac{2}{1-x}\sum_{j=0}^{\infty} a_j x^j \\ &= 2x(1-x)^{-3} + \frac{2}{1-x} f(x). \end{aligned}$$

This is now a fairly standard kind of differential equation. Multiplying both sides by $(1-x)^2$, we see

$$\begin{aligned} (1-x)^2 f'(x) &= 2x(1-x)^{-1} + 2(1-x)f(x) \\ \Longrightarrow (1-x)^2 f'(x) - 2(1-x)f(x) &= \frac{2}{1-x} - 2 \\ \Longrightarrow \big((1-x)^2 f(x)\big)' &= \frac{2}{1-x} - 2 \\ \Longrightarrow (1-x)^2 f(x) &= -2\log_e(1-x) - 2x + c. \end{aligned}$$

Setting $x = 0$, we get $f(0) = 0$ on the left-hand side, and on the right-hand side we get $c$, so $c = 0$. Therefore

$$f(x) = \frac{-2\Big(\log_e(1-x) + x\Big)}{(1-x)^2}.$$

Expanding out $\log_e(1-x)$ as a series, $-x - x^2/2 - x^3/3 - \ldots$, and similarly writing $1/(1-x)^2$ as the derivative of $1/(1-x)$, we obtain

$$f(x) = 2\sum_{k=2}^{\infty} \frac{x^k}{k} \sum_{j=0}^{\infty} (j+1)x^j.$$

Thus, looking at coefficients of $x^n$ on both sides, we get on the left-hand side $a_n$. On the right-hand side, we get the coefficient for each $x^j$ in the first series (which is $1/j$) times the term for the $x^{n-j}$ in the other, namely $(n-j+1)$. So we get

$$\begin{aligned} a_n &= 2\sum_{k=2}^{n} \frac{n-k+1}{k} = 2\sum_{k=2}^{n} \frac{n+1}{k} - 2\sum_{k=2}^{n} 1 \\ &= 2(n+1)\Bigg(\sum_{k=1}^{n} \frac{1}{k} - 1\Bigg) - 2(n-1) \\ &= 2(n+1)H_n - 4n. \end{aligned}$$

Some texts give a different argument for this, as follows.

**Theorem 2.1.4** (Mitzenmacher and Upfal [54]). *Suppose that a pivot is chosen independently and uniformly at random from an array of $n$ keys, in which Quicksort is applied. Then, for any input, the expected number of comparisons made by randomised Quicksort is $2n\log_e(n) + O(n)$.*

**Proof.** Let $\{x_1, x_2, \ldots, x_n\}$ be the input values and the output, after the termination of Quicksort, (i.e. keys in increasing order) be a permutation of the initial array $\{y_1, y_2, \ldots, y_n\}$. For $i < j$, let $X_{ij}$ be a $\{0, 1\}$-valued random variable, that takes the value $1$, if $y_i$ and $y_j$ are compared over the course of algorithm and $0$, otherwise. Then, the total number of comparisons $X$ satisfies

$$\begin{aligned} X &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \\ \Longrightarrow \mathbb{E}(X) &= \mathbb{E}\left( \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \right) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbb{E}(X_{ij}). \end{aligned}$$

Since $X_{ij} \in \{0, 1\}$, then $\mathbb{E}(X_{ij}) = \mathbb{P}(X_{ij} = 1)$. This event occurs when $y_i$ and $y_j$ are compared. Clearly, this happens when $y_i$ or $y_j$ is the first number chosen as pivot from the set $Y = \{y_i, y_{i+1}, \ldots, y_{j-1}, y_j\}$. (Because otherwise if some element between them is chosen, $y_k$ say, $y_i$ and $y_j$ are compared to $y_k$ and $y_i$ is put below $y_k$ and $y_j$ above it with the result that $y_i$ and $y_j$ are never compared).

Since the pivot is chosen uniformly at random, the probability the one of these two elements is the first of the $j - i + 1$ elements chosen is equal to $\frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1}$. Substituting $k = j - i + 1$, we obtain

$$\mathbb{E}(X) = \mathbb{E}\left( \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \right) = \sum_{i=1}^{n-1} \sum_{k=2}^{n-i+1} \frac{2}{k} = \sum_{k=2}^{n} \sum_{i=1}^{n+1-k} \frac{2}{k}.$$

The change of sum is justified by writing out the possible rows for fixed $i$ and columns for a fixed $k$, then changing round from summing rows first to summing columns first. This is, as $2/k$ does not depend on $i$, equal to

$$\sum_{k=2}^{n} (n + 1 - k) \frac{2}{k} = (n + 1) \sum_{k=2}^{n} \frac{2}{k} - 2 \sum_{k=2}^{n} 1.$$

Thus,

$$\begin{aligned} 2(n+1)\sum_{k=2}^{n}\frac{1}{k} - 2n + 2 &= 2(n+1)(H_n - 1) - 2n + 2 \\ &= 2(n+1)H_n - 4n, \end{aligned}$$

proving the claim. ∎

Also of some interest is the mean number of partitioning stages $\mathbb{E}(P_n)$ of the algorithm applied to $n$ keys as input. For the case where we simply use Quicksort for all the sorting, it is obvious that we will have $P_0 = 0$ and for $n \geq 1$, the number of partitioning stages $P_n$ obeys the following recurrence conditional on $j$ being the rank of the pivot

$$P_n = 1 + P_{j-1} + P_{n-j}.$$

Observe that $P_1 = 1$ and $P_2 = 2$. Taking expectations, and noting that the pivot is uniformly chosen at random and that the two sums $\sum_j \mathbb{E}(P_{j-1})$ and $\sum_j \mathbb{E}(P_{n-j})$ are equal, we see

$$\mathbb{E}(P_n) = 1 + \frac{2}{n}\sum_{j=0}^{n-1}\mathbb{E}(P_j).$$

Multiplying both sides by $n$ and differencing the recurrence relation, as we did for the derivation of the expected number of comparisons, we have

$$\begin{aligned} n\mathbb{E}(P_n) - (n-1)\mathbb{E}(P_{n-1}) &= 1 + 2\mathbb{E}(P_{n-1}) \\ \implies \frac{\mathbb{E}(P_n)}{n+1} &= \frac{1}{n(n+1)} + \frac{\mathbb{E}(P_{n-1})}{n} \\ \implies \mathbb{E}(P_n) &= (n+1)\Bigg(\sum_{j=2}^{n}\Big(\frac{1}{j} - \frac{1}{j+1}\Big) + \frac{1}{2}\Bigg). \end{aligned}$$

Finally, $\mathbb{E}(P_n) = n$.

## 2.2 Expected number of exchanges

Here we consider the number of exchanges or swaps performed by the algorithm, which is mentioned by Hoare [30] as a relevant quantity. We assume that each swap has a fixed cost and as in the previous section, we assume that the keys are distinct and that all $n!$ permutations are equally likely to be the input: this in particular implies that the pivot is chosen uniformly at random from the array.

We should specify the partitioning procedure. Assume that we have to sort $n$ distinct keys, where their locations in the array are numbered from left to right by $1, 2, \dots, n$. Set two pointers $i \leftarrow 1$ and $j \leftarrow n-1$ and select the element at location $n$ as a pivot. First, compare the element at location $1$ with the pivot. If this key is less than the pivot, increase $i$ by one until an element greater than the pivot is found. If an element greater than the pivot is found, stop and compare the element at location $n-1$ with the pivot. If this key is greater than the pivot, then decrease $j$ by one and compare the next element to the pivot.

If an element less than the pivot is found, then the $j$ pointer stops its scan and the keys that the two pointers refer are exchanged.

Increase $i$ by one, decrease $j$ by one and in the same manner continue the scanning of the array until $i \geq j$. At the end of the partitioning operation, the pivot is placed in its final position $k$, where $1 \leq k \leq n$, and Quicksort is recursively invoked to sort the subarray of $k-1$ keys less than the pivot and the subarray of $n-k$ keys greater than the pivot [29, 30].

Note that the probability of a key being greater than the pivot is

$$\frac{n-k}{n-1}.$$

The number of keys which are greater than pivot, and were moved during partition is

$$\frac{n-k}{n-1} \cdot (k-1).$$

Therefore, considering also that pivots are uniformly chosen and noting that we have to count the final swap with the pivot at the end of partition operation, we obtain

$$\sum_{k=1}^{n} \frac{(n-k)(k-1)}{n(n-1)} + 1 = \frac{n}{6} + \frac{2}{3}.$$

Let $S_n$ be the total number of exchanges, when the algorithm is applied to an array of $n$ distinct keys. We have that $S_0 = S_1 = 0$ and for $n \geq 2$, the following recurrence holds

$$S_n = \text{\textit{“Number of exchanges during partition routine”}} + S_{k-1} + S_{n-k}.$$

Since the pivot is chosen uniformly at random, the recurrence for the expected number of exchanges is

$$\begin{aligned}\mathbb{E}(S_n) &= \frac{n}{6} + \frac{2}{3} + \frac{1}{n}\sum_{k=1}^{n}\Big(\mathbb{E}(S_{k-1}) + \mathbb{E}(S_{n-k})\Big)\\ \Longrightarrow \mathbb{E}(S_n) &= \frac{n}{6} + \frac{2}{3} + \frac{2}{n}\sum_{k=0}^{n-1}\mathbb{E}(S_k).\end{aligned}$$

This recurrence relation is similar to the recurrences about the mean number of comparisons and will be solved by the same way. Subtracting $(n-1)\mathbb{E}(S_{n-1})$ from $n\mathbb{E}(S_n)$, the recurrence becomes

$$\begin{aligned}n\mathbb{E}(S_n) - (n-1)\mathbb{E}(S_{n-1}) &= \frac{2n+3}{6} + 2\mathbb{E}(S_{n-1})\\ \Longrightarrow \frac{\mathbb{E}(S_n)}{n+1} &= \frac{2n+3}{6n(n+1)} + \frac{\mathbb{E}(S_{n-1})}{n}.\end{aligned}$$

Telescoping, the last relation yields

$$\begin{aligned}\frac{\mathbb{E}(S_n)}{n+1} &= \sum_{j=3}^{n}\frac{2j+3}{6j(j+1)} + \frac{1}{3} = \sum_{j=3}^{n}\frac{1}{3(j+1)} + \sum_{j=3}^{n}\frac{1}{2j(j+1)} + \frac{1}{3}\\ &= \frac{1}{3}\sum_{j=3}^{n}\frac{1}{j+1} + \frac{1}{2}\left(\sum_{j=3}^{n}\frac{1}{j} - \sum_{j=3}^{n}\frac{1}{j+1}\right) + \frac{1}{3}\\ &= \frac{1}{3}\left(H_{n+1} - \frac{11}{6}\right) + \frac{1}{2}\left(\frac{1}{3} - \frac{1}{n+1}\right) + \frac{1}{3}.\end{aligned}$$

Tidying up, the average number of exchanges in course of the algorithm is

$$\mathbb{E}(S_n) = \frac{(n+1)H_n}{3} - \frac{n}{9} - \frac{5}{18}.$$

Its asymptotic value is

$$\frac{n\log_e(n)}{3}.$$

It follows that asymptotically the mean number of exchanges is about $1/6$ of the average number of comparisons.

In a variant of the algorithm analysed in [46] and [63], which we briefly mentioned in the introduction, partitioning of $n$ keys takes $n+1$ comparisons and subfiles of $m$ or fewer elements are sorted using insertion sort. Then the average number of comparisons, partitioning stages and exchanges respectively, are

$$\begin{aligned}
\mathbb{E}(C_n) &= (n+1)(2H_{n+1} - 2H_{m+2} + 1), \\
\mathbb{E}(P_n) &= 2 \cdot \frac{n+1}{m+2} - 1 \quad \text{and} \\
\mathbb{E}(S_n) &= (n+1)\Big(\frac{1}{3}H_{n+1} - \frac{1}{3}H_{m+2} + \frac{1}{6} - \frac{1}{m+2}\Big) + \frac{1}{2}, \text{ for } n > m.
\end{aligned}$$

For $m = 0$, we obtain the average quantities when there is no switch to insertion sort. Note that in this case the expected costs are

$$\begin{aligned}
\mathbb{E}(C_n) &= 2(n+1)H_n - 2n, \\
\mathbb{E}(P_n) &= n \quad \text{and} \\
\mathbb{E}(S_n) &= \frac{2(n+1)H_n - 5n}{6}.
\end{aligned}$$

## 2.3 Variance

We now similarly discuss the variance of the number of comparisons in Quicksort. Although the result has been known for many years – see [46], exercise 6.2.2-8 for an outline – there is not really a full version of all the details written down conveniently that we are aware of, so we have provided such an account

– this summary has been put on **arXiv**, [33]. The sources [43], [55] and [63] were useful in putting the argument together. Again, generating functions will be used. The result is:

**Theorem 2.3.1.** *The variance of the number of comparisons of Quicksort on $n$ keys, with a pivot chosen uniformly at random is*

$$\mathrm{Var}(C_n) = 7n^2 - 4(n+1)^2 H_n^{(2)} - 2(n+1)H_n + 13n.$$

We start with a recurrence for the generating function of $C_n$, namely $f_n(z) = \sum_{k=0}^{\infty} \mathbb{P}(C_n = k)z^k$. We will use this to reduce the proof of the Theorem to proving a certain recurrence formula for the expression $\frac{f_n''(1)}{2}$.

**Theorem 2.3.2.** *In Random Quicksort of $n$ keys, the generating functions $f_i$ satisfy*

$$f_n(z) = \frac{z^{n-1}}{n} \sum_{j=1}^{n} f_{j-1}(z) f_{n-j}(z).$$

**Proof.** We have, using the equation

$$C_n = C_{U_n - 1} + C^*_{n-U_n} + n - 1$$

that

$$\begin{aligned}\mathbb{P}(C_n = k) &= \sum_{m=1}^{n} \mathbb{P}(C_n = k | U_n = m)\frac{1}{n}\\ &= \sum_{m=1}^{n} \sum_{j=1}^{k-(n-1)} \mathbb{P}(C_{m-1} = j)\mathbb{P}(C_{n-m} = k-(n-1)-j)\frac{1}{n},\end{aligned}$$

noting that $C_{m-1}$ and $C_{n-m}$ are conditionally independent subject to the pivot. Thus

$$\mathbb{P}(C_n = k)z^k = \frac{1}{n}\sum_{m=1}^{n}\sum_{j=1}^{k-(n-1)} \mathbb{P}(C_{m-1} = j)z^j\mathbb{P}(C_{n-m} = k-(n-1)-j)z^{k-(n-1)-j}z^{n-1}.$$

Multiplying by $z^k$ and summing over $k$, so as to get the generating function $f_n$ of $C_n$ on the left, we obtain

$$\begin{aligned} f_n(z) &= \frac{1}{n}\sum_{k=1}^{n-1+j}\sum_{m=1}^{n}\sum_{j=1}^{k-(n-1)} \mathbb{P}(C_{m-1} = j)z^j\mathbb{P}(C_{n-m} = k-(n-1)-j)z^{k-(n-1)-j}z^{n-1} \\ &= \frac{z^{n-1}}{n}\sum_{m=1}^{n}\sum_{j=1}^{k-(n-1)} \mathbb{P}(C_{m-1} = j)z^j \sum_{k=1}^{n-1+j}\mathbb{P}(C_{n-m} = k-(n-1)-j)z^{k-(n-1)-j} \\ &= \frac{z^{n-1}}{n}\sum_{m=1}^{n} f_{m-1}(z)f_{n-m}(z), \qquad (2.1) \end{aligned}$$

as required. ∎

This of course will give a recursion for the variance, using the well-known formula for variance in terms of the generating function $f_X(z)$:

$$\mathrm{Var}(X) = f_X''(1) + f_X'(1) - \left(f_X'(1)\right)^2.$$

We use this formula together with Eq. (2.1). The first order derivative of $f_n(z)$ is

$$\begin{aligned} f_n'(z) = \frac{(n-1)z^{n-2}}{n}\sum_{j=1}^{n} f_{j-1}(z)f_{n-j}(z) + \frac{z^{n-1}}{n}\sum_{j=1}^{n} f_{j-1}'(z)f_{n-j}(z) \\ + \frac{z^{n-1}}{n}\sum_{j=1}^{n} f_{j-1}(z)f_{n-j}'(z). \end{aligned}$$

From standard properties of generating functions, it holds that

$$f_n'(1) = \mathbb{E}(C_n).$$

Differentiating again we obtain

$$\begin{aligned}
f_n''(z) = {} & \frac{(n-1)(n-2)z^{n-3}}{n}\sum_{j=1}^{n} f_{j-1}(z)f_{n-j}(z) + \frac{(n-1)z^{n-2}}{n}\sum_{j=1}^{n} f_{j-1}'(z)f_{n-j}(z) \\
& + \frac{(n-1)z^{n-2}}{n}\sum_{j=1}^{n} f_{j-1}(z)f_{n-j}'(z) + \frac{(n-1)z^{n-2}}{n}\sum_{j=1}^{n} f_{j-1}'(z)f_{n-j}(z) \\
& + \frac{z^{n-1}}{n}\sum_{j=1}^{n} f_{j-1}''(z)f_{n-j}(z) + \frac{z^{n-1}}{n}\sum_{j=1}^{n} f_{j-1}'(z)f_{n-j}'(z) \\
& + \frac{(n-1)z^{n-2}}{n}\sum_{j=1}^{n} f_{j-1}(z)f_{n-j}'(z) + \frac{z^{n-1}}{n}\sum_{j=1}^{n} f_{j-1}'f_{n-j}'(z) \\
& + \frac{z^{n-1}}{n}\sum_{j=1}^{n} f_{j-1}(z)f_{n-j}''(z).
\end{aligned}$$

Setting $z = 1$ [55],

$$\begin{aligned}
f_n''(1) = {} & (n-1)(n-2) + \frac{2}{n}(n-1)\sum_{j=1}^{n} M_{j-1} + \frac{2}{n}(n-1)\sum_{j=1}^{n} M_{n-j} \\
& + \frac{1}{n}\sum_{j=1}^{n}\big(f_{j-1}''(1) + f_{n-j}''(1)\big) + \frac{2}{n}\sum_{j=1}^{n} M_{j-1}M_{n-j},
\end{aligned}$$

where $M_{j-1}, M_{n-j}$ are $f_{j-1}'(1)$, $f_{n-j}'(1)$, i.e. the mean number of comparisons to sort an array of $(j-1)$ and $(n-j)$ elements respectively. Setting $B_n = \frac{f_n''(1)}{2}$, we obtain

$$B_n = \binom{n-1}{2} + \frac{2(n-1)}{n}\sum_{j=1}^{n} M_{j-1} + \frac{2}{n}\sum_{j=1}^{n} B_{j-1} + \frac{1}{n}\sum_{j=1}^{n} M_{j-1}M_{n-j},$$

using the symmetry of the sums. What this argument has shown for us is the following – compare [43] where it is also shown that this recurrence has to be solved, though no details of how to solve it are given.

**Lemma 2.3.3.** *In order to prove Theorem* 2.3.1*, it is sufficient to show that this recurrence is given by*

$$B_n = 2(n+1)^2 H_n^2 - (8n+2)(n+1)H_n + \frac{n(23n+17)}{2} - 2(n+1)^2 H_n^{(2)}.$$

**Proof.** Combining the equations $B_n = \frac{f_n''(1)}{2}$ and $\mathrm{Var}(X) = f_X''(1) + f_X'(1) - \big(f_X'(1)\big)^2$, the result follows. ∎

It will be convenient first to develop some theory on various sums involving harmonic numbers. Often we used MAPLE to verify these relations initially, but we provide complete proofs here. As a brief reminder, $M_j$ denotes the expected number of comparisons needed to sort an array of $j$ keys. Recall that

$$M_j = 2(j+1)H_j - 4j.$$

Thus,

$$\sum_{j=1}^{n} M_{j-1} = \sum_{j=1}^{n} \big(2jH_{j-1} - 4(j-1)\big) = 2\sum_{j=1}^{n} jH_{j-1} - 4\sum_{j=1}^{n}(j-1). \qquad (2.2)$$

For the computation of the first sum of Eq. (2.2), a Lemma follows

**Lemma 2.3.4.** *For* $n \in \mathbf{N}$

$$\sum_{j=1}^{n} jH_{j-1} = \frac{n(n+1)H_{n+1}}{2} - \frac{n(n+5)}{4}.$$

**Proof.** The sum can be written as

$$\begin{aligned}\sum_{j=1}^{n} jH_{j-1} &= 2+3\left(1+\frac{1}{2}\right)+\ldots+n\left(1+\frac{1}{2}+\ldots+\frac{1}{n-1}\right)\\ &= H_{n-1}(1+2+\ldots+n)-\left(1+\frac{1+2}{2}+\ldots+\frac{1+2+\ldots+n-1}{n-1}\right)\\ &= \frac{n(n+1)}{2}H_{n-1}-\sum_{j=1}^{n-1}\left(\frac{\sum_{i=1}^{j} i}{j}\right)\\ &= \frac{n(n+1)}{2}H_{n-1}-\frac{1}{2}\left(\frac{n(n+1)}{2}-1\right).\end{aligned}$$

The last equation can be easily seen to be equivalent with the statement of the Lemma. ∎

Thus we can find out about the sum of the $M_j$s, that it holds for $n \in \mathbf{N}$

**Corollary 2.3.5.**

$$\sum_{j=1}^{n} M_{j-1} = n(n+1)H_{n+1}-\frac{5n^2+n}{2}.$$

**Proof.** Using Lemma 2.3.4 and Eq. (2.2), the proof is immediate. ∎

Now, we will compute the term $\sum_{j=1}^{n} M_{j-1}M_{n-j}$. We shall use three Lemmas for the following proof.

**Lemma 2.3.6.** *For $n \in \mathbf{N}$, it holds that*

$$\begin{aligned}\sum_{j=1}^{n} M_{j-1}M_{n-j} = 4\sum_{j=1}^{n} &jH_{j-1}(n-j+1)H_{n-j}\\ &-\frac{8}{3}n(n^2-1)H_{n+1}+\frac{44n}{9}(n^2-1).\end{aligned}$$

**Proof.** To do this, we will again use the formula obtained previously for $M_j$. We have

$$\begin{aligned}\sum_{j=1}^{n} M_{j-1}M_{n-j} &= \sum_{j=1}^{n}\Big(\big(2jH_{j-1}-4j+4\big)\big(2(n-j+1)H_{n-j}-4n+4j\big)\Big)\\ &= 4\sum_{j=1}^{n} jH_{j-1}(n-j+1)H_{n-j} - 8n\sum_{j=1}^{n} jH_{j-1} + 8\sum_{j=1}^{n} j^2H_{j-1}\\ &\quad - 8\sum_{j=1}^{n} j(n-j+1)H_{n-j} + 16n\sum_{j=1}^{n} j - 16\sum_{j=1}^{n} j^2\\ &\quad + 8\sum_{j=1}^{n}(n-j+1)H_{n-j} - 16n^2 + 16\sum_{j=1}^{n} j.\end{aligned}$$

We need to work out the value of $\sum_{j=1}^{n} j^2H_{j-1}$:

**Lemma 2.3.7.** *For $n \in \mathbf{N}$ holds*

$$\sum_{j=1}^{n} j^2H_{j-1} = \frac{6n(n+1)(2n+1)H_{n+1} - n(n+1)(4n+23)}{36}.$$

**Proof.** Using the same reasoning as in Lemma 2.3.4,

$$\begin{aligned}\sum_{j=1}^{n} j^2H_{j-1} &= 2^2 + 3^2\left(1+\frac{1}{2}\right) + \ldots + n^2\left(1+\frac{1}{2}+\ldots+\frac{1}{n-1}\right)\\ &= H_{n-1}(1^2+2^2+\ldots+n^2)\\ &\quad - \left(1+\frac{1^2+2^2}{2}+\ldots+\frac{1^2+2^2+\ldots+(n-1)^2}{n-1}\right)\\ &= \frac{n(n+1)(2n+1)}{6}H_{n-1} - \sum_{j=1}^{n-1}\left(\frac{\sum_{i=1}^{j} i^2}{j}\right)\\ &= \frac{n(n+1)(2n+1)}{6}H_{n-1} - \frac{1}{36}(4n^3+3n^2-n-6),\end{aligned}$$

completing the proof. ∎

We also need to compute $\sum_{j=1}^{n} j(n-j+1)H_{n-j}$. A Lemma follows

**Lemma 2.3.8.** *For* $n \in \mathbf{N}$

$$\sum_{j=1}^{n} j(n-j+1)H_{n-j} = \frac{6nH_{n+1}(n^2+3n+2)-5n^3-27n^2-22n}{36}.$$

**Proof.** We can write $j = n+1-(n-j+1)$. Then, substituting $k = n-j+1$

$$\begin{aligned}\sum_{j=1}^{n} j(n-j+1)H_{n-j} &= \sum_{j=1}^{n}\big((n+1)-(n-j+1)\big)(n-j+1)H_{n-j}\\ &= (n+1)\sum_{j=1}^{n}(n-j+1)H_{n-j} - \sum_{j=1}^{n}(n-j+1)^2H_{n-j}\\ &= (n+1)\sum_{k=1}^{n}kH_{k-1} - \sum_{k=1}^{n}k^2H_{k-1}.\end{aligned}$$

These sums can be computed using Lemmas 2.3.4 and 2.3.7. ∎

In the same manner we shall compute $\sum_{j=1}^{n}(n-j+1)H_{n-j}$. Changing variables, the expression becomes $\sum_{k=1}^{n} kH_{k-1}$. Using the previous results, we have

$$\begin{aligned}
\sum_{j=1}^{n} M_{j-1}M_{n-j} &= 4\sum_{j=1}^{n} jH_{j-1}(n-j+1)H_{n-j} - 8n\left(\frac{n(n+1)H_{n+1}}{2} - \frac{n(n+5)}{4}\right) \\
&\quad + 8\left(\frac{6n(n+1)(2n+1)H_{n+1} - n(n+1)(4n+23)}{36}\right) + 16n\sum_{j=1}^{n} j \\
&\quad - 8\left(\frac{6nH_{n+1}(n^2+3n+2) - 5n^3 - 27n^2 - 22n}{36}\right) - 16\sum_{j=1}^{n} j^2 \\
&\quad + 8\left(\frac{n(n+1)H_{n+1}}{2} - \frac{n(n+5)}{4}\right) - 16n^2 + 16\sum_{j=1}^{n} j \\
&= 4\sum_{j=1}^{n} jH_{j-1}(n-j+1)H_{n-j} - 4n(n+1)(n-1)H_{n+1} \\
&\quad + \frac{4}{3}n(n^2-1)H_{n+1} + \frac{176n^3}{36} - \frac{176n}{36},
\end{aligned}$$

finishing the proof of Lemma 2.3.6. ∎

After some tedious calculations, the recurrence relation becomes

$$\begin{aligned}
B_n = {}& \frac{4\sum_{j=1}^{n} jH_{j-1}(n-j+1)H_{n-j}}{n} + \frac{2}{n}\sum_{j=1}^{n} B_{j-1} + \frac{-9n^2+5n+4}{2} \\
& - \frac{2}{3}(n^2-1)H_{n+1} + \frac{44}{9}(n^2-1).
\end{aligned}$$

Subtracting $nB_n$ from $(n+1)B_{n+1}$,

$$\begin{aligned}
&(n+1)B_{n+1} - nB_n \\
&= 4\Bigg(\sum_{j=1}^{n+1} jH_{j-1}(n-j+2)H_{n+1-j} - \sum_{j=1}^{n} jH_{j-1}(n-j+1)H_{n-j}\Bigg) \\
&\quad + 2B_n + (n+1)\frac{-9(n+1)^2+5(n+1)+4}{2} - n\frac{-9n^2+5n+4}{2} \\
&\quad - (n+1)\frac{2}{3}\big((n+1)^2-1\big)H_{n+2} + \frac{2}{3}n(n^2-1)H_{n+1} \\
&\quad + (n+1)\frac{44}{9}\big((n+1)^2-1\big) - n\frac{44}{9}(n^2-1) \\
&= 4\Bigg(\sum_{j=1}^{n} jH_{j-1}(n-j+2)H_{n+1-j} - \sum_{j=1}^{n} jH_{j-1}(n-j+1)H_{n-j}\Bigg) \\
&\quad + 2B_n - \frac{27n^2+17n}{2} - \frac{2}{3}n(n+1)(n+2)H_{n+2} \\
&\qquad + \frac{2}{3}nH_{n+1}(n^2-1) + \frac{44n(n+1)}{3}.
\end{aligned}$$

We obtain

$$\begin{aligned}
(n+1)B_{n+1} - nB_n = 4&\Bigg(\sum_{j=1}^{n} jH_{j-1} + \sum_{j=1}^{n} jH_{j-1}H_{n-j+1}\Bigg) \\
&+ 2B_n - 2n(n+1)H_{n+1} + \frac{1}{2}n(n+11).
\end{aligned}$$

We have to work out the following sum

$$\sum_{j=1}^{n} jH_{j-1}H_{n+1-j}.$$

We note that

$$\begin{aligned}\sum_{j=1}^{n} jH_{j-1}H_{n+1-j} &= 2H_1H_{n-1} + 3H_2H_{n-2} + \ldots + (n-1)H_{n-2}H_2 + nH_{n-1}H_1 \\ &= \frac{n+2}{2}\sum_{j=1}^{n} H_jH_{n-j}. \qquad (2.3)\end{aligned}$$

Sedgewick [63], presents and proves the following result:

**Lemma 2.3.9.**

$$\sum_{i=1}^{n} H_iH_{n+1-i} = (n+2)(H_{n+1}^2 - H_{n+1}^{(2)}) - 2(n+1)(H_{n+1}-1).$$

**Proof.** By the definition of harmonic numbers, we have

$$\sum_{i=1}^{n} H_iH_{n+1-i} = \sum_{i=1}^{n} H_i \sum_{j=1}^{n+1-i} \frac{1}{j}$$

and the above equation becomes

$$\sum_{j=1}^{n} \frac{1}{j} \sum_{i=1}^{n+1-j} H_i = \sum_{j=1}^{n} \frac{1}{j}\big((n+2-j)H_{n+1-j} - (n+1-j)\big), \qquad (2.4)$$

using the identity [44],

$$\sum_{j=1}^{n} H_j = (n+1)H_n - n.$$

Eq. (2.4) can be written as

$$\begin{aligned}
&(n+2)\sum_{j=1}^{n}\frac{H_{n+1-j}}{j}-\sum_{j=1}^{n}H_{n+1-j}-(n+1)H_n+n\\
&=(n+2)\sum_{j=1}^{n}\frac{H_{n+1-j}}{j}-\big((n+1)H_n-n\big)-(n+1)H_n+n\\
&=(n+2)\sum_{j=1}^{n}\frac{H_{n+1-j}}{j}-2(n+1)(H_{n+1}-1).
\end{aligned}$$

It can be easily verified that

$$\begin{aligned}
\sum_{j=1}^{n}\frac{H_{n+1-j}}{j}&=\sum_{j=1}^{n}\frac{H_{n-j}}{j}+\sum_{j=1}^{n}\frac{1}{j(n+1-j)}\\
&=\sum_{j=1}^{n-1}\frac{H_{n-j}}{j}+2\frac{H_n}{n+1}. \qquad (2.5)
\end{aligned}$$

Making repeated use of Eq. (2.5), we obtain the identity

$$\sum_{j=1}^{n}\frac{H_{n+1-j}}{j}=2\sum_{k=1}^{n}\frac{H_k}{k+1}.$$

We have then

$$\begin{aligned}
2\sum_{k=1}^{n}\frac{H_k}{k+1}&=2\sum_{k=2}^{n+1}\frac{H_{k-1}}{k}=2\sum_{k=1}^{n+1}\frac{H_{k-1}}{k}=2\sum_{k=1}^{n+1}\frac{H_k}{k}-2\sum_{k=1}^{n+1}\frac{1}{k^2}\\
&=2\sum_{k=1}^{n+1}\sum_{j=1}^{k}\frac{1}{jk}-2H_{n+1}^{(2)}=2\sum_{j=1}^{n+1}\sum_{k=j}^{n+1}\frac{1}{jk}-2H_{n+1}^{(2)}\\
&=2\sum_{k=1}^{n+1}\sum_{j=k}^{n+1}\frac{1}{kj}-2H_{n+1}^{(2)}.
\end{aligned}$$

The order of summation was interchanged. We can sum on all $j$ and for $k = j$, we must count this term twice. We obtain

$$2\sum_{k=1}^{n}\frac{H_k}{k+1} = \sum_{k=1}^{n+1}\left(\sum_{j=1}^{n+1}\frac{1}{kj}+\frac{1}{k^2}\right) - 2H_{n+1}^{(2)} = H_{n+1}^2 - H_{n+1}^{(2)}. \tag{2.6}$$

Finally

$$\sum_{i=1}^{n} H_i H_{n+1-i} = (n+2)(H_{n+1}^2 - H_{n+1}^{(2)}) - 2(n+1)(H_{n+1}-1). \qquad \blacksquare$$

The following Corollary is a direct consequence of Eq. (2.5) and (2.6).

**Corollary 2.3.10.** *For $n \in \mathbb{N}$, it holds*

$$H_{n+1}^2 - H_{n+1}^{(2)} = 2\sum_{j=1}^{n}\frac{H_j}{j+1}.$$

**Proof.**

$$\begin{aligned} &H_{n+1}^2 - H_{n+1}^{(2)} = H_n^2 - H_n^{(2)} + 2\frac{H_n}{n+1} \quad \text{and by iteration,} \\ &H_{n+1}^2 - H_{n+1}^{(2)} = 2\sum_{j=1}^{n}\frac{H_j}{j+1}. \end{aligned} \qquad \blacksquare$$

We will use the above Lemma and Corollary in our analysis. We have that

$$\sum_{i=1}^{n} H_i H_{n+1-i} = \sum_{i=1}^{n}\left(H_i\left(H_{n-i}+\frac{1}{n+1-i}\right)\right) = \sum_{i=1}^{n} H_i H_{n-i} + \sum_{i=1}^{n}\frac{H_i}{n+1-i}.$$

The second sum substituting $j = n+1-i$ becomes

$$\sum_{i=1}^{n}\frac{H_i}{n+1-i} = \sum_{j=1}^{n}\frac{H_{n+1-j}}{j}.$$

As we have seen it is equal to

$$\sum_{j=1}^{n} \frac{H_{n+1-j}}{j} = 2\sum_{j=1}^{n} \frac{H_j}{j+1} = H_{n+1}^2 - H_{n+1}^{(2)}.$$

Hence, by Lemma 2.3.9,

$$\begin{aligned}\sum_{j=1}^{n} H_j H_{n-j} &= (n+2)(H_{n+1}^2 - H_{n+1}^{(2)}) - 2(n+1)(H_{n+1}-1) - (H_{n+1}^2 - H_{n+1}^{(2)}) \\ &= (n+1)\big((H_{n+1}^2 - H_{n+1}^{(2)}) - 2(H_{n+1}-1)\big).\end{aligned}$$

Using the above result and Eq. (2.3), we have

$$\sum_{j=1}^{n} jH_{j-1}H_{n+1-j} = \binom{n+2}{2}\left((H_n^2 - H_n^{(2)}) + \frac{2n(1-H_n)}{n+1}\right).$$

Having worked out all the expressions involved in the following relation

$$\begin{aligned}(n+1)B_{n+1} - nB_n = 4&\left(\sum_{j=1}^{n} jH_{j-1} + \sum_{j=1}^{n} jH_{j-1}H_{n-j+1}\right) \\ &+ 2B_n - 2n(n+1)H_{n+1} + \frac{1}{2}n(n+11).\end{aligned}$$

This becomes

$$\begin{aligned}&(n+1)B_{n+1} - nB_n \\ &= 4\left(\frac{n(n+1)H_{n+1}}{2} - \frac{n(n+5)}{4} + \binom{n+2}{2}\Big((H_{n+1}^2 - H_{n+1}^{(2)}) - 2(H_{n+1}-1)\Big)\right) \\ &\qquad + 2B_n - 2n(n+1)H_{n+1} + \frac{1}{2}n(n+11) \\ &= 2(n+1)(n+2)(H_{n+1}^2 - H_{n+1}^{(2)}) - 4(n+1)(n+2)(H_{n+1}-1) - \frac{n(n-1)}{2} + 2B_n.\end{aligned}$$

Dividing both sides by $(n+1)(n+2)$ and unwinding the recurrence,

$$\frac{B_n}{n+1} = \frac{B_0}{1} + 2\sum_{i=1}^{n}(H_i^2 - H_i^{(2)}) - 4\sum_{i=1}^{n}(H_i - 1) - \sum_{i=1}^{n}\frac{(i-1)(i-2)}{2i(i+1)}.$$

Hence

$$\begin{aligned}\frac{B_n}{n+1} &= 2(n+1)(H_n^2 - H_n^{(2)}) + 4n - 4nH_n - 4\Big((n+1)H_n - 2n\Big)\\ &\quad - \sum_{i=1}^{n}\left(\frac{i+2}{2i} - \frac{3}{i+1}\right)\\ &= 2(n+1)(H_n^2 - H_n^{(2)}) - H_n(8n+2) + \frac{23n}{2} - 3 + \frac{3}{n+1}\\ &= 2(n+1)(H_n^2 - H_n^{(2)}) - H_n(8n+2) + \frac{23n^2 + 17n}{2(n+1)}.\end{aligned}$$

Finally, multiplying by $(n+1)$ we obtain

$$B_n = 2(n+1)^2(H_n^2 - H_n^{(2)}) - H_n(n+1)(8n+2) + \frac{23n^2 + 17n}{2}.$$

Consequently, by Lemma 2.3.3 the variance of the number of comparisons of randomised Quicksort is

$$7n^2 - 4(n+1)^2H_n^{(2)} - 2(n+1)H_n + 13n,$$

completing the proof of Theorem 2.3.1.

## 2.4 "Divide and Conquer" recurrences

We have computed the mean and variance of the number of comparisons made by Quicksort that mainly contribute to its time complexity. Because of the simple structure of the algorithm (dividing into smaller subproblems) we can in fact approach many other related problems in the same spirit. Let $F(n)$ denote the expected value of some random variable associated with randomised Quicksort and $T(n)$ be the average value of the “toll function”, which is the needed cost to divide the problem into two simpler subproblems. Then $F(n)$ is equal to the contribution $T(n)$, plus the measures required sort the resulting subarrays of $(i-1)$ and $(n-i)$ elements, where the pivot $i$ can be any key of the array with equal probability.

Thus, the recurrence relation is

$$\begin{aligned} F(n) &= T(n) + \frac{1}{n}\sum_{i=1}^{n}(F(i-1)+F(n-i)) \\ &= T(n) + \frac{2}{n}\sum_{i=1}^{n} F(i-1). \end{aligned}$$

This is the general type of recurrences arising in the analysis of Quicksort, which can be manipulated using the difference method or by generating functions. Since an empty array or an one having a unique key is trivially solved, the initial values of the recurrence is

$$F(0) = F(1) = 0.$$

The first method leads to the elimination of the sum, by subtracting $(n-1)F(n-1)$ from $nF(n)$ – see [46]. The recurrence becomes

$$nF(n) - (n-1)F(n-1) = nT(n) - (n-1)T(n-1) + 2F(n-1)$$

and dividing by $n(n+1)$ we have

$$\frac{F(n)}{n+1} = \frac{nT(n)-(n-1)T(n-1)}{n(n+1)} + \frac{F(n-1)}{n}.$$

This recurrence can be immediately solved by "unfolding" its terms. The general solution is

$$\begin{aligned} F(n) &= (n+1)\left(\sum_{j=3}^{n} \frac{jT(j)-(j-1)T(j-1)}{j(j+1)} + \frac{F(2)}{3}\right) \\ &= (n+1)\left(\sum_{j=3}^{n} \frac{jT(j)-(j-1)T(j-1)}{j(j+1)} + \frac{T(2)}{3}\right). \end{aligned}$$

When the sorting of subarrays of $m$ keys or less is done by insertion sort, the solution of the recurrence is

$$\begin{aligned} F(n) &= (n+1)\Bigg(\sum_{j=m+2}^{n} \frac{jT(j)-(j-1)T(j-1)}{j(j+1)} + \frac{F(m+1)}{m+2}\Bigg) \\ &= (n+1)\Bigg(\sum_{j=m+2}^{n} \frac{jT(j)-(j-1)T(j-1)}{j(j+1)} + \frac{T(m+1)}{m+2}\Bigg), \end{aligned}$$

since $n-1>m$.

Another classic approach, which is more transparent and elegant, is the application of generating functions. The recurrence is transformed to a differential equation, which is then solved. The function is written in terms of series and the extracted coefficient is the solution. Multiplying by $nx^n$ and then summing with respect to $n$, in order to obtain the generating function $G(x)=\sum\limits_{n=0}^{\infty} F(n)x^n$, we have

$$\sum_{n=0}^{\infty} nF(n)x^n = \sum_{n=0}^{\infty} nT(n)x^n + 2\sum_{n=0}^{\infty}\sum_{i=1}^{n} F(i-1)x^n.$$

The double sum is equal to

$$\sum_{n=0}^{\infty}\sum_{i=1}^{n} F(i-1)x^n = G(x)\sum_{n=1}^{\infty} x^n = \frac{xG(x)}{1-x}$$

and the differential equation is

$$xG'(x) = \sum_{n=0}^{\infty} nT(n)x^n + \frac{2xG(x)}{1-x}.$$

Cancelling out $x$ and multiplying by $(1-x)^2$,

$$\begin{aligned}
G'(x)(1-x)^2 &= (1-x)^2\sum_{n=1}^{\infty} nT(n)x^{n-1} + 2(1-x)G(x) \\
\big(G(x)(1-x)^2\big)' &= (1-x)^2\sum_{n=1}^{\infty} nT(n)x^{n-1} \\
\implies G(x)(1-x)^2 &= \int (1-x)^2\sum_{n=1}^{\infty} nT(n)x^{n-1}\,\mathrm{d}x + C \\
\implies G(x) &= \frac{\displaystyle\int (1-x)^2\sum_{n=1}^{\infty} nT(n)x^{n-1}\,\mathrm{d}x + C}{(1-x)^2},
\end{aligned}$$

where $C$ is constant, which can be found using the initial condition $G(0) = 0$. The solution then is being written as power series and the coefficient sequence found is the expected sought cost.

Now, one can obtain any expected cost of the algorithm, just by using these results. The “toll function” will be different for each case. Plugging in the average value, the finding becomes a matter of simple operations. This type of analysis unifies the recurrences of Quicksort into a single one and provides an intuitive insight of the algorithm.

## 2.5 Higher moments

We have effectively calculated the first and second moments of $C_n$ in Quicksort. Existing literature does not seem to address much questions about skewness and kurtosis, which are sometimes held to be interesting features of a random variable. Here, we present an inconclusive discussion about the sign of the skewness.

Using the probability generating function, we can obtain higher moments of the algorithm's complexity. A Lemma follows

**Lemma 2.5.1.** *Let a random variable $X$ having probability generating function:*

$$f_X(z) = \sum_{n=0}^{\infty} \mathbb{P}(X = n) z^n.$$

*For the $k$-th order derivative it holds that*

$$\left.\frac{d^k f_X(z)}{dz^k}\right|_{z=1} = \mathbb{E}\big(X \cdot (X-1) \cdot \ldots \cdot (X-k+1)\big).$$

**Proof.** Simply by computing successively the $k$-th order derivative of $f_X(z)$, we obtain

$$\frac{d^k f_X(z)}{dz^k} = \sum_{n=0}^{\infty} n \cdot (n-1) \cdot \ldots \cdot (n-k+1) \mathbb{P}(X = n) \cdot z^{n-k}$$

Setting $z = 1$, the proof follows directly. Note that the argument is similar to continuous random variables. ∎

Using MAPLE, we obtained a recursive form for the general $k$-th order derivative of the generating function.

**Lemma 2.5.2.** *Let*

$$f_n(z) = \frac{z^{n-1}}{n} \sum_{j=1}^{n} f_{j-1}(z) f_{n-j}(z)$$

*be the generating function of Quicksort's complexity in sorting $n$ keys. The $k \in \mathbf{N}$ order derivative is given by*

$$\frac{\mathrm{d}^k f_n(z)}{\mathrm{d}z^k} = \frac{1}{n} \cdot \left( \sum_{i=0}^{k} \binom{k}{i} \cdot \frac{\Gamma(n)}{\Gamma(n-i)} \cdot z^{n-i-1} \frac{\mathrm{d}^{k-i}}{\mathrm{d}z^{k-i}} \left( \sum_{j=1}^{n} f_{j-1}(z) \cdot f_{n-j}(z) \right) \right),$$

*where the $\Gamma$ function is defined for complex variable $z \neq 0, -1, -2, \ldots$ as*

$$\Gamma(z) := \int_0^\infty p^{z-1} e^{-p} \, \mathrm{d}p$$

*and when $z$ is a positive integer, then $\Gamma(z) = (z-1)!$.*

**Proof.** For $k = 0$, the result follows trivially. Assume that the statement of the Lemma holds for $k = m$. The $(m+1)$-th order derivative is

$$\begin{aligned}
\frac{\mathrm{d}^{m+1} f_n(z)}{\mathrm{d}z^{m+1}} &= \frac{1}{n} \sum_{i=0}^{m} \binom{m}{i} \frac{\Gamma(n)}{\Gamma(n-i)} \Bigg( (n-i-1) z^{n-i-2} \frac{\mathrm{d}^{m-i}}{\mathrm{d}z^{m-i}} \left( \sum_{j=1}^{n} f_{j-1}(z) \cdot f_{n-j}(z) \right) \\
&\quad + z^{n-i-1} \frac{\mathrm{d}^{m-i+1}}{\mathrm{d}z^{m-i+1}} \left( \sum_{j=1}^{n} f_{j-1}(z) \cdot f_{n-j}(z) \right) \Bigg) \\
&= \frac{1}{n} \sum_{i=1}^{m+1} \binom{m}{i-1} \frac{\Gamma(n)}{\Gamma(n-i+1)} (n-i) z^{n-i-1} \frac{\mathrm{d}^{m-i+1}}{\mathrm{d}z^{m-i+1}} \left( \sum_{j=1}^{n} f_{j-1}(z) \cdot f_{n-j}(z) \right) \\
&\quad + \frac{1}{n} \sum_{i=0}^{m} \binom{m}{i} \frac{\Gamma(n)}{\Gamma(n-i)} z^{n-i-1} \frac{\mathrm{d}^{m-i+1}}{\mathrm{d}z^{m-i+1}} \left( \sum_{j=1}^{n} f_{j-1}(z) \cdot f_{n-j}(z) \right).
\end{aligned}$$

Note that

$$\frac{\Gamma(n)}{\Gamma(n-i+1)}(n-i)=\frac{\Gamma(n)}{\Gamma(n-i)}.$$

Therefore,

$$\begin{aligned}\frac{\mathrm{d}^{m+1}f_n(z)}{\mathrm{d}z^{m+1}}&=\frac{1}{n}\sum_{i=1}^{m}\left(\binom{m}{i-1}+\binom{m}{i}\right)\frac{\Gamma(n)}{\Gamma(n-i)}z^{n-i-1}\\&\times\frac{\mathrm{d}^{m-i+1}}{\mathrm{d}z^{m-i+1}}\left(\sum_{j=1}^{n}f_{j-1}(z)\cdot f_{n-j}(z)\right).\end{aligned}$$

The well-known identity, (e.g. see [26])

$$\binom{m}{i-1}+\binom{m}{i}=\binom{m+1}{i},$$

completes the proof. ∎

We should point out that Lemma $2.5.2$ is an immediate consequence of Leibniz's product rule. Next, we shall ask a Question about the sign of the skewness of the time complexity of the algorithm, as it is moderately difficult to solve the recurrence involved, in order to compute the third moment. We already have seen that the possibility of worst-case performance of the algorithm is rather small and in the majority of cases the running time is close to the average time complexity which is $O\big(n\log_2(n)\big)$. Intuitively, this suggests that the complexity is negatively skewed. We present the following Question:

**Question 2.5.3.** *Is the skewness $\mathbb{S}(C_n)$ of the random number of key comparisons of Quicksort for the sorting of $n\geq 3$ keys negative?*

Note that the cases $n=1,2$ are deterministic, since we always make $0$ and $1$ comparisons for the sorting. This Question may have an affirmative answer, which can be possibly proven by an induction argument on the number of keys. However, great deal of attention must be exercised to the fact that the random number of comparisons required to sort the segment of keys less than the pivot and the segment of keys that are greater than the pivot are conditionally independent, subject to the choice of pivot.

## 2.6 Asymptotic analysis

After having examined the number of comparisons of Quicksort, in terms of average and worst case scenarios, and its variance, it is desirable also to study the concentration of the random variable $C_n$ about its mean. One might hope, by analogy with other probabilistic situations, that for large values of $n$ the number of comparisons is highly likely to be very close to the mean.

The analysis will be confined to the number of comparisons, because this is the most relevant measure for this thesis. Since our results will be asymptotic in nature, we need to have some relevant ideas about convergence of sequences of random variables. The following definitions come from [8], [21].

**Definition 2.6.1.**

*(i) A sequence of random variables $\{X_1, X_2, \ldots\}$ is said to converge in distribution (or converge weakly) to a random variable $X$ if and only if*

$$\lim_{n \to \infty} F_n(x) = F(x)$$

*at every point $x$ where $F$ is continuous. Here $F_n(x)$ and $F(x)$ are respectively the cumulative distribution functions of random variables $X_n$ and $X$. We shall denote* *this type of convergence by* $X_n \xrightarrow{\mathcal{D}} X$.

*(ii) A sequence of random variables* $\{X_1, X_2, \ldots\}$ *is said to converge in probability to a random variable* $X$ *if and only if* $\forall \varepsilon > 0$ *holds*

$$\lim_{n\to\infty} \mathbb{P}(|X_n - X| \geq \varepsilon) = 0.$$

*We will denote this type of convergence by* $X_n \xrightarrow{\mathcal{P}} X$.

*(iii) A sequence* $\{X_1, X_2, \ldots\}$ *of random variables is said to converge in* $L^p$*-norm to a random variable* $X$ *if and only if*

$$\lim_{n\to\infty} \mathbb{E}(|X_n - X|^p) = 0.$$

Note that convergence in $L^p$-norm, for $p \geq 1$, implies convergence in probability, and it is also easy to see that convergence in probability implies convergence in distribution: see e.g. [8]. Both converse statements are false in general.

We also present the definition of martingale, which shall be employed in a later stage of our analysis.

**Definition 2.6.2.** *Let* $\{Z_1, Z_2, \ldots\}$ *be a sequence of random variables. We say* $Z_n$ *is a martingale if and only if*

*(i)* $\mathbb{E}(|Z_n|) < \infty$*,* $\forall n \in \mathbf{N}$.

*(ii)* $\mathbb{E}(Z_n | Z_{n-1}, Z_{n-2}, \ldots, Z_1) = Z_{n-1}$.

### 2.6.1 Binary trees

In this subsection, a central notion to the analysis of Quicksort, and in general to the analysis of algorithms is discussed. We begin with a definition.

**Definition 2.6.3.** *A graph is defined as an ordered pair of two sets* $(V, E)$*. The set* $V$ *corresponds to the set of vertices or nodes. The set* $E$ *is the set of edges, which are pairs of distinct vertices.*

One kind of graph we concentrate on are trees. A definition of a tree (which suits us) is as follows [44]:

**Definition 2.6.4.** *Tree is a finite set* $\Delta$ *of nodes, such that:*
*(i) There is a unique node called the root of the tree.*
*(ii) The remaining nodes are partitioned into* $k \in \mathbf{N}$ *disjoint sets* $\Delta_1, \Delta_2, \ldots, \Delta_k$ *and each of these sets is a tree. Those trees are called the subtrees of the root.*

A particularly common tree is a binary tree. It is defined as a tree with the property that every node has at most $2$ subtrees, i.e. each node – excluding the root – is adjacent to one parent, so to speak, and up to two offspring, namely left and right child nodes. Note here that nodes which do not have any child nodes are called external. Otherwise, they are called internal. The size of a binary tree is the number of its nodes. The depth of a node is simply the number of edges from that node to the root in the (unique) shortest path between them and the height of a binary tree is the length from the deepest node to the root.

An extended binary tree is a binary tree with the property that every internal node has exactly two offspring [44]. Let $D_j$ be the depth of insertion of a key in a random binary tree of size $j - 1$. Since the root node is first inserted to an empty tree, it holds that $D_1 := 0$. The next inserted key is compared with the key at the root and if it is smaller, is placed to the left; otherwise is attached as a right subtree, thus $D_2 := 1$.

The internal path length of an extended binary tree having $n$ internal nodes is defined to be the sum over all internal nodes, of their distances to the root. Let us denote this quantity by $X_n$. We then have that

$$X_n = \sum_{j=1}^{n} D_j.$$

Similarly, the external path length is defined as the total number of edges in all the shortest paths from external nodes to the root node. The next Lemma gives a relationship between those two quantities.

**Lemma 2.6.5** (Knuth [44]). *For an extended binary tree with $n$ internal nodes it holds that*

$$Y_n = X_n + 2n,$$

*where $Y_n$ denotes the external path length of the tree.*

**Proof.** Suppose that we remove the two offspring of an internal node $v$, with its offspring being external nodes of the tree. We suppose that $v$ is at distance $h$ from the root. Then the external path length is reduced by $2(h+1)$, as its two offspring are removed. At the same time, the external path length is increased by $h$, because the vertex $v$ has just become an external node. Thus, the net change is equal to $-2(h+1)+h = -(h+2)$. For internal path length the change is a reduction by $-h$ as $v$ is no longer internal. Thus, overall, the change in $Y_n - X_n$ is equal to $-(h+2)-(-h) = -2$. The Lemma follows by induction. ∎

Binary trees play a fundamental and crucial role in computing and in the analysis of algorithms. They are widely used as data structures, because fast insertion, deletion and searching for a given record can be achieved. In Quicksort, letting nodes represent keys, the algorithm's operation can be depicted as a binary tree. The root node stores the initial pivot element. Since the algorithm at each recursion splits the initial array into two subarrays and so on, we have an ordered binary tree. The left child of the root stores the pivot chosen to sort all keys less than the value of root and the right child node stores the pivot for sorting the elements greater than the root.

The process continues until the algorithm divides the array into trivial subarrays having cardinality $0$ or $1$, which do not need any more sorting. These elements are stored as external or leaf nodes in this binary structure. It easily follows that for any given node storing a key $k$, its left subtree stores keys less than $k$ and similarly its right subtree contains keys greater than $k$.

In variants of Quicksort, where many pivots are utilised to partitioning process, the generalisation of binary trees provides a framework for the analysis, though we do not develop this in detail here. In the next subsection, the limiting distribution of the number of comparisons will be analysed, in terms of trees.

### 2.6.2 Limiting behaviour

We have previously seen that the operation of the algorithm can effectively be represented as a binary tree. Internal nodes store pivots selected at each recursion step of the algorithm: so the root vertex, for example, stores the first pivot with which all other elements are compared. We are interested in the total number of comparisons made. To understand this, we note that every vertex is compared with the first pivot which is at level $0$ of the tree. The array is divided into two parts – those above the pivot, and those below it. (Either of these subarrays may be empty). If a subarray is not empty, a pivot is found in it and is attached to the root as a child on the left, for the elements smaller than the first pivot or on the right for the elements larger than the pivot.

The process then continues recursively and each element at level $k$ in the tree is compared with each of the $k$ elements above it. Thus the total number of comparisons made is the sum of the depths of all nodes in the tree. This is equivalent to the internal path length of the extended binary tree. Thus,

$$Y_n = C_n + 2n,$$

where $Y_n$ is the external path length of the tree. To see this, we simply use Lemma 2.6.5. This fact can be also found in [49].

Generally, assume that we have to sort an array of $n$ distinct items with pivots uniformly chosen. All $n!$ orderings of keys are equally likely. This is equivalent to carrying out $n$ successive insertions [46]. Initially, the root node is inserted. The second key to be inserted is compared with the key at the root. If it is less than that key, it is attached as its left subtree. Otherwise, it is inserted as the right subtree. This process continues recursively by a series of comparisons of keys, until all $n$ keys have been inserted. Traversing the binary search tree in order, i.e. visiting the nodes of the left subtree, the root and the nodes of the right subtree, keys are printed in ascending order. Thus, Quicksort can be explicitly analysed in this way.

Recall that each internal node corresponds to the pivot at a given recursion of the Quicksort process (e.g. the depth of a given node). Thus the first pivot corresponds to the root node, its descendants or child nodes are pivots chosen from the two subarrays to be sorted, etc. Eventually, after $n$ insertions, we have built a binary search tree from top to bottom. We can use this approach to understand the following Theorem of Régnier [58].

**Theorem 2.6.6** (Régnier [58])**.** *Let random variables $Y_n$ and $X_n$ denote respectively the external and internal path length of binary search tree, built by $n$ successive insertions of keys. Then the random variables*

$$Z_n = \frac{Y_n - 2(n+1)(H_{n+1} - 1)}{n+1} = \frac{X_n - 2(n+1)H_n + 4n}{n+1}$$

*form a martingale with null expectations. For their variances it holds*

$$\mathrm{Var}(Z_n) = 7 - \frac{2\pi^2}{3} - \frac{2\log_e(n)}{n} + O\left(\frac{1}{n}\right).$$

**Proof.** By induction on $n$, the base case being trivial. Suppose we have an $(n-2)$-vertex tree and consider the insertion of the $(n-1)$-th key in the random binary tree. Its depth of insertion is $D_{n-1}$, so that a formerly external node becomes an internal and we see that its two (new) descendants are both at depth $D_{n-1}+1$. Recall that $D_n$ is the random variable counting the depth of insertion of a key in a random binary tree of size $n-1$ with $D_1 := 0$ and $D_2 := 1$. The following equation concerning conditional expectations holds:

$$n\mathbb{E}(D_n|D_1, \ldots, D_{n-1}) = (n-1)\mathbb{E}(D_{n-1}|D_1, \ldots, D_{n-2}) - D_{n-1} + 2(D_{n-1} + 1).$$

This recurrence yields

$$n\mathbb{E}(D_n|D_1,\dots,D_{n-1})-(n-1)\mathbb{E}(D_{n-1}|D_1,\dots,D_{n-2})=D_{n-1}+2.$$

Summing and using that the left-hand side is a telescopic sum,

$$n\mathbb{E}(D_n|D_1,\dots,D_{n-1})=\sum_{i=1}^{n-1}(D_i+2)=Y_{n-1}.$$

The last equation is justified by Lemma 2.6.5. Also, we have

$$\mathbb{E}(Y_n|D_1,\dots,D_{n-1})=\mathbb{E}(Y_{n-1}+2+D_n|D_1,\dots,D_{n-1})=\frac{n+1}{n}Y_{n-1}+2.$$

Thus, taking expectations and using $\mathbb{E}(\mathbb{E}(U|V))=\mathbb{E}(U)$,

$$\mathbb{E}(Y_n)=\frac{n+1}{n}\mathbb{E}(Y_{n-1})+2 \Longleftrightarrow \frac{\mathbb{E}(Y_n)}{n+1}=\frac{\mathbb{E}(Y_{n-1})}{n}+\frac{2}{n+1}.$$

This recurrence has solution

$$\mathbb{E}(Y_n)=2(n+1)(H_{n+1}-1).$$

For $Z_n$, we deduce that

$$\begin{aligned}\mathbb{E}(Z_n|D_1,\dots,D_n)&=\mathbb{E}\left(\frac{Y_n-\mathbb{E}(Y_n)}{n+1}|D_1,\dots,D_n\right)\\&=\frac{2}{n+1}+\frac{Y_{n-1}}{n}-\frac{\mathbb{E}(Y_n)}{n+1}\\&=Z_{n-1}.\end{aligned}$$

Therefore, $Z_n$ form a martingale. Further, note that $Z_n$ is a linear transformation of the internal path length $X_n$, so we get that

$$\mathrm{Var}(Z_n) = \frac{1}{(n+1)^2} \cdot \mathrm{Var}(X_n).$$

Previously, we saw that the number of comparisons is just the internal path length of a binary search tree. As a reminder the variance of the number of comparisons is equal to

$$\mathrm{Var}(C_n) = 7n^2 - 4(n+1)^2 H_n^{(2)} - 2(n+1)H_n + 13n.$$

Hence,

$$\mathrm{Var}(Z_n) = 7\left(\frac{n}{n+1}\right)^2 - 4H_n^{(2)} - \frac{2H_n}{n+1} + \frac{13n}{(n+1)^2}.$$

For the purpose of obtaining the asymptotics of the variance, an important family of functions, which is called polygamma functions are discussed. The digamma function is

$$\psi(z) = \frac{\mathrm{d}}{\mathrm{d}z} \log_e \Gamma(z)$$

and for complex variable $z \neq 0, -1, -2, \ldots$ can be written as [1],

$$\psi(z) = -\gamma + \sum_{j=0}^{\infty} \left( \frac{1}{j+1} - \frac{1}{j+z} \right). \tag{2.7}$$

In general, $\forall k \in \mathbf{N}$, the set

$$\psi^{(k)}(z) = \frac{\mathrm{d}^{k+1}}{\mathrm{d}z^{k+1}} \log_e \Gamma(z),$$

with $\psi^{(0)}(z) = \psi(z)$, forms the family of polygamma functions. Differentiating Eq. (2.7),

$$\psi^{(1)}(z) = \sum_{j=0}^{\infty} \frac{1}{(j+z)^2}. \tag{2.8}$$

By Eq. (2.7), it easily follows that

$$H_n = \psi(n+1) + \gamma.$$

Further, using the fact that $\zeta(2) = \lim_{n\to\infty} H_n^{(2)} = \dfrac{\pi^2}{6}$ [1], where $\zeta(s) = \sum_{j=1}^{\infty} \dfrac{1}{j^s}$ is the Riemann zeta function for $\mathfrak{Re}(s) > 1$, we obtain

$$H_n^{(2)} + \frac{2H_n}{n+1} = \frac{\pi^2}{6} - \psi^{(1)}(n+1) + \frac{2(\psi(n+1)+\gamma)}{n+1}. \tag{2.9}$$

Eq. (2.9) is asymptotically equivalent to

$$\frac{\pi^2}{6} + \frac{2\log_e(n)}{n},$$

thus, the asymptotic variance is

$$\begin{aligned} \mathrm{Var}(Z_n) &= 7 - \frac{2\pi^2}{3} - \frac{2\log_e(n)}{n} + O\left(\frac{1}{n}\right) \\ &= 7 - \frac{2\pi^2}{3} - O\left(\frac{\log_e(n)}{n}\right). \end{aligned} \qquad \blacksquare$$

**Remark 2.6.7.** *We note that there is a typo in the expression for the asymptotic variance in Régnier's paper [58], which is given as*

$$7 - \frac{2\pi^2}{3} + O\left(\frac{1}{n}\right).$$

*The correct formula for the asymptotic variance is stated in Fill and Janson [22].*

The key point about martingales is that they converge.

**Theorem 2.6.8** (Feller [21]). *Let $Z_n$ be a martingale, and suppose further that there is a constant $C$ such that $\mathbb{E}(Z_n^2) < C$ for all $n$. Then there is a random variable $Z$ to which $Z_n$ converges, with probability $1$. Further, $\mathbb{E}(Z_n) = \mathbb{E}(Z)$ for all $n$.*

We showed that

$$Z_n \xrightarrow{\mathcal{P}} Z.$$

It is important to emphasise that the random variable $Z$ to which we get convergence is not normally distributed. We saw that the total number of comparisons to sort an array of $n \geq 2$ keys, when the pivot is a uniform random variable on $\{1, 2, \ldots, n\}$ is equal to the number of comparisons to sort the subarray of $U_n - 1$ keys below pivot plus the number of comparisons to sort the subarray of $n - U_n$ elements above pivot plus $n - 1$ comparisons done to partition the array. Therefore,

$$X_n = X_{U_n - 1} + X^*_{n - U_n} + n - 1,$$

where the random variables $X_{U_n-1}$ and $X^*_{n-U_n}$ are identically distributed and independent conditional on $U_n$.

Consider the random variables

$$Y_n = \frac{X_n - \mathbb{E}(X_n)}{n}.$$

The previous equation can be rewritten in the following form

$$Y_n = \frac{X_{U_n - 1} + X^*_{n-U_n} + n - 1 - \mathbb{E}(X_n)}{n}.$$

By a simple manipulation, it follows that [22], [59]

$$Y_n = Y_{U_n - 1} \cdot \frac{U_n - 1}{n} + Y^*_{n-U_n} \cdot \frac{n - U_n}{n} + C_n(U_n),$$

where

$$C_n(j) = \frac{n-1}{n} + \frac{1}{n}\left(\mathbb{E}(X_{j-1}) + \mathbb{E}(X^*_{n-j}) - \mathbb{E}(X_n)\right).$$

The random variable $U_n/n$ converges to a uniformly distributed variable $\Xi$ on $[0, 1]$. A Lemma follows

**Lemma 2.6.9.** *Let $U_n$ be a uniformly distributed random variable on $\{1, 2, \ldots, n\}$. Then*

$$\frac{U_n}{n} \xrightarrow{\mathcal{D}} \Xi,$$

*where $\Xi$ is uniformly distributed on $[0, 1]$.*

**Proof.** The moment generating function of $U_n$ is given by

$$M_{U_n}(s) = \sum_{k=1}^{n} \mathbb{P}(U_n = k)e^{sk} = \frac{1}{n} \cdot \sum_{k=1}^{n} e^{sk} = \frac{1}{n} \cdot \frac{e^{s(n+1)} - e^s}{e^s - 1}.$$

For the random variable $U_n/n$, it is

$$M_{U_n/n}(s) = M_{U_n}(s/n) = \frac{1}{n} \cdot \sum_{k=1}^{n} e^{sk/n} = \frac{1}{n} \cdot \frac{e^{s(n+1)/n} - e^{s/n}}{e^{s/n} - 1}.$$

The random variable $\Xi$ has moment generating function

$$M_\Xi(s) = \int_0^1 e^{st}\,\mathrm{d}t = \frac{e^s - 1}{s}.$$

Now the moment generating function of $U_n/n$ is an approximation to the average value of $e^{sx}$ over the interval $[0,1]$ and so, as $n$ tends to infinity, we can replace it by its integral

$$\int_0^1 e^{sx}\,\mathrm{d}x = \left[\frac{e^{sx}}{s}\right]_0^1 = \frac{e^s - 1}{s}$$

and now all that is required has been proved. ∎

For the function

$$C_n(j) = \frac{n-1}{n} + \frac{1}{n} \cdot \big(\mathbb{E}(X_{j-1}) + \mathbb{E}(X^*_{n-j}) - \mathbb{E}(X_n)\big)$$

using the previous Lemma and recalling that asymptotically the expected complexity of Quicksort is $2n\log_e(n)$ it follows that

$$\begin{aligned}
\lim_{n\to\infty} C_n(n \cdot U_n/n) &= \lim_{n\to\infty} \left(\frac{n-1}{n} + \frac{1}{n} \cdot \big(\mathbb{E}(X_{U_n-1}) + \mathbb{E}(X^*_{n-U_n}) - \mathbb{E}(X_n)\big)\right) \\
&= \lim_{n\to\infty} \left(\frac{n-1}{n} + \frac{1}{n} \cdot \left(2\left(\frac{n \cdot U_n}{n} - 1\right)\log_e(U_n - 1)\right.\right. \\
&\qquad \left.\left. + 2\left(n - \frac{n \cdot U_n}{n}\right)\log_e(n - U_n) - 2n\log_e(n)\right)\right) \\
&= 1 + 2\xi\log_e \xi + 2(1-\xi)\log_e(1-\xi) = C(\Xi), \quad \forall\, \xi \in [0,1].
\end{aligned}$$

Thus $C_n(n \cdot U_n/n)$ converges to $C(\Xi)$, (see as well in [59]). Therefore, we obtain

$$\mathscr{L}(Y) = \mathscr{L}\big(Y \cdot \Xi + Y^* \cdot (1 - \Xi) + C(\Xi)\big).$$

But this does not work for the normal. Indeed, if $Y$ (and hence $Y^*$) are normals with mean zero and variance $\sigma^2$, a necessary condition for $Y\Xi + Y^*(1-\Xi) + C(\Xi)$ to have mean 0 (which would be needed for the equality to hold) would be that $C(\Xi) = 0$. This equality happens with probability equal to $0$, as we can easily deduce. Thus, the distribution which the sequence $Y_n$ converges is not Gaussian.

### 2.6.3 Deviations of Quicksort

It is desirable to derive bounds on the deviation of the random number of pairwise comparisons, needed to sort an array of $n$ distinct elements from its expected value, as $n$ gets arbitrarily large. We remind ourselves that the pivot is uniformly chosen at random.

We shall derive bounds on the following probability

$$\mathbb{P}\left(\left|\frac{C_n}{\mathbb{E}(C_n)} - 1\right| > \epsilon\right)$$

for $\epsilon > 0$, sufficiently small. From Chebyshev's inequality, we obtain a bound for the above probability. Chebyshev's inequality is as follows [21]:

**Theorem 2.6.10** (Chebyshev's inequality)**.** *Let $X$ be a random variable, with $\mathbb{E}(X^2) < \infty$. Then, for any real number $a > 0$*

$$\mathbb{P}\big(|X| \geq a\big) \leq \frac{\mathbb{E}(X^2)}{a^2}.$$

*If $\mathbb{E}(X) = m$ and $\mathrm{Var}(X) = \sigma^2$, then*

$$\mathbb{P}\big(|X - m| \geq a\big) \leq \frac{\sigma^2}{a^2}.$$

The random variable $Y_n = \dfrac{C_n}{\mathbb{E}(C_n)}$ has mean and variance,

$$\mathbb{E}(Y_n) = \mathbb{E}\left(\frac{C_n}{\mathbb{E}(C_n)}\right) = \frac{1}{\mathbb{E}(C_n)} \cdot \mathbb{E}(C_n) = 1$$

$$\begin{aligned}\mathrm{Var}(Y_n) &= \mathrm{Var}\left(\frac{C_n}{\mathbb{E}(C_n)}\right) = \frac{1}{\mathbb{E}^2(C_n)} \cdot \mathrm{Var}(C_n) \\ &= \frac{-4(n+1)^2 H_n^{(2)} - 2(n+1)H_n + (7n+13)n}{\big(2(n+1)H_n - 4n\big)^2}.\end{aligned}$$

For $\epsilon > 0$, we have

$$\mathbb{P}(|Y_n - 1| > \epsilon) < \frac{\mathrm{Var}(Y_n)}{\epsilon^2} = \frac{-4(n+1)^2 H_n^{(2)} - 2(n+1)H_n + (7n+13)n}{\epsilon^2\big(2(n+1)H_n - 4n\big)^2}.$$

It holds that,

$$\begin{aligned}\mathrm{Var}(C_n) &= -4(n+1)^2 H_n^{(2)} - 2(n+1)H_n + (7n+13)n \\ &\leq 7n^2 + 13n \\ &\leq 20n^2,\end{aligned}$$

using that the other terms are negative and $n \geq 1$. Further, using that

$$\lim_{n\to\infty} H_n^{(2)} = \sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6},$$

we get

$$\begin{aligned}\mathrm{Var}(C_n) &= -4(n+1)^2 H_n^{(2)} - 2(n+1)H_n + (7n+13)n \\ &\geq (7n+13)n - 4(n+1)^2\pi^2/6 - 2(n+1)\big(\log_e(n) + \gamma + o(1)\big) \\ &= (7 - 2\pi^2/3)n^2\big(1 + o(1)\big).\end{aligned}$$

From above inequalities, we deduce that $\mathrm{Var}(C_n) = \Theta(n^2)$. Thus,

$$\mathbb{P}(|Y_n - 1| > \epsilon) < \frac{\mathrm{Var}(Y_n)}{\epsilon^2} = \frac{\Theta(n^2)}{\epsilon^2\big(2(n+1)H_n - 4n\big)^2} = O\left((\epsilon \log_e(n))^{-2}\right).$$

Rösler [59] derived a sharper bound. He showed that this probability is $O(n^{-k})$, for fixed $k$. McDiarmid and Hayward [50] have further sharpened the bound. Their Theorem is

**Theorem 2.6.11** (McDiarmid and Hayward [50], McDiarmid [51]).
*Let $\epsilon = \epsilon(n)$ satisfy $\frac{1}{\log_e(n)} < \epsilon \leq 1$. Then as $n \to \infty$,*

$$\mathbb{P}(|Y_n - 1| > \epsilon) = n^{-2\epsilon\left(\log_e^{(2)}(n) - \log_e(1/\epsilon) + O(\log_e^{(3)}(n))\right)},$$

*where $\log_e^{(k+1)}(n) := \log_e\big(\log_e^{(k)}(n)\big)$ and $\log_e^{(1)}(n) = \log_e(n)$.*

In the recent paper of McDiarmid [51], Theorem 2.6.11 is revisited using concentration arguments. By this result, it can be easily deduced, that Quicksort with good pivot choices performs well, with negligible perturbations from its expected number of comparisons. As we previously saw, Quicksort can be depicted as an ordered binary tree. The root node or node $1$ corresponds to the input array of length $L_1 = n$ to be sorted.

A pivot with rank $U_n = \{1, 2, \ldots, n\}$ is selected uniformly at random and the initial array is divided into two subarrays, one with elements less than the pivot and a second, with elements greater than the pivot. Then, the node at the left corresponds to the subarray of keys that are less than the pivot, with length $L_2 = U_n - 1$ and the node at the right corresponds to the subarray of keys that are greater than the pivot, with length $L_3 = n - U_n$.

Recursively, Quicksort runs on these two subarrays and split them in four subarrays, until we get trivial subarrays and the initial array is sorted. For $j = 1, 2\ldots$ let $L_j$ be the length of the array to be sorted at $j$ node and $M_k^n$ be the maximum cardinality of the $2^k$ subarrays, after $k$ recursions of Quicksort. It is [50],

$$M_k^n = \max\{L_{2^k+i} : i = 0, 1, \ldots, 2^k - 1\}.$$

The following Lemma gives an upper bound for the probability that the maximum length of $2^k$ subarrays $M_k^n$, will exceed $\alpha$ times the initial set's length $n$. We easily see that the upper bound is rather small quantity. Thus, we deduce that the length of the array is rapidly decreasing, as Quicksort runs.

**Lemma 2.6.12** (McDiarmid and Hayward [50], McDiarmid [51]).
*For any* $0 < \alpha < 1$ *and any integer* $k \geq \log_e \left(\frac{1}{a}\right)$ *it holds*

$$\mathbb{P}\left(M_k^n \geq \alpha n\right) \leq \alpha \left(\frac{2e \cdot \log_e(1/\alpha)}{k}\right)^k.$$

So far, the analysis explicitly assumed the presence of distinct numbers. However, in many sorting problems, one can come across with duplicates. As we will see next, we have to consider the presence of equal numbers and how this affects the algorithm's performance.

## 2.7 Quicksorting arrays with repeated elements

In this section, we consider the presence of equal keys. In some cases, there may be multiple occurrences of some of the keys – so that there is a multiset of keys to be sorted. If a key $a_i$ appears $s_i$ times, we call $s_i$ the multiplicity of $a_i$.

Thus, we have a multiset of the array $\{s_1 \cdot a_1, s_2 \cdot a_2, \ldots, s_n \cdot a_n\}$, with $s_1 + s_2 + \ldots + s_n = N$ and $n$ is the number of distinct keys. Without loss of generality, we assume that we have to sort a random permutation of the array $\{s_1 \cdot 1, s_2 \cdot 2, \ldots, s_n \cdot n\}$. Obviously, when $s_1 = s_2 = \ldots = s_n = 1$, then the array contains $n = N$ distinct keys.

Consider an array consisting of keys with "large" multiplicities. The usual Quicksort algorithm is quite likely to perform badly, since keys equal to the pivot are not being exchanged. Recall that Hoare's partitioning routine [30] described in the previous Chapter, utilises two pointers that scan for keys greater than and less than the pivot, so at the end of partition, keys equal to pivot may be on either or both subarrays, leading to unbalanced partitioning which we saw is usually undesirable. The algorithm can be further tweaked for efficient sorting of duplicates, using a ternary partition scheme [7]. Keys less than pivot are on left, keys equal to pivot on middle and on right, keys greater than the pivot.

Hoare's partitioning scheme can be easily modified, in order for the pointers to stop on equal keys with the pivot. In other words, the lower pointer searches for a key greater than or equal to the pivot and the upper one for a key smaller than or equal to the pivot. Sedgewick [64] considers a partitioning routine, where only one of the pointers stops on keys either greater or smaller than the pivot.

The recurrence for the expected number of key comparisons $\mathbb{E}\big(C(s_1, s_2, \ldots, s_n)\big)$ is

$$\mathbb{E}\big(C(s_1, s_2, \ldots, s_n)\big) = N - 1 + \frac{1}{N}\sum_{i=1}^{n} s_i \Bigg( \mathbb{E}\big(C(s_1, \ldots, s_{i-1})\big) + \mathbb{E}\big(C(s_{i+1}, \ldots, s_n)\big) \Bigg).$$

The analysis has been done in [64]. The solution of the recurrence is

$$\mathbb{E}\big(C(s_1, s_2, \ldots, s_n)\big) = 2\left(1 + \frac{1}{n}\right) NH_n - 3N - n. \tag{2.10}$$

Note that when $n = N$, i.e. the keys to be sorted are distinct, Eq. (2.10) yields the expected number of comparisons, when the array is partitioned using $n - 1$ comparisons. As we saw in the previous Chapter, a different partitioning scheme proposed and analysed in [46], [63], utilises $N+1$ comparisons for partitioning an array of $N$ keys. In this case, an upper bound for large $n$ to the expected number of comparisons is [64],

$$\mathbb{E}\big(C(s_1, s_2, \ldots, s_n)\big) = 2N(H_N + 1) - 2 + O\left(\frac{N^2}{n}\right).$$

# Chapter 3

# Sorting by sampling

The preceding analysis has shown that Quicksort is prone to poor performance, when the chosen pivots happen to be (close to) the smallest or greatest keys in the array to be sorted. The partitioning yields trivial subarrays, leading to quadratic running time taken by the algorithm, and increasing the chances of ‘crashing’, i.e. an application of Quicksort may terminate unsuccessfully through running out of memory. On the other hand, good choices of pivot yield a much more efficient algorithm. The uniform model suggests that in fact any key has equal probability to be selected as pivot. In this Chapter, we discuss and analyse the general idea of how one can increase the probability that the selected pivots will produce (reasonably) balanced partitions, by trying to ensure that their ranks are ‘near’ the middle of the array.

A naive idea towards this, would be finding the median of the keys and use this as pivot. However, it is obvious that the finding of median imposes additional costs to the algorithm. Therefore, despite the always good choices, this method might not be better than choosing the pivot randomly. Instead of finding the median of the array to be sorted, it might be more efficient to randomly pick a sample from the array, find its median and use this as pivot. In what it follows, we will analyse this idea and its variants.

## 3.1 Median of (2k+1) partitioning

Singleton [68] suggested to randomly select three keys from the array to be sorted and use their median as pivot, leading to a better estimate of the median of the array and reducing further the chances of worst case occurrence.

This modification can be generalised as to choosing a sample of $(2k+1)$ keys at every recursive stage, computing their median and using that median as pivot, partitioning $n > 2k+1$ keys. Arrays that contain at most $(2k+1)$ keys are sorted by a simpler algorithm, such as insertion sort. The cost of sorting these small arrays is linear with respect to $n$.

Letting $C_{n\{2k+1\}}$ denote the number of comparisons required to sort $n$ keys when the pivot is the median of a random sample of $(2k+1)$ elements, uniformly selected from the relevant array, the recurrence for the average number of comparisons is given by (see [46])

$$\mathbb{E}(C_{n\{2k+1\}}) = n+1+\frac{2}{\dbinom{n}{2k+1}}\sum_{j=1}^{n}\binom{j-1}{k}\binom{n-j}{k}\mathbb{E}(C_{j-1\{2k+1\}}).$$

Multiplying both sides by $\dbinom{n}{2k+1}$,

$$\binom{n}{2k+1}\mathbb{E}(C_{n\{2k+1\}}) = \binom{n}{2k+1}(n+1)+2\sum_{j=1}^{n}\binom{j-1}{k}\binom{n-j}{k}\mathbb{E}(C_{j-1\{2k+1\}}).$$

Multiplying by $z^n$ and summing over $n$, in order to obtain the generating function for the expected number of comparisons, $f(z) = \sum_{n=0}^{\infty} \mathbb{E}(C_{n\{2k+1\}})z^n$

$$\begin{aligned}
&\sum_{n=0}^{\infty} \binom{n}{2k+1} \mathbb{E}(C_{n\{2k+1\}})z^n = \\
&\sum_{n=0}^{\infty} \binom{n}{2k+1} (n+1)z^n + 2\sum_{n=0}^{\infty}\sum_{j=1}^{n} \binom{j-1}{k}\binom{n-j}{k} \mathbb{E}(C_{j-1\{2k+1\}})z^n.
\end{aligned} \tag{3.1}$$

It holds

$$\begin{aligned}
\sum_{n=0}^{\infty} \binom{n}{2k+1} \mathbb{E}(C_{n\{2k+1\}})z^n &= \frac{1}{(2k+1)!}\sum_{n=0}^{\infty} n(n-1)\ldots(n-2k)\mathbb{E}(C_{n\{2k+1\}})z^n \\
&= \frac{z^{2k+1} f^{(2k+1)}(z)}{(2k+1)!},
\end{aligned}$$

where $f^{(2k+1)}(z)$ denotes the $(2k+1)$-th order derivative of $f(z)$. For the first sum in the right-hand side of Eq. (3.1)

$$\begin{aligned}
\sum_{n=0}^{\infty} \binom{n}{2k+1} (n+1)z^n &= \frac{z^{2k+1}}{(2k+1)!}\left(\sum_{n=0}^{\infty} z^{n+1}\right)^{(2k+2)} \\
&= \frac{z^{2k+1}}{(2k+1)!}\left(\frac{z}{1-z}\right)^{(2k+2)}.
\end{aligned}$$

The $(2k+2)$-th order derivative of $\big(z/(1-z)\big)$ can be easily seen by induction that is equal to

$$\frac{(2k+2)!}{(1-z)^{2k+3}}.$$

Expanding out the double sum of Eq. (3.1),

$$
\begin{aligned}
&\sum_{n=0}^{\infty}\sum_{j=1}^{n}\binom{j-1}{k}\binom{n-j}{k}\mathbb{E}(C_{j-1\{2k+1\}})z^n = \binom{0}{k}\binom{0}{k}\mathbb{E}(C_{0\{2k+1\}})z+\\
&\quad\left(\binom{0}{k}\binom{1}{k}\mathbb{E}(C_{0\{2k+1\}})+\binom{1}{k}\binom{0}{k}\mathbb{E}(C_{1\{2k+1\}})\right)z^2+\\
&\quad\left(\binom{0}{k}\binom{2}{k}\mathbb{E}(C_{0\{2k+1\}})+\binom{1}{k}\binom{1}{k}\mathbb{E}(C_{1\{2k+1\}})+\binom{2}{k}\binom{0}{k}\mathbb{E}(C_{2\{2k+1\}})\right)z^3+\dots\\
&\qquad=\left(\binom{0}{k}\mathbb{E}(C_{0\{2k+1\}})+\binom{1}{k}\mathbb{E}(C_{1\{2k+1\}})z+\dots\right)\left(\binom{0}{k}z+\binom{1}{k}z^2+\dots\right)\\
&\qquad=\left(\sum_{n=0}^{\infty}\binom{n}{k}\mathbb{E}(C_{n\{2k+1\}})z^n\right)\left(\sum_{n=0}^{\infty}\binom{n}{k}z^{n+1}\right)\\
&\qquad=\frac{z^k f^{(k)}(z)}{k!}\cdot\frac{z^{k+1}}{k!}\left(\sum_{n=0}^{\infty}z^n\right)^{(k)}\\
&\qquad=\frac{z^{2k+1}f^{(k)}(z)}{k!(1-z)^{k+1}}.
\end{aligned}
$$

The recurrence is transformed to the following differential equation

$$\frac{z^{2k+1}f^{(2k+1)}(z)}{(2k+1)!}=\frac{(2k+2)z^{2k+1}}{(1-z)^{2k+3}}+\frac{2z^{2k+1}f^{(k)}(z)}{k!(1-z)^{k+1}}.$$

Multiplying both sides by $\left(\frac{1-z}{z}\right)^{2k+1}$,

$$\frac{(1-z)^{2k+1}f^{(2k+1)}(z)}{(2k+1)!}=\frac{2(k+1)}{(1-z)^2}+\frac{2(1-z)^k f^{(k)}(z)}{k!},$$

which is a Cauchy–Euler differential equation. This type of differential equations arises naturally to the analysis of searching–sorting algorithms and urn models [14]. Substituting $x=1-z$, and putting $h(x)=f(1-x)$, we get

$$\frac{(-1)^{2k+1}x^{2k+1}h^{(2k+1)}(x)}{(2k+1)!}=\frac{2(k+1)}{x^2}+\frac{2(-1)^k x^k h^{(k)}(x)}{k!}.$$

We use the differential operator $\Theta$ for the solution of the differential equation. It is defined by

$$\Theta\big(h(x)\big) := xh'(x)$$

and by induction

$$\binom{\Theta}{k} h(x) = \frac{x^k h^{(k)}(x)}{k!}.$$

Applying the operator, our equation becomes

$$\mathcal{P}_{2k+1}(\Theta)h(x) = \frac{2(k+1)}{x^2},$$

where the indicial polynomial is equal to

$$\mathcal{P}_{2k+1}(\Theta) = (-1)^{2k+1}\binom{\Theta}{2k+1} - 2(-1)^k\binom{\Theta}{k}.$$

We proceed to identify the nature of the roots of the polynomial. A Lemma follows:

**Lemma 3.1.1.** *The indicial polynomial $\mathcal{P}_{2k+1}(\Theta)$ has $2k+1$ simple roots, with real parts greater than or equal to $-2$. The real roots are $0, 1, \dots, k-1$; $-2$; $3k+2$, if $k$ is odd and the $2\left\lfloor \frac{k}{2} \right\rfloor$ complex roots $\rho_1, \dots, \rho_{\lfloor \frac{k}{2} \rfloor}$ with their conjugates $\overline{\rho}_1, \dots, \overline{\rho}_{\lfloor \frac{k}{2} \rfloor}$.*

**Proof.** Let $\alpha = x + iy$ being a root of the polynomial. Then

$$\binom{\alpha}{2k+1} = -2(-1)^k\binom{\alpha}{k}. \tag{3.2}$$

From Eq. (3.2), we deduce that $k$ real roots of the polynomial $\mathcal{P}_{2k+1}(\Theta)$ are $0, 1, \ldots, k-1$. For $\alpha \neq 0, 1, \ldots, k-1$, we have that

$$\frac{(\alpha-k)}{(k+1)} \cdot \frac{(\alpha-k-1)}{(k+2)} \cdot \ldots \cdot \frac{(\alpha-2k)}{(2k+1)} = -2(-1)^k. \tag{3.3}$$

Suppose that $\mathfrak{Re}(\alpha) < -2$. Then

$$\begin{aligned}
\left|\frac{\alpha-k}{k+1}\right| &= \frac{\sqrt{(x-k)^2+y^2}}{k+1} > \frac{\sqrt{\left(-(k+2)\right)^2+y^2}}{k+1} \geq \frac{k+2}{k+1}. \\
\left|\frac{\alpha-k-1}{k+2}\right| &= \frac{\sqrt{(x-k-1)^2+y^2}}{k+2} > \frac{\sqrt{\left(-(k+3)\right)^2+y^2}}{k+2} \geq \frac{k+3}{k+2}. \\
&\vdots \\
\left|\frac{\alpha-2k}{2k+1}\right| &= \frac{\sqrt{(x-2k)^2+y^2}}{2k+1} > \frac{\sqrt{\left(-2(k+1)\right)^2+y^2}}{2k+1} \geq \frac{2k+2}{2k+1},
\end{aligned}$$

so the product of the moduli in Eq. (3.3) is greater than or equal to the telescoping product $(2k+2)/(k+1) = 2$, arriving in contradiction. Thus, for any root $\alpha$ of the polynomial, it is proved that $\mathfrak{Re}(\alpha) > -2$. Moreover, this argument shows that $-2$ is the unique root with real part equal to $-2$. To see all roots are simple, assume that there does exist a repeated root $\alpha$. Differentiating the polynomial,

$$\mathcal{P}'_{2k+1}(\Theta) = -\binom{\Theta}{2k+1}\sum_{j=0}^{2k}\frac{1}{\Theta-j} - 2(-1)^k\binom{\Theta}{k}\sum_{j=0}^{k-1}\frac{1}{\Theta-j}$$

and

$$\binom{\alpha}{2k+1}\sum_{j=0}^{2k}\frac{1}{\alpha-j} = -2(-1)^k\binom{\alpha}{k}\sum_{j=0}^{k-1}\frac{1}{\alpha-j}. \tag{3.4}$$

Eq. (3.2) and (3.4) imply that for a repeated root must hold

$$\sum_{j=0}^{2k}\frac{1}{\alpha-j}=\sum_{j=0}^{k-1}\frac{1}{\alpha-j}$$

or

$$\sum_{j=k}^{2k}\frac{1}{\alpha-j}=0. \tag{3.5}$$

Consequently, Eq. (3.5) implies that $\mathfrak{Im}(\alpha)=0$ and $\alpha\in(k,k+1)\cup(k+1,k+2)\cup\ldots\cup(2k-1,2k)$. Considering Eq. (3.2), the left-hand side has modulus less than $1$, since $k<\alpha<2k+1$. However, the right-hand side of Eq. (3.2) has modulus greater than $2$, leading to contradiction. ∎

Since the roots are simple, we can factor our polynomial in the form

$$(\Theta-r_1)(\Theta-r_2)\ldots(\Theta-r_{2k})(\Theta+2)h(x)=\frac{2k+2}{x^2}.$$

The solution to the differential equation $(\Theta-a)h(x)=x^b$ is [63],

$$h(x)=\begin{cases}\dfrac{x^b}{b-a}+cx^a, \text{ if } a\neq b\\ \\ x^b\log_e(x)+cx^a, \text{ if } a=b.\end{cases}$$

Therefore, applying $(2k+1)$ times the solution, we obtain

$$h(x)=\frac{2k+2}{(-2-r_1)(-2-r_2)\ldots(-2-r_{2k})}\frac{\log_e(x)}{x^2}+\sum_{j=1}^{2k+1}c_jx^{r_j},$$

where $c_j$ are the constants of integration. Note that

$$\mathcal{P}_{2k+1}(\Theta) = (\Theta + 2)\mathcal{S}_{2k}(\Theta),$$

thus, the expression in the denominator $\mathcal{S}_{2k}(-2)$ is

$$\begin{aligned}\mathcal{S}_{2k}(-2) &= \mathcal{P}'_{2k+1}(-2)\\ &= -\binom{-2}{2k+1}\sum_{j=0}^{2k}\frac{1}{-2-j} - 2(-1)^k\binom{-2}{k}\sum_{j=0}^{k-1}\frac{1}{-2-j}\\ &= -(2k+2)(H_{2k+2} - H_{k+1}).\end{aligned}$$

Reverting to the previous notation,

$$f(z) = -\frac{1}{H_{2k+2} - H_{k+1}}\frac{\log_e(1-z)}{(1-z)^2} + \sum_{j=1}^{2k+1} c_j(1-z)^{r_j}.$$

Using the identity [26],

$$\frac{1}{(1-z)^{m+1}}\log_e\left(\frac{1}{1-z}\right) = \sum_{n=0}^{\infty}\Big(H_{n+m} - H_m\Big)\binom{n+m}{m}z^n$$

and the binomial Theorem, the solution of the differential equation can be written in terms of series,

$$f(z) = \frac{1}{H_{2k+2} - H_{k+1}}\sum_{n=0}^{\infty}\Big((n+1)H_n - n\Big)z^n + \sum_{n=0}^{\infty}\sum_{j=1}^{2k+1} c_j(-1)^n\binom{r_j}{n}z^n.$$

Extracting the coefficients, the expected number of comparisons of 'median of $(2k+1)$' Quicksort is

$$\mathbb{E}(C_{n\{2k+1\}}) = \frac{1}{H_{2k+2}-H_{k+1}}\big((n+1)H_n - n\big) + \sum_{j=1}^{2k+1}(-1)^n \mathfrak{Re}\left(c_j\binom{r_j}{n}\right).$$

The real roots of the polynomial $0, 1, \ldots, (k-1)$ do not contribute to the expected number of comparisons, since $n > 2k+1$ and when $k$ is odd, the root $(3k+2)$ adds a negligible constant contribution. Further, note that the root $r_{2k+1} = -2$, contributes $c_{2k+1}(n+1)$, with $c_{2k+1} \in \mathbf{R}$. Therefore

$$\begin{aligned}\mathbb{E}(C_{n\{2k+1\}}) = &\frac{1}{H_{2k+2}-H_{k+1}}\big((n+1)H_n - n\big) + c_{2k+1}(n+1)\\ &+ 2\sum_{j=1}^{\lfloor\frac{k}{2}\rfloor}(-1)^n \mathfrak{Re}\left(c_j\binom{\rho_j}{n}\right) + O(1).\end{aligned}$$

The asymptotics of the real parts in the last sum is given in [27]. It is proved that

$$2(-1)^n \mathfrak{Re}\left(c_j\binom{\rho_j}{n}\right) = O(n^{-(\mathfrak{Re}(\rho_j)+1)})$$

and asymptotically the expected number of key comparisons is

$$\begin{aligned}\mathbb{E}(C_{n\{2k+1\}}) = &\frac{1}{H_{2k+2}-H_{k+1}} n\log_e(n) + \left(c_{2k+1} + \frac{1}{H_{2k+2}-H_{k+1}}(\gamma-1)\right)n\\ &+ o(n),\end{aligned}$$

since all the other roots have real parts greater than $-2$. The solution contains the case of ordinary Quicksort, where the pivot is randomly selected and of 'median of $3$' Quicksort. The coefficient of the leading term in the latter case is

$$\frac{1}{H_4 - H_2} = \frac{12}{7},$$

thus the expected number of key comparisons is

$$\mathbb{E}(C_{n\{3\}}) = \frac{12}{7}(n+1)H_n + O(n).$$

van Emden [19] obtained the asymptotic cost of the expected number of comparisons, using entropy arguments. The leading term is equal to

$$an \log_2(n),$$

where

$$a = \frac{1}{\mathbb{E}(H)} = -\frac{1}{2\displaystyle\int_0^1 xg(x)\log_2(x)\,\mathrm{d}x}.$$

Here $g(x)$ denotes the probability density of the median of a random sample of $(2k+1)$ elements and $\mathbb{E}(H)$ is the expected information yielded from a comparison. The nice and simple asymptotic form for the mean number of comparisons is

$$\mathbb{E}(C_{n\{2k+1\}}) \sim \frac{\log_e(2)}{H_{2k+2} - H_{k+1}} n \log_2(n).$$

Note that this result, yields the expected number of comparisons of standard Quicksort, for $k = 0$. In this case,

$$a = 2\log_e 2,$$

thus

$$\mathbb{E}(C_n) \sim 1.386n \log_2(n).$$

The notion of entropy is of great importance to the analysis presented in this thesis. Entropy is a measure of uncertainty regarding the events of a random variable. In other words, a higher uncertainty about the outcome of a random variable pertains to increased entropy. In the trivial case, where the probability of occurrence of an event is $1$, the entropy is equal to $0$, as there is no uncertainty. A formal definition follows [65].

**Definition 3.1.2.** *The Shannon's entropy $H$ of a discrete random variable $X$ taking the values $x_1, x_2, \ldots, x_n$ with probabilities $p(x_1), p(x_2), \ldots, p(x_n)$ is defined by,*

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b\bigl(p(x_i)\bigr).$$

The base $b$ of the logarithm will be normally equal to two; in this case we measure bits of entropy. We should mention that the notation $H(X)$ does not merely denote a function of $X$; entropy is a function of the probability distribution. Generally, in sorting algorithms that utilise comparisons for this task, entropy quantifies the amount of information gained from the sorting. Consider an unsorted array, with all the $n!$ permutations equally likely. A comparison gives $1$ bit of information, thus at least $\log_2(n!) = \Omega\bigl(n \log_2(n)\bigr)$ comparisons are needed for a complete sort – see in [5]. This quantity is called information–theoretic lower bound. The range of entropy is given in the following Lemma.

**Lemma 3.1.3.**

$$0 \leq H(X) \leq \log_b(n).$$

**Proof.** Since $\log_b p(x_i) \leq 0$, the left inequality follows immediately. Noting that the logarithm is concave function and applying Jensen's inequality [38], which for a random variable $X$ and a concave function $f$, states that

$$f\big(\mathbb{E}(X)\big) \geq \mathbb{E}\big(f(X)\big),$$

we have

$$\begin{aligned} \sum_{i=1}^{n} p(x_i) \log_b \left(\frac{1}{p(x_i)}\right) &\leq \log_b \left(\sum_{i=1}^{n} p(x_1) \frac{1}{p(x_i)}\right) \\ &= \log_b(n). \end{aligned}$$ ∎

The next four definitions can be found in [16], [53] and [65].

**Definition 3.1.4.** *The joint entropy $H(X, Y)$ of two discrete random variables $X$ and $Y$ is defined as*

$$H(X, Y) = -\sum_{x \in X} \sum_{y \in Y} p(x, y) \log_b p(x, y),$$

*where $p(x, y)$ is the probability that $X$ takes the value $x$ and $Y$ the value $y$.*

**Definition 3.1.5.** *The conditional entropy $H(X|Y)$ of two discrete random variables $X$ and $Y$ is defined as*

$$H(X|Y) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) \log_b \frac{p(x,y)}{p(y)}.$$

**Definition 3.1.6.** *The information content of a random variable $X$, with probability distribution $\mathbb{P}(X)$ is*

$$I(X) = -\log_b \mathbb{P}(X).$$

From definition 3.1.6, one can easily deduce, that

$$H(X) = \mathbb{E}\big(I(X)\big).$$

In other words, entropy is the expected value of the information. We proceed to the definition of mutual information.

**Definition 3.1.7.** *The mutual information of two discrete random variables $X$ and $Y$ is defined as:*

$$I(X \wedge Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log_b \frac{p(x,y)}{p(x)p(y)} = H(X) - H(X|Y)$$

*and quantifies the amount of information provided about $X$ by $Y$.*

These definitions will be used in a later part of the thesis.

We have to point out that up until now, our analysis did not take into account the added overhead of finding the median at each stage. In the simple case of three elements, the overhead is not significant, but for larger samples this might have adversary effects to the efficiency of Quicksort.

For the selection of the median, we use Hoare's Find algorithm [29] or Quickselect. This simple and intuitive algorithm searches for an element of a given rank $m$ in an array of $n$ keys. As in Quicksort, one partitions the array around a randomly chosen pivot, which at the end of the partition process is moved to its final position, $j$. If $m = j$, the pivot is the sought element and the search is completed. Otherwise, if $m < j$, Quickselect is recursively invoked to the left subarray of $j - 1$ keys. Conversely, if $m > j$, we search in the right subarray for the element of rank $(m - j)$.

Quickselect is ideal in situations where we want to identify order statistics, without the need to do a complete sort. The average number of comparisons $\mathbb{E}(C_{n;m})$ required for the retrieval of the $m$-th order statistic in an array of $n$ keys, is given by [46]

$$\mathbb{E}(C_{n;m}) = 2\big(n + 3 + (n+1)H_n - (m+2)H_m - (n-m+3)H_{n+1-m}\big).$$

For the sample of $(2k+1)$ keys, the rank of the median is $k+1$. Therefore,

$$\mathbb{E}(C_{2k+1;k+1}) = 2\big(2k + 4 + (2k+2)H_{2k+1} - 2(k+3)H_{k+1}\big).$$

The cost for finding the median of a random sample of $3$ keys is $8/3$ comparisons at each stage. However, as the sample of keys increases, in order to obtain a more accurate estimate of the median, the added overhead imposes a bottleneck to the efficiency of the algorithm.

The computation of other measures of this variant, such as the expected number of exchanges and passes can be performed by solving analogous recurrences, as the one for the number of comparisons. The average number of passes is recursively given by

$$\mathbb{E}(P_{n\{2k+1\}}) = 1 + \frac{2}{\binom{n}{2k+1}} \sum_{j=1}^{n} \binom{j-1}{k}\binom{n-j}{k} \mathbb{E}(P_{j-1\{2k+1\}}),$$

where the "toll function" is now one recursive call to the algorithm, after the chosen pivot is the median of $(2k+1)$ keys, which yields two subarrays. This recurrence can be turned to a differential equation with solution

$$\mathbb{E}(P_{n\{2k+1\}}) = \frac{n+1}{2(H_{2k+2} - H_{k+1})} - 1,$$

noting that for $k = 0$, the average number of passes is $n$. In the scheme, where arrays containing $m$ or fewer keys are sorted by insertion sort, the average costs are reduced. We refer to [27] where this variant is analysed.

## 3.2 Remedian Quicksort

In the previous section, we analysed the modification of Quicksort, where the pivot is selected as the median of a sample of $(2k+1)$ elements. This variant offers better protection against the occurrence of trivial partitions. However, there are some cases, where the running time of this partitioning scheme can go quadratic. Consider the application of 'median of $3$' Quicksort in an array of $n$ numbers, where the keys at positions $1$, $n$ and $\left\lfloor \frac{n+1}{2} \right\rfloor$ are selected as elements of the sample. In case that two keys of this sample happen to be the smallest (or greatest) elements of the array, the chosen pivot will be $2$ (or $n-1$), leading to trivial partitioning, making Quicksort everything else, except quick! In [63], a permutation of the array $\{1, \ldots, 15\}$ is given, which leads Quicksort to worst-case performance and in [20], an algorithm is presented which forms the worst-case permutation.

In order to remedy this, a bigger sample of $(2k+1)^{\beta}$ keys is randomly selected, its remedian is found and used as partitioning element of the array to be sorted. The remedian of the sample is recursively defined to be the median of $(2k+1)$ remedians of $(2k+1)^{\beta-1}$ elements, where the remedian of $(2k+1)$ elements is the median, and is shown to be a robust estimator of the median [60], [72].

A particular case of the remedian Quicksort widely used in sorting applications is Tukey's 'ninther', where the selected pivot is the median of three medians of three samples, each containing three elements [14], [72]. In practical implementations, this variant exhibits faster running time [7] with little added overhead. Specifically, the computation of the remedian of $9$ elements takes on average $4 \times \frac{8}{3}$ comparisons at each call – four times more than finding the median of $3$ randomly chosen keys.

Let $C_{n\{(2k+1)^{\beta}\}}$ denote the number of comparisons required for the complete sorting of an array of $n$ distinct keys, where the chosen pivot at each call is the remedian of a random sample of $(2k+1)^{\beta}$ elements. The recurrence relation is much more complicated than the previous ones and the probability $p_j^{\beta}$ that the remedian of $(2k+1)^\beta$ elements is the $(j+1)$-th element is

$$p_j^\beta = (2k+1)! \sum_{\alpha_1+\ldots+\alpha_{2k+1}=j} p(\alpha_1,\ldots,\alpha_{2k+1}) \frac{\binom{j}{\alpha_1,\ldots,\alpha_{2k+1}}\binom{(2k+1)^\beta-j-1}{(2k+1)^{\beta-1}-1-\alpha_1,\ldots,(2k+1)^{\beta-1}-\alpha_{2k+1}}}{\binom{(2k+1)^\beta}{(2k+1)^{\beta-1},\ldots,(2k+1)^{\beta-1}}},$$

where

$$\binom{j}{\alpha_1,\ldots,\alpha_{2k+1}} = \frac{j!}{\alpha_1!\ldots\alpha_{2k+1}!}$$

is the multinomial coefficient and $p(\alpha_1,\ldots,\alpha_{2k+1})$ is defined by

$$p(\alpha_1,\ldots,\alpha_{2k+1}) = p_{\alpha_1}^{(\beta-1)}(p_0^{(\beta-1)}+\ldots+p_{\alpha_2-1}^{(\beta-1)})\cdot\ldots\cdot(p_{\alpha_{2k+1}}^{(\beta-1)}+\ldots+p_{(2k+1)^\beta-1}^{(\beta-1)}),$$

with $p^{(0)}=1$. See as well [14], where the ‘splitting’ probabilities of $3^d$ remedian are presented.

Bentley’s and McIlroy’s experiments on the ‘remedian of $3^2$’ Quicksort [7] showed that the average number of key comparisons is $1.094n\log_2(n)-0.74n$, very close to the information–theoretic lower bound of $n\log_2(n)-1.44n$. This Quicksort utilises the ‘ninther’ partitioning for large arrays, then the ‘median of 3’ is used and as the algorithm proceeds, the partitioning strategy changes to the standard uniform pivot selection. The paper written by Durand [18] confirmed these experimental results, where the average number of comparisons is being given by

$$\mathbb{E}(C_{n\{3^2\}}) = 1.5697n\log_e(n) - 1.0363n + 1.5697\log_e(n) - 7.3484 + O\left(\frac{1}{n}\right).$$

From a theoretical point of view, this variant yields savings on the expected time needed for the sorting, with little additional cost of computing the remedian. However, the recurrences are quite involved, as the remedian has an inherent recursive definition. A different approach would be to randomly choose a larger sample and use its elements as pivots through complete sorting. An obvious advantage of this method, is that the cost of computing the median or remedian of a sample at each call of the algorithm is avoided and instead all the pivots for the subsequent calls belong in one sample.

## 3.3 Samplesort

Having examined the strategy of selecting the median of a sample as pivot and the more complicated 'remedian' variant, we proceed to the analysis of Samplesort algorithm invented by Frazer and McKellar [24]. Instead to randomly select a sample of keys at each stage, computing the median and using it as pivot, a larger sample of $2^k - 1$ keys is selected and extracted out of the array. It is sorted and its keys are being used as partitioning elements for the sorting of the array.

First the median of the sample is used as pivot, then the lower quartile to the lower subarray and the upper one to the subarray of the elements that are greater than the median. When the sample is exhausted, the resulting subarrays can be recursively sorted by the same procedure or by standard Quicksort.

For convenience, let $2t + 1 = 2^k - 1$ and let us denote the total number of comparisons of Samplesort applied to $n$ keys by $C_n^{\{2^k-1\}}$. Then, $C_n^{\{2^k-1\}}$ is equal to the number of comparisons to sort the sample of $2^k - 1$ elements, plus the number of comparisons to insert its elements to the remainder of the array, plus the number of comparisons required to sort the resulting $2^k$ subarrays, using ordinary Quicksort [24]. For the sorting of the sample, we use Quicksort and the average number of comparisons is

$$2(2t+2)H_{2t+1} - 4(2t+1). \tag{3.6}$$

Assume that the sorted sample is $x_1 < x_2 < \ldots < x_{t+1} < \ldots < x_{2t+1}$. First, the median $x_{t+1}$ is inserted to its final position in the array of $n-2t$ keys, by pairwise comparisons of the $n-2t-1$ keys to $x_{t+1}$. The cost of partitioning is $n-2t-1$ comparisons. Then, the first quartile is inserted to the subarray of the elements less than the median $x_{t+1}$ and the third quartile to the subarray of the elements greater than $x_{t+1}$. The cost of partitioning these two subarrays is $n-2t-1$ comparisons, since the sum of their lengths is $n-2t-1$ keys. The process is continued until all the elements of the sample are used as pivots and this will take $2t+1$ partitioning stages. Thus, an approximation to the average number of comparisons for the insertion of the sample is [24], [49],

$$(n-2t-1)\log_2(2t+1). \tag{3.7}$$

After all elements have been inserted, there are $2(t+1)$ subarrays to be sorted by Quicksort. The expected number of comparisons is [24]

$$2(n+1)(H_{n+1} - H_{2(t+1)}) - 2(n-2t-1). \tag{3.8}$$

Putting together Eq. (3.6), (3.7) and (3.8), we have that the expected number of comparisons of Samplesort is

$$\begin{aligned}
&2(n+1)(H_{n+1}-H_{2(t+1)})-2(n-2t-1)+(n-2t-1)\log_2(2t+1)\\
&\qquad+2(2t+2)H_{2t+1}-4(2t+1). \qquad (3.9)
\end{aligned}$$

For large values of $n$, the expected number of comparisons is given by the following Corollary.

**Corollary 3.3.1** (Frazer and McKellar [24]). *The asymptotic expected number of comparisons taken by Samplesort for the sorting of an array of $n$ keys, using a randomly chosen sample of $l$ keys, is*

$$\mathbb{E}(C_n^{\{l\}})=1.386n\log_2(n)-0.386\big(n-l\big)\log_2(l)-2n-0.846l.$$

It is worthwhile to note that the programming of Samplesort is simple and straightforward, thus making it a suitable candidate to sorting applications. It is proven in [24], that the procedure is asymptotically optimal, i.e. as $n \to \infty$, the expected number of comparisons approaches the information–theoretic bound. The process of randomly drawing the sample out of the array to be samplesorted should be carefully selected, to the effect that the elements of the sample will produce 'balanced' partitions. The main feature of Samplesort is that partitioning preserves the order of the sample, thus its keys are exchanged with the ones that they have to, so as to the exhaustion of the sample, its elements to be spread far apart.

In the direction of choosing a more ‘centered’ sample, one can proceed by randomly selecting three samples – each containing three keys – and computing their medians, at an extra cost of $8$ comparisons. For the sorting of larger arrays, the number of samples will obviously be greater. The medians will be used as elements of the sample and the remaining elements can be randomly chosen from the array. It should be noted, that this is an initial idea, lacking the mathematical analysis.

Albacea [2] derived a modification of Samplesort. This variant starts with $1$ key, that is used as pivot for the partition of $2$ keys, so to have a sorted array of $3$ keys. These keys are used as pivots for the partitioning of $4$ keys, so as to obtain a sorted array of $7$ keys and so on, until the whole array is sorted. It is proven [2] that the estimated expected number of key comparisons is

$$n\lceil \log_2(n+1)\rceil - 2^{\lceil \log_2(n+1)\rceil} - n + \lceil \log_2(n+1)\rceil + 1.$$

The derivation of the expected number of comparisons remains an open problem.

# Chapter 4

# Sorting by multiple pivots

In this Chapter, different partitioning routines are analysed. These schemes utilise many pivots for the partitioning of the array and naturally arise as a generalisation of the algorithm. The pivots are chosen uniformly at random and the array is partitioned into more than two subarrays. There is an additional overhead of comparing the pivots before the partitioning, which adds very small contributions to the running time. The aim of this modification is twofold: first to study if the possibility of worst-case scenario of the algorithm can be reduced further and secondly, to provide a theoretical basis for the analysis of the generalisation of the algorithm.

We show that the average case analyses of these variants can be fully described by a general recurrence model, which is transformed to a differential equation, whose solution provides the expected cost of these variants. Further, we demonstrate that the integration constants involved in the solution, can be efficiently computed using Vandermonde matrices.

## 4.1 Quicksorting on two pivots

Along the following lines, we present a variant of Quicksort, where $2$ pivots are used for the partitioning of the array. Let a random permutation of the keys $\{1, 2, \ldots, n\}$ to be sorted, with all the $n!$ permutations equally likely and let their locations in the array be numbered from left to right by $\{1, 2, \ldots, n\}$. The keys at locations $1$ and $n$ are chosen as pivots and since all the $n!$ permutations are equally likely to be the input, then all the $\binom{n}{2}$ pairs are equiprobable to be selected as pivots. At the beginning, the pivots are compared each other and are swapped, if they are not in order. If elements $i < j$ are selected as pivots, the array is partitioned into three subarrays: one with $(i-1)$ keys smaller than $i$, a subarray of $(j-i-1)$ keys between two pivots and the part of $(n-j)$ elements greater than $j$.

The algorithm then is recursively applied to each of these subarrays. The number of comparisons during the first stage is

$$\begin{aligned} A_{n,2} &= 1 + \big((i-1) + 2(j-i-1) + 2(n-j)\big) \\ &= 2n - i - 2, \end{aligned}$$

for $i = 1, \ldots, n-1$, and $j = i+1, \ldots, n$. Note that in the specific partitioning scheme, each element is compared once to $i$ and elements greater than $i$ are compared to $j$ as well. The average number of comparisons for the partitioning of $n$ distinct keys is

$$\mathbb{E}(A_{n,2}) = \frac{1}{\binom{n}{2}} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \left(2n - i - 2\right) = \frac{2}{n(n-1)} \left(\frac{5}{6}n^3 - 2n^2 + \frac{7}{6}n\right)$$

$$= \frac{5n-7}{3}.$$

Letting $C_{n,2}$ denote the number of comparisons of dual pivot Quicksort applied to an array of $n$ items, the recurrence for the expected number of comparisons is

$$\mathbb{E}(C_{n,2}) = \frac{5n-7}{3} + \frac{2}{n(n-1)} \times \left(\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbb{E}(C_{i-1,2}) + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbb{E}(C_{j-i-1,2}) + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbb{E}(C_{n-j,2})\right).$$

Note that the three double sums above are equal. Therefore, the recurrence becomes

$$\mathbb{E}(C_{n,2}) = \frac{5n-7}{3} + \frac{6}{n(n-1)} \sum_{i=1}^{n-1} (n-i)\mathbb{E}(C_{i-1,2}).$$

Letting $a_n = \mathbb{E}(C_{n,2})$, we have

$$a_n = \frac{5n-7}{3} + \frac{6}{n(n-1)} \sum_{i=1}^{n-1} (n-i)a_{i-1},\ n \geq 2.$$

It holds that $a_0 = a_1 = 0$. Multiplying both sides by $\binom{n}{2}$, we obtain

$$\begin{aligned}\binom{n}{2} a_n &= \binom{n}{2}\left(\frac{5n-7}{3} + \frac{6}{n(n-1)}\sum_{i=1}^{n-1}(n-i)a_{i-1}\right)\\ &= \frac{n(n-1)(5n-7)}{6} + 3\sum_{i=1}^{n-1}(n-i)a_{i-1}.\end{aligned}$$

This recurrence will be solved by the difference method. We have

$$\begin{aligned}\Delta F(n) &:= F(n+1) - F(n) \quad \text{and for higher orders}\\ \Delta^k F(n) &:= \Delta^{k-1}F(n+1) - \Delta^{k-1}F(n).\end{aligned}$$

Applying the difference operator

$$\begin{aligned}\Delta\binom{n}{2}a_n &= \binom{n+1}{2}a_{n+1} - \binom{n}{2}a_n = \frac{5n^2-3n}{2} + 3\sum_{i=0}^{n-1}a_i\\ \Delta^2\binom{n}{2}a_n &= \Delta\binom{n+1}{2}a_{n+1} - \Delta\binom{n}{2}a_n = 5n+1+3a_n.\end{aligned}$$

By definition,

$$\begin{aligned}\Delta^2\binom{n}{2}a_n &= \Delta\binom{n+1}{2}a_{n+1} - \Delta\binom{n}{2}a_n\\ &= \binom{n+2}{2}a_{n+2} - 2\binom{n+1}{2}a_{n+1} + \binom{n}{2}a_n\end{aligned}$$

and the recurrence becomes

$$\begin{aligned}&(n+1)(n+2)a_{n+2} - 2n(n+1)a_{n+1} + n(n-1)a_n = 2(5n+1+3a_n)\\ &\quad\Longrightarrow (n+1)\big((n+2)a_{n+2} - (n-2)a_{n+1}\big) - (n+2)\big((n+1)a_{n+1} - (n-3)a_n\big)\\ &\qquad = 2(5n+1).\end{aligned}$$

Dividing by $(n+1)(n+2)$, we obtain the telescoping recurrence

$$\frac{(n+2)a_{n+2}-(n-2)a_{n+1}}{n+2}=\frac{(n+1)a_{n+1}-(n-3)a_n}{n+1}+\frac{2(5n+1)}{(n+1)(n+2)},$$

which yields

$$\frac{(n+2)a_{n+2}-(n-2)a_{n+1}}{n+2}=2\sum_{j=0}^{n}\frac{5j+1}{(j+1)(j+2)}=\frac{18}{n+2}+10H_{n+1}-18.$$

The recurrence is equivalent to

$$na_n-(n-4)a_{n-1}=18+10nH_{n-1}-18n.$$

Multiplying by $\frac{(n-1)(n-2)(n-3)}{24}$, this recurrence is transformed to a telescoping one [63],

$$\binom{n}{4}a_n=\binom{n-1}{4}a_{n-1}+\frac{18(n-1)(n-2)(n-3)}{24}+10\binom{n}{4}H_{n-1}-18\binom{n}{4}.$$

Unwinding, we have

$$\begin{aligned}\binom{n}{4}a_n &= 18\sum_{j=1}^{n}\frac{(j-1)(j-2)(j-3)}{24}+10\sum_{j=1}^{n}\binom{j}{4}H_{j-1}\\ &\quad -18\sum_{j=1}^{n}\binom{j}{4}.\end{aligned}\tag{4.1}$$

The second sum of Eq. (4.1) is

$$\begin{aligned}\sum_{j=1}^{n}\binom{j}{4}H_{j-1} &= \sum_{j=1}^{n}\left(\binom{j}{4}\left(H_j-\frac{1}{j}\right)\right)=\sum_{j=1}^{n}\binom{j}{4}H_j-\sum_{j=1}^{n}\binom{j}{4}\frac{1}{j}\\ &=\binom{n+1}{5}\left(H_{n+1}-\frac{1}{5}\right)-\frac{1}{24}\sum_{j=1}^{n}(j-1)(j-2)(j-3)\\ &=\binom{n+1}{5}\left(H_{n+1}-\frac{1}{5}\right)-\binom{n}{4}\frac{1}{4},\end{aligned}$$

thus

$$\begin{aligned}&\binom{n}{4}a_n=\frac{9}{2}\binom{n}{4}+10\left(\binom{n+1}{5}\left(H_{n+1}-\frac{1}{5}\right)-\frac{1}{4}\binom{n}{4}\right)-18\binom{n+1}{5}.\\ &\Longrightarrow a_n=\frac{9}{2}+10\left(\frac{n+1}{5}\left(H_{n+1}-\frac{1}{5}\right)-\frac{1}{4}\right)-\frac{18(n+1)}{5}.\end{aligned}$$

The expected number of comparisons, when two pivots are chosen is

$$a_n=2(n+1)H_n-4n.$$

This is exactly the same as the expected number of comparisons for ordinary Quicksort.

Next, the expected number of key exchanges will be computed. The exchanges during partitioning are performed as follows. Set two pointers $l \leftarrow 2$, $u \leftarrow n-1$ and store temporarily the pivots in another array of size two, so the cells at locations $1$ and $n$ are empty, leaving two “holes”.

After the pivots are sorted by one comparison, the key at position $2$ is compared to the first pivot (i.e. the smaller of the two pivots); if it is less than the pivot, it is put into the left hole, which now is moved one position to the right and $l$ is increased by one. If it (i.e. the key at position $2$) is greater than the first pivot, it is compared to the second pivot (i.e. the greater of the two pivots). If it is less than the second pivot, $l$ is increased by one, otherwise $l$ stops.

Now, the $u$ pointer starts its downward scan. If an examined key is greater than both pivots, it is put into the right hole, which is moved one position to the left and $u$ is decreased by one. If a key is less than the second pivot and greater than the first, then $u$ is decreased by one. In case that a key is less than the first pivot, then $u$ stops its scan and the key that is greater to the second pivot, where $l$ has stopped is put to the right hole, which is moved one position to the left and the key where $u$ has stopped is put to the left hole, which is moved one position to the right. Then, $l$ is increased by one, $u$ is decreased by one and $l$ resumes its scan.

When pointers are crossed, the first pivot is put to the left hole, the second pivot is put to the right hole and partition is completed, since keys less than the first pivot are on its left, keys between two pivots on the middle and keys greater than the second pivot are on its right subarray. Note that the auxiliary space required for the storing of pivots is $O(1)$, since at the end of the partition routine, the pivots are moved back to the array and two other new pivots can be stored, as the algorithm operates on a given subarray. We refer to [63] for further details of this scheme.

The average number of swaps during the first stage is

$$\frac{1}{\binom{n}{2}} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (i-1) = \frac{1}{\binom{n}{2}} \sum_{i=1}^{n-1} (n-i)(i-1) = \frac{1}{\binom{n}{2}} \left( \sum_{i=1}^{n-1} (n-i)i - \sum_{i=1}^{n-1} (n-i) \right),$$

since $(i-1)$ keys are less than pivot $i$. Thus, the average contribution is

$$\frac{2}{n(n-1)}\left(\frac{n^3-n}{6}-\frac{n(n-1)}{2}\right)=\frac{n-2}{3}.$$

For the $(n-j)$ keys greater than $j$ the average value is the same, because the sums are equal. Adding the two final "exchanges" to get the pivots in place, the average number of exchanges during the partitioning routine is $\left(\frac{2(n+1)}{3}\right)$. Letting $S_{n,2}$ denote the number of exchanges of dual pivot Quicksort, the recurrence for the mean number of exchanges in course of the algorithm is

$$\begin{aligned}\mathbb{E}(S_{n,2}) &= \frac{2(n+1)}{3}+\frac{2}{n(n-1)}\\ &\quad\times\left(\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\mathbb{E}(S_{i-1,2})+\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\mathbb{E}(S_{j-i-1,2})+\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\mathbb{E}(S_{n-j,2})\right)\\ &= \frac{2(n+1)}{3}+\frac{6}{n(n-1)}\sum_{i=1}^{n-1}(n-i)\mathbb{E}(S_{i-1,2}).\end{aligned}$$

Multiplying both sides by $\binom{n}{2}$,

$$\binom{n}{2}\mathbb{E}(S_{n,2})=\frac{n(n-1)(n+1)}{3}+3\sum_{i=1}^{n-1}(n-i)\mathbb{E}(S_{i-1,2}).$$

This recurrence is solved in [63]: here we present a solution using generating functions. Letting $b_n=\mathbb{E}(S_{n,2})$ and $g(z)=\sum_{n=0}^{\infty}b_nz^n$ be the generating function of the average number of exchanges, the recurrence is transformed to the following differential equation:

$$\frac{z^2}{2}\frac{\mathrm{d}^2g(z)}{\mathrm{d}z^2}=\frac{z^2}{3}\frac{\mathrm{d}^3}{\mathrm{d}z^3}\left(\sum_{n=0}^{\infty}z^{n+1}\right)+3\sum_{n=1}^{\infty}\sum_{i=1}^{n}(n-i)b_{i-1}z^n.$$

The double sum is equal to

$$\begin{aligned}\sum_{n=1}^{\infty}\sum_{i=1}^{n}(n-i)b_{i-1}z^n &= b_0z^2+(2b_0+b_1)z^3+(3b_0+2b_1+b_2)z^4+\ldots\\ &= z^2(b_0+b_1z+b_2z^2+\ldots)+2z^3(b_0+b_1z+b_2z^2+\ldots)+\ldots\\ &= (z^2+2z^3+3z^4+\ldots)g(z)\\ &= \left(\sum_{n=0}^{\infty}nz^{n+1}\right)g(z)\end{aligned}$$

and our differential equation becomes

$$\frac{z^2}{2}\frac{\mathrm{d}^2g(z)}{\mathrm{d}z^2}=\frac{2z^2}{(1-z)^4}+3g(z)\left(\frac{z}{1-z}\right)^2.$$

Changing variables $v=1-z$, we have $f^{(k)}(v)=(-1)^kg^{(k)}(1-v)$. Thus,

$$\frac{(1-v)^2}{2}\frac{\mathrm{d}^2f(v)}{\mathrm{d}v^2}=\frac{2(1-v)^2}{v^4}+3f(v)\left(\frac{1-v}{v}\right)^2.$$

The differential equation can be simplified by multiplying both sides by $\left(\frac{v}{1-v}\right)^2$,

$$\frac{v^2}{2}\frac{\mathrm{d}^2f(v)}{\mathrm{d}v^2}=\frac{2}{v^2}+3f(v). \tag{4.2}$$

An elementary approach to solving this differential equation, is to assume that the solution is of the form $x^m$ [9]. Substituting the “trial solution” to Eq. (4.2), the characteristic or indicial polynomial is

$$\mathcal{P}_2(m)=m(m-1)-6,$$

with roots $m_1 = 3$ and $m_2 = -2$. Thus, the solution to the corresponding homogeneous equation is $c_1 v^3 + c_2 v^{-2}$, with $c_1, c_2 \in \mathbf{R}$. A particular solution of Eq. (4.2), which can be found e.g. using the method in [61], is

$$-\frac{4}{5}\log_e(v)v^{-2}.$$

By the initial conditions $f(1) = -f'(1) = 0$, the solution is

$$f(v) = \frac{4}{25}v^3 - \frac{20\log_e(v) + 4}{25v^2}.$$

In the next section, we will examine the generalised version of this differential equation. Reverting to variable $z$ and discarding terms for $n \leq 3$, we see that, expanding out the fraction term as a series,

$$g(z) = \sum_{n=0}^{\infty}\left(\frac{4}{5}\Big((n+1)H_n - n\Big) - \frac{4}{25}(n+1)\right)z^n.$$

Finally, the mean number of swaps of dual pivot Quicksort is

$$b_{n,2} = \frac{4}{5}(n+1)H_n - \frac{24n+4}{25},$$

which is nearly $2.4$ times greater than the expected number of exchanges of standard Quicksort.

The recurrence for the number of partitioning stages $P_{n,2}$ is much simpler;

$$P_{n,2} = 1 + P_{i-1,2} + P_{j-i-1,2} + P_{n-j,2}.$$

By the same reasoning, as in the derivation of the expected number of exchanges, the solution is

$$\mathbb{E}(P_{n,2}) = \frac{2}{5}(n+1) - \frac{1}{2}.$$

### 4.1.1 The variance of the number of key comparisons

It is desirable to compute the variance of the number of key comparisons of dual pivot Quicksort, as this measure provides a grip of the deviation of the random number of comparisons from its expected value. By the recursive relation, we have

$$\mathbb{P}(C_{n,2} = t) = \frac{1}{\binom{n}{2}} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbb{P}(A_{n,2} + C_{i-1,2} + C_{j-i-1,2} + C_{n-j,2} = t),$$

noting that the resulting subarrays are independently sorted, the above is

$$\frac{1}{\binom{n}{2}} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{l,m} \Big( \mathbb{P}(C_{i-1,2} = l)\mathbb{P}(C_{j-i-1,2} = m)\mathbb{P}(C_{n-j,2} = t - m - l - 2n + i + 2) \Big).$$

Letting $f_n(z) = \sum_{t=0}^{\infty} \mathbb{P}(C_{n,2} = t) z^t$ be the ordinary probability generating function for the number of comparisons needed to sort $n$ keys, we obtain

$$f_n(z) = \frac{1}{\binom{n}{2}} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} z^{2n-i-2} f_{i-1}(z) f_{j-i-1}(z) f_{n-j}(z). \tag{4.3}$$

It holds that $f_n(1) = 1$ and $f'_n(1) = 2(n+1)H_n - 4n$. The second order derivative of Eq. (4.3) evaluated at $z = 1$ is recursively given by

$$
\begin{aligned}
f''_n(1) \;&= \frac{2}{n(n-1)}\Bigg(\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(2n-i-2)^2 - \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(2n-i-2)\\
&+ 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(2n-i-2)\mathbb{E}(C_{i-1,2}) + 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(2n-i-2)\mathbb{E}(C_{j-i-1,2})\\
&+ 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(2n-i-2)\mathbb{E}(C_{n-j,2}) + 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\mathbb{E}(C_{i-1,2})\mathbb{E}(C_{j-i-1,2})\\
&+ 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\mathbb{E}(C_{i-1,2})\mathbb{E}(C_{n-j,2}) + 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\mathbb{E}(C_{j-i-1,2})\mathbb{E}(C_{n-j,2})\\
&+ \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}f''_{i-1}(1) + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}f''_{j-i-1}(1) + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}f''_{n-j}(1)\Bigg).
\end{aligned}
$$

The fourth and fifth sum turn out to be equal and by simple manipulation of indices, the sums involving products of expected values are equal. The double sum of the product of the mean number of comparisons can be simplified as follows, using Corollary 2.3.5:

$$
\begin{aligned}
&\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\mathbb{E}(C_{i-1,2})\mathbb{E}(C_{n-j,2}) = \sum_{i=1}^{n-1}\Bigg(\mathbb{E}(C_{i-1,2})\Big(\sum_{j=0}^{n-i-1}\mathbb{E}(C_{j,2})\Big)\Bigg)\\
&= \sum_{i=1}^{n-1}\Bigg(\Big((2iH_{i-1} - 4(i-1)\Big)\Big(2\binom{n-i+1}{2}H_{n-i} + \frac{n-i-5(n-i)^2}{2}\Big)\Bigg).
\end{aligned}
$$

Further, using $\binom{n-i+1}{2} = \binom{n-i}{2} + (n-i)$,

$$\begin{aligned}\sum_{i=1}^{n-1} i\binom{n-i+1}{2} H_{i-1}H_{n-i} &= \sum_{i=1}^{n-1}\Big((i-1)+1\Big)\binom{n-i+1}{2} H_{i-1}H_{n-i}\\ &= \sum_{i=1}^{n-1}(i-1)\binom{n-i}{2} H_{i-1}H_{n-i} + \sum_{i=1}^{n-1}\binom{n-i}{2} H_{i-1}H_{n-i}\\ &+ \sum_{i=1}^{n-1}(i-1)(n-i)H_{i-1}H_{n-i} + \sum_{i=1}^{n-1}(n-i)H_{i-1}H_{n-i}.\end{aligned}$$

The four sums can be evaluated using Corollary 3 in [70].

After some computations in MAPLE, that can be found in Appendix A, the recurrence is

$$\begin{aligned}f_n''(1) &= 2(n+1)(n+2)(H_n^2 - H_n^{(2)}) - H_n\left(\frac{17}{3}n^2 + \frac{47}{3}n + 6\right) + \frac{209}{36}n^2\\ &+ \frac{731}{36}n + \frac{13}{6} + \frac{6}{n(n-1)}\sum_{i=1}^{n-1}(n-i)f_{i-1}''(1).\end{aligned}$$

Subtracting $\binom{n}{2} f_n''(1)$ from $\binom{n+1}{2} f_{n+1}''(1)$, we have

$$\begin{aligned}\Delta\binom{n}{2} f_n''(1) &= 4n(n+1)(n+2)(H_n^2 - H_n^{(2)}) - \frac{nH_n}{9}(84n^2 + 198n + 42)\\ &+ 3\sum_{i=1}^{n} f_{i-1}'' + \frac{n}{9}(79n^2 + 231n + 14),\end{aligned}$$

using the identity [63]

$$H_{n+1}^2 - H_{n+1}^{(2)} = H_n^2 - H_n^{(2)} + \frac{2H_n}{n+1}.$$

Also, it holds that

$$\Delta^2 \binom{n}{2} f_n''(1) = 12(n+1)(n+2)(H_n^2 - H_n^{(2)}) - H_n(20n^2 + 32n - 12) \\ + 17n^2 + 37n + 3f_n''(1).$$

The left-hand side of the previous equation is the same as

$$\binom{n+2}{2} f_{n+2}''(1) - 2\binom{n+1}{2} f_{n+1}''(1) + \binom{n}{2} f_n''(1)$$

and the recurrence becomes

$$(n+1)(n+2)f_{n+2}''(1) - 2n(n+1)f_{n+1}''(1) + n(n-1)f_n''(1) \\ = 2\left(12(n+1)(n+2)(H_n^2 - H_n^{(2)}) - H_n(20n^2 + 32n - 12) + 17n^2 + 37n + 3f_n''(1)\right).$$

Dividing by $(n+1)(n+2)$, we obtain the telescoping recurrence

$$\frac{(n+2)f_{n+2}''(1) - (n-2)f_{n+1}''(1)}{n+2} \\ = \frac{(n+1)f_{n+1}''(1) - (n-3)f_n''(1)}{n+1} \\ + 2\left(12(H_n^2 - H_n^{(2)}) - \frac{H_n(20n^2 + 32n - 12)}{(n+1)(n+2)} + \frac{17n^2 + 37n}{(n+1)(n+2)}\right),$$

with solution

$$(n+2)f_{n+2}''(1) - (n-2)f_{n+1}''(1) = (24n^2 + 100n + 104)(H_{n+1}^2 - H_{n+1}^{(2)}) \\ - H_{n+1}(88n^2 + 292n + 224) + 122n^2 + 346n + 224,$$

which is equivalent to

$$
\begin{aligned}
nf''_n(1) - (n-4)f''_{n-1}(1) = (24n^2 + 4n)(H^2_{n-1} - H^{(2)}_{n-1}) \\
- H_{n-1}(88n^2 - 60n - 8) + 122n^2 - 142n + 20.
\end{aligned}
$$

Again as before, multiplying both sides by $\frac{(n-1)(n-2)(n-3)}{24}$, the recurrence telescopes with solution

$$
f''_n(1) = 4(n+1)^2(H^2_{n+1} - H^{(2)}_{n+1}) - 4H_{n+1}(n+1)(4n+3) + 23n^2 + 33n + 12.
$$

Using the well known fact that

$$
\mathrm{Var}(C_{n,2}) = f''_n(1) + f'_n(1) - \left(f'_n(1)\right)^2,
$$

the variance of the number of key comparisons of dual pivot Quicksort is

$$
7n^2 - 4(n+1)^2 H^{(2)}_n - 2(n+1)H_n + 13n. \tag{4.4}
$$

Note that the variance of dual pivot Quicksort is identical with the variance of ordinary Quicksort. In the next subsection, we provide the theoretical explanation of this fact.

### 4.1.2 Distribution of the number of key comparisons

Our results have shown that dual pivot Quicksort has the same expected number of comparisons, and the same variance, as in the case of ‘one-pivot’ Quicksort. Thus, it is natural to ask if the two random variables have the same distribution. We now show this, after an argument sketched by Prof. Colin McDiarmid [52].

Suppose that an array of $n$ distinct keys $x_1, x_2, \ldots, x_n$ is to be sorted by Quicksort and let, as usual, $C_n$ be the random number of comparisons required for the sorting. Obviously, $C_1 = 0$, $C_2 = 1$ and for $n \geq 3$ we pick uniformly at random an ordered pair of distinct indices $(I, J)$ in $[n] = \{1, 2, \ldots, n\}$ and we use $x_I$ as the first pivot. Given that $I = i$, the pivot $x_i$ partitions the array of $n$ keys to the subarray of $(i-1)$ keys less than $x_i$ and to the subarray of $(n-i)$ keys greater than $x_i$ by $(n-1)$ comparisons.

Given that $I = i$, if $x_J < x_i$, then $x_J$ is a uniformly at random chosen pivot from the subarray of $(i-1)$ elements less than $x_i$. In this case, for $I = i$ and $J = j$, the subarray of $(i-1)$ keys is partitioned to the subarray of $(j-1)$ keys less than $x_j$ and to the subarray of $(i-j-1)$ keys greater than $x_j$ by $(i-2)$ comparisons. Note that $(i-2)$ keys are compared to both pivots in two partitioning stages. Therefore, the following recurrence holds:

$$\begin{aligned}\mathbb{P}(C_n = t) &= \frac{1}{\binom{n}{2}} \sum_{j=1}^{n-1} \sum_{i=j+1}^{n} \mathbb{P}\big((n-1) + (i-2) + C_{j-1}^{(1)} + C_{i-j-1}^{(2)} + C_{n-i}^{(3)} = t\big) \\ &= \frac{2}{n(n-1)} \sum_{j=1}^{n-1} \sum_{i=j+1}^{n} \mathbb{P}\big((n+i-3) + C_{j-1}^{(1)} + C_{i-j-1}^{(2)} + C_{n-i}^{(3)} = t\big),\end{aligned}$$

where $C_n^{(1)}$, $C_n^{(2)}$ and $C_n^{(3)}$ are independent copies of $C_n$ – that is, are random variables with the same distribution as $C_n$, independent of it and each other.

If $x_J > x_i$, then $x_J$ is a uniformly at random selected pivot from the subarray of $(n-i)$ keys greater than $x_i$. Given that $I = i$ and $J = j$, the subarray of

$(n-i)$ keys is partitioned to the subarray of $(j-i-1)$ keys less than $x_j$ and to the subarray of $(n-j)$ keys greater than $x_j$ by $(n-i-1)$ comparisons. The recurrence relation is

$$\begin{aligned}\mathbb{P}(C_n = t) &= \frac{1}{\binom{n}{2}} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbb{P}\big((n-1) + (n-i-1) + C^{(1)}_{i-1} + C^{(2)}_{j-i-1} + C^{(3)}_{n-j} = t\big) \\ &= \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbb{P}\big((2n-i-2) + C^{(1)}_{i-1} + C^{(2)}_{j-i-1} + C^{(3)}_{n-j} = t\big),\end{aligned}$$

where as in the previous recurrence, $C^{(1)}_n$, $C^{(2)}_n$ and $C^{(3)}_n$ are independent copies of $C_n$. Observe that

$$\sum_{j=1}^{n-1} \sum_{i=j+1}^{n} (n+i-3) = \sum_{j=1}^{n-1} (n-j)(2n-j-2) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (2n-i-2),$$

thus for any two pivots selected uniformly at random, the recurrences are the same.

Recall that for the random number of comparisons $C_{n,2}$ of dual pivot Quicksort, it holds that $C_{1,2} = 0$, $C_{2,2} = 1$ and for $n \geq 3$ we choose uniformly at random an ordered pair of distinct indices $(I, J)$ in $[n] = \{1, 2, \ldots, n\}$. The pivots $x_I$ and $x_J$ are sorted by one comparison and we assume that its outcome is $x_I < x_J$. Given that $I = i$ and $J = j$, the array is partitioned to the subarray of $(i-1)$ keys less than $x_i$, the subarray of $(j-i-1)$ keys between two pivots and the subarray of $(n-j)$ keys greater than $x_j$. Since keys greater than $x_i$ are compared with the other pivot as well, the recurrence for the random number of comparisons is

$$\begin{aligned}\mathbb{P}(C_{n,2}=t) &= \frac{1}{\binom{n}{2}}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\mathbb{P}\big(1+(i-1)+2(j-i-1)+2(n-j)\\ &\qquad + C^{(1)}_{i-1,2}+C^{(2)}_{j-i-1,2}+C^{(3)}_{n-j,2}=t\big)\\ &= \frac{2}{n(n-1)}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\mathbb{P}\big((2n-i-2)+C^{(1)}_{i-1,2}+C^{(2)}_{j-i-1,2}+C^{(3)}_{n-j,2}=t\big),\end{aligned}$$

where $C^{(1)}_{n,2}$, $C^{(2)}_{n,2}$ and $C^{(3)}_{n,2}$ are independent copies of $C_{n,2}$. Note that when $x_I > x_J$, the recurrence is the same. Thus, since dual pivot Quicksort and ordinary Quicksort satisfy the same recurrence and have the same initial conditions for $n = 1, 2$, we deduce that the random variables $C_{n,2}$ and $C_n$ are identically distributed.

## 4.2 Multikey partitioning

A natural extension of having two pivots would be to have some other number $k$ of pivots. Here, we study the idea of randomly picking $k$ pivots $i_1, i_2, \dots, i_k$ and partitioning the array simultaneously according to these.

Again, let a random permutation of the array $\{1, 2, \dots, n\}$ be given to be sorted using this variant, with all the $n!$ permutations equally likely to be the input. The $k$ rightmost keys are chosen as pivots, are compared to each other and exchanged, if they are out of order. The sorting of the pivots can be efficiently implemented by insertion sort. Since all $n!$ permutations of the keys are equally likely to be the input, this amounts to the fact that any $k$-subset of keys has equal probability to be selected.

The remaining $(n - k)$ keys are compared to the pivots and the array is partitioned to $(k + 1)$ subarrays. The partitioning can be performed as follows. We compare the leftmost key to a randomly chosen pivot; if it is smaller than this pivot, it is compared with another smaller pivot (if one exists). Otherwise it is compared with a larger pivot (to the right) and after a series of comparisons, is inserted to its place between any two pivots, or to the left of the smallest pivot or to the right of the greatest pivot. We continue in the same fashion, until all keys are examined.

In [28], each of the $(n - k)$ keys is compared to the pivots by binary search, so a key is compared first to the median of the sorted array of the pivots. If it is less, is compared with the first quartile, otherwise is compared with the third quartile and after a series of comparisons is inserted to its position. In worst case, it takes $O\big(\log_2(k)\big)$ comparisons for the insertion of a key. Then, multipivot Quicksort is recursively applied to each of the resulting segments that contains at least $(k + 1)$ keys and arrays with less than $(k + 1)$ keys are sorted by insertion sort in $O(n)$ time.

This is equivalent to $(k+1)$-ary search trees, which is a generalisation of binary trees. Indeed, if $n \geq k + 1$, the $k$ pivots are stored in the root node of the tree in increasing order and the remaining $(n - k)$ keys are placed in the resulting $(k + 1)$ subtrees of the root. In case that $n = 0$, the tree is empty and if $n \leq k$, the tree has a single node, which stores the keys in order. Under the assumption of uniformity, this is a $(k + 1)$-random tree of $n$ nodes – see the article of Chern *et al.* [14] and Mahmoud's book [49] for the correspondence between trees and variants of Quicksort.

Let $f(n,k)$ denote the expected cost of the algorithm applied to an array of $n$ keys. We deliberately allow some flexibility in the form of cost; a typical example might be the number of comparisons. The expected cost of this variant is recursively given by

$$\begin{aligned} f(n,k) &= T(n,k) \\ &\quad + \frac{1}{\binom{n}{k}} \underbrace{\sum_{i'_1}\sum_{i'_2}\dots\sum_{i'_k}}_{i'_1<i'_2<\ldots<i'_k} \Big( f(i'_1-1,k) + f(i'_2-i'_1-1,k) + \ldots + f(n-i'_k,k) \Big), \end{aligned}$$

where $i'_1 < i'_2 < \ldots < i'_k$ are the pivots in increasing order, $T(n,k) = \overline{a}(k)n + \overline{b}(k)$ is the average value of a "toll function" $\tau(n,k)$ during the first recursive call and $f(i'_1-1,k)$ denotes the average cost for sorting the subarray of $(i'_1-1)$ elements less than $i'_1$ by multipivot Quicksort on $k$ pivots.

Though this looks a complex $k$-index summation, the recursion can be simplified, by noting that the pivots are randomly selected and the sums are equal,

$$\begin{aligned} f(n,k) &= T(n,k) \\ &\quad + \frac{1}{\binom{n}{k}} \underbrace{\sum_{i'_1}\sum_{i'_2}\dots\sum_{i'_k}}_{i'_1<i'_2<\ldots<i'_k} \Big( f(i'_1-1,k) + f(i'_2-i'_1-1,k) + \ldots + f(n-i'_k,k) \Big) \\ &= T(n,k) + \frac{1}{\binom{n}{k}} \sum_{i'_1=1}^{n-k+1} \sum_{i'_2=i'_1+1}^{n-k+2} \cdots \sum_{i'_k=i'_{k-1}+1}^{n} \Big( f(i'_1-1,k) + \ldots + f(n-i'_k,k) \Big) \\ &= T(n,k) + \frac{(k+1)!}{n(n-1)\ldots(n-k+1)} \sum_{i'_1=1}^{n-k+1} \binom{n-i_1}{k-1} f(i'_1-1,k). \end{aligned}$$

Multiplying both sides by $\binom{n}{k}$, the recurrence relation becomes

$$\binom{n}{k}f(n,k) = \binom{n}{k}T(n,k) + (k+1)\sum_{i'_1=1}^{n-k+1}\binom{n-i'_1}{k-1}f(i'_1-1,k).$$

For notational convenience, let $f(n,k) = a_n$ and consider the generating function $h(x) = \sum_{n=0}^{\infty} a_n x^n$;

$$\sum_{n=0}^{\infty}\binom{n}{k}a_n x^n = \sum_{n=0}^{\infty}\binom{n}{k}T(n,k)x^n + (k+1)\sum_{n=0}^{\infty}\left(\sum_{i'_1=1}^{n}\binom{n-i'_1}{k-1}a_{i'_1-1}\right)x^n.$$

The recurrence is transformed to a $k$-th order differential equation

$$\begin{aligned}
\frac{h^{(k)}(x)x^k}{k!} &= \sum_{n=0}^{\infty}\binom{n}{k}T(n,k)x^n + h(x)(k+1)\sum_{n=0}^{\infty}\binom{n-1}{k-1}x^n \\
&= \sum_{n=0}^{\infty}\binom{n}{k}\big(\overline{a}(k)n + \overline{b}(k)\big)x^n + (k+1)h(x)\left(\frac{x}{1-x}\right)^k \\
&= \frac{x^k\big(\overline{a}(k)(x+k) + \overline{b}(k)(1-x)\big)}{(1-x)^{k+2}} + (k+1)h(x)\left(\frac{x}{1-x}\right)^k,
\end{aligned}$$

since it can be easily seen by induction that the $k$-th order derivative of

$$\sum_{n=0}^{\infty}\big(\overline{a}(k)n + \overline{b}(k)\big)x^n = \frac{\overline{a}(k)x + \overline{b}(k)(1-x)}{(1-x)^2}$$

is

$$\frac{k!\big(\overline{a}(k)(x+k) + \overline{b}(k)(1-x)\big)}{(1-x)^{k+2}}.$$

Multiplying by $\left(\dfrac{x}{1-x}\right)^{-k}$, the differential equation is simplified to

$$\frac{h^{(k)}(x)(1-x)^k}{k!} = \frac{\overline{a}(k)(x+k) + \overline{b}(k)(1-x)}{(1-x)^2} + (k+1)h(x).$$

This differential equation is an equidimensional Cauchy–Euler equation, as the one encountered in the previous Chapter. Changing variables $x = 1 - z$, it is $h(x) = g(1-x)$. Applying the differential operator $\Theta$, where $\Theta g(z) = zg'(z)$, the differential equation becomes

$$\big((-1)^k \Theta(\Theta-1)\dots(\Theta-k+1) - (k+1)!\big)g(z) = \frac{k!\big(\overline{a}(k)(1-z+k) + \overline{b}(k)z\big)}{z^2}$$

and the indicial polynomial $\mathcal{P}_k(\Theta)$ is equal to

$$\mathcal{P}_k(\Theta) = (-1)^k \Theta^{\underline{k}} - (k+1)!.$$

Using the notation from [26], $\Theta^{\underline{k}} = \Theta(\Theta-1)\dots(\Theta-k+1)$ with $k \geq 0$, denotes the falling factorial. Again, we need a Lemma regarding the roots of the indicial polynomial.

**Lemma 4.2.1.** *The indicial polynomial $\mathcal{P}_k(\Theta)$ has $k$ simple roots with real parts in the interval $[-2, k+1]$. The real roots are $-2$; $(k+1)$, if $k$ is even and the $2\left\lfloor\frac{k-1}{2}\right\rfloor$ complex roots $\alpha_1, \dots, \alpha_{\left\lfloor\frac{k-1}{2}\right\rfloor}$ with their conjugates $\overline{\alpha}_1, \dots, \overline{\alpha}_{\left\lfloor\frac{k-1}{2}\right\rfloor}$.*

**Proof.** Let $\alpha = x + iy$ be a root of the polynomial. It holds

$$\begin{aligned}
&\alpha(\alpha-1)\ldots(\alpha-k+1) = (-1)^k(k+1)! \\
&\Longrightarrow \alpha(\alpha-1)\ldots(\alpha-k+1) = (-2)(-3)\ldots\big(-(k+1)\big) \\
&\Longrightarrow \frac{\alpha}{-2}\frac{\alpha-1}{-3}\cdots\frac{\alpha-k+1}{-(k+1)} = 1. \qquad (4.5)
\end{aligned}$$

Suppose that $\mathfrak{Re}(\alpha) < -2$. Then

$$\begin{aligned}
\left|\frac{\alpha}{-2}\right| &= \frac{\sqrt{x^2+y^2}}{2} > \frac{\sqrt{(-2)^2+y^2}}{2} \geq 1 \\
\left|\frac{\alpha-1}{-3}\right| &= \frac{\sqrt{(x-1)^2+y^2}}{3} > \frac{\sqrt{(-3)^2+y^2}}{3} \geq 1 \\
&\vdots \\
\left|\frac{\alpha-(k-1)}{-(k+1)}\right| &= \frac{\sqrt{\big(x-(k-1)\big)^2+y^2}}{k+1} > \frac{\sqrt{\big(-(k+1)\big)^2+y^2}}{k+1} \geq 1.
\end{aligned}$$

Considering the moduli in Eq. (4.5), we see that the left-hand side is a product of numbers which are all greater than 1, and so the overall product is greater than 1, but the right-hand side is equal to 1, leading to contradiction. Therefore every root has real part greater than or equal to $-2$. Further, looking over the same argument, we see that the only way we can have the real part being equal to $-2$ is if the imaginary part is equal to zero.

By the Fundamental Theorem of Algebra a polynomial of degree $n$ has $n$ complex roots with multiplicities. Note that $-2$ is always a simple root since,

$$\begin{aligned}\mathcal{P}_k(-2) &= (-1)^k(-2)(-3)\ldots\big(-(k+1)\big) - (k+1)! \\ &= (-1)^{2k}(k+1)! - (k+1)! = 0,\end{aligned}$$

$$\mathcal{P}_k'(-2) = (k+1)!\sum_{j=0}^{k-1}\frac{1}{-2-j} = -(k+1)!(H_{k+1}-1) < 0.$$

Suppose that $\alpha$ is a repeated root. Since $\mathfrak{Re}(\alpha) \geq -2$, we can write $\alpha = (x-2)+iy$ with $x \geq 0$. By the above comments we have that

$$\begin{aligned}\sum_{j=0}^{k-1}\frac{1}{\alpha-j} &= 0 \\ \Longrightarrow \sum_{j=0}^{k-1}\frac{1}{(x-2-j)+iy} &= 0 \\ \Longrightarrow \sum_{j=0}^{k-1}\frac{(x-2-j)-iy}{(x-2-j)^2+y^2} &= 0\end{aligned}$$

In particular, $\mathfrak{Im}(\alpha) = 0$. But that imaginary part is equal to $\sum_{j=0}^{k-1}\frac{y}{(x-2-j)^2+y^2}$ which is clearly only equal to zero if $y = 0$ i.e. the root is real. We will thus have obtained our contradiction if we can show that there are no real roots other than $-2$ and (for $k$ even) $k+1$.

To do this, suppose that we did have a real root $\alpha > -2$. We note first that $\alpha > 0$: because if not, then since $-2$ is also a root, there is a root of $\mathcal{P}'$ between $-2$ and $0$ by Rolle's Theorem. But since $\mathcal{P}_k'(\beta) = 0$ implies that $\sum_{j=0}^{k-1}\frac{1}{\beta-j} = 0$ and when $\beta < 0$ this number is clearly negative. Thus any real root $\alpha$ is positive. It is also $\leq k+1$ as if it were greater than $k+1$ we would have

$\alpha(\alpha-1)\dots(\alpha-k+1) > (k+1)!$. Further, note that $k+1$ is a root if and only if $k$ is even and there cannot be a root in $\alpha \in [k, k+1)$ as the product $\alpha(\alpha-1)\dots(\alpha-k+1)$ would be $< (k+1)!$.

Suppose then that $\alpha \in (j, j+1)$ for some $0 \leq j \leq k-1$. Then the product of the non-negative numbers in the sequence $\alpha, \alpha-1, \dots, \alpha-k+1$ is at most $\alpha(\alpha-1)\dots(\alpha-j) \leq (j+1)\dots 2 \cdot 1 = (j+1)!$. Thus, to get $\alpha$ being a root, we have to have that

$$(-1)^k(\alpha-j-1)\dots(\alpha-k+1) \geq \frac{(k+1)!}{(j+1)!} = (j+2)(j+3)\dots(k+1).$$

However the largest in modulus of $\alpha-j-1, \dots, \alpha-k+1$ is $\alpha-k+1 > j+1-k$ and so their product is less than $(k-1-j)!$. Consequently our inequalities together imply

$$\frac{(k+1)!}{(j+1)!(k-1-j)!} < 1 \Longrightarrow (k+1)\binom{k}{j+1} < 1$$

and this is a contradiction, completing the proof. ∎

The differential equation can be written as

$$\mathcal{S}_{k-1}(\Theta)(\Theta+2)g(z) = \frac{k!\big(\overline{a}(k)(1-z+k)+\overline{b}(k)z\big)}{z^2}.$$

Letting $r_k = -2$ and the remaining $(k-1)$ simple roots be $r_1, r_2, \dots, r_{k-1}$, we have

$$(\Theta-r_1)\dots(\Theta-r_{k-1})(\Theta+2)g(z) = \frac{k!\big(\overline{a}(k)(1-z+k)+\overline{b}(k)z\big)}{z^2}.$$

For the solution of our differential equation, let two functions $g_1(z) + g_2(z) = g(z)$. Then

$$(\Theta - r_1)\ldots(\Theta - r_{k-1})(\Theta + 2)\big(g_1(z) + g_2(z)\big) = \frac{\overline{a}(k)(k+1)!}{z^2} + \frac{(\overline{b}(k) - \overline{a}(k))k!}{z}$$

and by the property of linearity of differential operator

$$\begin{aligned}(\Theta - r_1)\ldots(\Theta - r_{k-1})(\Theta + 2)g_1(z) &= \frac{\overline{a}(k)(k+1)!}{z^2}\\ (\Theta - r_1)\ldots(\Theta - r_{k-1})(\Theta + 2)g_2(z) &= \frac{\big(\overline{b}(k) - \overline{a}(k)\big)k!}{z}.\end{aligned}$$

In the same manner as in the analysis of ‘median of $(2k+1)$’ Quicksort, applying $k$ times the solution, we obtain

$$\begin{aligned}g_1(z) &= \frac{\overline{a}(k)(k+1)!}{(-2-r_1)(-2-r_2)\ldots(-2-r_{k-1})}\frac{\log_e(z)}{z^2} + \sum_{i=1}^{k} c_i z^{r_i}\\ g_2(z) &= \frac{k!}{(-1-r_1)(-1-r_2)\ldots 1}\frac{(\overline{b}(k) - \overline{a}(k))}{z} + \sum_{i=1}^{k} d_i z^{r_i},\end{aligned}$$

where $c_i$ and $d_i$ are constants of integration. In order to evaluate $\mathcal{S}_{k-1}(-2)$, note that

$$\mathcal{S}_{k-1}(-2) = \mathcal{P}'_k(-2),$$

thus

$$\mathcal{S}_{k-1}(-2) = -(k+1)!(H_{k+1} - 1).$$

Moreover,

$$\mathcal{P}_k(-1) = -kk!.$$

Combining both solutions,

$$g(z) = -\frac{\overline{a}(k)}{H_{k+1}-1}\frac{\log_e(z)}{z^2} + \frac{1}{k}\frac{(\overline{a}(k)-\overline{b}(k))}{z} + \sum_{i=1}^{k} s_i z^{r_i}, \tag{4.6}$$

where $s_i = c_i + d_i$. The constants of integration can be found solving the following system of equations

$$g(1) = g'(1) = \ldots = g^{(k-1)}(1) = 0.$$

In terms of series;

$$\begin{aligned} h(x) = \frac{\overline{a}(k)}{H_{k+1}-1}\sum_{n=0}^{\infty}\big((n+1)H_n - n)\big)x^n + \sum_{n=0}^{\infty}\sum_{i=1}^{k} s_i(-1)^n\binom{r_i}{n}x^n \\ + \frac{\overline{a}(k)-\overline{b}(k)}{k}\sum_{n=0}^{\infty}x^n. \end{aligned} \tag{4.7}$$

The third sum of Eq. (4.7) adds to the solution a constant negligible contribution. Also, the root $(k+1)$, when $k$ is even, contributes a constant and the root $r_k = -2$, adds $s_k(n+1)$, with $s_k \in \mathbf{R}$. Extracting the coefficients, the expected cost of multipivot Quicksort is

$$a_n = \frac{\overline{a}(k)}{H_{k+1}-1}\big((n+1)H_n - n\big) + s_k(n+1) + 2\sum_{i=1}^{\lfloor\frac{k-1}{2}\rfloor}(-1)^n\mathfrak{Re}\left(s_i\binom{\alpha_i}{n}\right) + O(1).$$

The asymptotics of the last sum can be found by the well-known Stirling's formula, that states [26]

$$n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n.$$

Expressing the binomial coefficient in terms of $\Gamma$ functions, we have

$$(-1)^n\binom{\alpha_i}{n} = \binom{-\alpha_i+n-1}{n} = \frac{\Gamma(n-\alpha_i)}{n!\Gamma(-\alpha_i)}.$$

The relation 6.1.26 in [1] reads for $x, y \in \mathbf{R}$,

$$|\Gamma(x+iy)| \leq |\Gamma(x)|,$$

thus

$$|\Gamma(n-\alpha_i)| \leq |\Gamma\big(n-\mathfrak{Re}(\alpha_i)\big)|.$$

Using Stirling's formula,

$$\begin{aligned}
\frac{\Gamma(n-\mathfrak{Re}(\alpha_i))}{n!} &\sim \frac{\sqrt{2\pi\Big(n-\big(\mathfrak{Re}(\alpha_i)+1\big)\Big)}\left(\dfrac{n-\big(\mathfrak{Re}(\alpha_i)+1\big)}{e}\right)^{n-\big(\mathfrak{Re}(\alpha_i)+1\big)}}{\sqrt{2\pi n}\left(\dfrac{n}{e}\right)^n} \\
&\sim \frac{\left(\dfrac{n-\big(\mathfrak{Re}(\alpha_i)+1\big)}{e}\right)^{n-\big(\mathfrak{Re}(\alpha_i)+1\big)}}{\left(\dfrac{n}{e}\right)^n} \\
&= \left(\frac{n-\big(\mathfrak{Re}(\alpha_i)+1\big)}{e}\right)^{-\big(\mathfrak{Re}(\alpha_i)+1\big)}\left(1-\frac{\mathfrak{Re}(\alpha_i)+1}{n}\right)^n \\
&\sim Cn^{-(\mathfrak{Re}(\alpha_i)+1)},
\end{aligned}$$

where $C = e^{-(\mathfrak{Re}(\alpha_i)+1)}$ is an unimportant constant.

Therefore the term is bounded by

$$2(-1)^n\mathfrak{Re}\left(s_i\binom{\alpha_i}{n}\right) = O(n^{-(\mathfrak{Re}(\alpha_i)+1)})$$

and asymptotically, the expected cost is

$$\frac{\overline{a}(k)}{H_{k+1}-1} n \log_e(n) + \left( s_k + \frac{\overline{a}(k)}{H_{k+1}-1}(\gamma - 1) \right) n + o(n),$$

since all the other roots have real parts greater than $-2$.

Knowing the coefficients, any mean cost of the generalisation of the algorithm can be directly computed, using this solution, which assumes a simple form. These coefficients are related to the number of pivots used during the partitioning scheme. In [28], the average number of comparisons of the first stage is given by:

$$\overline{a}(s)n + O(1),$$

where $s$ denotes the number of partitions, when $s-1$ pivots are used or equivalently the maximum number of descendants of a node of an $s$–ary tree. The coefficient $\overline{a}(s)n$ is equal to

$$\overline{a}(s) = \lceil \log_2(s) \rceil + \frac{s - 2^{\lceil \log_2(s) \rceil}}{s}.$$

Thus, the average number of comparisons of Quicksort on $k$ pivots is

$$\left( \frac{\lceil \log_2(k+1) \rceil + 1 - \frac{2^{\lceil \log_2(k+1) \rceil}}{k+1}}{H_{k+1}-1} \right) (n+1)H_n + O(n).$$

### 4.2.1 Derivation of integration constants using Vandermonde matrices

The constants of integration can be found using Vandermonde matrices. Differentiating $m$ times Eq. (4.6),

$$\begin{aligned}
g^{(m)}(z) = {} & \frac{\overline{a}(k)}{H_{k+1}-1}\frac{(-1)^{m+1}m!\big((m+1)\log_e(z)-((m+1)H_m-m)\big)}{z^{m+2}} \\
& + (-1)^m m!\frac{\big(\overline{a}(k)-\overline{b}(k)\big)}{kz^{m+1}} + \sum_{i=1}^{k} s_i r_i^{\underline{m}} z^{r_i-m} \\
= {} & \frac{(-1)^m m!}{z^{m+1}}\Bigg(-\frac{\overline{a}(k)\big((m+1)\log_e(z)-((m+1)H_m-m)\big)}{z(H_{k+1}-1)} \\
& + \frac{\big(\overline{a}(k)-\overline{b}(k)\big)}{k}\Bigg) + \sum_{i=1}^{k} s_i r_i^{\underline{m}} z^{r_i-m}.
\end{aligned}$$

The result can be easily proven by induction or by Leibniz's product rule. Using the initial conditions, namely that $g$ and its first $(k-1)$ derivatives are $0$ when evaluated at $z=1$, we obtain

$$\sum_{i=1}^{k} s_i r_i^{\underline{m}} = (-1)^{m+1}m!\left(\frac{\overline{a}(k)\big((m+1)H_m-m)\big)}{H_{k+1}-1} + \frac{\overline{a}(k)-\overline{b}(k)}{k}\right),$$

for $m = 0, 1, \ldots, (k-1)$. In matrix form, the linear system is

$$\begin{bmatrix} 1 & 1 & \dots & 1 \\ r_1 & r_2 & \dots & -2 \\ \vdots & \vdots & \ddots & \vdots \\ r_1^{\underline{k-1}} & r_2^{\underline{k-1}} & \dots & (-2)^{\underline{k-1}} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_k \end{bmatrix} =$$

$$\begin{bmatrix} -\frac{1}{k}\big(\overline{a}(k) - \overline{b}(k)\big) \\ \frac{\overline{a}(k)}{H_{k+1} - 1} + \frac{1}{k}\big(\overline{a}(k) - \overline{b}(k)\big) \\ \vdots \\ (-1)^k (k-1)! \left( \frac{\overline{a}(k)\big(kH_{k-1} - (k-1)\big)}{H_{k+1} - 1} + \frac{\overline{a}(k) - \overline{b}(k)}{k} \right) \end{bmatrix}$$

Here we use the well-known identity $x^n = \sum_{k=0}^{n} {n \brace k} x^{\underline{k}}$ [1], where ${n \brace k}$ are the Stirling numbers of the second kind, to transform the coefficient matrix into a Vandermonde matrix. The determinant of this Vandermonde matrix is equal to

$$\prod_{1 \le i < j \le n} (r_j - r_i) \neq 0,$$

as the roots are all simple. Considering the expected number of passes of multipivot Quicksort, it holds that $a(k) = 0$ and $b(k) = 1$, for $k = 1, 2, \dots$ . The system is,

$$\begin{bmatrix} 1 & 1 & \dots & 1 \\ r_1 & r_2 & \dots & -2 \\ \vdots & \vdots & \ddots & \vdots \\ r_1^{\underline{k-1}} & r_2^{\underline{k-1}} & \dots & (-2)^{\underline{k-1}} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_k \end{bmatrix} = \begin{bmatrix} \frac{1}{k} \\ -\frac{1}{k} \\ \vdots \\ (-1)^{k-1} \frac{(k-1)!}{k} \end{bmatrix}$$

Turning the coefficient matrix into a Vandermonde one and using the identity $\sum_{j=0}^{n}(-1)^j j!\left\{ {n \atop j} \right\} = (-1)^n$, (see subsection 24.1.4 in [1]), we obtain

$$\begin{bmatrix} 1 & 1 & \dots & 1 \\ r_1 & r_2 & \dots & -2 \\ \vdots & \vdots & \ddots & \vdots \\ r_1^{k-1} & r_2^{k-1} & \dots & (-2)^{k-1} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_k \end{bmatrix} = \begin{bmatrix} \dfrac{1}{k} \\ -\dfrac{1}{k} \\ \vdots \\ (-1)^{k-1}\dfrac{1}{k} \end{bmatrix}$$

In [31] and [73] the inverse of Vandermonde matrix is given in terms of product of an upper and lower triangular matrices. Letting $\mathbf{A}^{-1}$ denote the inverse, it is equal to

$$\mathbf{A}^{-1} = \begin{bmatrix} 1 & \frac{1}{r_1-r_2} & \frac{1}{(r_1-r_2)(r_1-r_3)} & \dots \\ 0 & \frac{1}{r_2-r_1} & \frac{1}{(r_2-r_1)(r_2-r_3)} & \dots \\ 0 & 0 & \frac{1}{(r_3-r_1)(r_3-r_2)} & \dots \\ 0 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \dots \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & \dots \\ -r_1 & 1 & 0 & \dots \\ r_1r_2 & -(r_1+r_2) & 1 & \dots \\ -r_1r_2r_3 & r_1r_2+r_1r_3+r_2r_3 & -(r_1+r_2+r_3) & \dots \\ \vdots & \vdots & \vdots & \dots \end{bmatrix}$$

It is clear that the lower triangular matrix, post–multiplied by the vector $\left(1/k, -1/k, \dots (-1)^{k-1}/k\right)^{\mathbf{T}}$, will give us

$$\left(1/k, -(r_1+1)/k, (r_1+1)(r_2+1)/k, -\prod_{i=1}^{3}(r_i+1)/k, \dots, (-1)^{k-1}\prod_{i=1}^{k-1}(r_i+1)/k\right)^{\mathbf{T}}.$$

Thus, the solution is

$$s_i = (-1)^{k-1} \frac{\prod\limits_{\substack{j \neq i \\ 1 \leq j \leq k}} (r_j + 1)}{k \prod\limits_{\substack{j \neq i \\ 1 \leq j \leq k}} (r_i - r_j)}$$

and the expected number of partitioning stages of multipivot Quicksort on $k$ pivots is

$$(-1)^{k-1} \frac{\prod\limits_{j=1}^{k-1} (r_j + 1)}{k \prod\limits_{j=1}^{k-1} (-2 - r_j)} (n+1) + o(n).$$

Note that

$$\frac{\prod\limits_{j=1}^{k-1} (r_j + 1)}{\prod\limits_{j=1}^{k-1} (-2 - r_j)} = (-1)^{k-1} \frac{kk!}{(k+1)!(H_{k+1} - 1)},$$

therefore the mean number of partitioning stages is

$$\frac{n+1}{(k+1)(H_{k+1} - 1)} + o(n).$$

We remark that a generalised version of this result can be found in [28]. In [35], it was shown that the constants of integration can be computed for arbitrary values of the coefficients $a(k)$ and $b(k)$ by the same method, as in the derivation of the integration constants in the simple case of $a(k) = 0$ and $b(k) = 1$.

At the end of this section, it should be noted that the worst-case probability is not eliminated, but is less likely to occur. In an unfortunate situation, where the $k$ smallest or greatest keys are selected as pivots, partitioning will yield trivial subarrays and one containing the remaining elements. If the chosen pivots happen to be close to the quantiles of the array, this yields an optimal partitioning of the array. In the next section, we examine ways of a more efficient selection of pivots.

## 4.3 Multipivot–median partitioning

The preceding analysis of multipivot Quicksort, where $k$ pivots are uniformly selected at random has showed that the worst-case scenario is less likely from the standard 'one–pivot' model. Is any other way, where we can reduce further the probability of such scenario? We have seen that choosing the median from a random sample of the array to be sorted, yields savings to the running time of the algorithm. Since we have examined the analysis of multiple pivots, then we can select these pivots as the quantiles of a bigger random sample.

Thus, one can randomly choose a larger sample of $k(t + 1) - 1$ keys, find the $(t + 1)$-st, $2(t + 1)$-th, $\ldots$, $(k - 1)(t + 1)$-th smallest keys and use these $(k - 1)$ statistics as pivots. Note that for $k = 2$, this variant contains the median of $2t + 1$ Quicksort as a special case and for $t = 0$, we have the multipivot algorithm, whose mathematical analysis was presented in the previous section. This 'generalised Quicksort' was introduced by Hennequin [28]. Let $T(n_{\{k,t\}})$ be the average of a "toll function" during the first pass and $f(n_{\{k,t\}})$ the total expected cost of this variant, when applied to an array of $n$ keys. The following recurrence (which is not presented so simply in Hennequin) holds:

$$f(n_{\{k,t\}}) = T(n_{\{k,t\}}) + \frac{1}{\binom{n}{k(t+1)-1}}$$
$$\times \underbrace{\sum_{i_1}\sum_{i_2}\cdots\sum_{i_{k-1}}}_{i_1<i_2<\ldots<i_{k-1}}\Bigg(\binom{i_1-1}{t}\binom{i_2-i_1-1}{t}\cdots\binom{i_{k-1}-i_{k-2}-1}{t}\binom{n-i_{k-1}}{t}$$
$$\cdot \big(f((i_1-1)_{\{k,t\}}) + f((i_2-i_1-1)_{\{k,t\}}) + \ldots + f((i_{k-1}-i_{k-2}-1)_{\{k,t\}})$$
$$+ f((n-i_{k-1})_{\{k,t\}})\big)\Bigg),$$

since the pivots $i_1, \ldots, i_{k-1}$ are selected to be the $(t+1)$-st, …, $(k-1)(t+1)$-th smallest keys of the sample and each of the $k$ resulting subarrays $(i_1-1), (i_2-i_1-1), \ldots, (n-i_{k-1})$ contain $t$ elements of the sample.

As before, the general recurrence of average cost is translated to a differential equation with indicial polynomial [28],

$$\mathcal{P}_{k(t+1)-1}(\Theta) = (-1)^{k(t+1)-1}\binom{\Theta}{k(t+1)-1} - k(-1)^t\binom{\Theta}{t}.$$

A Lemma follows concerning the whereabouts of the roots of this polynomial:

**Lemma 4.3.1.** *The indicial polynomial $\mathcal{P}_{k(t+1)-1}(\Theta)$ has $k(t+1)-1$ simple roots, with real parts greater than or equal to $-2$. The real roots are the integers $0, 1, \ldots, (t-1)$, $-2$; $k(t+1)+t$, when $t$ is odd and $k$ is even or when $t$ is even and $k$ is odd and the $2\left\lfloor\frac{(k-1)t+k-2}{2}\right\rfloor$ complex roots $\lambda_1, \ldots, \lambda_{\lfloor\frac{(k-1)t+k-2}{2}\rfloor}$ with their conjugates $\overline{\lambda}_1, \ldots, \overline{\lambda}_{\lfloor\frac{(k-1)t+k-2}{2}\rfloor}$.*

**Proof.** It can be easily deduced that the integers $0, 1, \ldots, (t-1)$ and $-2$ are roots of the polynomial. Now, when $t$ is odd and $k$ is even, then $k(t+1)-1$ is odd and so a root $\alpha$ of the polynomial will satisfy

$$\binom{\alpha}{k(t+1)-1} = k\binom{\alpha}{t} \tag{4.8}$$

and by simple manipulations we can now verify that $k(t+1)+t$ is also a root. Similarly, if $t$ is even and $k$ is odd, we have that $k(t+1)-1$ is even and Eq. (4.8) is valid, again making $k(t+1)+t$ a root.

It will be proved by contradiction that all roots have real parts greater than or equal to $-2$. Note that the argument is similar with the proofs of Lemmas 3.1.1 and 4.2.1. For any root $r = x + iy$, with $x, y \in \mathbf{R}$ holds

$$(-1)^{k(t+1)-1}\binom{r}{k(t+1)-1} = k(-1)^t\binom{r}{t}. \tag{4.9}$$

For $r \neq 0, 1, \ldots, (t-1)$, Eq. (4.9) can be written as

$$(-1)^{k(t+1)-1}\frac{(r-t)(r-t-1)\ldots\big(r-k(t+1)+2\big)}{(t+1)\ldots\big(k(t+1)-1\big)} = (-1)^t k. \tag{4.10}$$

Assume that $\mathfrak{Re}(r) < -2$, then

$$\begin{aligned}
\left|\frac{r-t}{t+1}\right| &= \frac{\sqrt{(x-t)^2+y^2}}{t+1} > \frac{\sqrt{\big(-(t+2)\big)^2+y^2}}{t+1} \geq \frac{t+2}{t+1} \\
\left|\frac{r-t-1}{t+2}\right| &= \frac{\sqrt{(x-t-1)^2+y^2}}{t+2} > \frac{\sqrt{\big(-(t+3)\big)^2+y^2}}{t+2} \geq \frac{t+3}{t+2} \\
&\vdots \\
\left|\frac{r-\big(k(t+1)-2\big)}{k(t+1)-1}\right| &= \frac{\sqrt{\big(x-k(t+1)+2\big)^2+y^2}}{k(t+1)-1} > \frac{\sqrt{\big(-k(t+1)\big)^2+y^2}}{k(t+1)-1} \\
&\geq \frac{k(t+1)}{k(t+1)-1}.
\end{aligned}$$

Considering the product of moduli, we see that the left-hand side of Eq. (4.10) is greater than or equal to (in modulus) the telescoping product $k(t+1)/(t+1) = k$, which gives a contradiction. Further, this argument shows that $-2$ is the unique root with the least real part.

We now show the roots are simple. Assuming, for a contradiction, that $r$ is a repeated root, we get:

$$(-1)^{k(t+1)-1}\binom{r}{k(t+1)-1}\sum_{j=0}^{k(t+1)-2}\frac{1}{r-j} = k(-1)^t\binom{r}{t}\sum_{j=0}^{t-1}\frac{1}{r-j}. \tag{4.11}$$

Eq. (4.9) and (4.11) imply that

$$\sum_{j=0}^{k(t+1)-2}\frac{1}{r-j} = \sum_{j=0}^{t-1}\frac{1}{r-j}$$

or

$$\sum_{j=t}^{k(t+1)-2}\frac{1}{r-j} = 0. \tag{4.12}$$

From Eq. (4.12), we deduce that $\mathfrak{Im}(r) = 0$ and $r \in (t, t+1) \cup (t+1, t+2) \cup \ldots \cup \big(k(t+1) - 3, k(t+1) - 2\big)$. However, the modulus in the left-hand side of Eq. (4.9) is smaller than $1$, while the right-hand side is greater than $k$, proving that all roots are simple. ∎

By the Lemma, the polynomial can be written in terms of simple factors and the differential equation can be solved using the same way, as in other variants of Quicksort, previously analysed. The average cost of 'generalised Quicksort' is

$$\frac{\overline{a}(k,t)}{H_{k(t+1)} - H_{t+1}} \big((n+1)H_n - n\big) + O(n),$$

when the "toll function" is linear and its average is $\overline{a}(k,t)n + O(1)$.

Our analyses of the average cost of the algorithm and its variants have showed that any Quicksort needs on average $Cn\log_e(n) + O(n)$ key comparisons for the complete sorting of a file consisting of $n$ distinct keys. The constant $C$ can be made very close to the information–theoretic bound, as we saw. In many sorting applications the 'median of 3' is being used, with savings on the average time and little overhead for the computation of median. For large arrays, one can use the 'remedian of $3^2$' Quicksort.

# Chapter 5

# Partial order of keys

## 5.1 Introduction

Here, we investigate the analysis of Quicksort under the assumption of prior information of the order of keys. Specifically, we assume that there is a partial order on the keys. The rough idea is to see how much having partial information compatible with the true order allows us to speed up the process of finding the true order.

Let us illustrate the idea first with a simple example. Suppose that there are $d$ levels with $k$ keys at each level, so that $n = kd$. Anything in a higher level is known to be above everything in a lower level. Computing the ratio of the expected complexities, we have

$$\frac{\mathbb{E}(C_n)}{\mathbb{E}(C_n^*)} = \frac{d\big(2(k+1)H_k - 4k\big)}{2(n+1)H_n - 4n},$$

where $C_n^*$ and $C_n$ denote the number of comparisons of Quicksort with uniform pivot selection and in case of partial order, respectively. We consider the following cases:

1. When d is fixed number, then as $n$ tends to infinity,

$$\lim_{n\to\infty} \frac{n}{d} = \infty.$$

Thus,

$$\lim_{n\to\infty} \frac{\mathbb{E}(C_n)}{\mathbb{E}(C_n^*)} \sim \lim_{n\to\infty} \frac{d\big(2k\log_e(k)\big)}{2n\log_e(n)} = \lim_{n\to\infty} \frac{\log_e(k)}{\log_e(n)} = \lim_{n\to\infty} \frac{\log_e(n/d)}{\log_e(n)} = 1.$$

2. $d = k = \sqrt{n}$.

$$\lim_{n\to\infty} \frac{\mathbb{E}(C_n)}{\mathbb{E}(C_n^*)} = \lim_{n\to\infty} \frac{\sqrt{n}\big(2(\sqrt{n}+1)H_{\sqrt{n}} - 4\sqrt{n}\big)}{2(n+1)H_n - 4n} \sim \lim_{n\to\infty} \frac{2n\log_e(\sqrt{n})}{2n\log_e(n)} = \frac{1}{2}.$$

We see that Quicksort is on average twice as fast, when sorting a partially ordered array.

3. $k = \frac{1}{c}$ where $c$ is a constant. Then,

$$\lim_{n\to\infty} \frac{\mathbb{E}(C_n)}{\mathbb{E}(C_n^*)} = \frac{cn\big(2(\frac{1}{c}+1)H_{1/c} - \frac{4}{c}\big)}{2(n+1)H_n - 4n} = \frac{n\big((2+2c)H_{1/c} - 4\big)}{2n\log_e(n)} = \frac{c'}{\log_e(n)},$$

where $c' = \dfrac{(2+2c)H_{1/c} - 4}{2}$.

We should think a little about variability too. It is unsurprising that having the additional information about levels reduces variability of the number of comparisons, let us get a preliminary result. If we have the level structure, then $\operatorname{Var}(C_n^*)$ is the sum of the variances of sorting each of the $d$ independent levels. Each of these variances, since there are $k$ keys in each level, is just $\operatorname{Var}(C_k)$. For simplicity, we assume $k \to \infty$ as $n \to \infty$ and do asymptotics. We then have (for

$m$ either $n$ or $k$)

$$\mathrm{Var}(C_m) \sim \left(7 - \frac{2\pi^2}{3}\right) \cdot m^2$$

and thus we get

$$\frac{\mathrm{Var}(C_n)}{\mathrm{Var}(C_n^*)} \simeq \frac{(7 - 2\pi^2/3)n^2}{(7 - 2\pi^2/3)k^2 d} = d$$

so the variance of the version with the presorting is reduced by a factor of about $d$. These suggest there is interest in studying this situation, we now do so in more detail.

## 5.2 Partially ordered sets

An approach of having additional information is the partial order of the keys. We shall employ this assumption along the following lines. First, we present a definition [56].

**Definition 5.2.1.** *Let a finite set $P$ equipped with a binary relation '$\leq$' which has the following properties. (Here, $x$, $y$ and $z$ are elements of $P$).*
*(i) $x \leq x$, $\forall x \in P$. (That is, $\leq$ is reflexive)*
*(ii) If $x \leq y$ and $y \leq x$, then $x = y$. ($\leq$ is antisymmetric)*
*(iii) If $x \leq y$ and $y \leq z$, then $x \leq z$. ($\leq$ is transitive)*
*Then the pair (P, $\leq$) is called Partially Ordered Set.*

Henceforth, in this thesis we abbreviate 'partially ordered set' to 'poset'. We also present two key definitions [56].

**Definition 5.2.2.** *Let $(P, \leq)$ be a poset. We say that two elements $x$ and $y$ of this poset are comparable if $x \leq y$ or $y \leq x$. Otherwise they are incomparable.*

**Definition 5.2.3.** *Let $(P, \leq)$ be a poset.*

*(i) A minimal element of $(P, \leq)$, is an element with the property that no other element is smaller than it. A maximal element of $(P, \leq)$, is an element with the property that no other element is greater than it.*

*(ii) A chain in $P$ is a set $T$ of elements, where every pair of elements of $T$ are comparable. The number of elements of $P$ in the longest chain in $P$ is called the height of $P$ and denoted by $h(P)$.*

*(iii) An antichain in $P$ is a set $U$ of elements, no two of which are comparable. The number of elements of $P$ in the order of the largest antichain is called the width of $P$, and is denoted by $w(P)$.*

*(iv) A total order in $P$ is a partial order where every pair of elements are comparable.*

For example, the set of subsets of $X = \{1, 2\}$ has an antichain of order $2$, namely $\{1\}$ and $\{2\}$. A chain of length 3 in it, is $\emptyset \leq \{1\} \leq \{1, 2\}$. This partial order is not a total order as $\{1\}$ and $\{2\}$ are not comparable. Usually, if we have a partial order on a set, there will be several ways of extending it to a total order on that set. Often, we will use $\prec$ rather than $<$ to denote the partial order. Further, we present the following definition, that we will come across later.

**Definition 5.2.4.** *Let a poset (P, $\leq$). Its comparability graph $G(P)$ is the graph with the poset's vertex set, such that the elements are adjacent if and only if they are comparable in (P, $\leq$). Its incomparability graph $G(\tilde{P})$ is the graph, such that the elements are adjacent if and only if they are incomparable in (P, $\leq$).*

Another example of a partial order which usually is not a total order is the collection of subsets of a fixed set $X$, with the partial order $\leq$ being inclusion, normally denoted as $\subseteq$. It is easy to check that, for any $A \subseteq X$, we have that $A \leq A$ since any set is a subset of itself: if $A \leq B \leq A$ then we indeed have that $A = B$, giving asymmetry: and finally, if $A \subseteq B \subseteq C$ then of course $A \subseteq C$ and so $\leq$ will be transitive. This is not a total order if $X$ has order at least 2, as $\{x_1\} \subseteq X$ and $\{x_2\}$ are not comparable for $x_1 \neq x_2$ members of $X$. However, when we have a partial order on a set $P$ there will be at least one total order on $P$ extending it, and in fact usually there will be several such:

**Definition 5.2.5.** *A linear extension of a partial order $(P, \prec)$ is a total order $<$ on the set $P$ such that whenever $x \prec y$ in the partial order, then we have $x < y$ in the total order too. The number of linear extensions of a poset $P$ is denoted by $e(P)$.*

In other words, a linear extension of a partial order is a total order on the same set which is compatible with the partial order. This is of course of great relevance to us, as the situation we are in is that we are given partial information on the true order of the set of elements and want to know how many more pairwise comparisons we have to do to work out the true order on it: that is, we are trying to identify which of the numerous linear extensions of the partial order is the true order on it, with as few comparisons as possible.

The number of linear extensions of a poset can vary substantially according to the structure of the poset. For example, trivially, if the partial order happens already to be a total order there is only one extension, namely itself. Equally trivially, if the partial order contains no comparisons – i.e. it provides no information whatsoever – then all $n!$ possible orderings of the $n$ elements of $P$ are linear extensions.

Here is a generic lower bound on the number of pairwise comparisons we need to make in order to find the true order of our data, given a partial ordering $P$ of it.

**Theorem 5.2.6.** *Given a partial order $(P, \prec)$ which is partial information about the true total order on the underlying set $P$, it takes at least $\lceil \log_2\bigl(e(P)\bigr)\rceil$ pairwise comparisons to find the total order.*

**Proof.** Recall that linear extension or total order of a poset $(P, \prec)$ is a total order compatible with the partial one. All elements are comparable, forming a unique chain. Now let $x$, $y$ be incomparable, distinct members of poset. In $A$ total orders, we will have that $x < y$ and in $B$ linear extensions, $x > y$. It holds that $A + B = e(P)$ so,

$$\max(A, B) \geq \frac{e(P)}{2}.$$

Thus, after making one comparison, there are at least $\frac{e(P)}{2}$ candidates. Similarly, of those number of linear extensions, choosing again two elements, we have to consider at least $\frac{e(P)}{4}$ total orders. Thus, after $r$ comparisons, we will examine at least $\frac{e(P)}{2^r}$ linear extensions. We want to identify the unique total order among the number $e(P)$ of all possible linear extensions. Therefore, this fraction becomes equal to unity when $r = \lceil \log_2\bigl(e(P)\bigr)\rceil$ comparisons. ∎

**Remark 5.2.7.** *The quantity* $\log_2\bigl(e(P)\bigr)$ *is the information–theoretic lower bound, that we first came across in section* 3.1.

Kislitsyn [41] and independently Fredman [25], showed that often this lower bound is close to the truth.

**Theorem 5.2.8** (Fredman [25], Kislitsyn [41])**.** *Sorting an array of $n$ keys, which obeying a partial order $P$, can be achieved in worst case by $\log_2\big(e(P)\big)+2n$ comparisons.*

**Corollary 5.2.9.** *If $P$ is a poset for which $\log_2\!\big(e(P)\big)$ grows faster than $n$, we have that the number of comparisons of finding the true total order in worst case, is $\log_2\!\big(e(P)\big)\big(1+o(1)\big)$.*

**Proof.** We have that this quantity is bounded below by $\log_2\!\big(e(P)\big)$ and above by $\log_2\!\big(e(P)\big)+2n$ which is $\log_2\!\big(e(P)\big)\big(1+o(1)\big)$ by the assumption in the statement of the Corollary. ∎

Kahn and Kim [40] gave an algorithm for actually doing the finding of the true total order, which uses at most $54.45\log_2\!\big(e(P)\big)$ comparisons. This has recently been reproved by Cardinal *et al.* [13] whose proof manages to avoid Kahn and Kim's use of the ellipsoid method, a technique which though it is in theory polynomial-time, is difficult to do in practice. What all this makes clear is that, in considering how much information we can deduce from a random partial order, we will need to know about the logarithm of the number of linear extensions the partial order typically has.

Often a useful notion in studying posets is the theory of levels, [10], [42].

**Definition 5.2.10.** *If $(P,\leq)$ is a poset, we define*

$$L_1=\{x\in P:\nexists y\in P,\ y\leq x\ \wedge\ y\neq x\}$$

*the set of minimal elements of our poset to be the first level of our poset. The next level is the set of minimal elements in $P\setminus L_1$: each of these will have (at least one)* *element of $L_1$ below it. We then continue by induction, defining $L_i$ to be the level of minimal elements of $P \setminus \left(\cup_{j=1}^{i-1} L_j\right)$.*

Note that every level of a poset is an antichain: for two minimal elements in a poset cannot be comparable with each other. Moreover, every time you go up in a chain, you go up to a higher level. Thus the height of the poset will be the number of levels.

**Definition 5.2.11.** *The linear sum of two posets $(P_1, \prec_1)$ and $(P_2, \prec_2)$ is a poset with vertex set the disjoint union of $P_1$ and $P_2$ and with $x \prec y$ if and only if:*
*(i) if $x$ and $y$ are in $P_1$ and $x \prec_1 y$:*
*(ii) if $x$ and $y$ are in $P_2$ and $x \prec_2 y$:*
*(iii) if $x \in P_1$ and $y \in P_2$, then automatically we have $x \prec y$.*

A useful definition for us will be the following [56].

**Definition 5.2.12.**
*(i) Suppose $P$ is a partially ordered set, and $L$ is a particular total order on the same set which agrees with $P$ on every pair of elements which are comparable in $P$. (In other words, $L$ is one of the linear extensions of our partial order). Then a setup is a pair of elements $(x, y)$ which are incomparable in $P$ but are consecutive in $L$.*
*(ii) The number of setups $S(P, L)$ which must be made comparable to obtain the linear extension $L$ is denoted $s(P, L)$.*
*(iii) The setup number of $P$ is the minimum, over all the linear extensions $L$ of $P$, of $s(P, L)$ and it is denoted by $s(P)$.*

So the point is that, if information is given in the partial order $P$ and using pairwise comparisons to obtain the rest of the order, $s(P)$ is a lower bound on the number of comparisons which have to be done to find the true order.

The setup number is also known as the jump number. The following Lemma is used for the derivation of a simple lower bound on the setup number of a poset.

**Lemma 5.2.13** (Dilworth [17])**.** *The minimum number of chains into which a poset $P$ can be partitioned is the width $w(P)$.*

**Proof.** This is standard and there are several proofs, we refer to [17]. ∎

**Theorem 5.2.14.** *For any poset $P$, $s(P) \geq w(P) - 1$.*

**Proof.** The best case is to partition the poset into $w(P)$ chains $C_1, C_2, \ldots, C_{w(P)}$ and hope that there is some ordering of these chains (which without loss of generality is the order given) such that every element in $C_i$ is less than the minimum element of $C_{i+1}$ for each $1 \leq i \leq w - 1$. Because if this happens, then the total order is just the direct sum of the $C_i$ and we only have to do $w(P) - 1$ comparisons of the maximum element of $C_i$ with the minimum element of $C_{i+1}$. ∎

What we will do for the next while is consider various ways in which we could have a partial order given to us before we start using Quicksort to determine the complete order. So we are imagining that a previous researcher had carried out some of the comparisons and we want to know how many more comparisons we have to carry out to determine the total order. We will consider cases where the poset is randomly generated.

## 5.3 Uniform random partial orders

**Definition 5.3.1.** *A uniform random partial order is a partial order selected uniformly at random from all the partial orders on* $S = \{1, 2, \ldots, n\}$.

This means that all partial orders on $S$ are equally likely to be chosen. The basic structural result on such posets is the following, rather surprising, one.

**Theorem 5.3.2** (Kleitman and Rothschild [42]. Alternative proof by Brightwell, Prömel and Steger [11])**.** *Suppose that* $\leq$ *is a uniform random partial order on* $\{1, 2, \ldots, n\}$*. Then,* **whp.** *(henceforth, '***whp.***' stands for 'with high probability', which denotes the fact that as* $n \to \infty$*, the probability of an event approaches* $1$*) there are three levels: the bottom level has approximately* $n/4$ *elements in it, the middle layer approximately* $n/2$ *elements and the top layer about* $n/4$ *elements in it.*

**Remark 5.3.3.** *This feature of a uniform random partial order – that it has height only* $3$ *– is surprising to most mathematicians when they hear it, and perhaps suggests that "in nature" posets do not occur uniformly at random – some posets are favoured over others.*

Here is the key information on the number of linear extensions.

**Theorem 5.3.4** (Brightwell [10])**.** *Given any function* $\omega(n)$ *tending to infinity with* $n$ *(think of it as doing so extremely slowly) the number of linear extensions of a random partial order chosen uniformly at random is* **whp.***, between*

$$\frac{(n/2)!\big((n/4)!\big)^2}{\omega(n)} \text{ and } (n/2)!\big((n/4)!\big)^2\omega(n).$$

For the proof of this Theorem, we refer to Brightwell's survey [10].

**Corollary 5.3.5.** *When $P$ is selected uniformly at random, we have* **whp.**

$$\begin{aligned}\frac{1}{\log_e(2)}\big(n\log_e(n) - O(n)\big) \leq \log_2\big(e(P)\big)& \\ &\leq \frac{1}{\log_e(2)}\big(n\log_e(n) + O(n)\big).\end{aligned}$$

In other words, in this situation, the entropy lower bound on the number of comparisons required is essentially the right answer.

**Proof.** We have by Theorem 5.3.4, taking $\omega(n)$ to go to infinity very slowly, in particular more slowly than $\log_e(n)$, that

$$\begin{aligned}&\log_2\left(\frac{(n/2)!\big((n/4)!\big)^2}{\omega(n)}\right) \leq \log_2\big(e(P)\big) \leq \log_2\Big((n/2)!\big((n/4)!\big)^2\omega(n)\Big)\\ \Longrightarrow\ &\frac{1}{\log_e(2)}\log_e\left(\frac{(n/2)!\big((n/4)!\big)^2}{\omega(n)}\right) \leq \log_2\big(e(P)\big) \leq \frac{1}{\log_e(2)}\log_e\Big((n/2)!\big((n/4)!\big)^2\omega(n)\Big).\\ \Longrightarrow\ &\frac{1}{\log_e(2)}\big((n/2)\log_e(n/2) + 2(n/4)\log_e(n/4) - (n/2) - 2(n/4) + O(\log_e(n)\big).\\ &\leq \log_2\big(e(P)\big)\\ &\leq \frac{1}{\log_e(2)}\big((n/2)\log_e(n/2) + 2(n/4)\log_e(n/4) - (n/2) - 2(n/4) + O(\log_e(n)\big).\\ \Longrightarrow\ &\frac{1}{\log_e(2)}\big(n\log_e(n) - O(n)\big) \leq \log_2\big(e(P)\big) \leq \frac{1}{\log_e(2)}\big(n\log_e(n) + O(n)\big),\end{aligned}$$

as required. ∎

Using this Corollary, a key result follows regarding algorithm's time complexity:

**Corollary 5.3.6.** *Given a uniform partial order on a set of $n$ keys, the time taken to sort them by pairwise comparisons is approximately*

$$\frac{1}{2\log_e(2)} \approx 0.72135$$

*times the number of comparisons required by Quicksort to sort them without the partial information.*

**Proof.** The expected number of comparisons required by Quicksort for the sorting of $n$ keys is $2n\log_e(n)\big(1+o(1)\big)$, and since the variance of this is asymptotically $(7-2\pi^2/3)n^2\big(1+o(1)\big)$, we have by Chebyshev's inequality, letting $C_n$ be the number of comparisons

$$\begin{aligned} &\mathbb{P}\big(\big|(C_n-\mathbb{E}(C_n))\big| > n\log_e\big(\log_e(n)\big)\big) \leq \frac{(7-2\pi^2/3)n^2\big(1+o(1)\big)}{n^2(\log_e(\log_e(n)))^2} \\ \Longrightarrow\; &\mathbb{P}\big(\big|C_n - 2n\log_e(n)\big(1+o(1)\big)\big| > n\log_e\big(\log_e(n)\big)\big) \to 0, \text{ as } n\to\infty. \end{aligned}$$

Thus the probability of the complementary event, namely that $C_n$ is within $n\log_e\log_e(n)$ of its mean will tend to $1$. Therefore, **whp.** the number of comparisons is in

$$\Big(2n\log_e(n)\big(1+o(1)\big) - n\log_e\big(\log_e(n)\big), 2n\log_e(n)\big(1+o(1)\big) + n\log_e\big(\log_e(n)\big)\Big).$$

Thus, **whp.** it takes about

$$2n\log_e(n)\big(1+o(1)\big) \text{ comparisons.}$$

On the other hand, we have just seen that with a uniform partial order of keys, it takes about

$$\frac{1}{\log_e(2)} \cdot n \log_e(n) \text{ comparisons.}$$

This number is indeed $1/\big(2\log_e(2)\big)$ times the number Quicksort needs and the numerical value of this fraction is as stated. ∎

It is of interest to compare this with the naive lower bound on setup number, which performs rather poorly here.

**Corollary 5.3.7.** *The setup number of a uniform random poset is at least*

$$\frac{n}{2} + O(1).$$

**Proof.** This is immediate from the lower bound $s(P) \geq w(P) - 1$ and the fact that, by the Theorem of Kleitman and Rothschild [42], we clearly have that $w(P)$ is greater than $n/2 + O(1)$. Thus in this case the simplest setup number lower bound is not a very good one, as we have seen that the true answer is $O\big(n\log_2(n)\big)$ in this model. ∎

This in turn implies that if we were to use Quicksort, even in the optimal cases, we would have to compare $n/2 + O(1)$ pairs of keys. The expected time to do this would be asymptotically $2(n/2)\log_e(n/2)$.

## 5.4 Random bipartite orders

In this section, we inspect the number of linear extensions for bipartite orders. Let present a definition [10].

**Definition 5.4.1.** *Let $X$ and $Y$ be two disjoint sets, each one having cardinality equal to $n$. A random bipartite order $A_p(X, Y)$ is the poset $X \cup Y$ with both $X$ and $Y$ antichains, and for each pair $(x, y) \in X \times Y$ there is a relation $x \prec y$ with probability $p$ and no relation (i.e. they are incomparable) with probability $1 - p$, independently of all other pairs.*

We now think about the number of linear extensions. A linear extension of such an order will have to, amongst other things, put the set $X$ in order – there are $n!$ ways to do this – and there are similarly $n!$ ways to order $Y$. However there will also be some choices to make elsewhere, because while we have some relations $x < y$ for $(x, y) \in X \times Y$ in the partial order, we will also have some incomparable pairs. More precisely, the probability that a total order on $X$, a total order on $Y$ and a decision rule $\alpha$ for each pair $(x, y) \in X \times Y$ on whether $x \prec y$ or $y \prec x$, is a total order compatible with the partial order – i.e. a linear extension of the partial order – is $(1 - p)^{\ell(\alpha)}$, where $\ell(\alpha)$ is the number of reversals in $\alpha$, that is the number of pairs $(x, y) \in X \times Y$ such that $x > y$ in the total order.

Thus the expected number of linear extensions is $(n!)^2 \sum_{\alpha} (1 - p)^{\ell(\alpha)}$. The function multiplying $(n!)^2$ here is discussed at length in [10]: it is defined

$$\eta(p) := \prod_{i=1}^{\infty} \big(1 - (1 - p)^i\big).$$

Though some more work needs to be done to check this, it turns out that **whp.** the number of linear extensions is close to this mean value. We quote the result from Brightwell [10].

**Theorem 5.4.2** (Brightwell [10]). *The number of linear extensions $e(P)$ of a random bipartite partial order $A_p(X,Y)$ with $|X| = |Y| = n$ and probability $p$ such that $\lim_{n\to\infty} \frac{p(n)\cdot n^{1/7}}{\left(\log_2(n)\right)^{4/7}} = \infty$, satisfies* **whp.***,*

$$e(P) = (n!)^2 \cdot \frac{1}{\eta(p)} \cdot \big(1 + o(1)\big).$$

In particular this applies when $0 < p < 1$ is a constant. More precisely, we have that **whp.**

$$(n!)^2 \cdot \eta(p)^{-1} \cdot \left(1 - \frac{c\log_2^3(n)}{n}\right) \leq e(P) \leq (n!)^2 \cdot \eta(p)^{-1} \cdot \left(1 + \frac{c\log_2^3(n)}{n}\right).$$

Thus, for $p$ constant, letting $C = \log_2\big(\eta(p)\big)$ and noting that both logarithms approach $1$ as $n \to \infty$, we have that **whp.**

$$2\log_2(n!) - C + o(1) \leq \log_2\big(e(P)\big) \leq 2\log_2(n!) - C + o(1).$$

(Of course the two $o(1)$ terms are different). Thus the order of magnitude of $\log_2\big(e(P)\big)$ is **whp.**

$$2\log_2(n!) = \frac{2\cdot\log_e(n!)}{\log_e(2)} = \frac{2n\log_e(n)(1+o(1))}{\log_e(2)}.$$

We need to be careful about comparing this example with Quicksort: we must remember that the total number of keys being sorted in this example is $n + n = 2n$. Therefore, the expected time would be $4n\log_e(2n)\big(1 + o(1)\big) = 4n\log_e(n)\big(1 + o(1)\big)$. Thus the factor by which we are quicker here is again $2\log_e(2)$. In other words, we get the same speed-up as for the uniform and bipartite cases. We now move forward to the analysis of random $k$-dimensional orders in Quicksort.

## 5.5 Random k–dimensional orders

Here, the application of Quicksort in a random k–dimensional order is considered. For this purpose, a definition follows:

**Definition 5.5.1.** *A random $k$-dimensional partial order on the set $P = \{1, 2, \ldots, n\}$ is defined as follows. We select $k$ total orders on $\{1, 2, \ldots, n\}$ uniformly at random from all $n!$ total orders on that set, say we chose $\leq_1, \leq_2, \ldots, \leq_k$. We then define the partial order $\prec$ by*

$$x \preceq y \Leftrightarrow x \leq_i y \textit{ for all } 1 \leq i \leq k.$$

Of course it is highly likely that some of the total orders will be inconsistent with each other, and so we will only get a partial order. We now have to change perspective: we assume that the partial order which results from these $k$ total orders is given to us as partial information about the order on the set $P$, and that we have to use pairwise comparisons to find the true total order on $P$. Therefore, the total order we are looking for, and the $k$ total orders we used to define the partial order, may have little to do with each other.

We aim to estimate time complexity of finding the true order when the partial information given is a random $k$-dimensional order. Again, we use Theorem $5.2.8$ coupled with the information lower bound. Thus we need to know about the number of linear extensions of a random $k$-dimensional order. An important result follows,

**Theorem 5.5.2** (Brightwell [12]). *The number $e(P)$ of linear extensions of a random $k$-dimensional partial order $P$ with $|P| = n$ satisfies* **whp.**

$$\left(e^{-2}n^{1-1/k}\right)^n \leq e(P) \leq \left(2kn^{1-1/k}\right)^n .$$

Consequently we have that $\log_2(e(P))$ is bounded below by $n \cdot (1 - 1/k) \cdot \log_2(n)(1 + o(1))$ and similarly is bounded above by $n \cdot \big(\log_2(2k) + (1 - 1/k) \cdot \log_2(n)\big)(1+o(1))$. This is of course larger in order of magnitude (for fixed $k$, say) than $2n$ so the information lower bound is tight. Now, we deduce that

**Corollary 5.5.3.** *The time complexity of finding the true total order on a set given a random $k$-dimensional partial order on it, where $k$ is a constant, is* **whp.** *asymptotically equivalent to*

$$\frac{n \log_e(n)}{\log_e(2)} \cdot \left(1 - \frac{1}{k}\right) .$$

**Proof.** For the logarithm of the number of the linear extensions as $n$ tends to infinity holds,

$$\begin{aligned}
&n \cdot \left(1 - \frac{1}{k}\right) \cdot \log_2(n)\big(1 + o(1)\big) \leq \log_2\big(e(P)\big) \\
&\quad\leq n \cdot \big(\log_2(2k) + \left(1 - \frac{1}{k}\right) \cdot \log_2(n)\big) \\
&\Longrightarrow \frac{n}{\log_e(2)} \cdot \left(1 - \frac{1}{k}\right) \cdot \log_e(n) \\
&\quad\leq \log_2\big(e(P)\big) \leq \frac{n}{\log_e(2)} \cdot \big(\log_2(2k) + \left(1 - \frac{1}{k}\right) \cdot \log_e(n)\big).
\end{aligned}$$

Then, we obtain

$$\log_2\big(e(P)\big) = \frac{n \cdot \log_e(n)}{\log_e(2)} \cdot \left(1 - \frac{1}{k}\right) \cdot \big(1 + o(1)\big),$$

which completes the proof. ∎

Therefore, the speed up relating to Quicksort with no prior information is on average

$$\begin{aligned}
&\frac{n \cdot \log_e(n)}{2n \cdot \log_e(n)\log_e(2)} \cdot \left(1 - \frac{1}{k}\right) \\
&\quad= \frac{1}{2\log_e(2)} \cdot \left(1 - \frac{1}{k}\right) \\
&\quad\approx 0.72135 \cdot \left(1 - \frac{1}{k}\right).
\end{aligned}$$

**Remark 5.5.4.** *Note that the factor*

$$\frac{1}{2\log_e(2)} \approx 0.72135$$

*was previously encountered in uniform and bipartite random orders. It is worth to point out that for the multiplier*

$$1 - \frac{1}{k},$$

*when $k$ is arbitrarily large, the speed up is as of the case of uniform random orders. Whereas, having few $k$-dimensional orders, Quicksort runs much faster.*

## 5.6 Random interval orders

In this section, we examine the case where poset forms an interval order. A definition follows [23], [71].

**Definition 5.6.1.** *A poset $(P, \leq)$ is called an interval order if there exists a function $I$ such that each element $x \in P$ is mapped to a closed interval $I(x) = [a_x, b_x] \subseteq \mathbf{R}$. Then $\forall x, y \in P$, it holds that $x \prec y$ if and only if $b_x \leq a_y$.*

In other words, there exists a mapping $x \mapsto I(x) := [a_x, b_x]$ for every element of $(P, \leq)$, having the property that any two elements of the poset are comparable if and only if their corresponding intervals do not intersect. Otherwise, they are incomparable. Hence, the size of the largest chain of the poset is the maximum number of pairwise non-intersecting intervals. Conversely, the size of the largest antichain is the maximum number of intersecting intervals. We present the definition of random interval order.

**Definition 5.6.2.** *A random interval order is one where we generate $2n$ independent numbers $X_1, \ldots, X_n, Y_1, \ldots, Y_n$ from the uniform distribution on $[0, 1]$ and form $n$ closed intervals $I_j$, for $1 \leq j \leq n$, where $I_j = [X_j, Y_j]$ if $X_j < Y_j$*

*and $[Y_j, X_j]$ otherwise. (The event that $X_j = Y_j$ has probability zero so can be ignored). Then we define a partial order by saying that $I_i \prec I_j$ if and only if the maximum element of $I_i$ is less than the minimum element of $I_j$.*

**Remark 5.6.3.** *In fact any continuous probability distribution can be chosen to the analysis of random interval orders.*

Again, we want to estimate how many linear extensions there are of these. This time, there does not appear to be an immediate bound for the number of linear extensions in the literature. However one can obtain the relevant bound showing that $\log_2\big(e(P)\big)$ is **whp.** at least $cn \log_e(n)$ for some $c > 0$, which will of course be enough to show that the $2n$ term in the Fredman [25] – Kislitsyn [41] bound $\log_2\big(e(P)\big) + 2n$ is small compared with the term $\log_2\big(e(P)\big)$. The main results that we need in this direction are the following two Theorems, regarding the size of the largest antichain and chain of a random interval order respectively. For their proofs, we refer to [39].

**Theorem 5.6.4** (Justicz *et al.* [39])**.** *Let $A_n$ denote the size of a largest set of pairwise intersecting intervals in a family of $n$ random intervals. Then there exists a function $f(n) = o(n)$, such that* **whp.** *we have*

$$\frac{n}{2} - f(n) \leq A_n \leq \frac{n}{2} + f(n).$$

**Theorem 5.6.5** (Justicz *et al.* [39])**.** *Let $Y_n$ denote the maximum number of pairwise disjoint intervals in a family of $n$ random intervals. Then*

$$\lim_{n\to\infty} \frac{Y_n}{\sqrt{n}} = \frac{2}{\sqrt{\pi}}$$

*in probability.*

The following Corollary gives a lower bound for the number of comparisons required to sort a random interval order:

**Corollary 5.6.6.** *The number of comparisons for sorting $n$ keys, given a random interval order is* **whp.** *at least $cn\log_e(n)\big(1+o(1)\big)$, where one can take $c = 1/2\log_e(2) \approx 0.72135$.*

**Proof.** Theorem 5.6.4 shows that the largest antichain of the random interval order is **whp.** at least $r = \lceil(1-\epsilon)n/2\rceil$ for any $\epsilon > 0$. This is because a family of intersecting intervals forms an antichain. Thus we need to sort all these $r$ incomparable elements of the partial order in a total order extending it, and there are at least $r!$ ways of doing this. Using Stirling's formula, we obtain

$$r! \geq \left(\frac{(1-\epsilon)n}{2}\right)! \sim \sqrt{(1-\epsilon)n\pi}\left(\frac{(1-\epsilon)n}{2e}\right)^{(1-\epsilon)n/2}$$
$$\Longrightarrow \log_2\big(e(P)\big) \geq \log_2(r!) \geq \frac{(1-\epsilon)n}{2}\log_2\left(\frac{(1-\epsilon)n}{2e}\right) + \frac{1}{2}\log_2\big((1-\epsilon)n\pi\big).$$

Then we have that $\log_2\big(e(P)\big)$ has order of magnitude $n\log_2(n)$: in particular $2n = o\big(\log_2(e(P)\big)$ and so the complexity is $\log_2\big(e(P)\big)\big(1+o(1)\big)$. Further, we have that $\log_2\big(e(P)\big)$ is at least, by the above, $n(1-\epsilon)\log_2(n)\big(1+o(1)\big)/2$ which is equal to $n\log_e(n)\dfrac{1+o(1)}{2\log_e(2)}$ and so we can take the constant $c$ to be at least $1/2\log_e(2)$. ∎

We now present a much stronger result that gives sharp bounds on the number of linear extensions of a random interval order, following the insightful suggestions of Prof. Colin McDiarmid [52].

**Theorem 5.6.7.** *The number of comparisons for sorting $n$ keys, given a random interval order is* **whp.** *for $0 < \epsilon < 1$, between:*

$$\big(1 - \epsilon + o(1)\big) n \log_2(n) \leq \log_2\big(e(P)\big) \leq \big(1 + \epsilon + o(1)\big) n \log_2(n).$$

**Proof.** Let $0 < a < b < 1$ and consider the interval $(a, b)$. Let $I(i, j)$ be the interval $(\frac{i-1}{2^j}, \frac{i+1}{2^j})$, where $i$ and $j$ are positive integers with $i$ odd and $i < 2^j$. Further, let $j(a, b)$ be the least $j$ such that $\frac{i}{2^j} \in (a, b)$ for some positive integer $i$. There is a unique such $i$ since if there were at least two odd $i$, then there is at least one even $k$ between them and considering $k/2^j$, we can replace it by $(k/2)/2^{j-1}$ giving a smaller value of $j$ and contradicting the definition of $j$. Since $i$ is unique, we may call it $i(a, b)$.

We denote the interval $I\big(i(a, b), j(a, b)\big)$ by $J(a, b)$. Recall that $i$ is odd with $i < 2^j$ and observe that if $b - a > 2^{-j}$, then $j(a, b) \leq j$. For a given such $i$ and $j$, let $\mathcal{A}(i, j)$ be the set of all intervals $(a, b)$, such that $J(a, b) = I(i, j)$. The sets $\mathcal{A}(i, j)$ are antichains: for if we had two intervals $(a_1, b_1)$ and $(a_2, b_2)$ in $\mathcal{A}(i, j)$ with $(a_1, b_1)$ being less than $(a_2, b_2)$ (of course this is equivalent to $b_1 < a_2$) then saying that

$$J(a_1, b_1) = J(a_2, b_2) = \left(\frac{i-1}{2^j}, \frac{i+1}{2^j}\right)$$

would imply that $j(a_1, b_1) = j(a_2, b_2)$ and $i(a_1, b_1) = i(a_2, b_2)$. But given that $j(a_1, b_1) = j(a_2, b_2)$, there is clearly some $i/2^j$ with $i$ odd in $(a_1, b_1)$ which is less than any such in $(a_2, b_2)$ and the result follows. Indeed, if the midpoint of $I(i, j)$ (i.e. $i/2^j$) is less than the midpoint of $I(i', j')$ (i.e. $i'/2^{j'}$) then no interval in $\mathcal{A}(i', j')$ can precede any interval in $\mathcal{A}(i, j)$ in the interval order.

Let $X$ and $Y$ be independent and uniformly distributed random variables which denote the endpoints of a random interval in $(0,1)$. We have that

$$\mathbb{P}(|X-Y| \leq \epsilon/4) < \epsilon/2,$$

since

$$\begin{aligned}\mathbb{P}\big(|X-Y| \leq \frac{\epsilon}{4}\big) &= \mathbb{P}\big(Y-\frac{\epsilon}{4} \leq X \leq Y+\frac{\epsilon}{4}\big)\\ &= \int_0^1 \mathbb{P}\big(Y-\frac{\epsilon}{4} \leq X \leq Y+\frac{\epsilon}{4}\big|Y=y\big) f_Y(y)\,\mathrm{d}y\\ &= \int_0^1 \mathbb{P}\big(Y-\frac{\epsilon}{4} \leq X \leq Y+\frac{\epsilon}{4}\big|Y=y\big)\,\mathrm{d}y.\end{aligned}$$

The last equation follows because the probability density of $Y$, $f_Y(y)$ is $1$ on $[0,1]$ and $0$ elsewhere. Given that $Y=y$, the probability that $X$ is in the interval $(y-\epsilon/4, y+\epsilon/4)$ is (as it is uniformly distributed) at most the length of the interval $(y+\epsilon/4)-(y-\epsilon/4)=\epsilon/2$.

Thus the random number $\mathcal{N}$ of intervals with length at most $\epsilon/4$ is stochastically dominated by a binomial random variable $\mathcal{B}(n,\epsilon/2)$ with $n$ independent trials and success probability $\epsilon/2$. Therefore,

$$\mathbb{P}\big(\mathcal{N} \geq \epsilon n) \leq \mathbb{P}(\mathcal{B}(n,\epsilon/2) \geq \epsilon n\big)$$

and this probability tends to $0$, as $n \to \infty$. This is a consequence of Chernoff's inequality, in the following form: if $X$ is a binomially distributed variable with $n$ independent trials and success probability $p$, then for $\delta > 0$

$$\mathbb{P}\big(X \geq n(p+\delta)\big) \leq \left(\left(\frac{p}{p+\delta}\right)^{p+\delta}\left(\frac{1-p}{1-p-\delta}\right)^{1-p-\delta}\right)^n.$$

Chernoff's inequality appears in various places: we refer to [8] and to [54]. Since the number being raised to the power $n$ is $< 1$, this will indeed tend to $0$ (in fact will do so rapidly). The result in our case follows plugging in $p = \epsilon/2$ and $\delta = \epsilon/2$ and shows that the number of intervals with length at most $\epsilon/4$ is **whp.** less than $\epsilon n$.

Let $j_0$ be a positive integer and $m$ be the number of intervals $I(i,j)$ with $j \leq j_0$. Let $\mathcal{I}$ be a set of intervals and $\mathcal{I}'$ be the set of intervals in $\mathcal{I}$ with length $> 2^{-j_0}$. Let $n(i,j)$ be the number of intervals of $\mathcal{I}'$ in $\mathcal{A}(i,j)$. Then, the number of linear extensions $e(P)$ satisfies, since we have to put each of the antichains in order and there are $r!$ ways to order an antichain of $r$ elements,

$$e(P) \geq \prod n(i,j)! \geq \prod \left(\frac{n(i,j)}{e}\right)^{n(i,j)}$$

and by convexity

$$\log_2\bigl(e(P)\bigr) \geq \sum n(i,j)\log_2 \frac{n(i,j)}{e} \geq |\mathcal{I}'| \log_2 \frac{|\mathcal{I}'|}{em}.$$

Choosing $j_0$ sufficiently large that $2^{-j_0} < \epsilon/4$ , the number of intervals of length at most $2^{-j_0}$ is less than the number of intervals of length $< \epsilon/4$ which by Chernoff's inequality is **whp.** $< \epsilon n$: thus almost all intervals have length at least $2^{-j_0}$ and so $\mathcal{I}'$ has order at least $(1-\epsilon)n$. This will give us that **whp.**

$$\log_2\bigl(e(P)\bigr) \geq \bigl(1 - \epsilon + o(1)\bigr) n \log_2(n).$$

This completes the proof, as this gives the lower bound and the upper bound is just a consequence of the fact that there are at most $n!$ linear extensions of a partially ordered set with $n$ keys, and then we use Stirling's formula again. ∎

In the next Chapter we proceed to the analysis of partial orders where the information–theory lower bound is not $\omega(n)$. Central to the subsequent analysis are random graphs.

# Chapter 6

# Linear extensions of random graph orders

Recall that in the previous Chapter, we examined the number of linear extensions of various partial orders, where the information–theoretic lower bound dominated the linear term. In this Chapter, we examine the case where both terms are asymptotically equivalent. We obtain bounds of the expected height of a random graph and we derive a new bound on the number of linear extensions of a random graph order.

## 6.1 Random graph orders

In this section, we consider random graphs. A definition follows [37]:

**Definition 6.1.1.** *The Erdős–Rényi random graph $G(n, p)$ has labelled vertex set $\{1, 2, \dots, n\}$ and for each pair of vertices, the probability of an edge arising between them is $p$, independently of all the other pairs of vertices.*

By the definition, here and throughout this thesis, a random graph $G(n, p)$ denotes a simple graph, without loops or multiple edges. An independent set of a graph is a subset of the vertex set, such that there is no edge connecting any two vertices. On the other hand, a clique of a random graph $G(n, p)$ is a subset of its vertex set, with the property that an edge is arising between every two vertices. Note that $p$ may very well depend on $n$. We will usually be interested in the behaviour as $n \to \infty$.

**Definition 6.1.2.** *The random graph order $P(n, p)$, with partial order relation $\prec$, is a partially ordered set with underlying set the vertices of $G(n, p)$ and we initially say that $i \prec j$ if and only if $i < j$ and the edge $i \sim j$ is present in the random graph $G(n, p)$. We then take the transitive closure of this relation to get a partial order.*

*In the same manner, we write $P(\mathbf{Z}, p)$ for the infinite partially ordered set obtained by taking vertex set $\mathbf{Z}$, the set of integer numbers, saying initially $i \prec j$ if and only if $i < j$ in the usual total order on $\mathbf{Z}$ and the edge $i \sim j$ is present (which it is with probability $p$ independent of all other edges), and then taking the transitive closure.*

The point is that for a partial order we of course require transitivity by the axioms for a partially ordered set. But of course this is not guaranteed in a random graph. We could, for example, have the edges $1 \sim 2$ and $2 \sim 3$ in the random graph, but no edge between $1$ and $3$. Then of course we would have put in the edge $1 \sim 3$ as since $1 \prec 2$ and $2 \prec 3$ we must have $1 \prec 3$ by transitivity. Note that there are efficient algorithms for finding the transitive closure of a relation. What we aim to do next, following the analysis in the last section, is to consider how many comparisons will be needed to finalise the order of a set of keys when a random graph partial order on the set is given already. For our needs, we present the definition of graph entropy, introduced by Körner [47]. This definition is from Simonyi's survey on graph entropy [67].

**Definition 6.1.3.** *Let $G = G(n,p)$ be a random graph. Let $X$ be a random variable taking its values on the vertices of $G$ and $Y$ taking its values on the stable (independent) sets of $G$. Suppose further that their joint distribution is such that $X \in Y$ with probability $1$. Also, the marginal distribution of $X$ on $V(G)$ is identical to the given distribution $P$. Then, the graph entropy $H(G,P)$ of the random graph $G$ is*

$$H(G,P) = \min I(X \wedge Y),$$

*where $I(X \wedge Y)$ is as in Definition 3.1.7.*

As in the last Chapter, we need to know about the number of linear extensions. The following Theorem from [4] will give us what we need.

**Theorem 6.1.4** (Alon *et al.* [4])**.** *Let $0 < p < 1$ be fixed and consider $e(P)$, where the partial order is from $P(n,p)$. Then we have that there are $\mu(p) > 0$ and $\sigma^2(p) > 0$ such that*

$$\frac{\log_e\big(e(P)\big) - \big(\mu(p) \cdot n\big)}{\sigma(p) \cdot \sqrt{n}} \xrightarrow{\mathcal{D}} \mathbb{N}(0,1).$$

**Corollary 6.1.5.** *There is a constant $c(p)$ such that* **whp.** *we have*

$$\log_2\big(e(P)\big) = c(p)n + O\big(n^{1/2}\omega(n)\big),$$

*where $\omega(n)$ is any function tending to infinity with $n$ (we usually think of it as doing so very slowly).*

**Proof.** For random variable $\mathbb{N}(0,1)$, the probability that it is between $-\omega(n)$ and $\omega(n)$ is $1-o(1)$ as $n \to \infty$. Thus for large enough $n$

$$\frac{\log_e\bigl(e(P)\bigr) - \mu(p)\cdot n}{\sigma(p)\cdot\sqrt{n}} \in \bigl(-\omega(n), \omega(n)\bigr).$$

Changing the base of the logarithm, we get:

$$\frac{\frac{\log_2(e(P))}{\log_2(e)} - \mu(p)\cdot n}{\sigma(p)\cdot\sqrt{n}} \in \bigl(-\omega(n), \omega(n)\bigr)$$
$$\Longrightarrow \log_2\bigl(e(P)\bigr) \in \Bigl(\log_2(e)\cdot\bigl(\mu(p)n - \omega(n)\sigma(p)\sqrt{n}\bigr), \log_2(e)\cdot\bigl(\mu(p)n + \omega(n)\sigma(p)\sqrt{n}\bigr)\Bigr).$$

and this gives the claim, with $c(p) = \mu(p)\cdot\log_2(e) > 0$. ∎

**Corollary 6.1.6.** *For a random graph order $P(n,p)$ with $0 < p < 1$ constant,*

$$\log_2\bigl(e(P)\bigr) = \log_2(e)\mu(p)n\bigl(1+o(1)\bigr).$$

**Proof.** The proof follows directly from the previous Corollary. ∎

In other words, the logarithm of the number of linear extensions is linear in $n$. Thus when we use Theorem 5.2.8, we see that both terms in it $\log_2\bigl(e(P)\bigr)$ and $2n$ are linear. So we do not get the exact asymptotics as in the previous Chapter. Recall that in that Chapter, we always had $\log_2\bigl(e(P)\bigr) = \omega(n)$ so outweighed the linear term. (In many cases, $\log_2(e(P))$ was of order of magnitude $n\log_e(n)$ so won comfortably). However here it is not clear what multiple of $n$ will be the time complexity of finding the total order by pairwise comparisons. Note that it will be at most some constant multiple of $n$, so this will be quicker than just using about $2n\log_e(n)$ comparisons in Quicksort – in other words, the partial order here does substantially speed up the process of finding the true order.

To deal with this question in more detail, we need to know about the structure of a random graph order. We concentrate to begin with on the case where $p$ is a constant. In this case, we shall see that basically the partial order consists of a linear sum of smaller partial orders.

## 6.2 Additive parameters and decomposing posets

**Definition 6.2.1.** *A vertex $v$ in any partial order is said to be a post if and only if every other vertex of the partial order is comparable with it.*

If $v$ is a post in a partial order $(P, \prec)$, then we can write the partial order as a linear sum of two subposets $(P_1, \prec)$ and $(P_2, \prec)$, namely

$$P_1 = \{x \in P : x \preceq v\}\, P_2 = \{x \in P : x \succ v\}.$$

Clearly two elements in the same $P_i$ which were comparable before still are, and any element $x$ in $P_1$ is smaller than any element $y$ in $P_2$ since $x \prec v \prec y$ by assumption.

**Definition 6.2.2.** *A parameter $f$ of partial orders is said to be additive if and only if, whenever $P$ is the linear sum of two subposets $P_1$ and $P_2$ we have $f(P) = f(P_1) + f(P_2)$.*

An example of an additive parameter is the height, as if we have a longest chain in $P_1$ and a longest chain in $P_2$, we can concatenate them to form a chain in the linear sum, so $f(P) \geq f(P_1) + f(P_2)$: and in the other direction, given a longest chain in $P$, we restrict to the subsets and get chains in $P_1$ and $P_2$, so $f(P) \leq f(P_1) + f(P_2)$, and so they are equal. Another example is the number of further comparisons needed to sort a partially ordered set which is a linear sum of two subposets. An important Lemma follows

**Lemma 6.2.3.** *The number of further comparisons $c(P)$ of Quicksort which need to be applied to a poset $P$ to obtain the true ordering, with knowledge of where the posts are and where elements are relative to the posts, is an additive parameter.*

**Proof.** Write $P = P_1 \oplus P_2$. If we sort the whole linear sum of $P_1$ and $P_2$ with $k$ comparisons, we have sorted $P_1$ and $P_2$ as well. Thus, taking $k = c(P)$, we see that $c(P) \geq c(P_1) + c(P_2)$. (We have sorted $P_1$ and $P_2$ using $k$ comparisons: we might have done it with fewer). Conversely, if we have sorted $P_1$ and $P_2$ with a total of $k$ comparisons, then we have sorted the whole of $P$ because, by definition, everything in $P_1$ is above everything in $P_2$, thus $c(P) \leq c(P_1)+c(P_2)$. The result follows. ∎

Now here is a key result from Brightwell [10].

**Theorem 6.2.4** (Brightwell [10]). *With probability $1$, the set of posts in $P(\mathbf{Z}, p)$ for $0 < p < 1$ constant, is infinite.*

Basically, each bit between posts will be small. Indeed Brightwell's survey [10] also shows that for sufficiently large $k$, given our $p$ (which remember is constant) there is a constant dependent of $p$, $c(p) > 1$ such that the probability that none of $\{2k, 4k, \ldots, 2k^2\}$ are posts is less than or equal to $c^{-k}$. We can now start showing how to use this idea to break down various invariants of $P(n, p)$ into small units. We need some notation about posts. Let their positions be

$$\dots, U_{-1}, U_0, U_1, \dots$$

where $U_0$ is the first post at or to the right of $0$. Then we say that $P_j$ is the poset induced on the interval $(U_j, U_{j+1}]$. These posets are called the factors of the partial order. The next Theorem from [10] presents an important result, regarding convergence in distribution of additive parameters.

**Theorem 6.2.5** (Brightwell [10])**.** *Let $p$ be a constant with $0 < p < 1$. Let $f$ be an additive parameter of partial orders which is not proportional to $|P|$. Let $Y$ and $Z$ be the random variables $f(P_0)$ and $f(P_{-1})$ respectively. Further, suppose that the moments $\mathbb{E}(Y^r)$ and $\mathbb{E}(Z^r)$ are finite for all $r \in \mathbf{N}$. Then there exist constants $\mu = \mu(p) > 0$ and $\sigma = \sigma(p) > 0$, such that $\mathbb{E}\big(f(P(n,p))\big)/n \to \mu$ and $\mathrm{Var}\big(f(P(n,p))\big)/n \to \sigma^2$. Furthermore*

$$\frac{f\big(P(n,p)\big) - \mu(p) \cdot n}{\sigma(p) \cdot \sqrt{n}} \xrightarrow{\mathcal{D}} \mathbb{N}(0,1),$$

*with convergence of all moments.*

The following Corollary is a consequence of Theorem $6.2.5$.

**Corollary 6.2.6.** *Given $0 < p < 1$ constant, the height of $P(n,p)$, which is an additive parameter is* **whp.** *equal to $\mu(p) \cdot n \cdot \big(1 + o(1)\big)$.*

**Proof.** This follows easily from the previous Theorem. ∎

## 6.3 Average height of random graph orders

What we really need to do now is to obtain bounds for the average number of linear extensions. Albert and Frieze [3], derived estimates for the average height of a random graph order, which is an additive parameter as we previously saw. The idea is the following. A random graph is sequentially constructed; at each step a new vertex $j$ is added and the probability of an edge from it to any previously existing vertex $i$ with $i \leq j$ is $1/2$. They consider both an underestimate of the height and an overestimate. We present a generalisation of this construction with a constant probability $p \in (0, 1)$ of an edge arising.

**Theorem 6.3.1.** *The underestimate $f(p)$ and overestimate $h(p)$ increments of the average height are given by:*

$$f(p) = 1 - \frac{\displaystyle\sum_{j=1}^{\infty}\left(\left(\prod_{i=1}^{j-1}\frac{p(1-p)^i}{1-(1-p)^{i+2}}\right)\left((1-p)^j\right)\right)}{\displaystyle\sum_{j=1}^{\infty}\left(\prod_{i=1}^{j-1}\frac{p(1-p)^i}{1-(1-p)^{i+2}}\right)}$$

$$h(p) = \frac{1}{\displaystyle\sum_{j=1}^{\infty}(1-p)^{\frac{j(j-1)}{2}}}.$$

**Proof.** Let $l_k$ be the length of the longest chain in a random graph order of size $k \in \mathbf{N}$ and $d_k$ be the number of top endpoints of longest chains. Consider the addition of vertex $(k+1)$. The event that the new vertex $(k+1)$ is the new, unique, endpoint of a longest chain is the event that one or more of the edges from $(k+1)$ to the $d_k$ endpoints actually arises. In this event, what will happen is that the length of the longest chain will increase by $1$ and the number of endpoints of longest chains will drop to $1$, with probability $1-(1-p)^{d_k}$. The complementary event is that the number of endpoints will be increased by one.

It is at the next step that we make a pessimistic assumption. The pessimistic assumption is that, in these cases where $(k+1)$ does not become the unique endpoint of a longest chain, the number of endpoints increases by $1$ with probability only $p$. In fact, though there will certainly be at least one vertex one level below all the $d_k$ upper endpoints, in most cases there will be more than that – say $r$ of them – so if $(k+1)$ is joined to any of them the number of longest chains will increase, and the probability of this happening will be $1-(1-p)^r$. In this event, the number of endpoints will increase by at least $1$, but the length of the longest chain will remain unchanged.

Let the random variables $\eta$ and $\theta$ denote the underestimates of the length of the longest chain and the number of endpoints of the longest chain(s) respectively. These variables obey the following recurrence:

$$(\eta_{k+1},\theta_{k+1}) = \begin{cases} (\eta_k+1,1) & \text{with probability } 1-(1-p)^{\theta_k} \\ (\eta_k,\theta_k+1) & \text{with probability } p(1-p)^{\theta_k} \\ (\eta_k,\theta_k) & \text{with  probability } (1-p)(1-p)^{\theta_k} \end{cases}$$

Also, let $\mu$ and $\phi$ be the overestimates of the length of the longest chain and the number of endpoints of the longest chain(s) respectively. In this case, the recurrence relation is:

$$(\mu_{k+1}, \phi_{k+1}) = \begin{cases} (\mu_k + 1, 1) & \text{with probability } 1 - (1-p)^{\phi_k} \\ (\mu_k, \phi_k + 1) & \text{with probability } (1-p)^{\phi_k} \end{cases}$$

We consider only the second component. This is clearly a positive recurrent, irreducible, aperiodic, Markov process with state space the positive integers. Thus a stationary distribution does exist with limiting probabilities

$$p_j = \lim_{n \to \infty} \mathbb{P}(\theta_n = j)$$

and when $\theta_{k+1} = j + 1$, then this could come about from $\theta_k$ being $j$ (with probability $p(1-p)^j$) or from $\theta_k$ being $j+1$ (with probability $(1-p)^{(j+1)+1} = (1-p)^{j+2}$). The solution giving the stationary distribution $p_j = \lim_{n \to \infty} \mathbb{P}(\theta_n = j)$ for the first case is

$$\begin{aligned} p_{j+1} &= p(1-p)^j p_j + (1-p)(1-p)^{j+1} p_{j+1} \\ \implies p_{j+1} &= \frac{p(1-p)^j}{1-(1-p)^{j+2}} \cdot p_j \\ \implies p_{j+1} &= \prod_{i=1}^{j} \frac{p(1-p)^i}{1-(1-p)^{i+2}} \cdot p_1. \end{aligned}$$

The value of $p_1$ can be found by the following equation

$$p_1 \sum_{j=1}^{\infty} \left( \prod_{i=1}^{j-1} \frac{p(1-p)^i}{1-(1-p)^{i+2}} \right) = 1.$$

The expected height increment generally for $p \in (0,1)$ is

$$p_1 \sum_{j=1}^{\infty}\left(\left(\prod_{i=1}^{j-1}\frac{p(1-p)^i}{1-(1-p)^{i+2}}\right)\left(1-(1-p)^j\right)\right).$$

Substituting $p_1$, which is a function of $p$, the average height increment is

$$f(p) = \frac{1}{\sum_{j=1}^{\infty}\left(\prod_{i=1}^{j-1}\frac{p(1-p)^i}{1-(1-p)^{i+2}}\right)} \sum_{j=1}^{\infty}\left(\left(\prod_{i=1}^{j-1}\frac{p(1-p)^i}{1-(1-p)^{i+2}}\right)\left(1-(1-p)^j\right)\right),$$

which is equal to

$$1 - \frac{\sum_{j=1}^{\infty}\left(\left(\prod_{i=1}^{j-1}\frac{p(1-p)^i}{1-(1-p)^{i+2}}\right)\left((1-p)^j\right)\right)}{\sum_{j=1}^{\infty}\left(\prod_{i=1}^{j-1}\frac{p(1-p)^i}{1-(1-p)^{i+2}}\right)}.$$

For the second (overestimate) case, the stationary distribution obeys the following recursive relation

$$\begin{aligned} & p_{j+1} = (1-p)^j p_j \\ \Longrightarrow\ & p_{j+1} = (1-p)^j \cdot (1-p)^{j-1} \cdot \ldots \cdot (1-p)p_1 = (1-p)^{\frac{j(j+1)}{2}} p_1. \end{aligned}$$

Thus, the value of $p_1$ can be retrieved by the following equation

$$\sum_{j=1}^{\infty} p_j = p_1 \sum_{j=1}^{\infty}(1-p)^{\frac{j(j-1)}{2}} = 1$$

and the average height increment is

$$p_1 \sum_{j=1}^{\infty} \left( (1-p)^{\frac{j(j-1)}{2}} \left(1-(1-p)^j\right) \right),$$

which further simplified yields the simple expression

$$p_1 \sum_{j=1}^{\infty} \left( (1-p)^{\frac{j(j-1)}{2}} - (1-p)^{\frac{j(j+1)}{2}} \right) = p_1.$$

Therefore, the average height increment in the overestimate case is

$$h(p) = \frac{1}{\sum\limits_{j=1}^{\infty} (1-p)^{\frac{j(j-1)}{2}}}.$$

The argument is complete. ∎

We can rewrite the inverse of the overestimate as

$$\begin{aligned}
\sum_{j=1}^{\infty} (1-p)^{\frac{j(j-1)}{2}} &= (1-p)^0 + (1-p)^1 + (1-p)^{1+2} + (1-p)^{1+2+3} + \ldots \\
&= 1 + (1-p) + (1-p)(1-p)^2 + (1-p)(1-p)^2(1-p)^3 + \ldots \\
&= \sum_{j=1}^{\infty} \left( \prod_{i=1}^{j-1} (1-p)^i \right).
\end{aligned}$$

The average height increment can be computed in terms of $\theta$ elliptic functions. This class of functions arises in many different areas of Mathematics and a comprehensive account of their analyses can be found at [1], [74]. The overestimate height increment assumes the simple form of

$$\frac{2\sqrt[8]{1-p}}{\theta_2(0, \sqrt{1-p})},$$

where the $\theta_2$ function is defined by, (see section 16.27 in Abramowitz and Stegun [1]),

$$\theta_2(z,q) = 2q^{1/4}\sum_{n=0}^{\infty} q^{n(n+1)}\cos\big((2n+1)z\big),$$

with $q = 1-p$.

The previous equation can be expressed as a product, instead of summation. It holds [74],

$$\theta_2(z,q) = 2q^{1/4}G\cos(z)\prod_{n=1}^{\infty}\left(1+2q^{2n}\cos(2z)+q^{4n}\right),$$

where $G = \prod_{n=1}^{\infty}(1-q^{2n})$. Therefore, the expected overestimate height increment becomes

$$\begin{aligned}&\frac{1}{\prod_{n=1}^{\infty}\big(1-(1-p)^n\big)\prod_{n=1}^{\infty}\big(1+2(1-p)^n+(1-p)^{2n}\big)}\\ &\qquad=\frac{1}{\prod_{n=1}^{\infty}\Big(\big(1-(1-p)^n\big)\big(1+(1-p)^n\big)^2\Big)}.\end{aligned}$$

The following MAPLE graph plots the underestimate and the overestimate functions. Note that for $p \geq 0.7$, both $f(p)$ and the overestimate $h(p)$ are in fact very close to $p$.

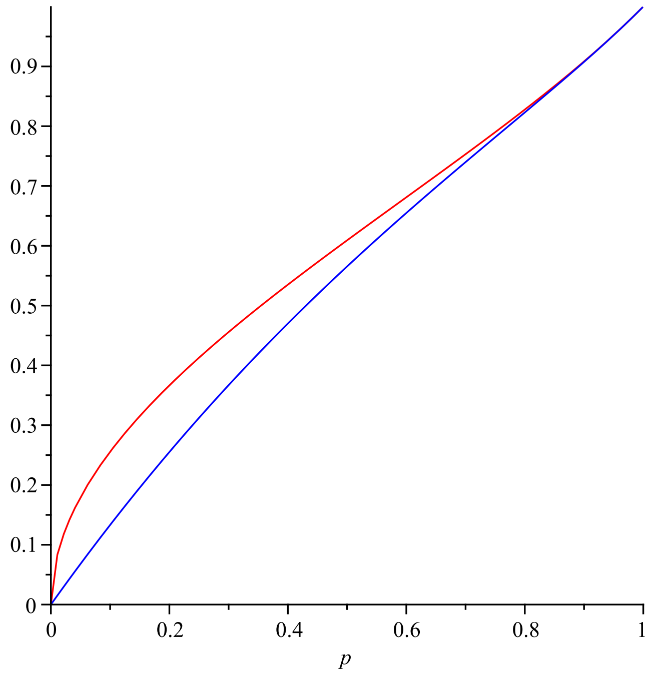

**Figure 6.1:** Plot of underestimate (blue) and overestimate (red) functions of the expected height increment of a random graph order.

In the next section, we give sharp bounds for the average height.

## 6.4 Bounds on the height of random graph orders

We have managed to obtain functions $f(p)$, which is the underestimate increment of the expected height and $h(p)$, which is the overestimate of the expected height of a random graph partial order. We first give a lower bound on the underestimate function, which is easier to work with (more tractable).

**Lemma 6.4.1.** *For all $p \in (0,1)$, $f(p) \geq p$.*

**Proof.** The probability of an edge arising between a newly added vertex $(k+1)$ and any of the $k$ existing vertices in a random graph is $p$, independently of the other edges. Thus, with probability $p$ the length of a greedy chain will increase by one and with probability $(1-p)$, the length will remain unchanged. Thus, the expected increase in the height of a chain is $p$ and the claim follows immediately. ∎

Here is an upper bound on the overestimate function.

**Lemma 6.4.2.** *It holds that*

$$h(p) \le p + \frac{(p-1)^2}{2-p}.$$

**Proof.** To obtain an upper bound on

$$h(p) = \frac{1}{1+(1-p)+(1-p)^3+\dots}$$

it is enough to bound below the denominator $1+(1-p)+(1-p)^3+\dots$ . A crude lower bound is $2-p$ (as all other terms are positive). Thus

$$h(p) \le \frac{1}{2-p}.$$

This in turn is

$$\begin{aligned}\frac{1}{2-p} &= \frac{1}{1-(p-1)} = 1+(p-1)+(p-1)^2+\dots\\ &= p+(p-1)^2+(p-1)^3+\dots\\ &= p+\frac{(p-1)^2}{1-(p-1)}\\ &= p+\frac{(p-1)^2}{2-p}.\end{aligned}$$ ∎

Collecting more terms, we will be able to obtain a sharper bound in the following manner. It holds that

$$1 + (1-p) + (1-p)^3 + \ldots > 2 - p + (1-p)^3 + \ldots + (1-p)^K,$$

for a finite number $K = \dfrac{m(m-1)}{2}$, where $m \in \mathbf{N}$, so the bound has the form

$$\frac{1}{2 - p + \sum_{j=3}^{m} \left( (1-p)^{\frac{j(j-1)}{2}} \right)}.$$

Its sharper than the previous one but again infinitely many terms are discarded. In order to obtain a bound using all terms, note that $j(j-1)/2 \leq j^2$ and

$$1 + (1-p) + (1-p)^3 + \ldots \geq (1-p) + (1-p)^4 + (1-p)^9 + \ldots .$$

The overestimate can be bounded above by another theta function. A sharper bound is

$$h(p) \leq \frac{2}{\theta_3(0, 1-p) + 1},$$

where the $\theta_3$ function is defined [1]

$$\theta_3(z, q) = 1 + 2 \sum_{n=1}^{\infty} q^{n^2} \cos(2nz).$$

Thus, we derived suitable bounds, such that

$$p \leq f(p) < h(p) \leq \frac{2}{\theta_3(0, 1-p) + 1}.$$

In the following section, we present bounds on the number of linear extensions.

## 6.5 Expected number of linear extensions

Here we obtain bounds on the average number of linear extensions of a random graph order. We start by quoting a useful Theorem from [4],

**Theorem 6.5.1** (Alon *et al.* [4]). *Let a random variable $Y$ be geometrically distributed, i.e. $\mathbb{P}(Y = k) = pq^{k-1}$, for $k = 1, 2, \dots$. Then,*

$$\begin{aligned}\mathbb{E}\big(\log_e(Y)\big) = \sum_{k=1}^{\infty} \log_e(k) pq^{k-1} < \mu(p)\\ \leq \big(1 - k(p)\big) \log_e \frac{1/p - k(p)}{1 - k(p)}\\ < \log_e \frac{1}{p} = \log_e\big(\mathbb{E}(Y)\big),\end{aligned}$$

where $k(p)$ is defined as,

$$k(p) = \prod_{k=1}^{\infty} (1 - q^k),$$

with $q = 1 - p$.

The following Theorem regarding the expected number of linear extensions is the main contribution in this Chapter.

**Theorem 6.5.2.** *Let $p \in (0, 1)$. The average increment of the natural logarithm of the number of linear extensions $\mu(p) = \dfrac{\mathbb{E}\big(\log_e\big(e(P)\big)\big)}{n}$ is* **whp.** *bounded below by:*

$$\mu(p) \geq -\frac{\log_2 h(p) \log_e(2)}{2}.$$

**Proof.** We consider the longest chain $C$. We know that **whp.** $\mathbb{E}(|C|) \leq h(p)n$ by Theorem 6.3.1. Now, by Lemma $5$ of Cardinal *et al.* [13], which states that $|C| \geq 2^{-H(\tilde{P})}n$, we deduce that

$$h(p) \geq \mathbb{E}\big(2^{-H(\tilde{P})}\big),$$

where $H(\tilde{P})$ denotes the entropy of the incomparability graph. By Lemma $4$ of that paper, we also have that

$$nH(\tilde{P}) \leq 2\log_2\left(e(P)\right).$$

Therefore

$$h(p) \geq \mathbb{E}\big(e(P)^{-2/n}\big).$$

Taking on both sides logarithms and applying Jensen's inequality [38], we get that **whp.**

$$\begin{aligned}\log_2 h(p) &\geq \log_2 \mathbb{E}\left(e(P)^{-2/n}\right)\\ &\geq \mathbb{E}\left(\log_2(e(P)^{-2/n})\right)\\ &= -\frac{2}{n}\mathbb{E}\big(\log_2(e(P)\big)\\ &= -\frac{2}{\log_e(2)}\mu(p).\end{aligned}$$

Thus, $\mu(p)$ is **whp.** bounded below by

$$\mu(p) \geq -\frac{\log_2 h(p)\log_e(2)}{2}$$

and the proof of the Theorem is complete. ∎

As we can see from the following graph, the bound isn't tight compared with the one of Alon *et al.* [4]. However, its derivation provides an insight into the relation of the height of a random graph order with the number of linear extensions.

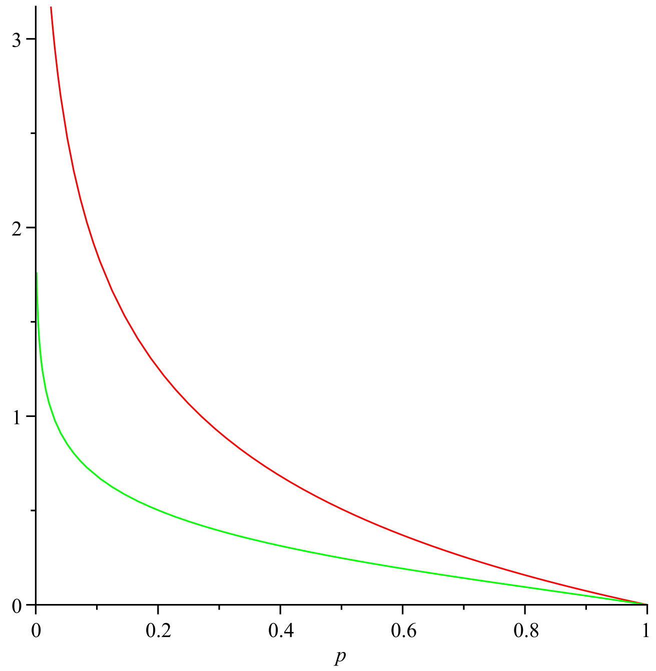

**Figure 6.2:** Plot of Alon *et al.* [4] bound (red) and of bound of Theorem 6.5.2 (green) on $\mu(p)$.

# Chapter 7

# Conclusions and future research directions

At the last Chapter of this thesis, the conclusions of the research and possible future directions are discussed. In the first section, we consider the sorting of partially ordered sets and in the second section, the fast merging of chains.

## 7.1 Sorting partially ordered arrays

Central to the analysis of the time complexity of sorting partially ordered sets, was the number of linear extensions, as a measure of the 'presortedness' of the array. Recall that the quantity $\log_2(n!)$ is the lower bound of comparisons needed to sort an array of $n$ keys, with no prior information. In all cases of partially ordered sets considered in Chapter $5$, the constant $1/2 \log_2(e)$ appeared to the asymptotic number of comparisons. A future direction to research might be the sharpening of these results. For example, one might ask, what is the average number of key exchanges or the computation of exact expected costs.

Generalising Albert–Frieze argument [3] and using entropy arguments, a new result was the lower bound on the number of linear extensions of a random graph order. However, we have seen that it does not directly compete the bounds of Alon *et al.* [4], thus there is space for further improvement of this bound or to derivation of new sharper ones and this might be a suitable topic for further research.

A different bound on the number of linear extensions of a random interval order can be derived as follows. By Theorem 5.6.5, one can deduce that **whp.**, the size of the longest chain C, is

$$|C| = \frac{2\sqrt{n}}{\sqrt{\pi}}.$$

Using the result $|C| \geq 2^{-H(\tilde{P})}n$ from Cardinal *et al.*, as in Theorem 6.5.2, we have

$$\frac{2}{\sqrt{n\pi}} \geq 2^{-H(\tilde{P})}.$$

Taking logarithms, a lower bound for the entropy of the incomparability random interval graph is,

$$H(\tilde{P}) \geq \frac{1}{2}\log_2(n) + O(1).$$

By the following inequality [13]

$$nH(\tilde{P}) \leq 2\log_2\bigl(e(P)\bigr),$$

$\log_2\big(e(P)\big)$ is **whp.** bounded below by

$$\log_2\big(e(P)\big) \geq \frac{n}{4}\log_2(n) + O(n).$$

Thus, **whp.**

$$\begin{aligned}\log_2\big(e(P)\big) &\geq \frac{n}{4}\log_2(n)\big(1+o(1)\big)\\ &= \frac{n}{4\log_e(2)}\log_e(n)\big(1+o(1)\big)\\ &\approx 0.36n\log_e(n)\big(1+o(1)\big).\end{aligned}$$

**Remark 7.1.1.** *The speed up factor of the derived bound of the number of linear extensions of a random interval order is* $1/4\log_e(2) \approx 0.36$ *– exactly half of the constant stated in Corollary 5.6.6. This fact shows that the bound in this Chapter performs rather poorly, comparing with the results in section 5.6. Note that, with further information about the entropy, improvement might be possible.*

## 7.2 Merging chains using Shellsort

In the last section, we consider the merging of chains. This problem is analysed in the paper of Hwang and Lin [32], where an efficient algorithm is presented.

Here, we discuss some preliminary ideas, that might be worthwhile for further study. Specifically, we propose the application of Shellsort for the merging of linearly ordered sets. Shellsort was invented by Donald Shell [66] in 1959 and is based on insertion sort. The algorithm runs from left to right, by comparing elements at a given gap or increment $d \in \mathbf{N}$ and exchanging them, if they are in reverse order, so in the array $\{a_1, a_2, \ldots, a_n\}$ the $d$ subarrays $\{a_j, a_{j+d}, a_{j+2d}, \ldots\}$, for $j = 1, 2, \ldots, d$ are separately sorted. At the second pass, Shellsort runs on smaller increment, until after a number of passes, the increment becomes $d = 1$. This final insertion sort completes the sorting of the array.

The sequence of the increments is crucial for the running time of the algorithm, as the pivot selection is important to Quicksort. Shell [66] proposed the sequence $\left\lfloor \frac{n}{2} \right\rfloor, \left\lfloor \frac{n}{4} \right\rfloor, \ldots, 1$, which leads to quadratic time. Pratt [57] suggested a sequence of the form $2^a 3^b < n$, where $a, b \in \mathbf{N}$, which yields $\Theta\big(n \log_2^2(n)\big)$ time. Incerpi and Sedgewick [36] have shown that there do exist $\log_a(n)$ increments, for which the running time of the algorithm is $O(n^{1+\frac{\epsilon}{\sqrt{\log_2(n)}}})$, with $a = 2^{\epsilon^2/8}$ and $\epsilon > 0$. Despite the extensive analysis of Shellsort, there are many open problems, as whether the algorithm can achieve on the average $O\big(n \log_2(n)\big)$ run-time.

In our problem, Shellsort can be fruitfully applied to merging chains, which can be done quite fast, using the knowledge of the partial order. Consider two chains $C_1$ and $C_2$,

$$C_1 = \{a_1 < a_2 < \ldots < a_n\}$$
$$C_2 = \{b_1 < b_2 < \ldots < b_m\}.$$

Starting from $a_1$, with initial increment $d = \max\{n, m\}$, the algorithm separately sorts $d$ subarrays. At the second pass, the algorithm iterates from $a_2$ with increment $d-1$. The final comparison of the element occupying the location $d$ with its adjacent key in the position $d+1$, terminates the merging process.

We illustrate this informal idea, with a simple example. Suppose that we want to merge the chains $C_1 = \{5,7,9,11,12\}$ and $C_2 = \{4,6,10\}$. We start with the array $\{5,7,9,11,12,4,6,10\}$ and initial increment $\max\{5,3\} = 5$. Then, the following subarrays are independently sorted: $\{5,4\}$, $\{7,6\}$, $\{9,10\}$, so at the end of the first iteration, the array becomes $\{4,6,9,11,12,5,7,10\}$. Starting from $6$, with increment equal to $4$, the algorithm proceeds to the subarrays $\{6,5\}$, $\{9,7\}$, $\{11,10\}$, so we obtain $\{4,5,7,10,12,6,9,11\}$. In the same manner, starting from $7$, with gap equal to $3$, the subarrays $\{7,6\}$, $\{10,9\}$, $\{12,11\}$ are sorted, giving $\{4,5,6,9,11,7,10,12\}$. Then, the subarrays $\{9,7,12\}$ and $\{11,10\}$ are sorted, yielding $\{4,5,6,7,10,9,11,12\}$. The final comparison to $\{10,9\}$ returns the merged chain. This algorithm took $13$ comparisons for the merging of $8$ keys. A different increment sequence might speed up the process, noting that there are some redundant comparisons, e.g. the comparison of $\{9,7\}$ in the second pass. Its appealing feature is that it merges 'in-place', without the need of auxiliary memory.

Central to the argument is the number of inversions of a permutation of $n$ distinct keys $\{a_1, a_2, \ldots, a_n\}$. An inversion is a pair of elements, such that for $i < j$, $a_i > a_j$. Obviously, an upper bound for the number of inversions is $1+2+\ldots+(n-1) = \binom{n}{2}$. On the other hand, a sorted array has $0$ inversions. In other words, the number of inversions determine the 'amount' of work needed for the complete sorting. Generally, when one has to merge $k$ chains, with cardinalities $m_1, m_2, \ldots, m_k$ and $\sum_{j=1}^{k} m_j = n$, an upper bound to the number of inversions is

$$\binom{n}{2} - \left(\binom{m_1}{2} + \binom{m_2}{2} + \ldots + \binom{m_k}{2}\right).$$

Note that this bound corresponds to the case, where the chains are presented in completely wrong order, e.g. the elements of a chain $C_j$ are greater from the elements of chains, which lie to its right. In practice, this case occurs rarely, so the bound can be greatly improved.

The application of Mergesort, as proposed by Cardinal *et al.* [13] for the merging of chains completes the sorting in $(1+\epsilon)\log_2\bigl(e(P)\bigr) + O(n)$ time – see their Theorem $3$. Cardinal *et al.* remark in their paper that their Mergesort algorithm is better than an earlier one of Kahn and Kim [40], provided $\log_2\bigl(e(P)\bigr)$ is super–linear. As we have seen in this Chapter, $\log_2\bigl(e(P)\bigr)$ is linear, thus the application of Shellsort might constitute an alternative choice for the merging of chains.

# Appendix A

# MAPLE Computations

Here is the MAPLE worksheet for the computation of the variance of the number of key comparisons of dual pivot Quicksort.

```
> restart;

> f_{n}(z):= 1/binomial(n, 2)sum(sum(z^2n-i-2f_{i-1}(z)f_{j-i-1}(z)

  f_{n-j}(z), j=i+1..n), i=1..n-1);

> diff(f_{n}(z), z$2);

> subs(z = 1, diff(f_{n}(z), z$2));
```

$$\begin{aligned} f_n''(1) = \frac{2}{n(n-1)}\Bigg(&\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(2n-i-2)^2 - \sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(2n-i-2) \\ &+ 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(2n-i-2)\mathbb{E}(C_{i-1,2}) + 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(2n-i-2)\mathbb{E}(C_{j-i-1,2}) \\ &+ 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}(2n-i-2)\mathbb{E}(C_{n-j,2}) + 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\mathbb{E}(C_{i-1,2})\mathbb{E}(C_{j-i-1,2}) \\ &+ 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\mathbb{E}(C_{i-1,2})\mathbb{E}(C_{n-j,2}) + 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\mathbb{E}(C_{j-i-1,2})\mathbb{E}(C_{n-j,2}) \\ &+ \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} f_{i-1}''(1) + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} f_{j-i-1}''(1) + \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} f_{n-j}''(1)\Bigg). \end{aligned} \tag{A.1}$$

The following MAPLE commands compute the sums of Eq. (A.1)

```
> sum(sum((2n-i-2)^2, j=i+1..n), i=1..n-1);
> sum(sum((2n-i-2), j=i+1..n), i=1..n-1);
> 2(sum(sum((2n-i-2)(2iharmonic(i-1)-4(i-1)), j = i+1..n), i = 1..n-1));
> simplify(%);
> 2(sum(sum((2n-i-2)((2(j-i))harmonic(j-i-1)-4(j-i-1)),
  j=i+1..n), i=1..n-1));
> sum(4 n harmonic(k)(k+i+1)-4nharmonic(k)i-2iharmonic(k)(k+i+1)
  +2harmonic(k) i^2-4harmonic(k)(k+i+1)+4iharmonic(k), k=0..n-i-1);
> 2(sum(3n-3i+2nharmonic(n-i)(n-i)^2 -iharmonic(n-i)(n-i)
  +2nharmonic(n-i)(n-i)-(3(n-i))n+(3/2(n-i))i-iharmonic(n-i)(n-i)^2
  -2harmonic(n-i) (n-i)^2
  +(1/2)i(n-i)^2-2harmonic(n-i)(n-i)-n (n-i)^2+(n-i)^2, i=1..n-1));
> 2(sum(4nharmonic(j)(n-j)^2-5n^2harmonic(j)(n-j)-2nharmonic(j)
  +(n-j)harmonic(j)n+(2(n-j))harmonic(j)-(n-j)^2
  harmonic(j)+2n^3 harmonic(j)-(n-j)^3 harmonic(j), j=1..n-1));
> sum((2(k+1))harmonic(k)-4k, k = 0..n-i-1);
> 2(sum((2iharmonic(i-1)-4(i-1))(2binomial(n-i+1, 2)
  harmonic(n-i)+(n-i-5(n-i)^2)(1/2)), i=1..n-1));
> sum((8(n-k))binomial(k+1, 2)harmonic(k), k=1..n-1);
> sum(8binomial(k+1, 2)harmonic(k), k=1..n-1);
```

Using these results, the second-order derivative evaluated at $1$ is given recursively by:

$$f_n''(1) = 2(n+1)(n+2)(H_n^2 - H_n^{(2)}) - H_n\left(\frac{17}{3}n^2 + \frac{47}{3}n + 6\right) + \frac{209}{36}n^2 + \frac{731}{36}n + \frac{13}{6} + \frac{6}{n(n-1)}\sum_{i=1}^{n-1}(n-i)f_{i-1}''(1). \tag{A.2}$$

The MAPLE input for the solution of the recurrence is as follows

```
> f''_n(1)=2 (harmonic(n))^2 n^2-17/3 harmonic(n) n^2+209/36 n^2
  -2 harmonic(n,2) n^2+731/36 n-6 harmonic(n,2) n-47/3 harmonic(n) n
  +6 (harmonic(n))^2 n+4 (harmonic(n))^2-4 harmonic(n,2)-6 harmonic(n)
  +13/6+6/(n(n-1))sum((n-i)f''_i-1(1), i=1..n-1);
> binomial(n, 2)f''_{n}(1);
> binomial(n+1, 2)f''_{n+1}(1)- binomial(n, 2)f''_{n}(1);
> binomial(n+2, 2)f''_{n+2}(1)-2binomial(n+1, 2)f''_{n+1}(1)
  + binomial(n, 2)f''_{n}(1);
```

The output of these commands gives the second–order difference

$$\Delta^2\binom{n}{2}f_n''(1) = 12(n+1)(n+2)(H_n^2 - H_n^{(2)}) - H_n(20n^2 + 32n - 12) + 17n^2 + 37n + 3f_n''(1) \tag{A.3}$$

and after standard manipulations of Eq. (A.3), (recall subsection 4.1.1), the solution of the recurrence is

$$f_n''(1) = 4(n+1)^2(H_{n+1}^2 - H_{n+1}^{(2)}) - 4H_{n+1}(n+1)(4n+3) + 23n^2 + 33n + 12.$$

# Bibliography

[1] Abramowitz, M. and Stegun, I. A. (1972) *"Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables."* Dover Publications.

[2] Albacea, E. A. (2012) *"Average-case analysis of Leapfrogging samplesort."* Philipp. Sci. Lett. **5** (1): 14–16.

[3] Albert, M. and Frieze A. (1989) *"Random graph orders."* Order **6** (1): 19–30.

[4] Alon, N., Bollobás, B., Brightwell, G. and Janson, S. (1994) *"Linear Extensions of a Random Partial Order."* Ann. Appl. Probab. **4** (1): 108–123.

[5] Bell, D. A. (1958) *"The Principles of Sorting."* Comput. J. **1** (2): 71–77.

[6] Bentley, J. L. (2000) *"Programming Pearls."* Addison-Wesley Publishing, second edition.

[7] Bentley, J. L. and McIlroy, M. D. (1993) *"Engineering a Sort Function."* Software Pract. Exper. **23** (11): 1249–1265.

[8] Billingsley, P. (2012) *"Probability and measure."* John Wiley & So., third edition.

[9] Boyce, W. E., DiPrima, R. C. (2001) *"Elementary Differential Equations and Boundary Value Problems."* John Wiley & So., seventh edition.

[10] Brightwell, G. (1993) *"Models of Random Partial Orders."* Surveys in Combinatorics, Cambridge University Press.

[11] Brightwell, G., Prömel, H. J. and Steger, A. (1996) *"The Average Number of Linear Extensions of a Partial Order."* J. Comb. Theory, Ser. A **73** (2): 193–206.

[12] Brightwell, G. (1992) *"Random k-dimensional orders: Width and number of Linear extensions."* Order **9** (4): 333–342.

[13] Cardinal, J., Fiorini, S., Joret, G., Jungers, R. and Munro, I. (2010) *"Sorting under partial information (without the Ellipsoid algorithm)."* STOC' 10 Proceedings of the 42nd ACM symposium on Theory of computing, 359–368.

[14] Chern, H. H., Hwang, H. K. and Tsai, T. H. (2002) *"An asymptotic theory for Cauchy–Euler differential equations with applications to the analysis of algorithms."* J. Algorithm. **44** (1): 177–225.

[15] Cormen, T. H. Leiserson, C. E. Rivest, R. L. and Stein, C. (2001) *"Introduction to Algorithms."* M.I.T. Press, second edition.

[16] Cover, T. M. and Thomas, J. A. (1991) *"Elements of Information Theory."* John Wiley & So., second edition.

[17] Dilworth, R. P. (1950) *"A Decomposition Theorem for Partially Ordered Sets."* Ann. Math. **51** (1): 161–166.

[18] Durand, M. (2003) *"Asymptotic analysis of an optimized quicksort algorithm."* Inform. Process. Lett. **85** (2): 73–77.

[19] van Emden, M. H. (1970) *“Increasing the efficiency of Quicksort.”* Comm. ACM. **13** (9): 563–567.

[20] Erkiö, H. (1984) *“The Worst Case Permutation for Median-of-Three Quicksort.”* Comput. J. **27** (3): 276–277.

[21] Feller, W. (1957) *“An Introduction to Probability Theory and its Applications, Vol. II.”* John Wiley & So., first edition.

[22] Fill, J. A. and Janson, S. (2002) *“Quicksort Asymptotics.”* J. Algorithm. **44** (1): 4–28.

[23] Fishburn, P. (1985) *“Interval Orders and Interval Graphs.”* Discrete Math. **55** (2): 135–149.

[24] Frazer, W. D. and McKellar, A. C. (1970) *“Samplesort: A Sampling Approach to minimal storage Tree Sorting.”* J. ACM. **17** (3): 496–507.

[25] Fredman, M. L. (1976) *“How good is the information theory bound in sorting?”* Theor. Comput. Sci. **1** (4): 355–361.

[26] Graham, R. L., Knuth, D. E. and Patashnik, O. (1994) *“Concrete Mathematics: A Foundation for Computer Science.”* Addison-Wesley Publishing, second edition.

[27] Hennequin, P. (1989) *“Combinatorial analysis of quicksort algorithm.”* RAIRO Theor. Inform. Appl. **23** (3): 317–333.

[28] Hennequin, P. (1991) *“Analyse en moyenne d’algorithmes, tri rapide et arbres de recherche.”* Ph.D. thesis. École Polytechnique.

[29] Hoare, C. A. R. (1961) *“Algorithm 63: Partition, Algorithm 64: Quicksort, and Algorithm 65: Find.”* Comm. ACM. **4** (7): 321–322.

[30] Hoare, C. A. R. (1962) *"Quicksort."* Comput. J. **5** (1): 10–15.

[31] Hou, S. H. and Hou, E. (2008) *"Triangular Factors of the Inverse of Vandermonde Matrices."* Proceedings of the International MultiConference of Engineers and Computer Scientists. Vol. II, IMECS.

[32] Hwang, F. K. and Lin, S. (1972) *"A simple algorithm for merging two disjoint linearly ordered sets."* SIAM J. Comput. **1** (1): 31–39.

[33] Iliopoulos, V. and Penman, D. B. (2010) *"Variance of the number of Comparisons of Randomised Quicksort."* **arXiv:** 1006.4063.

[34] Iliopoulos, V. and Penman, D. B. (2012) *"Dual Pivot Quicksort."* Discrete Math. Algorithm. Appl. **4** (3): 1250041.

[35] Iliopoulos, V. (2013) *"Quicksorting on multiple pivots and a Vandermonde matrix."* Seminar in the Department of Mathematical Sciences, University of Essex.

[36] Incerpi, J. and Sedgewick, R. (1985) *"Improved Upper Bounds on Shellsort."* J. Comput. Syst. Sci. **31** (2): 210–224.

[37] Janson, S., Łuczak, T. and Ruciński, A. (2000) *"Random Graphs."* John Wiley & So., first edition.

[38] Jensen, J. L. W. V. (1906) *"Sur les fonctions convexes et les inégalités entre les valeurs moyennes."* Acta Math. **30** (1): 175–193.

[39] Justicz, J., Scheinerman, E. and Winkler, P. (1990) *"Random Intervals."* Amer. Math. Monthly **97** (10): 881–889.

[40] Kahn, J. and Kim, J. H. (1992) *"Entropy and sorting."* STOC' 92 Proceedings of the 24th annual ACM symposium on Theory of computing, 178–187.

[41] Kislitsyn, S. S. (1968) *"A finite partially ordered set and its corresponding set of permutations."* Mat. Zametki **4** (5): 511–518.

[42] Kleitman, D. J. and Rothschild, B. L. (1975) *"Asymptotic enumeration of partial orders on a finite set."* Trans. Amer. Math. Soc. **205**: 205–220.

[43] Knessl, C. and Szpankowski, W. (1999) *"Quicksort algorithm again revisited."* Discrete Math. Theor. Comput. Sci. **3** (2): 43–64.

[44] Knuth, D. E. (1997) *"The Art of Computer Programming, Vol. I: Fundamental Algorithms."* Addison-Wesley Publishing, third edition.

[45] Knuth, D. E. (1997) *"The Art of Computer Programming, Vol. II: Seminumerical Algorithms."* Addison-Wesley Publishing, second edition.

[46] Knuth, D. E. (1998) *"The Art of Computer Programming, Vol. III: Sorting and Searching."* Addison-Wesley Publishing, second edition.

[47] Körner, J. (1973) *"Coding of an Information source having ambiguous alphabet and the entropy of graphs."* Proceedings of the 6th Prague Conference on Information Theory, 411–425.

[48] Mahmoud, H. M. (2010) *"Distributional analysis of swaps in Quick Select."* Theor. Comput. Sci. **411** (16-18): 1763–1769.

[49] Mahmoud, H. M. (2000) *"Sorting: A Distribution Theory."* John Wiley & So., first edition.

[50] McDiarmid, C. J. H. and Hayward, R. B. (1996) *“Large Deviations for Quicksort.”* J. Algorithm. **21** (3): 476–507.

[51] McDiarmid, C. J. H. (2013) *“Quicksort and Large Deviations.”* Mathematical and Engineering Methods in Computer Science, Lecture Notes in Computer Science **7721:** 43–52.

[52] McDiarmid, C. J. H. (2013) *Personal Communications.*

[53] McEliece, R. (2002) *“The theory of Information and Coding: A Mathematical Framework for Communication.”* University Cambridge Press, second edition.

[54] Mitzenmacher, M. and Upfal, E. (2005) *“Probability and Computing: Randomized Algorithms and Probabilistic Analysis.”* Cambridge University Press.

[55] Mizoi, T. and Osaki, S. (1996) *“Probabilistic Analysis of Time Complexity of Quicksort.”* Electron. Comm. Jpn. Pt. III **79** (3): 34–42.

[56] Neggers, J. and Kim, H. S. (1998) *“Basic Posets.”* World Scientific Publishing.

[57] Pratt, V. (1979) *“Shellsort and Sorting Networks.”* Ph.D. thesis. Garland Publishing.

[58] Régnier, M. (1989) *“A limiting distribution for Quicksort.”* RAIRO Theor. Inform. Appl. **23** (3): 335–343.

[59] Rösler, U. (1991) *“A Limit Theorem for Quicksort.”* RAIRO Theor. Inform. Appl. **25** (1): 85–100.

[60] Rousseeuw, P. J. and Bassett, G. W. (1990) *"The remedian: A robust averaging method for large data sets."* J. Am. Statist. Assoc. **85** (409): 97–104.

[61] Sabuwala, A. H. and De Leon, D. (2011) *"Particular solution to the Euler-Cauchy equation with polynomial non-homogeneities."* Discret. Contin. Dyn. S. **2011:** 1271–1278.

[62] Scowen, R. S. (1965) *"Algorithm 271: Quickersort."* Comm. ACM. **8** (11): 669–670.

[63] Sedgewick, R. (1980) *"Quicksort."* Ph.D. thesis. Garland Publishing.

[64] Sedgewick, R. (1977) *"Quicksort with Equal Keys."* SIAM J. Comput. **6** (2): 240–267.

[65] Shannon, C. E. (1948) *"A Mathematical Theory of Communication."* Bell Syst. Tech. J. **27** (3): 379–423.

[66] Shell, D. L. (1959) *"A High-Speed Sorting Procedure."* Comm. ACM. **2** (7): 30–32.

[67] Simonyi, G. (1995) *"Graph Entropy: A Survey."* In Combinatorial Optimization (Ed. W. Cook, L. Lovász, and P. Seymour). DIMACS Series in Discrete Mathematics and Theoretical Computer Science. Amer. Math. Soc., 399–441.

[68] Singleton, R. C. (1969) *"Algorithm 347: An efficient algorithm for sorting with minimal storage [M1]."* Comm. ACM. **12** (3): 185–186.

[69] Skiena, S. S. (2008) *"The algorithm design manual."* Springer, second edition.

[70] Spieß, J. (1990) *"Some Identities involving Harmonic numbers."* Math. Comp. **55** (192): 839–863.

[71] Trotter, W. (1997) *"New perspectives of Interval Orders and Interval Graphs."* Surveys in Combinatorics, Cambridge University Press.

[72] Tukey, J. W. (1978) *"The ninther, a technique for low-effort robust (resistant) location in large samples."* In Contributions to Survey Sampling and Applied Statistics in Honor of H. O. Hartley, Ed. by H. A. David. 251–258, Academic Press.

[73] Turner, L. R. (1966) *"Inverse of the Vandermonde Matrix with Applications."* National Aeronautics and Space Administration, technical note D-3547.

[74] Whittaker, E. T. and Watson, G. N. (1996) *"A Course of Modern Analysis."* Cambridge University Press, fourth edition.