Multi-Pivot Quicksort: Theory and Experiments

Shrinu Kushagra
skushagr@uwaterloo.ca

Alejandro Lopez-Ortiz
alopez-o@uwaterloo.ca

J. Ian Munro
imunro@uwaterloo.ca

Aurick Qiao
a2qiao@uwaterloo.ca

David R. Cheriton School of Computer Science
University of Waterloo

February 7, 2014

Background

- Quicksort was introduced by C.A.R. Hoare in 1960.
- Divide and conquer algorithm

procedure $\textsc{Quicksort}$(Array A)
    pivot $\leftarrow$ arbitrary element in A
    partition A into elements $\leq$ and $>$ pivot
            // hope that parts are about the same size
    Quicksort($\leq$ part of A)
    Quicksort($>$ part of A)
end procedure

Background

- Worst case running time is $\Theta(n^2)$

Background

- Worst case running time is $\Theta(n^2)$
- But happens very rarely

Background

- Worst case running time is $\Theta(n^2)$
- But happens very rarely
- How rarely?

Background

- Worst case running time is $\Theta(n^2)$
- But happens very rarely
- How rarely?
- Sort 100 random numbers using quicksort

Background

- Worst case running time is $\Theta(n^2)$
- But happens very rarely
- How rarely?
- Sort 100 random numbers using quicksort
- Repeat 1 billion times

Background

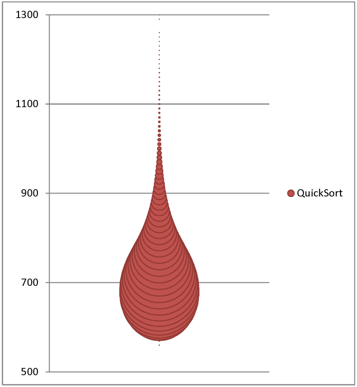
Fastest possible time $n \lg n = 100 \lg 100 \approx 600$ steps

Background

Fastest possible time $n \lg n = 100 \lg 100 \approx 600$ steps
Slowest possible time $n(n-1)/2 = 100(100-1)/2 \approx 5000$ steps

Background

Fastest possible time $n \lg n = 100 \lg 100 \approx 600$ steps
Slowest possible time $n(n-1)/2 = 100(100-1)/2 \approx 5000$ steps



Background

- Expected running time is $\Theta(n \log n)$

Background

- Expected running time is $\Theta(n \log n)$
- Very fast running time in practice $\approx 2n \ln n + O(n)$

Background

- Expected running time is $\Theta(n \log n)$
- Very fast running time in practice $\approx 2n \ln n + O(n)$
- Extensively studied by Bob Sedgewick in his 1975 PhD thesis

Background

- Expected running time is $\Theta(n \log n)$
- Very fast running time in practice $\approx 2n \ln n + O(n)$
- Extensively studied by Bob Sedgewick in his 1975 PhD thesis
- Key issue: obtain a good partition

Background

- Expected running time is $\Theta(n \log n)$
- Very fast running time in practice $\approx 2n \ln n + O(n)$
- Extensively studied by Bob Sedgewick in his 1975 PhD thesis
- Key issue: obtain a good partition
- I.e. we need a "good" pivot

Background

- Expected running time is $\Theta(n \log n)$
- Very fast running time in practice $\approx 2n \ln n + O(n)$
- Extensively studied by Bob Sedgewick in his 1975 PhD thesis
- Key issue: obtain a good partition
- I.e. we need a "good" pivot
- Use median-of-three strategy: select three items, sort them, use the one in the middle as pivot

Background

- Expected running time is $\Theta(n \log n)$
- Very fast running time in practice $\approx 2n \ln n + O(n)$
- Extensively studied by Bob Sedgewick in his 1975 PhD thesis
- Key issue: obtain a good partition
- I.e. we need a "good" pivot
- Use median-of-three strategy: select three items, sort them, use the one in the middle as pivot
- Optimal, ultimate quicksort introduced by Sedgewick in 1978

BREAKING NEWS!

BREAKING NEWS!

In 2009, Vladimir Yaroslavskiy proposed a new quicksort variant using a dual-pivot partitioning scheme.

BREAKING NEWS!

In 2009, Vladimir Yaroslavskiy proposed a new quicksort variant using a dual-pivot partitioning scheme.

- Outperforms classic quicksort under the Java JVM by close to 10%.

BREAKING NEWS!

In 2009, Vladimir Yaroslavskiy proposed a new quicksort variant using a dual-pivot partitioning scheme.

- Outperforms classic quicksort under the Java JVM by close to 10%.
- Replaced Java's internal sorting algorithm in Java 7.

BREAKING NEWS!

In 2009, Vladimir Yaroslavskiy proposed a new quicksort variant using a dual-pivot partitioning scheme.

- Outperforms classic quicksort under the Java JVM by close to 10%.
- Replaced Java's internal sorting algorithm in Java 7.

This contradicts prior work (especially Sedgewick 1977) showing that using multiple pivots is an inferior strategy!

Analysis of Yaroslavskiy

Wild and Nebel, Uni-Kaiserslautern (ESA 2012, best paper award) provided a rigorous average-case analysis of Yaroslavskiy's quicksort.

Analysis of Yaroslavskiy

Wild and Nebel, Uni-Kaiserslautern (ESA 2012, best paper award) provided a rigorous average-case analysis of Yaroslavskiy's quicksort.

- An implementation detail in the partitioning process cuts down the number of comparisons from Sedgewick's approach.

Analysis of Yaroslavskiy

Wild and Nebel, Uni-Kaiserslautern (ESA 2012, best paper award) provided a rigorous average-case analysis of Yaroslavskiy's quicksort.

- An implementation detail in the partitioning process cuts down the number of comparisons from Sedgewick's approach.
- Yaroslavskiy's quicksort uses on average $1.9n \ln n - 2.46n + O(\ln n)$ comparisons.

Analysis of Yaroslavskiy

Wild and Nebel, Uni-Kaiserslautern (ESA 2012, best paper award) provided a rigorous average-case analysis of Yaroslavskiy's quicksort.

- An implementation detail in the partitioning process cuts down the number of comparisons from Sedgewick's approach.
- Yaroslavskiy's quicksort uses on average $1.9n \ln n - 2.46n + O(\ln n)$ comparisons.
- Classic quicksort uses on average $2.0n \ln n - 1.51n + O(\ln n)$ comparisons.

Analysis of Yaroslavskiy

Aumüller and Dietzfelbinger, Uni-Ilmenau (ICALP 2013) captured all dual-pivot quicksort schemes in a single model.

Analysis of Yaroslavskiy

Aumüller and Dietzfelbinger, Uni-Ilmenau (ICALP 2013) captured all dual-pivot quicksort schemes in a single model.

- Lower bound of $1.8 n \ln n + O(n)$ comparisons for *all* dual-pivot quicksort algorithms.

Analysis of Yaroslavskiy

Aumüller and Dietzfelbinger, Uni-Ilmenau (ICALP 2013) captured all dual-pivot quicksort schemes in a single model.

- Lower bound of $1.8n \ln n + O(n)$ comparisons for *all* dual-pivot quicksort algorithms.
- Presented an implementation of dual-pivot quicksort achieving that bound.

Analysis of Yaroslavskiy

Aumüller and Dietzfelbinger, Uni-Ilmenau (ICALP 2013) captured all dual-pivot quicksort schemes in a single model.

- Lower bound of $1.8n \ln n + O(n)$ comparisons for *all* dual-pivot quicksort algorithms.
- Presented an implementation of dual-pivot quicksort achieving that bound.
- 3% slower than Yaroslavskiy's algorithm on integer data.

Analysis of Yaroslavskiy

Aumüller and Dietzfelbinger, Uni-Ilmenau (ICALP 2013) captured all dual-pivot quicksort schemes in a single model.

- Lower bound of $1.8n \ln n + O(n)$ comparisons for *all* dual-pivot quicksort algorithms.
- Presented an implementation of dual-pivot quicksort achieving that bound.
- 3% slower than Yaroslavskiy's algorithm on integer data.
- 2% faster than Yaroslavskiy's algorithm on strings.

Analysis of Yaroslavskiy

Yaroslavskiy's quicksort uses 5-8% fewer comparisons but achieves more than a 10% performance gain.

- Another factor must be contributing to its performance.

There is a disparity between theory and what is observed in practice.

Our Work

We make several contributions to the topic:

Our Work

We make several contributions to the topic:

1. Confirm experimental results in C, removing potential artifacts introduced by the JVM.

Our Work

We make several contributions to the topic:

1. Confirm experimental results in C, removing potential artifacts introduced by the JVM.
2. Describe a quicksort variant using three pivots that (in our experiments) outperforms Yaroslavskiy's quicksort.

Our Work

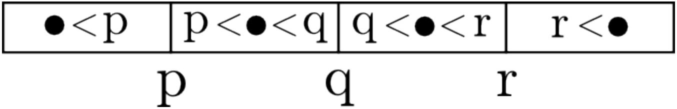
We make several contributions to the topic:

1. Confirm experimental results in C, removing potential artifacts introduced by the JVM.
2. Describe a quicksort variant using three pivots that (in our experiments) outperforms Yaroslavskiy's quicksort.
3. Propose **cache behavior** as an explanation for the performance of multi-pivot quicksort algorithms.

3-Pivot Quicksort

Intuitively the same as classic quicksort:

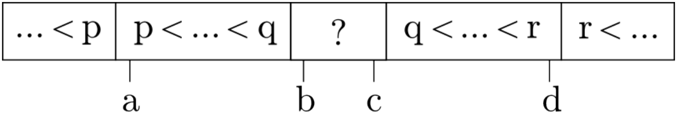
- Choose three elements as pivots and partition the array around them.
- Recursively sort the subarrays defined by the pivots.



3-Pivot Partition

Use four pointers *a*, *b*, *c*, and *d*.

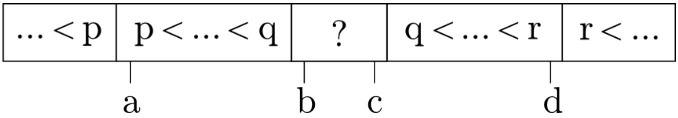
- Initialize *a* and *b* to the beginning of the array and *c* and *d* to the end of the array.
- Advance pointers *b* and *c* toward each other while maintaining the invariant shown in the figure.
- End when *b* and *c* cross each other.



3-Pivot Partition

In order to maintain the invariant, we must swap each new element into place.

1. Keep advancing *b* while the element is less than *q*, swapping it into place with the element at *a* or leaving it alone. Keep advancing *c* in the same way.
2. Now both elements at *b* and *c* must go into "opposite" sides of the array. Swap them into place according to the four cases.
3. Repeat.



Comparisons and Swaps

The standard method of analysis by solving recurrences gives the average number of comparisons and swaps for the 3-pivot quicksort:

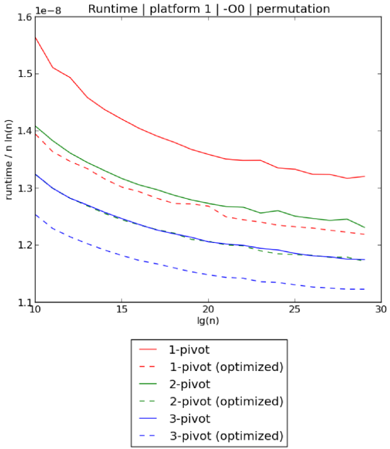
- $\approx 1.846 n \ln n + O(n)$ comparisons
- $\approx 0.615 n \ln n + O(n)$ swaps

Experimental Results

Experiments were run on the following algorithms:

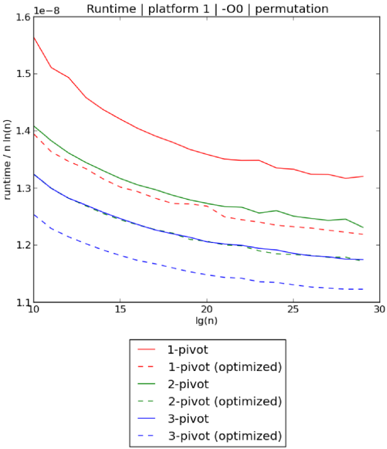
- Classic 1-pivot quicksort.
- 1-pivot quicksort using median of 3 pivot selection.
- Yaroslavskiy's 2-pivot quicksort.
- 2-pivot quicksort using 2nd and 4th of 5 pivot selection.
- Our 3-pivot quicksort.
- 3-pivot quicksort using 2nd, 4th and 6th of 7 pivot selection.

Experimental Results



- Yaroslavskiy's algorithm performs just as well written in C, confirming previous experimental results.

Experimental Results



- Yaroslavskiy's algorithm performs just as well written in C, confirming previous experimental results.
- The 3-pivot algorithm performs especially well under this setup, and mostly outperforms the other variants under multiple rigorous tests.

Experimental Results

Interesting observation:

- 3-pivot quicksort outperforms median-of-3 1-pivot quicksort.
- **Comparisons**: $1.85n \ln n$ vs. $1.71n \ln n$
- **Swaps**: $0.62n \ln n$ vs. $0.34n \ln n$

3-pivot quicksort uses **more** comparisons and **more** swaps but has **better** performance.

This further suggests the presence of another factor contributing to performance.

Cache Behavior Analysis

Method used:

1. Count the number of cache misses incurred by a single partition step for any three pivots.
2. Define a recurrence based on the recursion of the quicksort being analyzed.
3. Use symbolic math package to solve the recurrence and manually simplify the expression.

Cache Behavior Analysis – Results

Let M be the size of the cache and B be the size of each block of cache.

1-Pivot Quicksort: $2\left(\frac{n+1}{B}\right)\ln\left(\frac{n+1}{M+2}\right)+O(\frac{n}{B})$

2-Pivot Quicksort: $\frac{8}{5}\left(\frac{n+1}{B}\right)\ln\left(\frac{n+1}{M+2}\right)+O(\frac{n}{B})$

Leading constants of 2 and 1.6 for cache faults versus 2 and 1.9 for comparisons.

Cache Behavior Analysis – Results

More interestingly, the results for 3-pivot quicksort compared with median-of-3 1-pivot quicksort:

3-Pivot Quicksort: $\frac{18}{13}\left(\frac{n+1}{B}\right)\ln\left(\frac{n+1}{M+2}\right) + O(\frac{n}{B})$

Median-of-3 Quicksort: $\frac{12}{7}\left(\frac{n+1}{B}\right)\ln\left(\frac{n+1}{M+2}\right) + O(\frac{n}{B})$

Leading constant of $\sim \mathbf{1.38}$ for 3-pivot quicksort and $\sim \mathbf{1.71}$ for median-of-3 quicksort.

Cache Behavior Experiments

Experiments using valgrind tool cachegrind reinforces the cache analyses.

Sorting $10,000,000$ integers:

- **1-pivot**: $\sim 3,700,000$ cache misses
- **2-pivot**: $\sim 3,100,000$ cache misses
- **3-pivot**: $\sim 2,700,000$ cache misses

Conclusion

- We have confirmed that multi-pivot quicksort schemes outperform classic quicksort.
- Cache behavior explains the performance differences seen in practice.
- Fastest quicksort

Conclusion

- We have confirmed that multi-pivot quicksort schemes outperform classic quicksort.
- Cache behavior explains the performance differences seen in practice.
- Fastest quicksort ...yet.

Conclusion

The number of layers of cache seems to be constantly increasing in hardware. This means:

- Cache effect are constantly becoming more pronounced.
- Past performance results may no longer be valid in modern architecture.
- Present results may change in the future.

Future Work

Future work regarding multi-pivot quicksort may be directed toward:

- Experimentation on different caching architectures.
- Exploiting caches in more complex ways.

Thank you!