Average-Case and Distributional Analysis of Java 7’s Dual Pivot Quicksort

Markus E. Nebel

based on joint work with Ralph Neininger and Sebastian Wild

AofA 2013
Menorca, Spain
Many inventions by algorithms community vs. Few methods successful in practice

- C
- C++
- Java 6
- .NET
- Haskell
- Python

Quicksort + Mergesort variant as stable sort

Sorting methods listed on Wikipedia
Sorting methods of standard libraries for random access data
Many inventions by algorithms community vs. Few methods successful in practice

- C
- C++
- Java 6
- .NET
- Haskell
- Python

Quicksort
+ Mergesort variant as stable sort

Sorting methods listed on Wikipedia

Sorting methods of standard libraries for random access data
History of Quicksort in Practice

- **1961,62 Hoare**: first publication, average case analysis
- **1969 Singleton**: median-of-three & Insertionsort on small subarrays
- **1975-78 Sedgewick**: detailed analysis of many optimizations
- **1993 Bentley, McIlroy**: *Engineering a Sort Function*
- **1997 Musser**: $\mathcal{O}(n \log n)$ worst case by bounded recursion depth

Basic algorithm settled since 1961; latest tweaks from 1990’s. Since then: Almost identical in all programming libraries!
History of Quicksort in Practice

- **1961,62 Hoare:** first publication, average case analysis
- **1969 Singleton:** median-of-three & Insertionsort on small subarrays
- **1975-78 Sedgewick:** detailed analysis of many optimizations
- **1993 Bentley, McIlroy:** *Engineering a Sort Function*
- **1997 Musser:** $O(n \log n)$ worst case by bounded recursion depth

~ Basic algorithm settled since 1961; latest tweaks from 1990’s.

**Since then:** Almost identical in all programming libraries!

---

History of Quicksort in Practice

- **1961,62 Hoare**: first publication, average case analysis
- **1969 Singleton**: median-of-three & Insertionsort on small subarrays
- **1975-78 Sedgewick**: detailed analysis of many optimizations
- **1993 Bentley, McIlroy**: *Engineering a Sort Function*
- **1997 Musser**: $O(n \log n)$ worst case by bounded recursion depth

Basic algorithm settled since 1961; latest tweaks from 1990’s.

Since then: Almost identical in all programming libraries!

- **Until 2009**: Java 7 switches to a new dual pivot Quicksort!

Sept. 2009 Vladimir Yaroslavski announced algorithm on Java core library mailing list ~ July 2011 public release of Java 7 with Yaroslavski’s Quicksort.
Why switch to new, unknown algorithm?

Normalized Java runtimes (in ms).
Average and standard deviation of 1000 random permutations per size.

$n \cdot \ln n$
Why switch to new, unknown algorithm? Because it is faster!

Normalized Java runtimes (in ms). Average and standard deviation of 1000 random permutations per size.
Running Time Experiments

Why switch to new, unknown algorithm? Because it is faster!

![Graph showing normalized Java runtimes (in ms). Average and standard deviation of 1000 random permutations per size.]

-remains true for basic variants of algorithms: -○- vs. -□-!
Dual Pivot Quicksort

High Level Algorithm:

1. Partition array around two pivots \( p \leq q \).
2. Sort 3 subarrays recursively.

How to do partitioning?
Dual Pivot Quicksort

High Level Algorithm:

1. Partition array around two pivots $p \leq q$.
2. Sort 3 subarrays recursively.

How to do partitioning?

1. For each element $x$, determine its class
   - small for $x < p$
   - medium for $p < x < q$
   - large for $q < x$
   by comparing $x$ to $p$ and/or $q$

2. Arrange elements according to classes

Markus E. Nebel
Dual Pivot Quicksort – Previous Work

- **Robert Sedgewick, 1975**
  - in-place dual pivot Quicksort implementation
  - more comparisons and swaps than classic Quicksort

- **Pascal Hennequin, 1991**
  - comparisons for list-based Quicksort with $r$ pivots
  - $r = 2 \Rightarrow$ same #comparisons as classic Quicksort
    - in one partitioning step: $\frac{5}{3}$ comparisons per element
  - $r > 2 \Rightarrow$ very small savings, but complicated partitioning
Dual Pivot Quicksort – Previous Work

- Robert Sedgewick, 1975
  - in-place dual pivot Quicksort implementation
  - more comparisons and swaps than classic Quicksort

- Pascal Hennequin, 1991
  - comparisons for list-based Quicksort with $r$ pivots
  - $r = 2 \rightsquigarrow$ same #comparisons as classic Quicksort in one partitioning step: $\frac{5}{3}$ comparisons per element
  - $r > 2 \rightsquigarrow$ very small savings, but complicated partitioning

$\rightsquigarrow$ Using two pivots does not pay, and ...

... no theoretical explanation for impressive speedup.
How many comparisons to determine classes (small, medium or large)?

- Assume, we first compare with $p$.
  $\sim$ small elements need 1, others 2 comparisons

- on average: $\frac{1}{3}$ of all elements are small
  $\sim \frac{1}{3} \cdot 1 + \frac{2}{3} \cdot 2 = \frac{5}{3}$ comparisons per element

- if inputs are uniform random permutations, classes of $x$ and $y$ are independent

- $\sim$ Any partitioning method needs at least
  $\frac{5}{3} (n - 2) \sim \frac{20}{12} n$ comparisons on average?
Dual Pivot Quicksort – Comparisons

How many comparisons to determine classes (small, medium or large)?

- Assume, we first compare with \( p \).
  - small elements need 1, others 2 comparisons

- on average: \( \frac{1}{3} \) of all elements are small
  - \( \frac{1}{3} \cdot 1 + \frac{2}{3} \cdot 2 = \frac{5}{3} \) comparisons per element

- if inputs are uniform random permutations, classes of \( x \) and \( y \) are independent

- Any partitioning method needs at least
  - \( \frac{5}{3} (n - 2) \approx \frac{20}{12} n \) comparisons on average?

- No! (Stay tuned . . . )
Beating the “Lower Bound”

- \( \sim \frac{20}{12} n \) comparisons only needed, if there is one comparison location (giving rise to fixed order like first compare with \( p \)), then checks for \( x \) and \( y \) independent

- **But:** Can have several comparison locations!

  Here: Assume two locations \( C_1 \) and \( C_2 \) s.t.

  - \( C_1 \) first compares with \( p \).
  - \( C_2 \) first compares with \( q \).
  - \( C_1 \) executed often, iff \( p \) is large.
  - \( C_2 \) executed often, iff \( q \) is small.

- \( \sim \) \( C_1 \) executed often
  - iff many small elements
    - iff good chance that \( C_1 \) needs only one comparison
      (\( C_2 \) similar)

- \( \sim \) less comparisons than \( \frac{5}{3} \) per elements on average
Yaroslavskiy’s Quicksort

while \( k \leq g \)

\[ C_k \]
if \( A[k] < p \)

Swap \( A[k] \) and \( A[\ell] \) ; \( \ell := \ell + 1 \)

\[ C'_k \]
else if \( A[k] \geq q \)

\[ C_g \]
while \( A[g] > q \) and \( k < g \) do \( g := g - 1 \) end while

Swap \( A[k] \) and \( A[g] \) ; \( g := g - 1 \)

\[ C'_g \]
if \( A[k] < p \)

Swap \( A[k] \) and \( A[\ell] \) ; \( \ell := \ell + 1 \)

end if

end if

\( k := k + 1 \)

end while

Invariant:

\[ \begin{array}{cccccccc}
\text{\(< p\)} & \ell & \text{\(p \leq \circ \leq q\)} & k & \text{\(?\)} & g & \text{\(> q\)} \\
\rightarrow & & & & & & & \\
\end{array} \]
Analysis of YaroslavskiŁy’s Algorithm

In this talk:

* leading term asymptotics of comparisons
  (we have results for swaps and Java bytecodes too)
* distribution and correlation of costs
* effect of pivot sampling

\[ C_n \] expected number of comparisons to sort random permutation of \( \{1, \ldots, n\} \)

\[ C_n \] satisfies recurrence relation

\[
C_n = c_n + \frac{2}{n(n-1)} \sum_{1 \leq p < q \leq n} \left( C_{p-1} + C_{q-p-1} + C_{n-q} \right),
\]

with \( c_n \) expected number of comparisons in \textbf{first} partitioning step

recurrence solvable by standard methods

\[ \sim \textbf{linear} \quad c_n \sim a \cdot n \quad \text{yields} \quad C_n \sim \frac{6}{5} a \cdot n \ln n. \]

\[ \sim \text{need to compute} \quad c_n \]
Analysis of Yaroslavskiy’s Algorithm

- **first** comparison for all elements (at $C_k$ or $C_g$)
  $\sim \sim n$ comparisons

- **second** comparison for some elements at $C'_k$ resp. $C'_g$
  ... but how often are $C'_k$ resp. $C'_g$ reached?

- $C'_k$: all non-small elements reached by pointer $k$.
- $C'_g$: all non-large elements reached by pointer $g$.

- second comparison for **medium** elements **not avoidable**
  $\sim \sim \frac{1}{3} n$ comparisons in expectation

- $\sim$ it remains to count:
  - large elements reached by $k$ and
  - small elements reached by $g$. 
Analysis of Yaroslavskiy’s Algorithm

- **Second** comparisons for small and large elements? Depends on **location**!

- \( C'_k \sim l @ K \): number of large elements at positions \( K \).
- \( C'_g \sim s @ G \): number of small elements at positions \( G \).

1. Recall invariant:

\[
\begin{array}{cccccc}
< p & \ell & p & \leq \circ & \leq q & k & ? & g & > q \\
\rightarrow & & \rightarrow & & \leftarrow & \\
\end{array}
\]

\( k \) and \( g \) cross at (rank of) \( q \)

\[
\begin{align*}
& l @ K = 3 \\
& s @ G = 2
\end{align*}
\]

positions \( K = \{2, \ldots, q - 1\} \)

positions \( G = \{q, \ldots, n - 1\} \)

\( |K| \sim q \), \( |G| \sim n - q \)
Assume \( p \) and \( q \) are fixed.

2. How many **small** and **large** elements?

- \(#\text{small} = |\{1, \ldots, p - 1\}| = p - 1\)
- \(#\text{large} = |\{q + 1, \ldots, n\}| = n - q\)

\(|K|, |G|, #\text{small} \text{ and } #\text{large} \text{ are constant} \text{ (for given } p \text{ and } q).\)

But: \( l @ K \) and \( s @ G \) are random even for fixed \( p \) and \( q \).
Distribution of \( l @ K \) and \( s @ G \)

Assume \( p \) and \( q \) are fixed.

How many small and large elements?

- \( \#\text{small} = |\{1, \ldots, p - 1\}| = p - 1 \)
- \( \#\text{large} = |\{q + 1, \ldots, n\}| = n - q \)

\( \sim |K|, |G|, \#\text{small} \text{ and } \#\text{large} \) are constant (for given \( p \) and \( q \)).

But: \( l @ K \) and \( s @ G \) are random even for fixed \( p \) and \( q \).
Conditional Distribution of $l@K$

- We draw positions of large elements at random.
- $n - 2$ positions ($\equiv$ urn with $n - 2$ balls)
- draw #large positions without replacement ($\equiv$ number of draws is #large)
- $|K|$ positions contribute to $l@K$ ($\equiv$ maximum number of successes is $|K|$)

$\sim l@K \sim \text{Hypergeometric} (#\text{large}, |K|, n - 2)$

- $E[l@K \mid p, q] = \frac{#\text{large} \cdot |K|}{n - 2} \sim \frac{(n - q) \cdot q}{n - 2}$
Conditional Distribution of \( l@K \)

- We draw positions of large elements at random.
- \( n - 2 \) positions (\( \equiv \) urn with \( n - 2 \) balls)
- draw \#large positions without replacement (\( \equiv \) number of draws is \#large)
- \( |K| \) positions contribute to \( l@K \) (\( \equiv \) maximum number of successes is \( |K| \))

\[ l@K \sim \text{Hypergeometric} (\#\text{large}, |K|, n - 2) \]

\[ \mathbb{E} [l@K \mid p, q] = \frac{\#\text{large} \cdot |K|}{n - 2} \sim \frac{(n - q) \cdot q}{n - 2} \]
Analysis of Yaroslavskiy’s Algorithm

- **law of total expectation:**

\[
\mathbb{E}[l @ K] = \sum_{1 \leq p < q \leq n} \Pr[pivots(p, q)] \cdot (n - q) \frac{q-2}{n-2} \sim \frac{1}{6}n
\]

- **Similarly:** \( \mathbb{E}[s @ G] \sim \frac{1}{12}n \).

- **Summing up contributions:**

\[
\begin{align*}
    c_n & \sim n \\
    & + \frac{1}{3}n \quad \text{medium elements} \\
    & + \frac{1}{6}n \quad \text{large elements at } C'_k \\
    & + \frac{1}{12}n \quad \text{small elements at } C'_g \\
    = & \quad \frac{19}{12}n
\end{align*}
\]

- **Recall:** “lower bound” was \( \frac{20}{12}n \).
The **contraction method** can be used to show

**Theorem**

For the number \( C_n \) of key comparisons used by Yaroslavskiy’s Quicksort when operating on a uniformly at random distributed permutation we have

\[
\frac{C_n - \mathbb{E}[C_n]}{n} \to C^*, \quad (n \to \infty),
\]

where the convergence is in distribution and with second moments. The distribution of \( C^* \) is determined as the unique fixed point, subject to \( \mathbb{E}[X] = 0 \) and \( \mathbb{E}[X^2] < \infty \), of

\[
X \overset{p}{=} 1 + (D_1 + D_2)(D_2 + 2D_3) + \sum_{j=1}^{3} \left( D_j X^{(j)} + \frac{19}{10} D_j \ln D_j \right),
\]

where \((D_1, D_2, D_3), X^{(1)}, X^{(2)} \) and \( X^{(3)} \) are independent and \( X^{(j)} \) has the same distribution as \( X \) for \( j \in \{1, 2, 3\} \). Moreover, we have, as \( n \to \infty \),

\[
\text{Var}(C_n) \sim \sigma^2_C n^2 \quad \text{with} \quad \sigma^2_C = \frac{2231}{360} - \frac{361}{600} \pi^2 = 0.25901\ldots
\]
Distribution of costs

Exact distribution \( \frac{C_n}{n} \),

and \( \frac{C_n - \mathbb{E}[C_n]}{n}, n = 5 \ldots 25 \).
Covariance between comparisons and swaps

**Theorem**

For the number \( C_n \) of key comparisons and the number \( S_n \) of swaps used by Yaroslavskiy’s algorithm on a random permutation, we have for \( n \to \infty \)

\[
\text{Cov}(C_n, S_n) \sim \sigma_{C,S} n^2 \quad \text{with} \quad \sigma_{C,S} = \frac{28}{15} - \frac{19}{100} \pi^2 = -0.00855817\ldots
\]

The **correlation coefficient** of \( C_n \) and \( S_n \) is consequently

\[
\rho = \frac{\text{Cov}(C_n, S_n)}{\sqrt{\text{Var}(C_n)} \sqrt{\text{Var}(S_n)}} \approx -0.0512112\ldots
\]

**Remark:** Note the changed behavior compared to classic quicksort where a **strong** negative correlation (\(-0.86404\)) is observed!
Pivot Sampling

- Black (white) dot shows optimal (symmetric) choice for the pivots (exact order statistics $\Delta k \to \infty$);
- dashed black (dotted white) lines represent “equi-cost-ant” to optimum (equidistant from symmetric) pivot choices;
- a smaller sample together with optimal choice can beat symmetric choice for larger sample (in number of Java bytecodes).

$\alpha_1 (\alpha_2)$ the percentage of small (medium) elements.
Pivot Sampling

- Black (white) dot shows **optimal** (symmetric) choice for the pivots (exact order statistics $\Delta k \rightarrow \infty$);
- dashed black (dotted white) lines represent "equi-cost-ant" to optimum (equidistant from symmetric) pivot choices;
- a smaller sample together with optimal choice can **beat symmetric choice for larger sample** (in number of Java bytecodes).

$\alpha_1$ ($\alpha_2$) the percentage of small (medium) elements.
Discussion

- **Comparisons:**
  
  - Yaroslavskiy needs $\sim \frac{6}{5} \cdot \frac{19}{12} n \ln n = 1.9 n \ln n$ on average.
  
  - Classic Quicksort needs $\sim 2 n \ln n$ comparisons!
Discussion – we did not really succeed

- **Comparisons:**
  - Yaroslavskiy needs $\sim \frac{6}{5} \cdot \frac{19}{12} n \ln n = 1.9 n \ln n$ on average.
  - Classic Quicksort needs $\sim 2 n \ln n$ comparisons!

- **Swaps:**
  - $\sim 0.6 n \ln n$ swaps for Yaroslavskiy’s algorithm vs.
  - $\sim 0.3 n \ln n$ swaps for classic Quicksort

- **Bytecodes:**
  - $\sim 21.7 n \ln(n) - 3.56319n$ Java bytecodes for Yaroslavskiy’s algorithm vs.
  - $\sim 18 n \ln(n) + 6.21488n$ Java bytecodes for classic Quicksort
Conclusion

Yaroslavskiy’s quicksort is a perfect textbook example to demonstrate
- how well methods from AofA are developed;
- the depth of results obtainable (precise expectations, distributions, covariances, ...) by those methods;
- how AofA can guide engineering of an algorithm (pivot sampling, switch to insertionsort, ...).

However, our sophisticated machinery fails to explain the practical efficiency of Yaroslavskiy’s algorithms; (presumably) it would be important to get access to
- branch mispredictions and/or
- cache misses.
Many thanks for your attention!