- $O(n \log(n))$ Average Case Run Time
- In place algorithm
- Pick Pivots
- Partitioning Data
- Recurse to a smaller sub-array
- Use Insertion Sort for a small sub-array

Data is partitioned By Comparing and Swapping elements. These will be our metrics.

- $2.13n \log n - 2.57n + O(\log(n))$ Comparisons
- $0.8n \log n - 0.3n + O(\log(n))$ Swaps
- Two Pivots
  - First and Last Element
  - Middle 2 of 5 elements (Evenly Spaced Out)
- Partitions Smalls then Bigs (Middle is automatic)
- Three Recursive Calls

[?]

Tertile Elements = Middle 2 of 5 elements (Evenly Spaced Out)

- $1.9n \log n - 2.46n + O(\log(n))$ Comparisons
- $0.6n \log n + 0.08n + O(\log(n))$ Swaps
- Two Pivots
  - Middle 2 of 5 elements (Evenly Spaced Out)
  - Uses 5-element sorting network
- Simultaneous Partition algorithm
- Two Recursive Calls

[?]

It turns out that Yaroslavskiys partitioning method is able to take advantage of certain asymmetries in the outcomes of key comparisons. While the Regular Dual Pivot quicksort fails to utilize them, even though being based on the same abstract algorithmic idea.

- $1.846n \log n + O(n)$ Comparisons
- $0.615n \log n + O(n)$ Swaps
- Three Pivots
  - Middle 3 of 7 elements (Evenly Spaced Out)
- Simultaneous Partition algorithm
- Four Recursive Calls

[?]

This algorithm had incomplete pseudo-code. Only the partition algorithm was specified. Later on we see that this algorithm doesn't do well. It may be due to some choices we made in our implementation.

Elements are chosen at each quartile.

- $O(n \log n)$ Comparisons
- $O(n \log n)$ Swaps
- M Pivots
  - Sort $2M$ elements then pick ever other element
- Partition each segment one at a time
- $M + 1$ Recursive Calls

[?]

The run times discussed thus far have been for specific pivots. The analysis for each pivot and how one partition changes the analysis, which depends on studying the decision tree generated.

Theoretical Average Case Run Time

Sort Method	Comparisons
Classic	$2n \log n - 1.51n + O(\log(n))$
Dual Pivot	$2.13n \log n - 2.57n + O(\log(n))$
Optimal Dual Pivot	$1.8n \log n + O(n)$
Three Pivot	$1.846n \log n + O(n)$
Yaroslavskiy	$1.9n \log n - 2.46n + O(\log(n))$
M Pivot	$O(n \log n)$

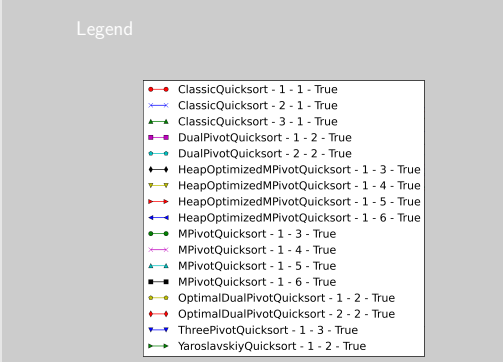
Sort Method	Swaps
Classic	$0.33n \log n - 0.58n + O(\log(n))$
Dual Pivot	$0.8n \log n - 0.3n + O(\log(n))$
Optimal Dual Pivot	$0.33n \log n + O(n)$
Three Pivot	$0.615n \log n + O(n)$
Yaroslavskiy	$0.6n \log n + 0.08n + O(\log(n))$
M Pivot	$O(n \log n)$

[?]

[?]

[?] [?]

These are average case run times. The M-Pivot quicksort is not studied very well. As you see, people study each quicksort with a fixed number of pivots.



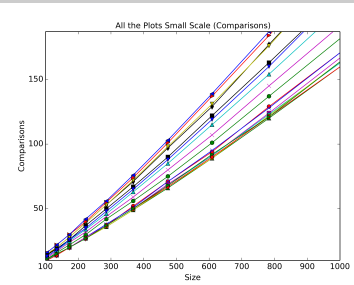
You can see the color and shape combination for each version of quick sort. These color and shape choices will be consistent throughout the following plots. There are 17 distinct derivatives of quicksort that have been run.

Note that for the large scale plot. We overlay the function :

$A \cdot n \log(n) + B \cdot n + C \log(n)$

With the function parameters A, B and C where found using a curve fitter on the data. As you will see in the following plots. The function are 'good ' fits. We have not done analysis to see if we have over fit our data. Although from the analysis done in many of the papers, the functions have that form.

Also for the small scale plots. The curve fitted function may not be seen, as a result, we just connect the data points and only show the fitted curve in the large scale plots.

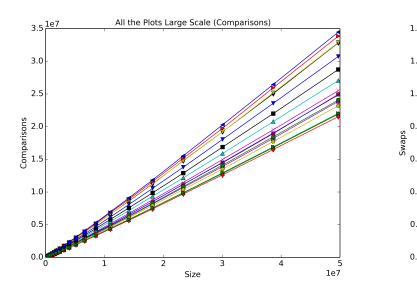


Comparisons :

The Heap 6-pivot sort is the worst. Close with all the other heap m-pivot sorts. 5-Pivot sort, 4-pivot sort, 3-pivot sort and Three-pivot quicksort all seem to be mediocre. The classic quicksorts, yaro, dual pivot, and optimal dual pivot all seem to be close and doing well

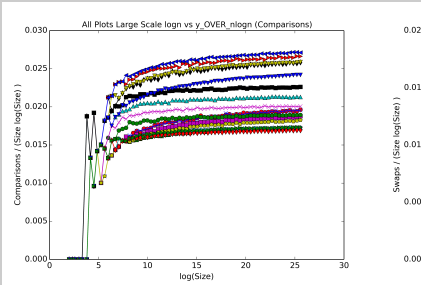
Swaps :

Similar trend as comparisons. Although the classic quicksorts are much higher than expected. They are close to the heap m-pivot sorts.



Comparisons : Optimized dual pivot quicksort, yaro, M-pivot sorts do well Heap Optimized m-pivot sorts do badly Swaps : Heap Optimized Sorts still do badly relative to the rest. Classic quicksorts also doing badly.

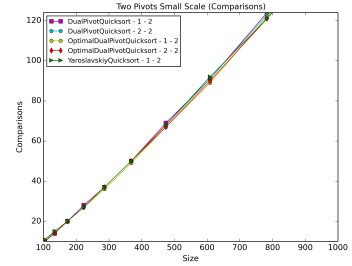
Mass Comparison $\log(n)$ vs $\frac{y}{n\log(n)}$



The point of these two plots is to emphasize the trend we see. In the last figure, all the plots look linear-like and doing it's own thing. In this plot there is a clear fact that all the plots have a trend.

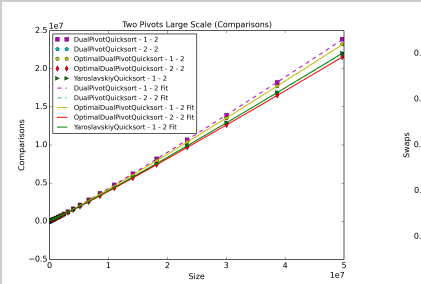
Comparisons :

Swaps :



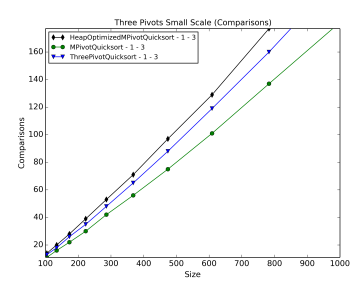
Comparisons : They are all doing very similarly. Swaps : Optimal Dual Pivot quicksort and dual pivot quicksort are doing really well. Yaro isn't.

Two Pivot Comparison Large Scale



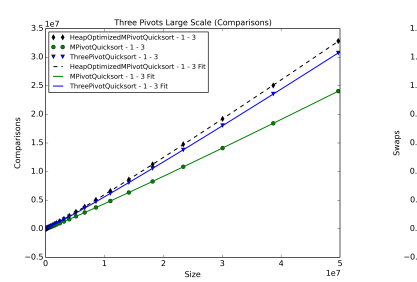
Comparisons : We can start to see them split. Optimal Dual Pivot Quicksort [second option] still rocks. Dual Picot [1] and Optimal Dual Pivot [1] don't do well. Swaps : They all do very similarly. Optimal Dual [1] and Dual Pivot [1] start to split away.

Three Pivot Comparison Small Scale

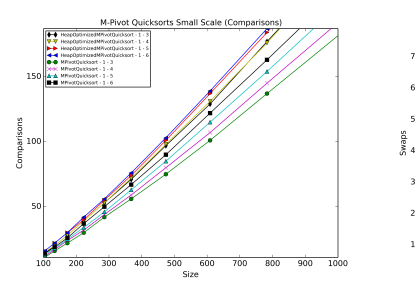


Comparisons : Interpertations Clear From plot. 3-Pivot Sort is great then the Three-pivot-Quicksort Swaps : Three-pivot-Quicksort is great then the 3-Pivot Sort

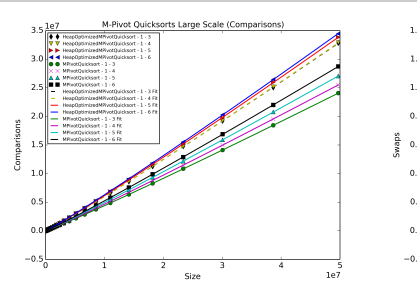
In both cases heap 3-pivot sort is the worst.



Comparisons : Same as the small scale Swaps : Same as the small scale



Comparisons : 3-pivot sort rocks. Heap 6-pivot sort sucks. Swaps : 3,4,5-pivot sort are really good. Heap 3-pivot sort is bad.



Comparisons : 3,4,5-pivot sort are really good. All the heap-m-pivot sorts are not Swaps : 6-pivot sort is the best. Although 3,4,5-pivot sorts are good as well. All the heap-m-pivot sorts are not good.

Fit Coefficients of $A \cdot n\log(n) + B \cdot n + C\log(n)$

Sort Method	Comparisons	Swaps
Classic - 1 - 1	0.02219	0.01060
Classic - 2 - 1	0.02126	**0.01110**
Classic - 3 - 1	0.01799	0.00828
Dual Pivot - 1 - 2	0.02109	0.00636
Dual Pivot - 2 - 2	0.01787	0.00603
Optimal Dual Pivot - 1 - 2	0.02044	0.00636
Optimal Dual Pivot - 2 - 2	**0.01754**	0.00603
Three Pivot - 1 - 3	0.02595	0.00616
Yaroslavskiy - 1 - 2	0.01811	0.00584

Sort Method	Comparisons	Swaps
Heap M Pivot - 1 - 3	0.02755	0.00999
Heap M Pivot - 1 - 4	0.02782	0.00885
Heap M Pivot - 1 - 5	**0.02903**	0.00809
Heap M Pivot - 1 - 6	0.02801	0.00769
M Pivot - 1 - 3	0.01955	0.00640
M Pivot - 1 - 4	0.02039	0.00594
M Pivot - 1 - 5	0.02136	0.00532
M Pivot - 1 - 6	0.02369	**0.00524**

Name of sort - Pivot Selection Method - Number of Pivots
The values in the table are of the coefficient of the $n\log(n)$ term.
Best Comparisons : Optimal Dual Pivot - 2 - 2
Best Swaps : M Pivot - 1 - 6
Worst Comparisons : Heap M Pivot - 1 - 5
Worst Swaps : Classic - 2 - 1
Overall we see that the Heap-M-Pivot sort do terribly. This may be a result of how often we call Heapify. We may have not needed to call heapify at every recursive call. This can be an error on our part.
We are currently using these as a qualitative comparison between each sorting method.