**Analysis Document**
CS 6012
Lindsay Haslam

**Who are your team members?**

Corinne Jones.

**Mergesort Threshold Experiment**

In the following experiment, we discovered that the optimal threshold value for the given array predominantly lies around 20. While there were occasional variations,, with a threshold of 10 outperforming in some instances, the consistently superior performance was with a threshold value of 20. Also, the threshold corresponding to the maximum value (denoted in red) does not resort to insertion sort. The red line increases in execution time when in contrast with alternative threshold values.

Threshold 5

| | |
|---|---|
| 100 | 329458 |
| 500 | 402083 |
| 1000 | 1240916 |
| 5000 | 2243542 |

Threshold 20

| | |
|---|---|
| 100 | 12709 |
| 500 | 102417 |
| 1000 | 203666 |
| 5000 | 1410125 |

Threshold 10

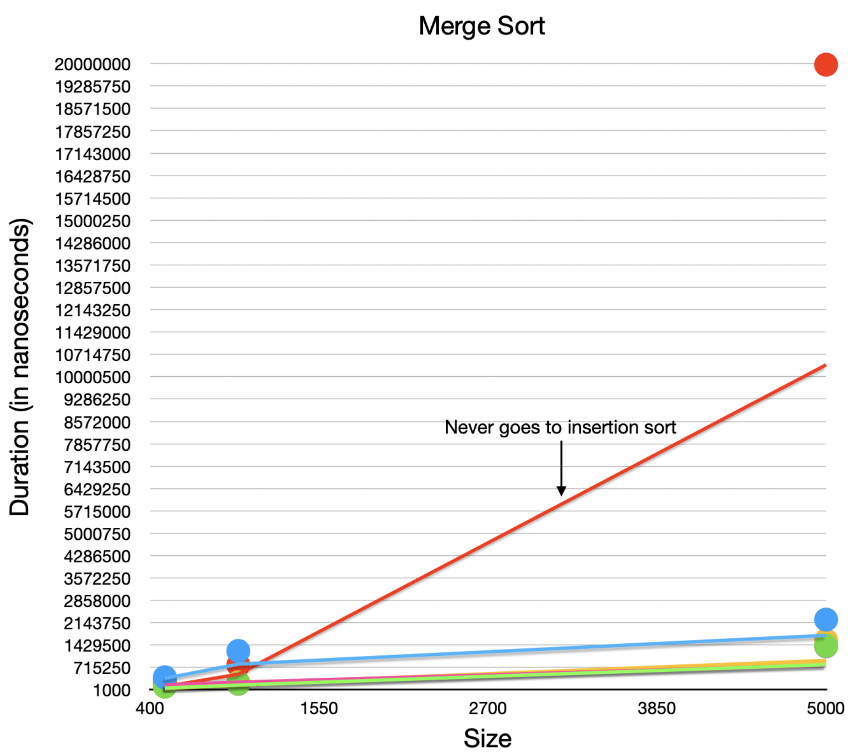
| | |
|---|---|
| 100 | 30959 |
| 500 | 296791 |
| 1000 | 192625 |
| 5000 | 1390583 |

Threshold 50

| | |
|---|---|
| 100 | 20417 |
| 500 | 104583 |
| 1000 | 241208 |
| 5000 | 1582625 |

ThreshMaxVal

| | |
|---|---|
| 100 | 8916 |
| 500 | 202375 |
| 1000 | 793417 |
| 5000 | 19968791 |

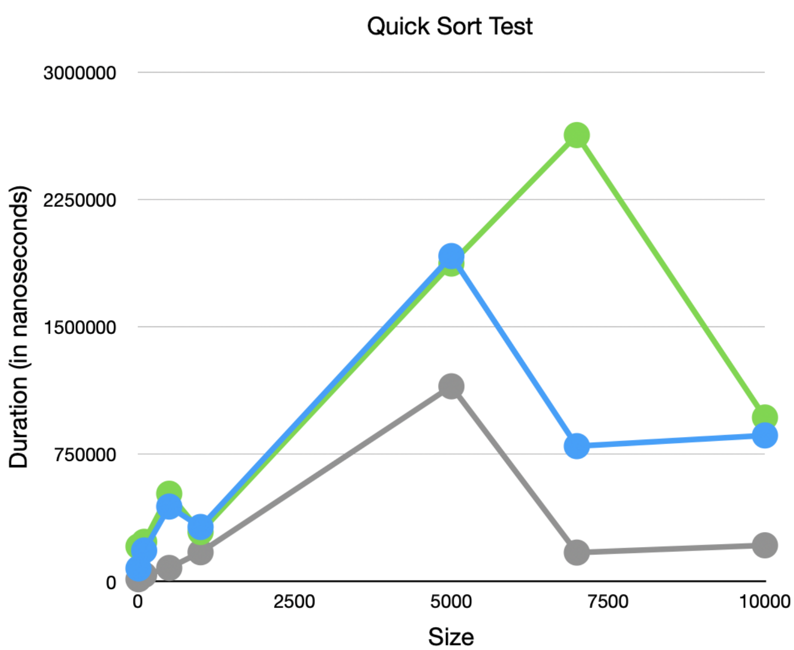


## Quicksort Pivot

The best pivot point identified for quicksort was the median index. This choice aligns logically with our approach, where we consider the first and last indices (represented by 0 and array.length-1, respectively) and calculate their midpoint. Professor Jones, in his lectures, asserted that the median index consistently emerges as the optimal choice, with random selection ranking at second.

Random Index

| | |
|---|---|
| 10 | 76542 |
| 100 | 182083 |
| 500 | 440791 |
| 1000 | 320666 |
| 5000 | 1916000 |
| 7000 | 796542 |
| 10000 | 859625 |




Left Index

| | |
|---|---|
| 10 | 205875 |
| 100 | 233208 |
| 500 | 516542 |
| 1000 | 290875 |
| 5000 | 1871792 |
| 7000 | 2628458 |
| 10000 | 965375 |

Median Index

| | |
|---|---|
| 10 | 13042 |
| 100 | 38542 |
| 500 | 79416 |
| 1000 | 171917 |
| 5000 | 1148917 |
| 7000 | 169833 |
| 10000 | 212583 |

## Quick Sort vs Merge Sort

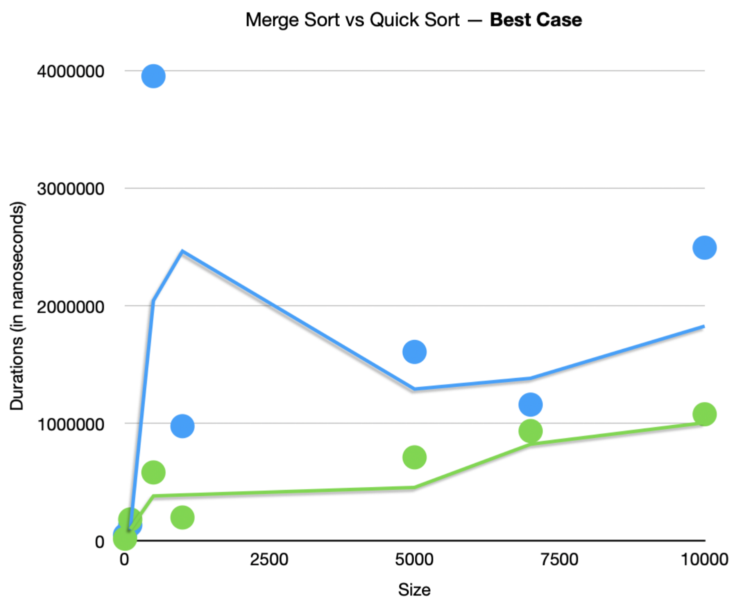
After running best, average, and worst case. It is apparent that quicksort is the ideal sort every time. When it comes to average sort, the differences between the two aren't as drastic, as they seem to follow a similar pattern. For best case, merge sort takes a significant amount of time for smaller sized arrays, and drops at a certain size.

Merge

| | |
|---|---|
| 10 | 50083 |
| 100 | 136959 |
| 500 | 3953250 |
| 1000 | 976458 |
| 5000 | 1608125 |
| 7000 | 1159834 |
| 10000 | 2494625 |

Quick

| | |
|---|---|
| 10 | 19500 |
| 100 | 183542 |
| 500 | 583000 |
| 1000 | 199959 |
| 5000 | 711459 |
| 7000 | 934916 |
| 10000 | 1077958 |

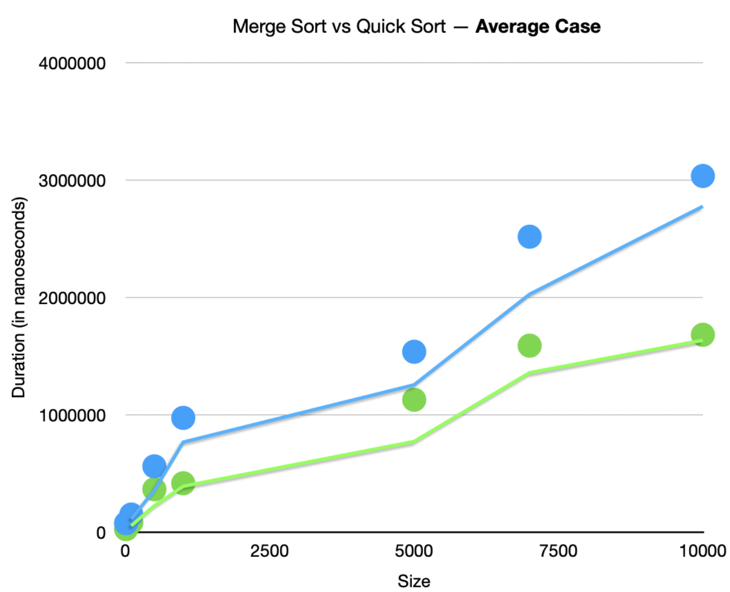



Merge

| | |
|---|---|
| 10 | 77458 |
| 100 | 150875 |
| 500 | 561500 |
| 1000 | 974084 |
| 5000 | 1536958 |
| 7000 | 2516459 |
| 10000 | 3033292 |

Quick

| | |
|---|---|
| 10 | 27000 |
| 100 | 84375 |
| 500 | 366625 |
| 1000 | 417417 |
| 5000 | 1128666 |
| 7000 | 1588833 |
| 10000 | 1681417 |

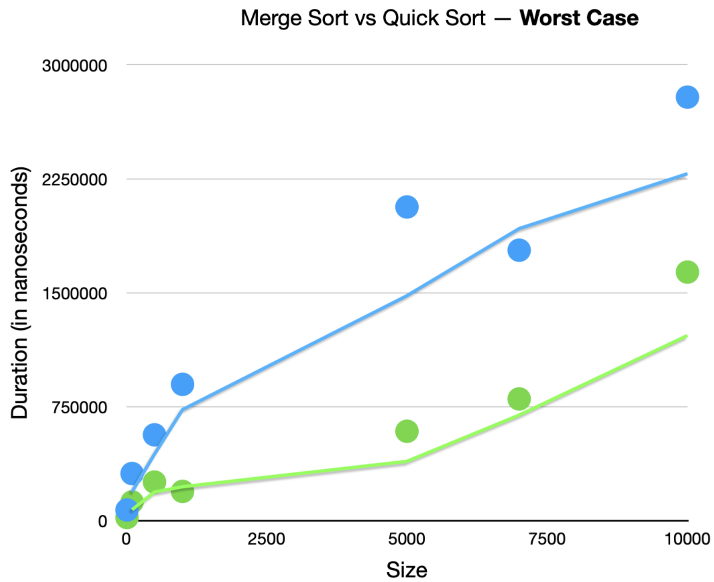


Merge

| | |
|---|---|
| 10 | 71083 |
| 100 | 310959 |
| 500 | 565541 |
| 1000 | 897750 |
| 5000 | 2064583 |
| 7000 | 1780292 |
| 10000 | 2787250 |

Quick

| | |
|---|---|
| 10 | 21792 |
| 100 | 124208 |
| 500 | 254916 |
| 1000 | 193459 |
| 5000 | 589083 |
| 7000 | 801708 |
| 10000 | 1636208 |




**Do the actual running times of your sorting methods exhibit the growth rates you expected to see?**

I did not anticipate that different thresholds would create such an impact on the efficiency of merge and insertion sort. It was genuinely surprising to see the substantial difference made by entering insertion sort either too early or too late in size. I'm interested to see how engineers frequently combine various algorithms to achieve optimal runtimes.

I was expecting the median index to be the best pivot, but I was surprised that the random index method's results weren't closer to the median's results. I think a better way to test the random index method would be to iterate through it many times (meaning a new random array is created each time) and create some sort of average with it. The specific graph that I have in this document is a representation of one array, and it could have been a record of any index. There's a chance the results would have been better than, closer to, or drastically different than median index's results.

As for the growth rates in best, average, and worst cases, my expectations were kind of met. Professor Jones mentioned that merge sort tends to be slower and less efficient, so I assumed that at least one case might favor merge sort. While I initially expected most graphs to lean towards quicksort, I was surprised that *all* scenarios consistently demonstrate quicker durations with quicksort. It was also interesting to see how sometimes my lines didn't look "obviously" like the Big O notation that it should.. Occasionally, by zooming out from the graph or using logarithmic settings in Numbers, the resemblance to a specific Big O notation becomes clearer.