# *Programming Assignment 1 Report*

## Results

- $N_{min}$: the smallest power of 10 array size that takes 20 milliseconds or more per run to sort.
- $N_{max}$: the largest power of 10 array size that takes 10 minutes or less per run to sort (30 mins for all 3 runs), or the largest input size you can successfully sort if no input took more than 30 minutes total.
- $T_{min}$: the time to sort an array of size Nmin.
- $T_{max}$: the time to sort an array of size Nmax.

| | $n_{min}$ | $t_{min}$ | $n_{max}$ | $t_{max}$ |
|---|---|---|---|---|
| SC | $10^5$ | 895,458,292 ns or 895.46 ms | $10^6$ | 91,047,884,000 ns or 1.51min |
| SS | $10^5$ | 898,365,208 ns or 898.37 ms | $10^6$ | 97,103,288,000 ns or 1.62min |
| SR | $10^4$ | 35,514,917 ns or 35.51 ms | $10^5$ | 3,702,927,959ns or 3.7s |
| IC | $10^8$ | 22,761,542 ns or 22.76 ms | $10^9$ | 1,010,319,750ns or 1.01s |
| IS | $10^8$ | 22,304,375 ns or 22.30 ms | $10^9$ | 1,503,439,708ns or 1.5s |
| IR | $10^8$ | 160,165,292 ns or 160.17 ms | $10^9$ | 3,149,739,959ns or 3.1s |
| MC | $10^6$ | 24,770,250 ns or 24.77 ms | $10^9$ | 80,632,596,458ns or 1.3 min |
| MS | $10^6$ | 26144584 ns or 26.14 ms | $10^9$ | 437,035,572,667 ns or 7.2 min |
| MR | $10^6$ | 34,242,875 ns or 34.24 ms | $10^9$ | 504,712,349,250 ns or 8.4 min |

| | | | | |
|---|---|---|---|---|
| QC | $10^5$ | 889,896,250ns or 889.90ms | $10^6$ | 91,973,526,625ns or 1.53 min |
| QS | $10^7$ | 123,493,000ns or 123.49ms | $10^8$ | 1,424,522,792ns or 1.42 seconds |
| QR | $10^7$ | 118,248,916ns or 118.25ms | $10^8$ | 1,414,923,625ns or 1.41 seconds |

## Analysis

| | $t_{max}/t_{min}$ | n ratio | nln(n) ratio | $n^2$ ratio | Behavior |
|---|---|---|---|---|---|
| SC | 102ns | $10^6/10^5 = 10$ | $(10^6\ln(10^6)) / (10^5\ln(10^5)) = 12$ | $(10^6)^2/(10^5)^2 = 100$ | $n^2$ |
| SS | 108ns | $10^6/10^5 = 10$ | $(10^6\ln(10^6)) / (10^5\ln(10^5)) = 12$ | $(10^6)^2/(10^5)^2 = 100$ | $n^2$ |
| SR | 104ns | $10^5/10^4 = 10$ | $(10^5\ln(10^5)) / (10^4\ln(10^4)) = 11$ | $(10^5)^2/(10^4)^2 = 100$ | $n^2$ |
| IC | 44ns | $10^9/10^8 = 10$ | $(10^9\ln(10^9)) / (10^8\ln(10^8)) = 11$ | $(10^9)^2/(10^8)^2 = 100$ | nln(n) |
| IS | 67ns | $10^9/10^8 = 10$ | $(10^9\ln(10^9)) / (10^8\ln(10^8)) = 11$ | $(10^9)^2/(10^8)^2 = 100$ | $n^2$ |
| IR | 20ns | $10^9/10^8 = 10$ | $(10^9\ln(10^9)) / (10^8\ln(10^8)) = 11$ | $(10^9)^2/(10^8)^2 = 100$ | nln(n) |
| MC | 3255ns | $10^9/10^6 = 1000$ | $(10^9\ln(10^9)) / (10^6\ln(10^6)) = 1,500$ | $(10^9)^2/(10^6)^2 = 1,000,000$ | nln(n) |
| MS | 16716ns | $10^9/10^6 = 1000$ | $(10^9\ln(10^9)) / (10^6\ln(10^6)) = 1,500$ | $(10^9)^2/(10^6)^2 = 1,000,000$ | nln(n) |
| MR | 14739ns | $10^9/10^6 = 1000$ | $(10^9\ln(10^9)) / (10^6\ln(10^6)) = 1,500$ | $(10^9)^2/(10^6)^2 = 1,000,000$ | nln(n) |

| | | | | | |
|---|---|---|---|---|---|
| QC | 103ns | $10^8/10^7 = 10$ | $(10^8\ln(10^8)) / (10^7\ln(10^7)) = 11$ | $(10^8)^2/(10^7)^2 = 100$ | $n^2$ |
| QS | 12ns | $10^8/10^7 = 10$ | $(10^8\ln(10^8)) / (10^7\ln(10^7)) = 11$ | $(10^8)^2/(10^7)^2 = 100$ | nln(n) |
| QR | 12ns | $10^8/10^7 = 10$ | $(10^8\ln(10^8)) / (10^7\ln(10^7)) = 11$ | $(10^8)^2/(10^7)^2 = 100$ | nln(n) |

**Your report should contain a summary of (1) how your results compare to the theoretical analysis for all four algorithms, and (2) why your results make sense or are surprising. You should spend more time explaining your results when they are unusual or unexpected.**

## Summary

My results overall are somewhat surprising. First, for the selection algorithm, however, the results match up. When using selection sort, it is known to have a best, average, and worst time case of $O(n^2)$. In our analysis we can see that it is accurate to show us that. For insertion sort, it is known to have a best case of O(n) with an average and worst case $O(n^2)$. We know that worst case, when sorting an already sorted array, the time complexity should be $O(n^2)$, which matches our analysis, however, what is surprising is that the results for constant and random are 44ns and 20ns respectively which I determined was closer to 11ns than 10ns, meaning they are more fitting with nln(n), but that seems like such a small way of being able to tell the difference between whether it's n or nln(n). For merge sort, the time complexity is supposed to be nln(n) for best, average and worst case. We can see from our analysis that that is true. For quicksort, we know that it has a time complexity of O(nlog(n)) for best and average cases with a worst case of $O(n^2)$. When quicksorting a constant array, we get the worst case of $O(n^2)$, which is accurate, however we know that quicksorting an already sorted array should give us worst case of $O(n^2)$, however my analysis tells me that sorting the sorted array gives us a time complexity of O(nln(n)), which is surprising. It should be $O(n^2)$. For sorting random numbers, we know that it should have the best case scenario of O(nln(n)), meaning that matches up.