EN.605.202.81 Section 84

Alexander Shah

Lab 4: Sort Analysis

December 5, 2023

This program implements five sorting methods, using four variations of quicksort and natural merge sort. The versions of quicksort pick different locations to create pivot points and may use insertion sort to recombine them. The different methods of picking mid points causes the algorithm to vary depending on the length and ordering of the data. By running different length tests with varied ordering, we are able to analyze the sorting efficiency based on the number of comparisons the algorithm makes as well as exchanges of elements during its runtime. This data shows us that as the number of elements increases, so does the number of comparisons and exchanges but at different rates depending on the sort and order. For example, ascending and descending orders make quicksort based approaches very inefficient and quicksort tends to have extremely high numbers of comparisons for ordered data which can be costly just to determine that the items are already in order.

A good example is quicksort using first pivot running on ascending and descending data. When ascending the sort required only 49 exchanges for 50 numbers but the descending order required 674. This large step up in exchanges illustrates the inefficiencies some sorts can experience just by ordering. Choosing the median pivot was better than using the first number as a pivot. And the difference in threshold for insertion sort, 100 vs 50, made little difference for large files. As the sizes of files grew, the effect that ordering had on quicksort grew as well. Overall the most important factors as the size grew were the pivot choice for quick sort, with median being preferred over insert, over first pivot, and the order of the data. Comparatively, natural merge sort was more resistant to ordering changes and file sizes. Natural merge sort operated more efficiently compared to a straight merge sort which recursively divides data and uses extra space to hold the arrays. By choosing an in place sort that uses a stack, a quick structure that lends itself well to popping on and off ordered sequences of data, the natural merge

sort ran fairly consistently and efficiently but crucially it used far less space than a straight merge sort would have. Natural merge sort is also faster for data that has partial order, compared to straight merge which performs similarly regardless of order. Natural merge sort is more consistent overall compared to quicksort, and would be faster than straight merge sort regardless of order or size.

Justification for Design Decisions

Initially I tried implementing the sorts using recursion but ran into trouble running them with large amounts of data. I was getting recursion errors, that max depth had been reached, leading to most of the sorts failing to complete. I went back and reimplemented the sorts using iterative approaches by using the MyStack class made in previous labs to act as a stack structure. Choosing a stack was a natural fit given our experience in previous labs, as well as for the task, the stack was a good choice for time and space efficiency of operations.

What I Learned/What I would Do Differently

Implementing 5 recursive sorts resulted in a lot of recursion failures, so I would first attempt iterative designs over recursion until I am sure the method would benefit from recursion. It has been much harder to debug recursive problems than iterative ones.

| Sort Name | Order | Size | Comparisons | Exchanges |
|---|---|---|---|---|
| Quicksort first pivot | random | 50 | 277 | 119 |
| Quicksort first pivot | random | 1000 | 10569 | 5653 |
| Quicksort first pivot | random | 2000 | 23399 | 12554 |
| Quicksort first pivot | random | 5000 | 68270 | 35059 |
| Quicksort first pivot | random | 10000 | 157753 | 80463 |
| Quicksort first pivot | ascending | 50 | 1225 | 49 |
| Quicksort first pivot | ascending | 1000 | 499500 | 999 |
| Quicksort first pivot | ascending | 2000 | 1999000 | 1999 |
| Quicksort first pivot | ascending | 5000 | 12497500 | 4999 |
| Quicksort first pivot | ascending | 10000 | 49995000 | 9999 |
| Quicksort first pivot | descending | 50 | 1225 | 674 |
| Quicksort first pivot | descending | 1000 | 499500 | 250999 |
| Quicksort first pivot | descending | 2000 | 1999000 | 1001999 |
| Quicksort first pivot | descending | 5000 | 12497500 | 6254999 |

| Sort Name | Order | Size | Comparisons | Exchanges |
|---|---|---|---|---|
| Quicksort first pivot | descending | 10000 | 49995000 | 25009999 |
| Quicksort insert 100 | random | 50 | 725 | 727 |
| Quicksort insert 100 | random | 1000 | 23416 | 20537 |
| Quicksort insert 100 | random | 2000 | 48713 | 42335 |
| Quicksort insert 100 | random | 5000 | 127638 | 105268 |
| Quicksort insert 100 | random | 10000 | 274484 | 218722 |
| Quicksort insert 100 | ascending | 50 | 49 | 49 |
| Quicksort insert 100 | ascending | 1000 | 494550 | 999 |
| Quicksort insert 100 | ascending | 2000 | 1994050 | 1999 |
| Quicksort insert 100 | ascending | 5000 | 12492550 | 4999 |
| Quicksort insert 100 | ascending | 10000 | 49990050 | 9999 |
| Quicksort insert 100 | descending | 50 | 1225 | 1274 |
| Quicksort insert 100 | descending | 1000 | 499500 | 253449 |
| Quicksort insert 100 | descending | 2000 | 1999000 | 1004449 |
| Quicksort insert 100 | descending | 5000 | 12497500 | 6257449 |
| Quicksort insert 100 | descending | 10000 | 49995000 | 25012449 |
| Quicksort insert 50 | random | 50 | 725 | 727 |
| Quicksort insert 50 | random | 1000 | 16172 | 12778 |
| Quicksort insert 50 | random | 2000 | 31958 | 24297 |
| Quicksort insert 50 | random | 5000 | 92209 | 66858 |
| Quicksort insert 50 | random | 10000 | 207732 | 146252 |
| Quicksort insert 50 | ascending | 50 | 49 | 49 |
| Quicksort insert 50 | ascending | 1000 | 498275 | 999 |
| Quicksort insert 50 | ascending | 2000 | 1997775 | 1999 |
| Quicksort insert 50 | ascending | 5000 | 12496275 | 4999 |
| Quicksort insert 50 | ascending | 10000 | 49993775 | 9999 |
| Quicksort insert 50 | descending | 50 | 1225 | 1274 |
| Quicksort insert 50 | descending | 1000 | 499500 | 251599 |
| Quicksort insert 50 | descending | 2000 | 1999000 | 1002599 |
| Quicksort insert 50 | descending | 5000 | 12497500 | 6255599 |
| Quicksort insert 50 | descending | 10000 | 49995000 | 25010599 |
| Quicksort median pivot | random | 50 | 218 | 173 |
| Quicksort median pivot | random | 1000 | 9107 | 5570 |
| Quicksort median pivot | random | 2000 | 20861 | 12116 |
| Quicksort median pivot | random | 5000 | 60807 | 37644 |
| Quicksort median pivot | random | 10000 | 134190 | 81497 |
| Quicksort median pivot | ascending | 50 | 193 | 148 |
| Quicksort median pivot | ascending | 1000 | 7987 | 4960 |
| Quicksort median pivot | ascending | 2000 | 17964 | 10916 |

| Sort Name | Order | Size | Comparisons | Exchanges |
|---|---|---|---|---|
| Quicksort median pivot | ascending | 5000 | 51822 | 30713 |
| Quicksort median pivot | ascending | 10000 | 113631 | 66421 |
| Quicksort median pivot | descending | 50 | 245 | 187 |
| Quicksort median pivot | descending | 1000 | 14205 | 9001 |
| Quicksort median pivot | descending | 2000 | 32880 | 20499 |
| Quicksort median pivot | descending | 5000 | 98004 | 60501 |
| Quicksort median pivot | descending | 10000 | 218585 | 133464 |
| Natural merge sort | random | 50 | 217 | 269 |
| Natural merge sort | random | 1000 | 8737 | 9738 |
| Natural merge sort | random | 2000 | 19395 | 21396 |
| Natural merge sort | random | 5000 | 56820 | 61825 |
| Natural merge sort | random | 10000 | 123668 | 133673 |
| Natural merge sort | ascending | 50 | 161 | 213 |
| Natural merge sort | ascending | 1000 | 5052 | 6053 |
| Natural merge sort | ascending | 2000 | 11104 | 13105 |
| Natural merge sort | ascending | 5000 | 33356 | 38361 |
| Natural merge sort | ascending | 10000 | 71712 | 81717 |
| Natural merge sort | descending | 50 | 133 | 185 |
| Natural merge sort | descending | 1000 | 4932 | 5933 |
| Natural merge sort | descending | 2000 | 10864 | 12865 |
| Natural merge sort | descending | 5000 | 29804 | 34809 |
| Natural merge sort | descending | 10000 | 64608 | 74613 |

Figure 1. Comparisons and Exchanges for 5 different sorts

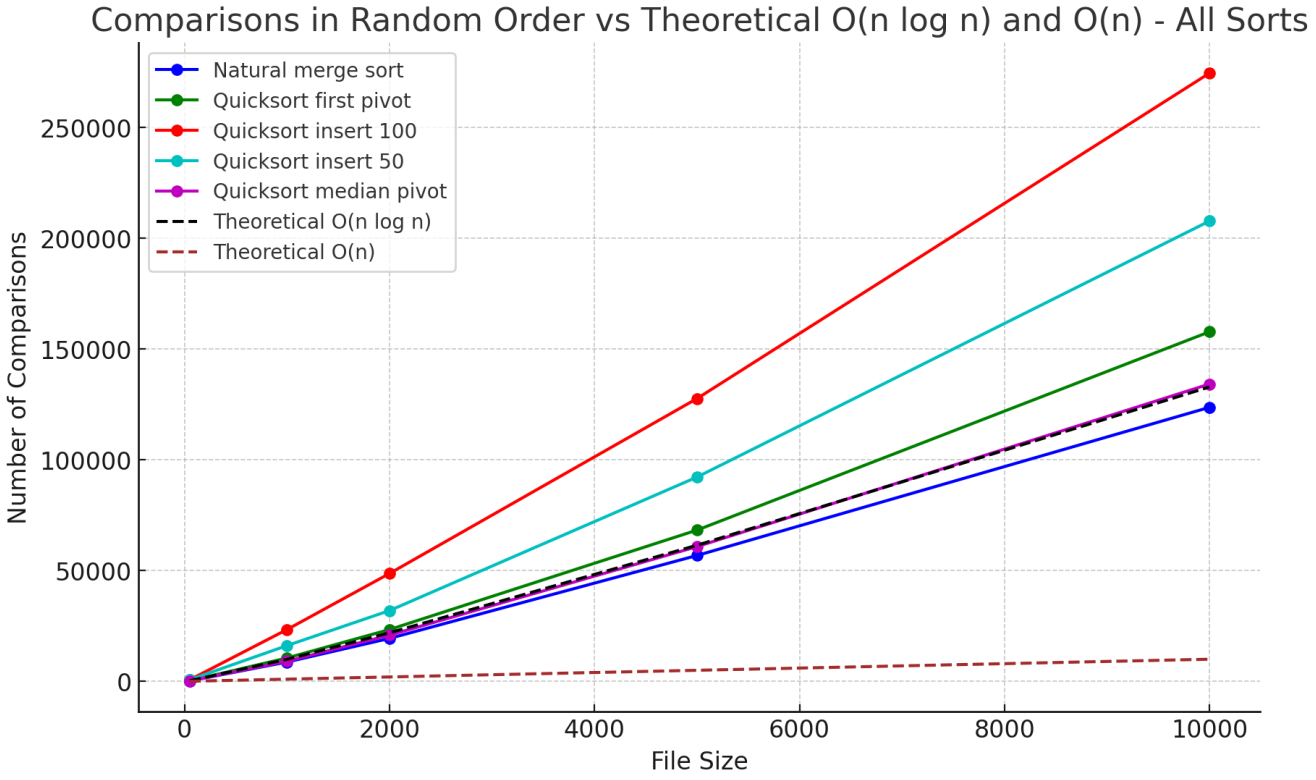


Figure 2. Chart comparing random ordered performance

We can see from the chart above that natural merge sort performed the best, followed closely by quicksort using the median pivot. Randomly ordered data shows an average case for quicksort, making it a better visual comparison but this doesn't reflect the performance in all cases. Looking at nlogn and n lines, we can see how natural merge sort falls under n logn for random data, making it a good sort.

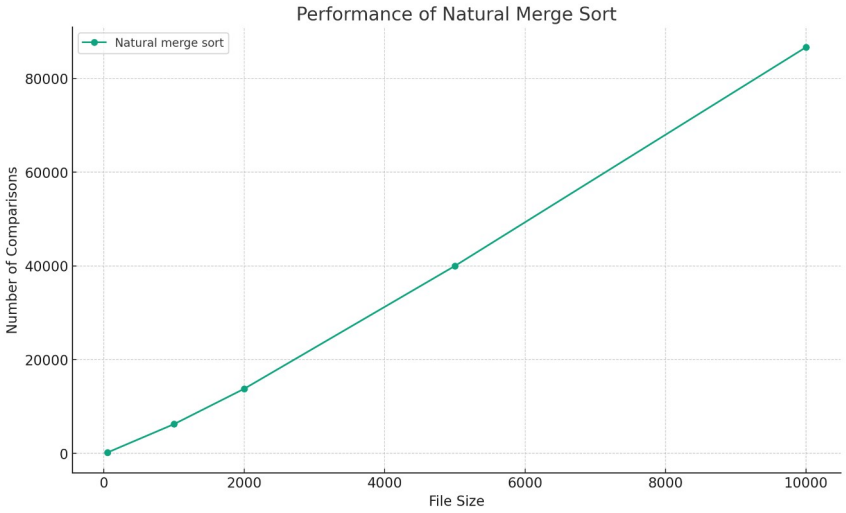


Figure 3. Natural merge sort average comparisons

Natural merge sort shows a nearly linear correlation to file size.

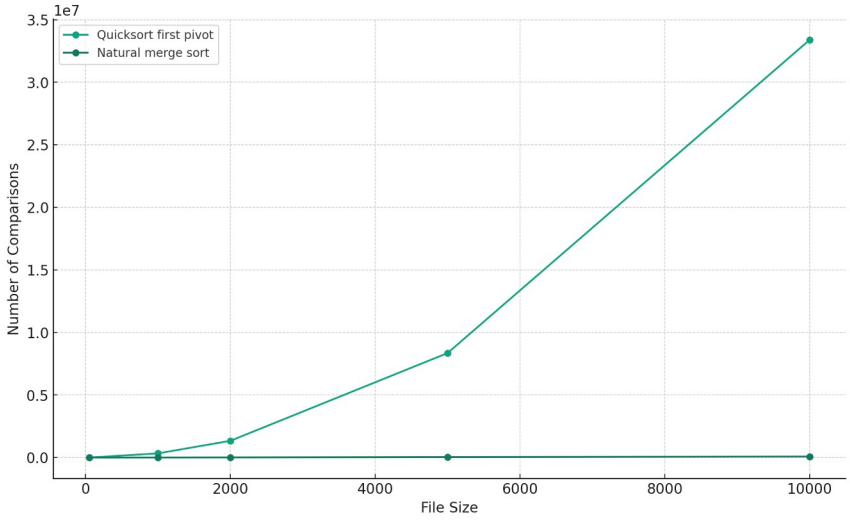


Figure 4. Natural merge and quicksort first pivot comparison

Comparatively, the number of comparisons becomes much greater at the same file size for quicksort, with noticeable growth as the file size grows.