Sidney Sanders

Dr. Tianyi Song

CS 4120 Design and Analysis of Algorithms

15 November 2020

Project 1 report

Section I

The expected performance of Insertion sort would be given to us using the time complexity that insertion sort has which is O(n2). Insertion sort is one of the simple algorithms and is not a very complex algorithm to understand. Insertion gets its speed from small datasets and sorted lists. The performance for the Merge sort would be the time complexity of O(n log(n)). Merge sort is a sorting algorithm that splits the data in two parts, sorts, and then merges the two halves back together. Merge sort is faster with sorting larger data sets and is a bit faster in this project than insertion sort due to the large amounts of data given to these algorithms. Quick sort has an average time complexity of O(n log(n)). Quick sort is a more complex algorithm compared to the other two types of sorting algorithms this takes a pivot location in the array which is similar with Merge where it creates two arrays, but the way it fills the arrays and sorts is a bit different. This algorithm has a pivot point and fills the two arrays with integers either lower or higher than the pivot point. Works well with large amounts of data.

|  |  |  |  |
| --- | --- | --- | --- |
| Merge Sort | Size | Comparisons | Time |
| Identical | 10000 | 19996 | 155 |
|  | 20000 | 39996 | 459 |
|  | 40000 | 79996 | 668 |
|  | 80000 | 159996 | 1291 |
| Rand | 10000 | 19996 | 209 |
|  | 20000 | 39996 | 789 |
|  | 40000 | 79996 | 577 |
|  | 80000 | 159996 | 1305 |
| Reversed | 10000 | NULL | NULL |
|  | 20000 | NULL | NULL |
|  | 40000 | 79996 | 682 |
|  | 80000 | 159996 | 1303 |
| Sorted | 10000 | 19996 | 413 |
|  | 20000 | 39996 | 463 |
|  | 40000 | 79996 | 731 |
|  | 80000 | 159996 | 1183 |

Section II

|  |  |  |  |
| --- | --- | --- | --- |
| Insertion Sort | Size | Comparisons | Time |
| Identical | 10000 | 9999 | 87 |
|  | 20000 | 19999 | 56 |
|  | 40000 | 39999 | 111 |
|  | 80000 | 79999 | 224 |
| Rand | 10000 | 37285538 | 89916 |
|  | 20000 | 148833080 | 343602 |
|  | 40000 | 467970368 | 1113480 |
|  | 80000 | 2179729071 | 5233155 |
| Reversed | 10000 | 50004999 | 120402 |
|  | 20000 | 200009999 | 475969 |
|  | 40000 | 800019999 | 1972447 |
|  | 80000 | 3200039999 | 7524586 |
| Sorted | 10000 | 9999 | 28 |
|  | 20000 | 19999 | 171 |
|  | 40000 | 39999 | 112 |
|  | 80000 | 79999 | 313 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Quick Sort Mid | | Size | Comparisons | Time |
| Identical | 10000 | | 100000000 | 287528 |
|  | 20000 | | 400000000 | 1216835 |
|  | 40000 | | 1600000000 | 4476977 |
|  | 80000 | | 6400000000 | 17844641 |
| Rand | 10000 | | 241724 | 3576 |
|  | 20000 | | 517952 | 3111 |
|  | 40000 | | 1116604 | 7282 |
|  | 80000 | | 2431992 | 14136 |
| Reversed | 10000 | | 74459534 | 213363 |
|  | 20000 | | 297757051 | 765919 |
|  | 40000 | | 1176648086 | 3073103 |
|  | 80000 | | 4623059418 | 11935962 |
| Sorted | 10000 | | 49607877 | 118743 |
|  | 20000 | | 198016213 | 447116 |
|  | 40000 | | 784027031 | 1683750 |
|  | 80000 | | 3075332158 | 6627007 |

|  |  |  |  |
| --- | --- | --- | --- |
| Quick Sort | Size | Comparisons | Time |
| Identical | 10000 | 100000800 | 483894 |
|  | 20000 | 400009392 | 1727020 |
|  | 40000 | 1600000480 | 6930715 |
|  | 80000 | 6400013647 | 27620406 |
| Rand | 10000 | 225532 | 14083 |
|  | 20000 | 523528 | 17997 |
|  | 40000 | 1148547 | 33130 |
|  | 80000 | 2306605 | 61954 |
| Reversed | 10000 | 242692 | 7218 |
|  | 20000 | 507642 | 13775 |
|  | 40000 | 1076290 | 28106 |
|  | 80000 | 2352414 | 56672 |
| Sorted | 10000 | 207822 | 5319 |
|  | 20000 | 483752 | 12646 |
|  | 40000 | 1110843 | 22290 |
|  | 80000 | 2396493 | 52669 |

Section III

I believe that the theoretical performance expectations were met. The insertion sort algorithm did well on the identical and sorted lists due to the fact it travels the list from the beginning and travels to the end. In those lists the algorithm did not have to do any moving of data and there for did not have to go back through the list. Merge sort did well with all the different types, it was uniform across the board with the amount of comparisons the algorithm needed to get through the files. Quick sort using the random pivot did not do well with the identical data files but other than that was also uniform in number of comparisons in the random, reversed, and sorted files. Quick sort using the middle element as the pivot did surprisingly bad, I believe. The number of comparisons were high compared to the random pivot quick sort I compared it to. The time it took to complete the datasets was also high and did worse than the merge sort, and the other quick sort with the random pivot. That was something that really surprised me about this project. I had the idea that quick sort would be one of the best algorithms with the larger sets of data but overall, I think that merge sort was the best overall. Something else that stood out to me in the data tables is that both quick sort algorithms struggles with the lists of identical dataset files.