Gürkan Bıyık 2020510019

**Homework 1 Report**

**Performance Table**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **EQUAL INTEGERS** | | | **RANDOM INTEGERS** | | | **INCREASING INTEGERS** | | | **DECREASING INTEGERS** | | |
| **1.000** | **10.000** | **100.000** | **1.000** | **10.000** | **100.000** | **1.000** | **10.000** | **100.000** | **1.000** | **10.000** | **100.000** |
| **Merge Sort** | **Two Parts** | 2ms | 1ms | 13ms | 1ms | 6ms | 22ms | 0ms | 0ms | 5ms | 0ms | 0ms | 5ms |
| **Three Parts** | 1ms | 1ms | 10ms | 0ms | 1ms | 39ms | 0ms | 1ms | 6ms | 0ms | 1ms | 4ms |
| **Quick Sort** | **First Element** | 2ms | 12ms | 887  ms | 1ms | 2ms | 12ms | 1ms | 14ms | 1285ms | 1ms | 15ms | 1415ms |
| **Random Element** | 2ms | 16ms | 927  ms | 0ms | 1ms | 13ms | 0ms | 1ms | 8ms | 1ms | 1ms | 4ms |
| **MidOfFirstMidLastElement** | 2ms | 14ms | 941 ms | 1ms | 1ms | 13ms | 1ms | 5ms | 274  ms | 0ms | 2ms | 130  ms |

**INTRODUCTION**

The merge and quick sort are divide-and-conquer algorithms. Both algorithms divide data into small pieces and make the ordering process on those small pieces. This process generally continues increasingly speedily because, after every small piece ordering process, the amount of the process to be applied data is getting decreased. The merge sort divides pieces continuously almost half, then orders the small pieces recursively to order and joins the data into bigger pieces. Quick sort chooses a pivot point to stack the data on both sides by comparing itself. Then, makes the same process on both sides by choosing a new pivot. Quick sort places the pivot in the right place in every recursive loop. So, in each process, one piece takes place where it should be on the actual data. The merge sort is generally efficient and suitable for all data, but the quick sort may not be efficient for all data. Because if data is ordered or all elements are equal, the data stacks at either left or right of the point. One side of the pivot takes a huge process, on the other side there is no process. So, quick sort works slower when the ratio of the left and right sides of the pivot is unbalanced. If both sides of the pivot are a similar amount of data, the algorithm works efficiently, and faster than merge sort.

**REPORT**

Due to graphs that were created from the test. When, the data amount increases, the time to order getting increasing, due to an increase in the ordering process. Comparing two-way merge to three-way merge, three-way merge divides the array into three parts, while two-way merge sort divides the array into half. Both algorithms' worst space complexity is O(n). On the other hand, three-way merge sorts are faster than two-way merge sorts in general. Because the time average complexity of the two-way merge sort is θ(n.log2(n)), the three-way merge sort is θ(n.log3(n)).

Furthermore, quick sort’s worst-case space complexity is 0(log(n)), which is less than merge sort. So, we can say that quick sort uses less memory than merge sort for ordering. Quick sort average time complexity is θ(n.log(n)). But, if the key thing on time complexity is the ratio. If the ratio is unbalanced, it cause the algorithm works in 0(n^2) complexity. As shown in the graphs, selecting the pivot as random makes the quick sort much faster. Because it prevents an unbalanced ratio.

In summary, if we have huge data and we don’t know how the data is, it is better to use merge sort, due to the worst time complexity. If we know that data is randomized, the quick sort will be a good choice due to its space complexity. Because both merge and quick sort has equal average time complexity.

**Part A**

The element quantity is over a thousand, so we should use divide and conqueror sorting algorithms instead of linear sorting algorithms like insertion sort, and selection sort. Our data is randomized because all words are sorted in Turkish equivalents, so their English equivalent is randomized. Because of that, there will not happen a worst-case scenario. But merge sort would be better because any unbalanced ratio occurs in the subarray for quick sort due to a huge amount of data

**Part B**

Suppose that a person has about 100 relatives. Our data will be partly randomized. So, we can choose the person on the bound of the Sub-Upper Pedigree as a pivot. I think comparing the age to this person, the ratio will be balanced and the algorithm performs efficiently.

**Resoruces i got help:**

* <https://www.interviewkickstart.com/learn/merge-sort-vs-quicksort-performance-analysis>
* <https://www.interviewkickstart.com/learn/quicksort-vs-merge-sort#:~:text=Merge%20sort%20is%20an%20external,memory%20throughout%20the%20sorting%20process>.