

**MARMARA UNIVERSITY**

**FACULTY OF ENGINEERING  
 DEPARMENT OF COMPUTER SCIENCE   
 ENGINEERING**

**CSE 2046 – ANALYSIS OF ALGORITHM**

**Submitted To: Due Date:**

**Asst.Prof.Ömer Korçak 11/05/2022**

|  |  |
| --- | --- |
| **STUDENTS** | **SCHOOL ID** |
| Murat ÖZCAN | 150119008 |
| Berkkan RENÇBER | 150119011 |
| Yasin Sefa KIRMAN | 150119034 |

Contents

**Introduction to Project2**

**1-) Designing The Experiment2**

Input Generation2

File Operation2

**2-) Designing and Coding2**

Programming Language2

Deciding on Metrics2

Theoretical Analysing Sorting Algorithm2

**3-) Illustrating and Analyzing Results3**

Time Complexity Of Insertion Sort3

Time Complexity Of Merge Sort 3

Time Complexity Of Quick Sort 3

Time Complexity Of Partial Selection Sort4

Time Complexity Of Partial Heap Sort 5

Time Complexity Of Unsorted Quick Select 6

Time Complexity Of Median-Of-Three7

Comparing Results with Different Inputs and Searching Key7

Theoretical Expectations12

**4-) Distribution of Tasks13**

**5-) References13**

**INTRODUCTION TO PROJECT**

The purpose of this Project is to find number of work is done by each different sorting algorithms and checking the whether they fit in theoretical results or not and compare the count number and make a comparission on results.

Same inputs will be tested for each algorithm for getting consistent comparision.

**1-) DESIGNING THE EXPERIMENT**

First neccessary thing we completed is that creating input file and reading them. FileOperation class is the center of the project. Any class can get same input array or input list reaching from outside of the class and do sorting using same input. Depends on list or array, algorithm calls inputArray or inputList function and get the input which size is assigned earlier.

The reason we used count as a reference of making observations is that there was a bug in Java System.currentTimeMillis() function. Sometimes the values were misleading. Therefore, to make realiable observation we decided to use count variable for being more accurate.

To make better better observations and comparissions between the count number of algorithms our input sizes are consist of 1000,2000,5000,1000 and 15000 number of random integers.

**2-) DESIGNING AND CODING**

We implemented the algorithms in the experiment with Java. We created the input lists to identify the best and worst cases of these algorithms. In this experiment, we analyzed and compiled algorithms to find the k'th smallest element in the list in 7 different ways.

• The first of these is Insertion-sort. Insertion-sort sorts elements in the list by decreasing and conquering. Time complexity of Insertion-sort is O(n) in the best case and O(n2) in the worst case.

• The second one is Merge-sort. Merge-sort sorts elements in the list by dividing and conquering. Time complexity of Merge-sort is O(nlogn) in the all cases.

• The third one is Quick-sort. Quick-sort sorts elements in the list by dividing and conquering. Time complexity of Quick-sort is O(nlogn) in the best case and is O(n2) in the worst case.

• The fourth one is partial selection-sort. Partial selection-sort sorts elements with brute force but it find the minimum element k times to find the k’th smallest element. Time complexity of it is O(k\*n) in the all cases.

• The fifth one is partial heap-sort. Partial heap-sort sorts elements by transforming and coquering. It uses max heap or min heap. For exapmle, if it sorts with max heap, it inserts n times. In this situation root of max heap is biggest key in the list. Then, it removes n-k times root to find k’th smallest element in the list. Time complexity of it is O(nlogn) in all cases.

• The sixth one is quick select algorithm. In this part, we did not sort but we applied quick select by partitioning. We chose the pivot element as the first element in an array. Time complexity of it is O(n) in the best case and O(n2) in the worst case.

• In this part, we applied again quick select but this time is median of-three pivot selection. The median of three selects the median of the first, middle and last element of the list as pivot. Time complexity of it is O(n) in the all cases.

**3-) ILLUSTRATING AND ANALYZING RESULTS**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **INSERTION SORT** | | | | | |
| Input Sizes | 1000 | 2000 | 5000 | 10000 | 15000 |
| Sorted Input List Time Complexity (Count) | 999 | 1999 | 4999 | 9999 | 14999 |
| Reversed Input List Time Complexity (Count) | 499991 | 1999994 | 12499925 | 49999892 | 112500027 |
| Random Input List Time Complexity (Count) | 258067 | 1013878 | 6277243 | 25123457 | 56046878 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **MERGE SORT** | | | | | |
| Input Sizes | 1000 | 2000 | 5000 | 10000 | 15000 |
| Sorted Input List Time Complexity (Count) | 19952 | 43904 | 123616 | 267232 | 417232 |
| Reversed Input List Time Complexity (Count) | 19952 | 43904 | 123616 | 267232 | 417232 |
| Random Input List Time Complexity (Count) | 19952 | 43904 | 123616 | 267232 | 417232 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **QUICK SORT** | | | | | |
| Input Sizes | 1000 | 2000 | 5000 | 10000 | 15000 |
| Sorted Input List Time Complexity (Count) | 618763 | 2538450 | 15541086 | 62079421 | 143056766 |
| Reversed Input List Time Complexity (Count) | 568739 | 2325864 | 14392810 | 57117600 | 131137905 |
| Random Input List Time Complexity (Count) | 18476 | 40926 | 105871 | 247379 | 400441 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **PARTIAL SELECTION SORT** | | | | | |
| Input Sizes | 1000 | 2000 | 5000 | 10000 | 15000 |
| Sorted Input List Time Complexity (Count)  Of Finding First Quarter | 218625 | 874750 | 5468125 | 21873750 | 49216875 |
| Reversed Input List Time Complexity (Count) Of Finding First Quarter | 218625 | 874750 | 5468125 | 21873750 | 49216875 |
| Random Input List Time Complexity (Count)  Of Finding First Quarter | 218625 | 874750 | 5468125 | 21873750 | 49216875 |
| Sorted Input List Time Complexity (Count)  Of Finding Median | 374750 | 1499500 | 9373750 | 37497500 | 84371250 |
| Reversed Input List Time Complexity (Count) Of Finding Median | 374750 | 1499500 | 9373750 | 37497500 | 84371250 |
| Random Input List Time Complexity (Count) Of Finding Median | 374750 | 1499500 | 9373750 | 37497500 | 84371250 |
| Sorted Input List Time Complexity (Count)  Of Finding Third Quarter | 468375 | 1874250 | 11716875 | 46871250 | 105463125 |
| Reversed Input List Time Complexity (Count) Of Finding Third Quarter | 468375 | 1874250 | 11716875 | 46871250 | 105463125 |
| Random Input List Time Complexity (Count) Of Finding Third Quarter | 468375 | 1874250 | 11716875 | 46871250 | 105463125 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **PARTIAL HEAP SORT** | | | | | |
| Input Sizes | 1000 | 2000 | 5000 | 10000 | 15000 |
| Sorted Input List Time Complexity (Count)  Of Finding First Quarter | 749 | 1499 | 3749 | 7499 | 11249 |
| Reversed Input List Time Complexity (Count) Of Finding First Quarter | 12759 | 28495 | 81075 | 177255 | 279935 |
| Random Input List Time Complexity (Count) Of Finding First Quarter | 5427 | 11273 | 34919 | 67607 | 93137 |
| Sorted Input List Time Complexity (Count)  Of Finding Median | 499 | 999 | 2499 | 4999 | 7499 |
| Reversed Input List Time Complexity (Count) Of Finding Median | 8773 | 19515 | 55753 | 121475 | 191049 |
| Random Input List Time Complexity (Count) Of Finding Median | 3961 | 7857 | 23723 | 40591 | 60831 |
| Sorted Input List Time Complexity (Count)  Of Finding Third Quarter | 249 | 499 | 1249 | 2499 | 3749 |
| Reversed Input List Time Complexity (Count) Of Finding Third Quarter | 4569 | 10137 | 28845 | 62669 | 98149 |
| Random Input List Time Complexity (Count) Of Finding Third Quarter | 2187 | 5031 | 12641 | 24231 | 34413 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **UNSORTED QUICK SELECT** | | | | | |
| Input Sizes | 1000 | 2000 | 5000 | 10000 | 15000 |
| Sorted Input List Time Complexity (Count)  Of Finding First Quarter | 218875 | 875250 | 5469375 | 21876250 | 49220625 |
| Reversed Input List Time Complexity (Count) Of Finding First Quarter | 289468 | 1192794 | 7258580 | 29036850 | 67066620 |
| Random Input List Time Complexity (Count) Of Finding First Quarter | 1934 | 4434 | 14706 | 35593 | 33398 |
| Sorted Input List Time Complexity (Count)  Of Finding Median | 375250 | 1500500 | 9376250 | 37502500 | 84378750 |
| Reversed Input List Time Complexity (Count) Of Finding Median | 228433 | 954090 | 5828516 | 23077494 | 53606503 |
| Random Input List Time Complexity (Count) Of Finding Median | 3244 | 4052 | 12692 | 47480 | 36966 |
| Sorted Input List Time Complexity (Count)  Of Finding Third Quarter | 469125 | 1875750 | 11720625 | 46878750 | 105474375 |
| Reversed Input List Time Complexity (Count) Of Finding Third Quarter | 135580 | 549703 | 3406413 | 13300456 | 31299191 |
| Random Input List Time Complexity (Count) Of Finding Third Quarter | 3632 | 5543 | 9369 | 28155 | 45649 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **QUICK SELECT (MEDIAN-OF-THREE)** | | | | | |
| Input Sizes | 1000 | 2000 | 5000 | 10000 | 15000 |
| Sorted Input List Time Complexity (Count) | 9830 | 23620 | 67381 | 144116 | 221669 |
| Reversed Input List Time Complexity (Count) | 12638 | 29318 | 82977 | 185947 | 289131 |
| Random Input List Time Complexity (Count) | 8750 | 20825 | 56831 | 131141 | 189965 |

In this section, we tabulate how sorting algorithms behave according to different input sizes and situations created by unsorted elements. At this stage, we tried to find the position of the first quartile, median and third quartile elements within the ordered elements. In the next steps, we will compare how the algorithms perform, at least on our inputs.

We have taken into account the input sizes of 1000, 5000 and 15000 when making the comparisons. We used the elements in the first quartile, median and third quartile indices as the kth element in each. We also added to our comparisons how the algorithms behave according to the state of the inputs.

**Note:** Time complexity axis is the order of complexity of algorithms according to the count results in histogram graphs.

Although there are slight differences when we compare them according to the sorted inputs, in general the time complexity order of the algorithms does not change. When we examined the histograms, we saw that the fastest algorithm is partial heap sort and the slowest algorithm is quick sort.

When we compared against reversed entries, we found that mostly partial heap sort was the fastest. Also, we can say quick sort among the slowest.

A few things caught our attention here. Insertion sort works fast when the number of inputs is small. In addition, partial heap sort and median of three algorithms have been working consistently fast so far.

When we compared the random inputs, a ranking that met our expectations emerged. The remarkable point here is that as the number of inputs increases, the unsorted quick select algorithm works slower than the partial heap sorta. When we looked at the results again, we saw that the partial heap sort was the fastest, and the partial selection sort was the slowest.

Our expectations and results:

* **Insertion Sort** -> We noticed that it is similar to the O(n) growth factor with the theoretical best case in the inputs sorted in the insertion sort. We also saw that the algorithm gave bad results in the reversed input list, so we understood that it has a worst case and a growth factor of O(n^2). In the random input list, we decided that it could be a worst case again. In addition, we saw that the algorithm is fast for small input sizes.
* **Merge Sort** -> We have seen that this algorithm is at an average speed compared to the others. We noticed that the algorithm works at the same speed in all possible situations. This matched with the theoretical knowledge. We think that this is due to the use of divide and conquer structure.
* **Quick Sort** -> We noticed that this algorithm is worst for sorted, reversed and same input situations. When we compared it with the theoretical data as well, we realized that this might be consistent because the results and theoretical information overlapped with time complexity, which showed us O(n^2). In random data, we saw that the algorithm started to work faster.
* **Partial Selection Sort** -> We noticed that the algorithm does not work very fast compared to the others, but we saw that this algorithm works regardless of whether the inputs are sorted or unordered. When we looked at the theoretical information, we found these similarities. We thought that the reason why the algorithm did not work very fast was due to the brute-force structure.
* **Partial Heap Sort** -> We have seen that this algorithm works quite fast compared to other algorithms. What caught our attention was that the algorithm was a bit slow on reversed input lists. The reason for this is due to the max-heap structure we set up. Our outputs were realized in line with theoretical knowledge.
* **Unsorted Quick Select** -> We noticed that the algorithm did not work fast in sorted and reversed, but it worked very fast in random. When we made a comparison with the theoretical data, we realized that these are the worst cases. The reason was that in the worst case, the algorithm theoretically showed a growth factor of O(n^2).
* **Quick Select (Median-Of-Three)** -> We have seen that the algorithm works faster than the others. When we looked at the theoretical data, we saw similarities because the growth factor of the algorithm was O(n) and it did not make much difference according to the state of the inputs. This proves that the algorithm works stably.

**4-) DISTRIBUTION OF TASKS**

We used Discord and Github platforms while doing the project. In general, everyone helped each other in everything, but to be specific, we create the table below.

|  |  |
| --- | --- |
| **NAME AND SURNAME** | **TASKS** |
| Murat ÖZCAN | Coding of Merge Sort  Coding of Partial Heap Sort  Preparing Designing and Coding Part of Report  Organizing All Codes |
| Berkkan RENÇBER | Coding Inserion Sort  Coding Quick Sort  Preparing Tables  Preparing Histograms  Preparing Illustrating and Analyzing Result |
| Yasin Sefa KIRMAN | Coding Unsorted Quick Select  Coding Median-Of-Three  Coding File Operation  Preparing Input Lists  Preparing Outputs  Testing Codes Own Computer  Preparing Designing the Experiment |

**5-) REFERENCES**

* CSE2026 Analysis Of Algorithm Lecture Slides
* Levitin, "Introduction to Design and Analysis of Algorithms", 3/e, Pearson
* <https://www.geeksforgeeks.org/>
* <https://en.wikipedia.org/>
* <http://www.java2s.com/>