

**MARMARA UNIVERSITY**

**FACULTY OF ENGINEERING DEPARTMENT OF COMPUTER SCIENCE ENGINEERING**

**CSE 2046 – ANALYSIS OF ALGORITHMS**

**COMPARING SORTING ALGORITHMS**

**STUDENTS NUMBERS**

Serkan Koç 150118073

İbrahim Hakkı Candan 150118061

Atila İlhan Yatağan 150118033

**Submitted To:** **Due Date:**

Asst.Prof.Ömer Korçak 11.05.2022

Purpose of the Project

The main goal of this project is to design an experiment to compare the theoretical and empirical results of sorting algorithms. According to our observations, we will do some analysis with plots and tables.

Generating Inputs

1. Insertion Sort:

Insertion sort is a decrease and conquer algorithm and a simple sorting algorithm that is based on splitting an array into two parts as sorted and unsorted.

The best case is a sorted array in ascending order for insertion sort. The time complexity of the best case is O(n).

The worst case is a reversed sorted array in descending order for insertion sort. The time complexity of the worst case is O(n2).

For the average case, we considered it should be better than worst case and should be worse than better case, so it may be a random array for this sort. The time complexity of the average case is O(n2).

2. Merge Sort:

Merge sort is a divide and conquer algorithm. It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves. The time complexity of all cases is O(nlogn).

For the best case, we used a sorted array in ascending order algorithm to reduce the comparisons between the elements.

For the worst case, we found a permutation while researching the merge sort. This permutation parses the array into n parts and reorganizes according to the algorithm. We took inputs that have 2n elements.

For the average case, we used random inputs that are generated with different sizes.

3. Quick Sort (pivot is always selected as the first element):

Quick sort is a divide and conquer algorithm. It selects the first element as the pivot and partitions the array.

For the best case, we generated inputs that include the median of the array as the first element. The time complexity of the best case is O(nlogn).

For the worst case, we generated inputs with reversed sorted and sorted arrays. Because the worst case occurs when the largest or smallest element is selected as the pivot. The time complexity of the worst case is O(n2).

For the average case, we used random inputs that are generated with different sizes. The time complexity of the average case is O(nlogn).

4. Partial Selection Sort

The concept used in selection sort helps us to partially sort the array up to kth smallest (or largest) element for finding the kth smallest (or largest) element in an array. Thus, a partial selection sort yields a simple selection algorithm that takes **O(k\*n)** time to sort the array. This is asymptotically inefficient but can be sufficiently efficient if k is small and is easy to implement.

5. Partial Heap Sort

6. Quick Select Algorithm (pivot is always selected as the first element)

7. Quick Select Algorithm (with median-of-three pivot selection)

Deciding on Metrics

There are two alternatives for deciding on efficiency metrics. First alternative is inserting counters into our program and counting the basic operations of sorting algorithms. Second alternative is to time our program for algorithms.

We decided to do the second alternative, we put the ‘start time’ before calling methods of algorithms and ‘end time’ right after the methods. We used nanoseconds to measure the time difference because it is more precise than the millisecond. With this time scale, we observed the difference easily. To plot the measurements, we transformed nanoseconds to milliseconds for getting more understandable observations.

We repeated the execution of the experiment for each algorithm several times to get accurate results. After executions, we took the average of the results we obtained.

Analyzing Results

1. Insertion Sort:

For the best case sorted array in ascending order is appropriate to use. We observed sorted array inputs gave lowest values and time amount is about to increase n times when input size increased as expected. Our input sizes are 100, 1000, and 10000 (2

*Table 1 Insertion Sort Best Case*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | n | ms\_1 | ms\_2 | ms\_3 | AVERAGE |
| best | 100 | 0,0049 | 0,0057 | 0,0042 | 0,0049 |
| best | 1000 | 0,0222 | 0,0224 | 0,0224 | 0,0223 |
| best | 10000 | 0,2359 | 0,2052 | 0,1981 | 0,2131 |
| best | 10000 | 0,2083 | 0,2011 | 0,2073 | 0,2056 |