**** **EXPERIMENT DESIGN**

**Taylan Rojen Döğer 150117055**

**Ömer Öztürk 150117008**

**Designing the Experiment**

First of all, we use Java programming language to evaluate our project.

Secondly, we use System.*currentTimeMillis*() to calculate and compare execution times for each given algorithms. Due to fact that unit time can be affected and changed by computer’s features, we use only 1 computer to gain more accurate results.

About inputs we think that it is better to differentiate **according to size and orders of inputs**. For instance, size can be 1.000, 2.000, 4.000, 6.000, 8.000, 10.000, 12.000, 14.000, 16.000, 18.000 and 20.000. And input orders can be:

**Ordered**: We generate random numbers and put them in increasing order.

**Reverse**-**Ordered**: We generate random numbers and put them in decreasing order.

**Random**-**Ordered**: We just generate random numbers without any intervention.

With this approach, we think that we can see more sensible best-worst case scenarios.

**Coding and Running**

In **inputOrders** class, we basically generate numbers according to orders given above in order to use them according to our choice.

In **QuickSortMedianOfThree, QuickSortFirstElementPivot, HeapSort, MergeSort, CountingSort, InsertionSort** and **BinaryInsertionSort:**

We basically iterate each of algorithms that requested from us and print the execution times.

In **Main**, we take input from user to determine size and order of number sequence. Then we call objects of each classes and use them to send needed information.

**Theoretical Expectations**

**Insertion-sort:**

Best Case: O(n)

Average Case: O()

Worst Case: O(

**Binary Insertion-sort:**

Best Case: O(n)

Average Case: O(

Worst Case: O(

**Merge-sort:**

Best Case: O(

Average Case: O(

Worst Case: O(

**Quick-sort:**

Best Case: O(

Average Case: O(

Worst Case: O(

**Quick-sort (median-of-three pivot):**

Best Case: O()

Average Case: O()

Worst Case: O()

**Heap-sort:**

Best Case: O()

Average Case: O()

Worst Case: O()

**Counting-sort:**

Best Case: O(n + k) = O(n)

Average Case: O(n + k) = O(n)

Worst Case: O(n + k) = O(n)

**Illustrating and Analyzing Results**

In here we will illustrate the results.

**Insertion Sort**

**Binary Insertion Sort:**

**Merge-Sort:**

**Quick-sort**

**Heap-sort**

**Quick-sort (median-of-three)**

**Counting-sort**

In order to obtain more accurate results, we take calculation results by calculating 5 times and get the average of any order type with any order size.

If we plot the results to compare them according to order types:

**Conclusion**

Before starting, we provide you extra text file that includes all sorting algorithms’ order type-size results. We calculate all of them 5 times so that we can get average of them and more accurate results.

**Insertion Sort** has O(n) in best(sorted) case. Extra text file verifies this assumption. But as input increases, we get some open-ended results. We think it comes from computer’s power level&management and also background programs.

**Binary Insertion Sort’s** reverse-ordered(different than others) and random&normal-ordered(pretty much same) results can be seen.

**Merge Sort** has O(nlogn) time complexity of all best-avg-worst cases and we think it is verified.

**Quick Sort** has : O() for both best&avg cases and O( for worst case. We don’t sure that it is verified.

**Quick Sort with Median-Of-Three** has o(nlogn) for all cases and we think it is verified.

**Heap Sort** has O(nlogn) time complexity of all best-avg-worst cases and it is same according to our results.

**Counting Sort** has O(n) time complexity of all our cases so it matches with our experiment.