PROGRAMMING TASK I

In this part of the assignment, you are asked to write different sorting algorithms. The goal of the homework is to exercise the details of sorting algorithms and have hands-on experience with programming with sorting APIs.

IMPORTANT: The homework consists of steps, and you should follow them sequentially, i.e. do not go to the next step before fixing this step. It is recommended that you complete one step per day.

Here are the steps of the homework:

Step 1. In this step, first write a program (i) that loads N integers from a file and (ii) that creates a random array and uses the insertion sort algorithm on Page 251 of the textbook. (In the text file, the first line in the input file should be the size of the array, and each following line should consist of a single integer).

Step 2. In this second step, modify this insertion sort algorithm such that it sorts the input integer array in **descending** order, and start sorting the array **FROM-RIGHT-TO-LEFT**.

Step 3. Use the `Insertion sort` algorithm on Page 251, now to sort an array of **double** values instead of integers.

Step 4. Modify the top-down `Merge Sort` algorithm on page 273 to sort the input integer array in **descending** order**.**

Step 5. In this step, create a `Route` class. Each route should have two fields: a `String` attribute, called *source*, and a second `String` attribute called *destination*. The `Route` class should implement the `Comparable` interface to be able to compare `Route` objects with respect to their source and destination. The `Route` objects are first ordered (alphabetically) with respect to their source. If their sources are the same, they are ordered (alphabetically) with respect to their destinations. Check the example on Page 247 of the textbook for implementing the `Comparable` interface.

Step 6. Use the `Quick Sort` method on Page 289 to sort the given `Route` objects. For this purpose, create 10 different `Route` objects, each with a *source* and *destination* from of one these four cities: Ankara, Istanbul, Antalya, and Izmir. Then, call this method to sort these objects with respect to source and destination.

Step 7. In this step, modify the `Quick Sort` partition method on Page 291 to sort the elements in **descending** order.

Step 8. In this last step, assume that (recursive) `Quick Sort` method receives an `int` parameter `depth`; from the main driver that is initially approximately 2 log N. Modify this recursive `Quick Sort` to call `Merge Sort` on its current subarray if the level of recursion has reached depth. (**Hint**: Decrement depth as you make recursive calls; when it is 0, switch to `Merge Sort`.) Call this new function to sort the elements in **descending** order.

PROGRAMMING TASK II

In this part of the homework, first, you are expected to write a static method that takes an array of n integers and prints the smallest pairwise absolute difference between them along with the corresponding pair of numbers.

Your solution is expected to be composed of two steps:

- Sort the given array.
- Print **the smallest pairwise absolute difference** along with the corresponding pair of numbers. The running time complexity of this step has to be *O(n)*.

If there is more than one pair with the smallest absolute difference, only the ones whose sum is the smallest are printed.

Example input and outputs should be as follows:
->There is one pair with the smallest absolute difference
Input: 23, 1, 5, 102, 34, 99
Output: 3 [99 102]

->There are two pairs with the smallest absolute difference, only the one whose sum is the smallest is printed
Input: 23,1,4,102,34,99
Output: 3[1 4]

After writing your method as described above, you are expected to make its experimental run time analysis in the four different settings where in each setting you are going to use a different sorting algorithm:

- Setting 1: `Selection sort` (use the algorithm on Page 249 of the textbook)
- Setting 2: `Insertion sort` (use the algorithm on Page 251 of the textbook)
- Setting 3: `Merge sort` (use the algorithm on Page 273 and 271 of the textbook)
- Setting 4: `Quick sort` (use the algorithm on Page 289 and 291 of the textbook)

Note: To make your work easier while testing your method in four different settings, you could design your method in a way that it takes an additional input specifying which sorting algorithm is to be used.

In this way, you are going to observe and report how the performance of your method changes when different sorting algorithms are used. Based on the experimental results that you obtain, comment on the complexity of your method using each sorting method.

While doing experimental analysis of the method in each setting, you are expected to run it for **different sizes** of **ascending/descending** and **random ordered** integer arrays.

1. To make experimental analysis of your method, create a new *SortingAlgorithmTester* class, and within the main method create an array with (i) ascending, (ii) descending, or (iii) random integers, to be sorted with each algorithm. You can get a random list using the java class Random, located in *java.util.Random.*

2. As you test your method in different settings, collect time measures. Time is measured in milliseconds. You should use the `System.currentTimeMillis()` method, which returns a long that contains the current time (in milliseconds). It is recommended that you do several runs of your method on a given array and take the median value (throw out the lowest and highest time). In order to get the most accurate runtimes possible, you may want to close other programs that might be taking a lot of processing power and memory and affecting the runtimes of your method.

3. For your homework, you should submit the corresponding .java file of your method, *SortingAlgorithmTester.java* file and your at most 3-page report in PDF format. The report should give a detailed explanation of your experimental setup, procedure, and experimental results showing how the performance of your method changes when different sorting algorithms are used for different types of inputs (ascending / descending /random ordered). You should also write the individual steps that you took to complete the homework.

You are expected to add visualization (table, plot, graph, etc.) showing the time complexity of your method when different sorting algorithms are used with different size and type of inputs (e.g. ascending, descending, random input). For this purpose, you can use tools like MS Excel, R etc.