Lab Assignment 11, Object-Oriented Programming, CSE 271, Spring 2020
Department of Computer Science and Engineering, Miami University
Sorting Algorithms

Lab Overview:

A sorting algorithm is used to arrange a given array, list, or a collection of elements in ascending order such that,

$$entry_1 \leq entry_2 \leq entry_3 \leq \cdots \leq entry_n$$

where, n is the number of elements. In this lab, you will implement four sorting algorithms we discussed in the lecture, selection sort, insertion sort, quicksort and merge sort. You will also compare their performance (time takes to sort) for different size of arrays.

Sorting Algorithms:

Please download the **Sorting.java** file from Canvas. You need to implement the four sorting algorithms mentioned above. That is, you need to define the following four methods,

- `public static void selectionSort (int[] array)`
- `public static void insertionSort(int[] array)`
- `public static void quickSort(int[] array)`
- `public static int[] mergeSort(int[] array)`

You will find the algorithms and code for these sorting algorithms in the textbook. You are welcome to use the code from the textbook, but you must follow the given structure of the methods in the Sorting class. You must not change the header of any of the given methods.

The given methods are just skeleton implementations for four sorting algorithms that throw an *UnsupportedOperationException*. You will find a couple of helper methods for quick sort and merge sort methods that you also need to define. Your job is to complete all four sorting methods and their helper methods.

You will find the instructions for the following helper methods as a comment in the respective methods. Comment out the following line as you start to define the incomplete methods: `throw new UnsupportedOperationException().`

The list of helper methods are given below that you need to define except for the swap() method.

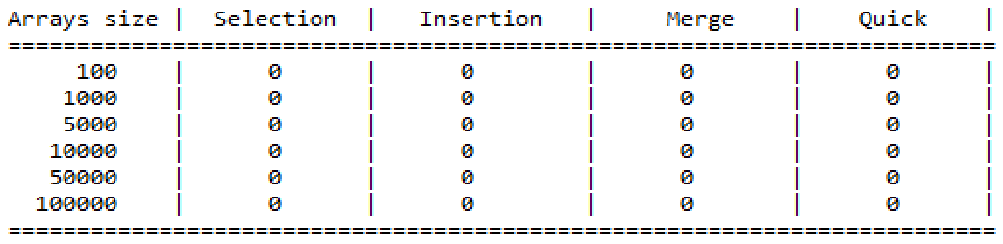
- `private static void quickSort(int[] array, int begin, int end)`
- `private static int partition(int[] array, int begin, int end)`
- `private static void swap(int[] array, int i, int j)`
- `private static int[] mergeSort(int[] array, int begin, int end)`
- `private static int[] merge(int[] left, int[] right)`

There is also a method void swap(int[] array, int i, int j) that swaps two elements of an array, that you may find useful. It is already fully implemented.

Lab Assignment 11, Object-Oriented Programming, CSE 271, Spring 2020
Department of Computer Science and Engineering, Miami University
Sorting Algorithms

Running Time Table:

Please download the *RunTime.java* class from Canvas, that reports the run time of the sorting algorithms you implemented in Sorting class. It displays the running time in the following format:



```
Arrays size |  Selection   |   Insertion    |     Merge     |    Quick     |
==========================================================================
    100     |      0       |      0         |       0       |      0       |
   1000     |      0       |      0         |       0       |      0       |
   5000     |      0       |      0         |       0       |      0       |
  10000     |      0       |      0         |       0       |      0       |
  50000     |      0       |      0         |       0       |      0       |
 100000     |      0       |      0         |       0       |      0       |
==========================================================================
```

Once you are done with implementing the **Sorting** class you need to run **RunTime**.java. You only need to replace the following statement in **RunTime**.java:

```
System.out.format("%7d      |", 0);
```

with the following line **for all four algorithms** so that it computes the executing time.

```
System.out.format("%7d      |", endTime - startTime);
```

Important Note:

Make sure the file name is correct. No late submission. You will get zero for late submission.

Submission:

You should submit your final Sorting.java and RunTime.java to Canvas. You don't need to submit Javadoc.

Grading:

Task	**Points**
Selection Sort	20
Insertion Sort	20
Quick Sort	25
Merge Sort	25
Runtime report	10
Total	**100**