Assignment #6 - Sort Comparison

Objectives:

The main objectives of this assignment are:

- Implementing and testing 5 different sorting algorithms
- Perform the test testing and plot the time testing results
- Compare the growth rates of different sorting algorithms
- Experience the big difference of $O(n^2)$ vs. O(n lg n)

Allowed Imports:

java.util.Arrays;

In merge sort, you may import java.util.Arrays to use method CopyOfRange()

The Assignment:

This project is to implement different sorting algorithms and compare their performance.

1. Download and import the zip file.

   A new project named Sorts should be added to the workspace. Please take some time to study the starter code.

2. **Copy** your `HeapAPQ` implementation from previous project into the current project `src/cs2321 folder.`

   Please contact instructor/TAs to get the approval to use the HeapAPQ in the library cs2321util packages.

3. Implement the 5 sorting algorithms correctly. Note some test cases have been provided to you under test dir.

   1) InPlaceSelectionSort.java
   2) InPlaceInsertionSort.java
   3) HeapPQSort.java
   4) MergeSort.java
   5) QuickSort.java

   Important notes and FAQ for implementation:

1) Don't cast the generic type E to int in your code. We will test with data other than int type.
2) What does this <E extends Comparable<E>> mean? It means that E is not any type, but a type that extends Comparable.
3) How to compare two objects that extends Comparable? Use compareTo method E1.compare(E2).
4) How to create an array of generic type <E extends Comparable<E>> ? (E[]) new Comparable[size]
5) How to fix stack overflow error? Change the eclipse java virtual machine stack size
   Run -> Run configuration -> Auguments -> VM arguments
   add -Xss12800k

4. Analyze the time complexity of sorting algorithms.

- Annotate the running time complexity with `@TimeComplexity` annotation and `@TimeComplexityExpected` annotation if it is needed.
- Write justification comments.

5. Time Testing

After implementing the sorting algorithms correctly, you need to test how long it will take to sort random numbers. The time testing code has been implemented in **`SortTiming.java`.**

1) It will take somewhere between 10-20 minutes to finish all time testing.
2) Please consider running the time testing part on the lab computer if your laptop is slow.
3) Read the code carefully and understand what it does.
4) You might want to run a certain type of sort first, not all of them at once. You can do this by commenting and uncommenting part of the codes. Then run the main method.
5) You will see the progress of the code as it runs through all 5 sorting algorithms for different data size, from 1,000 to 100,000 elements. It measures the time in million second.
6) It outputs the result to the console **AND** to a file called `output.txt` in the project folder.

After the running of SortTiming.java is complete, load the data file output.txt to Excel or other tools to plot the following two charts. You need to put the data from output.txt and the two graphs in ONE pdf file.

If you don't know how to create a line graph, please watch the short video file first.

Create a line graph of the time testing for all 5 sorting algorithms:

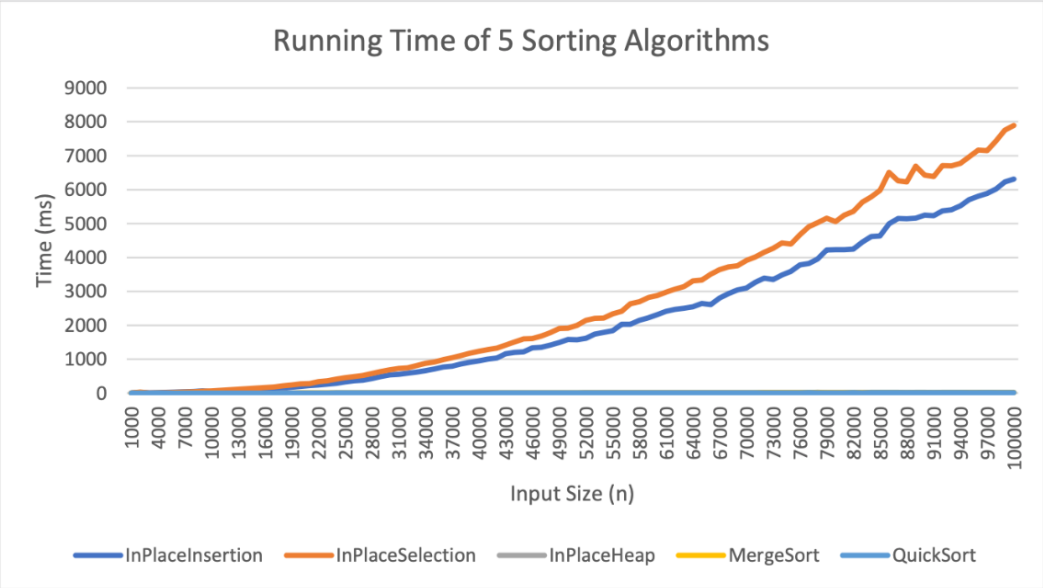
- The horizontal (x) axis should be input size

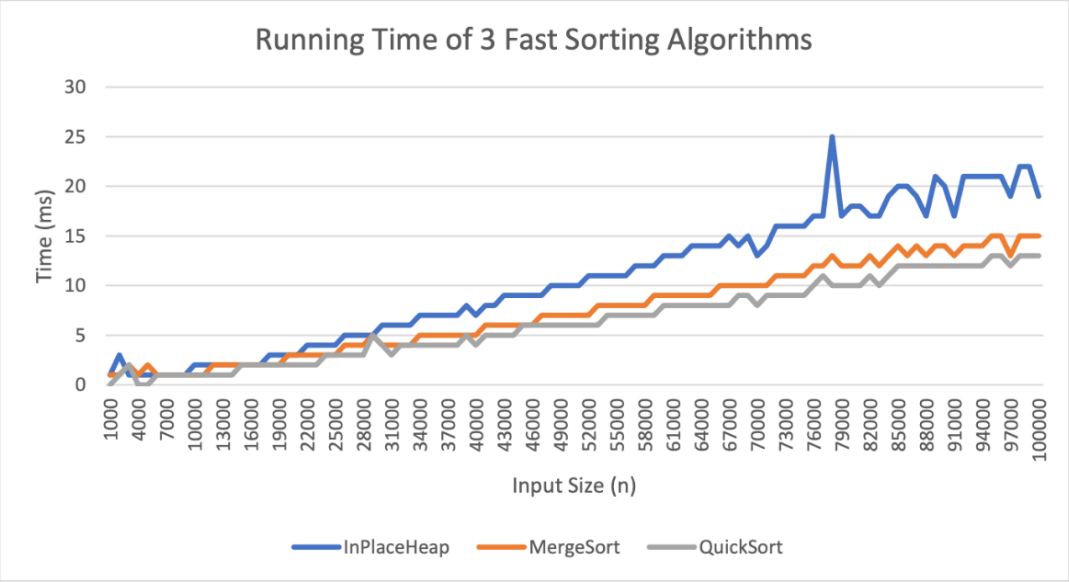
- The vertical axis (y) should be "time in ms".
- The title should be "Running Time of 5 Sorting Algorithms"

Create a line graph of the time testing for the three fast sorting algorithms: MergeSort, QuickSort, HeapPQSort :

- The horizontal (x) axis label should be "input size"
- The vertical axis (y) label should be "time in ms".
- The title should be "Running Time of 3 Fast Sorting Algorithms"

Here is a sample of what the graph should look like. It is ok to have spikes here and there. But the trend should be obvious. Please note that the sample graph has “inplace heap sort”, but you are asked to do heapPQSort for this assignment.





Submission:

Please copy your report prog6.pdf in the src folder of the project, then export the project and save it as prog6.zip, submit it. Submit the pdf file, which contains the data and the graphs, before the due date.

Grading Criteria:

1. Sorting correctly: 60 points

   TA's check for appropriate implementation of each algorithm. If your implementation is wrong, for example, you use bubble sort in place of insertion sort, you will lose all points even you passed the test cases: -60 points

2. Two line graphs: 15 points
3. Time Complexity annotation for 5 sorting algorithms: 10 points
4. Time Complexity Justification, Comments and Style: 15 points total. 3 points for each sorting algorithm.

Here is what you will see in your grading report:

```
/ 10.0  Sorting Complexity Annotations
InPlaceSelectionSort
InPlaceInsertionSort
HeapPQSort
MergeSort
QuickSort
10.0 / 10.0  InPlaceSelectionSort Test Results
 / 2.0  Timing test for InPlaceSelectionSort() (above graph)
 / 0.0  Deduction for wrong algorithm  (negative points, up to -12p)  - selection sort
 / 3.0  TCJ, Comment, Style
10.0 / 10.0  InPlaceInsertionSort Test Results
 / 2.0  Timing test for InPlaceInsertionSort() (above graph)
 / 0.0  Deduction for wrong algorithm  (negative points, up to -12p)  - insertion sort
 / 3.0  TCJ, Comment, Style
10.0 / 10.0  HeapPQSort Test Results
 / 2.0  Timing test for HeapPQSort() (above graph)
 / 0.0  Deduction for wrong algorithm  (negative points, up to -12p)  -  heap PQ sort
 / 3.0  TCJ, Comment, Style
10.0 / 10.0  MergeSort Test Results
 / 2.0  Timing test for MergeSort() (above graph)
 / 0.0  Deduction for wrong algorithm  (negative points, up to -12p)  - merge sort
 / 3.0  TCJ, Comment, Style
10.0 / 10.0  QuickSort Test Results
 / 2.0  Timing test for QuickSort() (above graph)
 / 0.0  Deduction for wrong algorithm  (negative points, up to -12p)  - quick sort
 / 3.0  TCJ, Comment, Style
 / 15.0  Time Testing Result in the pdf file (data + 2 line graphs)
```