Data Structures – Assignment 5

IDC, Spring 2019

Submission day: 7.5.2019– (you can use an extension and submit by 10.5.2019)

Honor code:

1. Do not copy the answers from any source
2. You may work in small groups but write your own.
3. Cheating students will face Committee on Discipline.

Introduction:

In this assignment you will write Java implementation of several algorithms studied in class:

- Quick sort (with an arbitrary pivot)
- Quick sort (with a random pivot).
- Merge sort
- Counting sort
- Insertion sort (as introduced in the previous assignment)

After completing these functions, you will be able to compare their performances on different types and sizes of input.

Do not use any external libraries, except the ones provided.

Do not change the package declaration.

The assignment:

Open the provided *SortingComparisons.java* file. It contains the skeleton of several functions as well as some testing functions.

Start by implementing the functions. It is highly recommended that you thoroughly test each function before proceeding to the next function.

When you are satisfied with the operation of your functions, you will be able to run the *main* function (also provided) to compare between the performance qualities of the different algorithms. The runtimes will be graphically depicted for visualization purposes. In order to view the graphs you will need to add several classes to your project.

The files *plotter.jar, jfreechart-1.0.17.jar, jcommon-1.0.21.jar* contain all the necessary classes. You will need to let Java know where they are. In Eclipse, right click your project and select **properties**, choose **Java Build Path** on the left menu and from there press the **Libraries** tab. Once there, press the **Add External JARs…** button. Choose all 3 *Jar* files and verify they appear in Eclipse. This will allow you to compile the *SortingComparisons* class and execute all functions.

Remarks:

- When implementing *quickSortArbitraryPivot* choose the pivot, in an arbitrary way, to be the rightmost element in the current subarray.

- When implementing the *quickSortRandomPivot* algorithm, the pivot should be chosen random using the *random* function of Java.

- When implementing the *countingSort* algorithm, there is no need to make it stable, as shown in class. In particular, you are only allowed to use one auxiliary array of size k.

Comparisons:

The code provided will execute and plot several comparisons:

- insertion sort vs quick sort on random arrays.
- merge sort vs quick sort on random arrays.
- insertion sort vs quick sort on sorted arrays.
- quicksort with an arbitrary pivot vs quicksort with random pivot on a random array.
- counting sort vs quicksort on integer arrays with elements whose value is smaller than n, the size of the array.

In each plot, the runtime of each algorithm will be represented by a simple curve. A green curve representing the function $nlog(n)$ is presented for reference.

The constants at the top of the class determine the input sizes for each comparison. You should experiment with different sizes to get a feel for how the runtimes behave.

In addition to submitting your code, **submit a text file named *Explanation.txt*.** The file should contain 5 paragraphs, one paragraph for each comparison. In each paragraph provide a short explanation for the shape of the obtained graph.

We supply an example of a graph and a text explaining the graph. Use this as a demonstration of how your explanation file is expected to look. The graph shows the runtime of accessing the middle element in an array against the runtime of accessing the middle element of a linked list.

Submission:

- You may submit the assignment in pairs, this is not mandatory but recommended.
- Make sure your code is presentable and is written in good format. Any deviations from these guidelines will result in a point penalty.
- Make sure your code can be compiled. **Code which does not compile will not be graded.**
- Your grade will be primarily based on the correctness of your code.
- Your code may be tested for different possible edge cases; make sure to handle such cases. **It is your responsibility to test your code thoroughly before submission.**
- Submit a zip file with the following file only:
    - SortingComparisons.java
    - Explanation.txt
- The name of the zip file must be in the following format "ID-NAME.zip", where "ID" is your id and "NAME" is your full.
- For example, "03545116-Allen_Poe.zip".
- If you submit as a pair the zip file should be named in the following format "ID-NAME-ID-NAME.zip".
- For example, "03545116-Allen_Poe-02238761-Paul_Dib.zip".