Data Structures – Assignment 5
IDC, Spring 2021

Lecturer: Prof. Yael Moses

TAs: Dan Mikulincer, Yael Hitron, Yuval Belfer

Submission day: 14.5.21 – (you can use an extension and submit by 18.5.21).

Honor code:

- Do not copy the answers from any source.
- You may work in small groups but write your own solution - write whom you worked with.
- Cheating students will face Committee on Discipline (COD).
- Do not forget – you are here to learn!

Introduction

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

- Weird sort
- Quick sort (with an arbitrary pivot)
- Merge Sort
- Bubble Sort

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

Do not import or use any libraries, except the ones provided.

The assignment

Open the provided *Sorting.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.

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 sorting class and execute all functions.

Remarks

- When implementing *quickSort* choose the pivot in an arbitrary way, to be the right-most element in the current subarray.
- When implementing *bubbleSort* you need to add the following optimization: If at any iteration, no inversions were made (which means the array is sorted), then the function should stop. In particular, your code should run in linear time if the array is sorted.
- The *weirdSort* algorithm should be implemented according to the pseudo-code introduced in the recitation.

Comparisons

The code provided will execute and plot several comparisons:

1. bubble sort vs quick sort on random arrays.
2. merge sort vs quick sort on random arrays.
3. bubble sort vs quick sort on a sorted array.
4. weird sort vs bubble sort.

In each plot, the runtime of the algorithms (measured in milliseconds) will be represented by a simple curve. A green curve representing the function $n \log(n)$ 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 4 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.

Before submitting this assignment, take some time to inspect your code, check that your functions are short and precise. If you find some repeated code, consider making it into a function. 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**.

Submit a zip file with he following files only:

- Sorting.java

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 name. 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".