9 Sorting: Time Trials; Generating Javadocs

Goals

The first part of this lab will let you experience first-hand the differences between the time required to sort a (medium) large sets of data when using different sorting algorithms. It will also illustrate a technique one can use to measure the timing of algorithms in general.

In the second part of the lab you will learn how to generate the *Javadoc* documentation, and practice reading *Javadoc* style documentation for programs. **After today we require that all your programs include documentation written in the *Javadoc* style.**

9.1 Sorting: Time Complexity of Algorithms

One of the concerns of program designers is the efficiency of the program. Programs that deal with large amounts of data and complex calculations may take a long time to complete. The designer must be able to judge beforehand whether the program can complete its task in a reasonable time, and look for ways to improve the program efficiency when necessary.

We will use several different sorting algorithms (programs that produce a sorted list of data from the given unsorted one) to illustrate the need for understanding program complexity, and will learn how we can measure and compare the timing performance of several alternative solutions to the same problem.

9.1.1 Selection sort

In Lab 8, the last part (8.2) asked you to implement the *Selection sort* algorithm. It looks like nobody had the chance to complete that part of the lab.

Start this lab by finishing the last part of Lab 8 that deals with the *Selection sort*.

9.1.2 Quicksort

In this part you will learn a bit more about the QuickSort sorting algorithm and do some preliminary timing measurements for a couple of sorting algorithms.

Start with a new project **SortingAlgorithms** and import all files in **Sorting.zip** file.

1. Save the file **citydb.txt** in the same directory where *Eclipse* has your `src` and `bin` folder.

   Set up the *Configuration* as usual and run the project. It will come up with a file dialog asking you to select the input file. Choose the file **citydb.txt**. Now look at the results. Besides the tests, the output includes the timing results for the insertion sort included in the code.

   Change the number of data items to be read to 30000 and run the tests again. Be patient, it may take a while.

2. Add the code for the two variants of *QuickSort* posted on the course wiki. Add the code that invokes each algorithm with the data from the **citydb.txt** file. Add the necessary test cases. Finally add the code that measures the time needed to complete each algorithm.

   Run the program and observe the timing results.

3. The **Algorithms.java** files includes the definition of a list of items of the type `T`. Add the method `quicksort` to these classes and the interface - using the technique similar to what we did in class and what you did in Fundies 1 last semester.

4. Add tests, the code that invokes the algorithm with the full database of cities, and the code that measures the timing.

   Run the program and observe the timing results.

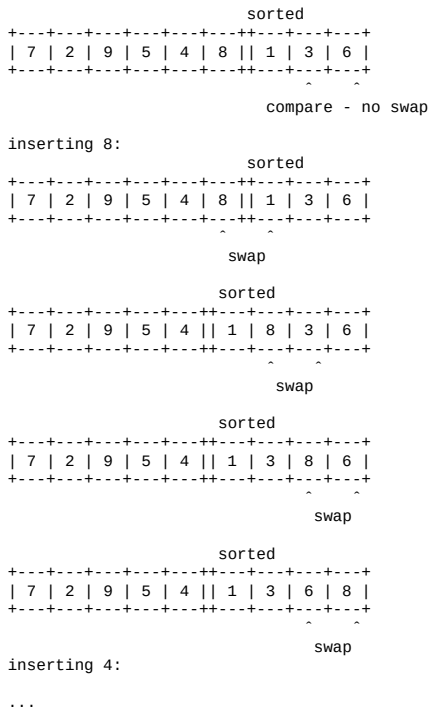
5. The last part is a *pencil and paper* exercise.

   For the following starting data show how each of the three versions of the *QuickSort* proceeds. We have shown you the beginning of the *pencil and paper* analysis of the given *Insertion sort* algorithm.

```
+---+---+---+---+---+---+---+---+---+---+
| 7 | 2 | 9 | 5 | 4 | 8 | 3 | 1 | 6 |
+---+---+---+---+---+---+---+---+---+---+

inserting 1:
                                    sorted
+---+---+---+---+---+---+---+---+---++---+
| 7 | 2 | 9 | 5 | 4 | 8 | 3 | 1 || 6 |
+---+---+---+---+---+---+---+---+---++---+
                                      ^   ^
                            compare - no swap

inserting 3:
                                  sorted
+---+---+---+---+---+---+---+---++---+---+
| 7 | 2 | 9 | 5 | 4 | 8 | 3 || 1 | 6 |
+---+---+---+---+---+---+---+---++---+---+
                              ^   ^
                             swap
```



9.2 Documentation

This is partly

For this lab download the following files:

- The file *Balloon.java* — our sample data class
- The file *TopThree.java* that has been used to practice working with `ArrayList` in imperative style (using mutation).
- The *Examples.java* file that defines examples of all data and defines all tests.

Create a new **Project** *Lab9* and import into it all files from the zip file. Import the tester.jar and colors.jar.

9.2.1 Generating Documentation

Once Eclipse shows you that there are no errors in your files select **Generate Javadoc...** from the **Project** pull-down menu. Select to generate docs for all

files in your project with the destination *Lab9/doc* directory. Make sure you select all files for which you wish to generate the documentation.

You should be able to open the *index.html* file in the *Lab9/doc* directory and see the documentation for this project. Compare the documentation for the class `Balloon` with the web pages. You see that all comments from the source file have been converted to the web document.

Observe the format of the comments, especially the `/**` at the beginning of the comment. If you do not understand the rules, ask the TA or one of the tutors, or experiment with new comments. From now on all of your work should have a proper *Javadoc* style documentation.

9.2.2 Stack, Queue, Priority Queue, LinkedList; Vector

Look up the documentation for the following Java classes and interfaces: `Stack, Queue, PriorityQueue, List, LinkedList` and `Vector`. Identify which of them represent interfaces, which represent abstract classes, and which provide a complete implementation that you can use in your program. Draw a class diagram that shows the relationship between these classes and interfaces.