Semester assignment II
Inf102 fall 2011
Deadline: 22:00hr Thursday 10. of November

There will be two mandatory assignments and one classroom test in this course. In total these will count for 30% of the final grade. Both mandatory assignments have to be passed in order to qualify for the exam but if the classroom test has a better score than one of the mandatory assignments then this grade will be used in stead. In other words each mandatory assignment counts for 15% of the final grade.

In order to get the mandatory assignment evaluated the following conditions have to be met:

- **Overview**
  A text document containing an overview of the completed assignment. This should contain your name, e-mail address, and a short description of the added files.

- **Source code**
  During the evaluation we will check that the source code compiles and produces correct answer for a set test data not available to the students. All java files necessary to compile the project have to be included in the submission, but do not add .class files. Other files that are of importance to the project can also be included in the submission.

  All source code should be properly commented and indented in a way that makes it easy to read and understand how the code is supposed to work.

- **Analysis**
  Add a separate text document(any normal format is fine) where the running time for each implemented method is given in big O-notation. If the method is not trivial it is also required to justify the running time. Points will be given according to the quality of the analysis but the most important thing is to use correct arguments.

The hand in should consist of a single zip-file with a name composed by your first and last name. If your name is Jon Doe and you want to deliver files MyBinaryTree.java, MyOblig1.java, oversikt.txt from a Linux system then the zip file with name JonDoe.zip can be crated by the command
"zip JonDoe MyBinaryTree.java MyOblig1.java oversikt.txt". Bytecode or files of type *.class shall not be included in the zip file. The assignment should be delivered through the course web page. On the main page of the course there is a rectangle at the right named assignments. In this rectangle there is a folder called

Assignment folder and in this folder there is another folder called Assignment 2, where the assignment should be delivered. Remember that the code handed in will be recompiled and tested on new data which are not available for the students.

Individual work

Each student is responsible of writing and completing his or here own code. If two delivered assignments are so similar that we suspect cheating then both will fail. It is your own responsibility to ensure that noone copies your code. The same band also holds for copying from books, internet, or other sources.

Task 1

The first task is to sort a set of numbers or integers. In the file storage there are two files named "numbers1.txt" and "numbers2.txt" containing respectively 1000000 and 10 integers. Objective of this task is as follows:

- **Load** these integers from file into a suitable data structure. The file to load should be given with a program parameter, e.i. one can run the program like "java Task1 numbers1.txt".
- Sort the integers in non-decreasing order using **bucket**-sort.
- Sort the integers in non-decreasing order using regular **quick**-sort. By regular quick-sort we mean that sets L,E,G are defined and used in class.
- Verify that the two algorithms output the **same** sequence of integers.
- Use the method System.currentTimeMillis()to measure the time used by the two algorithms and print a **report** of this. Be careful to only time the sorting part.
- (Extra,**optional**) Implement in-place quick-sort and compare the running time with the two other algorithms.

The format of the input files are as follows: A sequence of integers is given by blank space separation. First integer gives the number of integers following it in the file and will not be among the integers to be sorted. Below is an output example for Task1.

```
size = 1000000
Load complete
Bucket-sort completed
Regular quick-sort completed
Outputs of Bucket-sort and Quick-sort are equal.

We are sorting 1000000 integers in this test:
Bucket-sort completed in 28 ms.
Quick-sort completed in  748 ms.
```

Task 2

In the file storage there are three files “Node.java,Edge.java,Graph.java” containing a set of method declarations that are not implemented. Implement these to get a working graph data structure. The graph data structure should be a **adjacency-list** structure, and all methods should be implemented such that the best possible running time in big O-notation is achieved. These data structures will come in handy in Task 3.

Task 3

Final task is to compute a route between pairs of airports. Files “name.data” and “edge.data” describe a graph where vertices represent airports and edges represent flights between pairs of airports. In this task class “LoadData” found in the file storage can be used to read the data from files “name.data” and “edge.data”.

Use the data structure created in Task2 to make a program that computes and print the shortest paths between two airports in the graph. Like in task 1 we will use command line parameters to give the start and stop destinations. Start and stop destinations are given by the integer id of the airport. From the example below we can notice that Bergen has id 503 while Vienna has id 410.

Below is an example output when running command
“ java Task3 503 410 448 445 1082 503 1016 410”.

```
2631 vertices loaded
7558 edges loaded

Printing path from [Bergen, Norway] to [Vienna, Austria]
[Bergen, Norway]
[Gainesville, FL]
[Astoria/Seaside, OR]
[Vienna, Austria]

Printing path from [Lyon, France] to [Las Piedras, Venezuela]
[Lyon, France]
[Lexington/Frankfort, KY]
[Fayetteville, NC]
[Las Piedras, Venezuela]

Printing path from [Cagliari, Italy] to [Bergen, Norway]
[Cagliari, Italy]
[Atlantic City, NJ]
[Gainesville, FL]
[Bergen, Norway]

No path exists from [Vienna, Austria] to [Kristiansand, Norway]
```