Data Structures and Algorithms I 26.1.2017

Exercise 1

These first tasks can/should be responded more with common sense that on the basis of CS studies. Draw a picture of each task.

1. You'll get a task to *sort manually* stacks of A4-papers to alphabetical order according to a name on each paper. What technique/method you would use if the number of papers in a stack is

   (a) 10

   (b) 100

   (c) 1000

   (d) 10000

   Estimate how long time the sorting of each stack takes (in seconds, minutes, or hours). Calculate the time usage from your sorting method. Estimate the time needed for each *simple* operation (such as comparing the names of two papers) and the calculate how many such operations you'll do in each stage.

2. Calculate how long it would take for each task of 1 using *selection sort.* At each stage of the selection sort, we look for the smallest element of the remaining stack, and set it upside down on top of the stack of the sorted papers. Use paper scanning speed 1 sec / sheet. What are the best and the worst of cases of this algorithm, and how long time they will take?

3. Again we have the stacks of task 1. How long it would take to *find* a given paper (by name) when

   (a) the papers are not sorted

   (b) papers are in sorted order

   Reason again your time estimate. If the stack a not initially sorted, then it is not sensible to sort them in order to make just one search. However, if we are going to make thousands of searches, the sorting helps a lot. For each stack size, what is the limit on the number of searches before we should do the sorting to reduce the overall work?

4. We still use the paper stacks of task 3, but this time the printer did malfunction so that randomly about 20% of the papers are empty (extra blanc papers). How this impacts to time of task 3b? When (for how many searches) you should remove the blanc papers from disturbing?

In the following ”write an algorithm which” tasks, you should make a functional Java method that gets parameter input and possibly returns a new collection in accordance with the assignment, but does nothing else. Thus, for example, does not alter the input data (unless requested to do so) or print anything (at least in the final version). Please take the input generating main program from course web page. For exercise classes, we'll show your answers using projector, thus bring it on by saving it to the cs.uef.fi server, somewhere else in the network, or memory stick.

5. Write and algorithm (Java-method) that gets two parameter integer arrays (*Integer*[] A, *Integer*[] B) and which creates and returns a new integer array that has all such elements that are contained in array A, but are not contained in array B (i.e., the *difference* of the arrays). What is the time complexity of your algorithm? Can the time complexity be improved? Take a main program and skeleton from course www-page.

6. Unless you know it already, familiarize yourself with class *ArrayList*. See `http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html`. Modify the algorithm of previous task 5 to work with *ArrayList*s instead of arrays. Do not use ready *.removeAll*() -method. What is the time complexity of your algorithm? Can the time complexity be improved? Take a main program and skeleton from course www-page.