Aalto University School of Science
Department of Computer Science
CSE-A1140 Data Structures and Algorithms

EXAMINATION
17.2.2015

Write on each paper your name, student number, degree programme, and the course code with name. Also write the date, hall, the number of papers you return, and your *signature*. Using any extra devices is prohibited in this examination.

1) Ten Questions (10 x 1p)

This is a <u>compulsory part</u> of the final exam. You need to get at least 5p out of the maximum 10p so that the rest of the exam will be checked. However, this part alone is not enough to pass the whole exam. On the other hand, in order to get 5p, you are not required to give "the exactly correct answer", but more or less show that *<u>you have understood the functionality of the code fragments</u>* related to this part. Thus, pay attention to the reasoning. Refer to the code line numbers if possible.

In the following, you can see two algorithms computing power function (x^n) for integers x and n. Read through all the questions below without answering them and after that <u>familiarize yourself with the code throughout</u>. After this, answer all the questions and take time for pondering and explaining your reasoning. Note, however, that all the questions refer to the given algorithms. In addition, the *argumentation* is the only thing that matters for the points!

```
1 int pow1(int x, int n) {
2     if (n == 0)
3           return 1; else
4     if (n == 1)
5           return x; else
6     if ("n is odd")
7           return pow1(x², ⌊n/2⌋) *
x; else
8     if ("n is even")
9           return pow1(x², ⌊n/2⌋));
10 }
```

```
11 int pow2(int x, int n) {
12          p = 1;
13    for (int i=1; i<=n; i++)
14          p = p * x;
15    return p;
16 }
17
18
19
20
```

a) *Describe* in your own words how `pow1` works.
b) *Describe* in your own words how `pow2` works.
c) *In which code lines* and *how many multiplications* `pow1` does? *Give an example* in case the algorithm is called with parameter *n=9*.
d) *In which code lines* and *how many multiplications* `pow2` does? *Give an example* in case the algorithm is called with parameter *n=9*.
e) *Analyse* the time complexity of Algorithm 1 in terms of the input size *n*.
f) *Analyse* the time complexity of Algorithm 2 in terms of the input size *n*.
g) *Argue* whether it is *true or false:* Algorithm 1 is more efficient than Algorithm 2.
h) *Argue* whether it is *true or false:* Algorithm 1 computes the same function than Algorithm 2.
i) *What* would be *the order of multiplications* in Algorithm 1 if the line 7 would be changed to "`return x * pow1(x*x, n/2); else`"? *Give an example*.
j) Is it possible to replace the for-loop in Algorithm 2 with another loop construction? *Argue* either why not or give an example how to replace it (write the algorithm anew).

Bonus:
k) *Ponder and compare* the memory consumption of Algorithm 1 and 2.

2) Terminology (2p + 2p + 2p + 2p)

Define briefly the following *concepts* (4 x 1p). *Give an example* of each (4 x 1p).

a) Abstract Data Type
b) Stack
c) Selection
d) A stable sorting method

3) Dictionaries/Search structures (2p + 2p + 2p + 2p + 2p)

Compare balanced search trees with hashing. Outline your answer as follows:

a) *Define* Abstract Data Type *Dictionary* (Search structure).
b) *Mention* at least *one balanced search tree*, and *one hashing method.*
c) *Which operations* (give at least 2) can be implemented more efficiently in balanced search trees than by hashing? Argue your answer.
d) *Which operations* (give at least 2) can be implemented more efficiently by hashing than in balanced search trees? Argue your answer.
e) *Suggest a method* to combine a balanced search tree and a hashing method in order to come up with an even better search structure.

You can choose to do EITHER 4.1 OR 4.2. If you do both, you get points only from one of them (the best points count).

4.1) Priority queues (2p + 2p + 4p)

a) *Define* the term *priority queue*.

b) *Mention an algorithm* that utilizes a priority queue as an auxiliary data structure. *Describe briefly* for what purpose the algorithm needs it.

c) *Show step by step* how the linear-time *BuildHeap* algorithm (a.k.a., *FixHeap*, *Bottom-Up Heap Construction*) works if the input array has the keys 7, 2, 25, 1, 10, 23, 14, 20, 3, 5, 6, 15, and 13. The heap requirement is that each node is smaller than its children. **Note:** this is not the same as inserting all the keys one at a time!

4.2) Determining Minimun Spanning Tree (4p + 4p)

a) Explain *an algorithm* that computes the minimum spanning tree for a graph. Argue what is the time complexity of this algorithm, if the graph has V vertices and E edges?

b) Lets consider the following undirected graph that has the vertices A-F and edges AB(2), AD(2), AE(6), BC(4), BE(3), CE(1), CF(4), DE(1), EF(3). The weight of an edge follows in the parenthesis. Show how the algorithm you just explained finds the mininum spanning tree for the given graph.