Data Structures and Algorithms II — 30./31.3.2017

Exercise 1

These first tasks are partly recap of Data Structures and Algorithms I course. Making these will orient you back to the subject and then making following exercises will be easier. Draw a picture of each task.

1. Draw the recursion trees of following algorithms and calculate the asymptotic time complexity as a function of parameter n.

```
void A(int n) {
    if (n > 1) {
        A(n/2);
        A(n/2);
} }
```

```
void B(int n) {
    if (n > 1) {
        B(n−1);
        B(n−1);
} }
```

```
void C(int n) {
    if (n > 1) {
        int a = 0;
        for (int i = 0; i < n; i++)
            a += i;
        C(n−1);
} }
```

```
void D(int n) {
    if (n > 1) {
        int a = 0;
        D(n−1);
        for (int i = 0; i < n; i++)
            a += i;
        D(n−1);
} }
```

In the following ”write an algorithm” tasks, you should make a working 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. At exercise classes, we'll show your answers using projector, thus bring it with you by saving it to the cs.uef.fi server, somewhere else in the network, or a memory stick.

2. Write an algorithm that finds which element of a given collection has most occurrences. The parameter is a *Collection<E>* and the return value is an element (of type E) that has most occurrences in the collection. If there are multiple elements with equal number of occurrences, return any of those. If the collection is empty, return *null*. Compare elements with *.equals*() operation. Use a mapping (*Map<E, Integer>*) as a helping structure. What is the time complexity of your algorithm?

3. Compare the running time of task 2 when the helping data structure is (a) *HashMap*, or (b) *TreeMap*. Write a program that measures the time difference of these when input grows. How you explain the results?

4. Write a program that tests the performance difference of *LinkedList* and *ArrayList*. The algorithm should add n elements to the list, traverse the elements, and remove the elements one by one. Write a single algorithm that uses operations of interface *List* operations and then write a test method that calls the algorithm using both list implementations.

5. Add to the task 4 an algorithm that does the same performance testing for a standard array (E[]) instead of collection.

6. Add to the task 5 comparison of performance of *Integer* array and *int* array.