Linnaeus University
Department of Computer Science
Ola Petersson

Exam in Algorithms and Advanced Data Structures, 1DV516, 3 credits

Thursday 26 October 2017, 14.00–19.00

The exam consists of **6** questions, worth 10 points each. To pass you need 30 points. Write your name, personal identity number and the course code on each paper you hand in. You may answer the questions in Swedish or English.
When an algorithm is asked for, it should be understood that it should be as efficient as possible, and it should be expressed in such a way that it is understandable.
Allowed aids: Pen, pencil, eraser, and paper.
Not allowed: Consultation of any media outside of this document.

1. (a) Formally show that $n/100 + \sqrt{n} = \Omega(n)$. (2p)

(b) Consider the function $f(n) = a\log(n+b)$, where a and b are positive constants. Formally show that $f(n) = O(\log n)$. (2p)

(c) Consider the following method:

```
public void test (int n) {
    int a = 0;
    for (int i=0; i <n*n; i++)
        for (int j = 0 ; j <= i; j++)
            a = a + i * j;
}
```

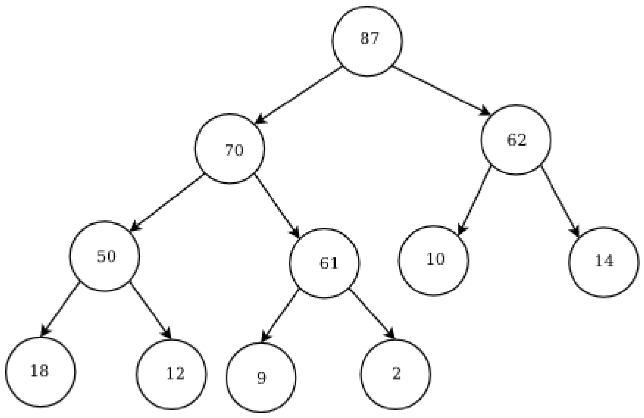
What is the asymptotic time complexity of this method? Motivate! (3p)

(d) Consider the following pseudo-code:

```
public void test2 (array A) {
   if size of A is 0, 1 or 2
         do some fixed amount of work;
   otherwise
         divide the array A into 3 equal parts, and call test2 recursively on each piece;
         do work proportional to the size of the array A;
}
```

State the recurrence equation for the running time. (3p)

2. (a) Consider the following max-heap:



Draw the heap *and* its array representation after a deleteMax operation has been performed and the tree has been reorganized to again be a heap. You should use the reorganization algorithm discussed in class. (3p)

(b) Consider the following algorithm to randomly permute an array of n elements, with all permutations equally likely

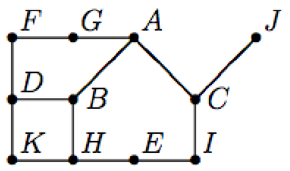
1.Split the array into two roughly equal-size subarrays

2.Recursively permute the two subarrays

3.Randomly merge the two subarrays

To randomly merge two arrays, we iteratively take an element from the front of one or the other, each with probability 1/2, until one of the arrays is exhausted, then take the remaining elements. What is the asymptotic complexity of this algorithm? Motivate! (4p)

(c) Consider the following graph:



List the nodes in the order visited by DFS, starting in node A. When there is an arbitrary choice to make, visit nodes in alphabetical order. (3p)

3. An item in an array A of length n is called a *majority item* if it appears more than $n/2$ times.

(a) What is the minimum and maximum number of majority items in A? Formally prove your answer. (2p)

(b) Give an algorithm that determines if A has a majority item, and if so, outputs its location in A. Your algorithm should adhere to the following constraints:

- The array A is stored in *read-only memory*
- Your algorithm cannot use more than $O(\log n)$ words of extra read/write memory. Each word can hold a number bounded by $O(n)$
- For full credit, your algorithm should run in time $O(n \log n)$, assuming constant-time comparisons (8p)

4. Say we are given a weighted, directed graph G with positive edge weights $w(u, v)$, and a source node s in G. We would like to modify Dijkstra's shortest-path algorithm to also produce a *count* of the number of different minimal-length paths from s to v, for every node v.

Recall that Dijkstra's algorithm builds a set K of nodes incrementally, starting with $K = \{s\}$. Normally, the algorithm keeps a value $D(v)$ for each node v giving the length of a shortest path from s to v passing through intermediate nodes in K.

We will modify the algorithm to maintain an extra value, $N(v)$, giving *the number of paths* of length $D(v)$ from s to v passing through intermediate nodes in K.

(a) Describe how to initialize $D(v)$ and $N(v)$ for all nodes v. (3p)

(b) Describe how to update $D(v)$ and $N(v)$ when a new node is added to K. Describe your modifications in words; that is, do not write any code. (7p)

5. Suppose we want to make change for n cents, using the least number of coins of denominations 1c, 10c, and 25c.

(a) Consider the following greedy strategy: if the amount left to change is m; take the largest coin that is no more than m; subtract this coin's value from m, and repeat until $m = 0$. Either give a counterexample to prove that this algorithm can result in a non-optimal solution, or argue that it always produces an optimal solution. (3p)

(b) Give a $\Theta(n)$-time dynamic programming algorithm that produces an optimal solution. (7p)

6. (a) Given a set of real numbers $\{x_1, x_2, \ldots, x_n\}$, where $x_1 \leq x_2 \leq \ldots \leq x_n$, we want to find the smallest number of unit-length closed intervals that together contain all the points. For example, the interval $[2.35, 3.35]$ includes all x_i such that $2.35 \leq x_i \leq 3.35$. Give the most efficient algorithm you can to solve this problem, argue why it's correct, and analyze its time complexity. (5p)

(b) DP is a complexity class that lies between NP and PSPACE (polynomial space). That is, NP is a subset of DP and DP is a subset of PSPACE. DP-complete problems are defined in the usual way for complexity classes. What is the definition of a DP-complete problem? (5p)