Algorithms and Data Structures (DAT3/SW3/INF5)

Exam Assignments

Simonas Šaltenis

31 January 2012

Full name:	
CPR-number:	
E-mail at student.aau.dk:	

This exam consists of three exercises and there are three hours to solve them. When answering the questions in exercise 1, write directly in the provided fields on this paper. Remember also to put your name and your CPR number on any additional sheets of paper you will use.

- *Read* carefully *the text of each exercise before solving it!*
- *For exercises 2 and 3, it is important that your solutions are presented in a readable form. In particular, you should provide precise descriptions of your algorithms using pseudo-code. It is also worth to write two or three lines describing informally what the algorithm is supposed to do as well as to justify your complexity analyses.*
- *Make an effort to use a readable handwriting and to present your solutions neatly. This will make it easier to understand your answers.*

In exercise 1, CLRS refers to T.H. Cormen, Ch. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms* (both 2nd and 3rd editions).

During the exam you are allowed to consult books and notes. The use of pocket calculators is also permitted, but you are not allowed to use any electronic communication devices.

Exercise 1 [50 points in total]

1. (*8 points*)

1.1. $2^n + n^3 + 4n$ is:

☐ **a)** $\Omega(2^n n)$ ☐ **b)** $\Omega(n^n)$ ☐ **c)** $\Omega(3^n)$ ☑ **d)** $\Omega(n^4)$

1.2. $8^{\log_2 n} + 9n \lg n^9 + n^2$ is:

☐ **a)** $\Theta(n^9)$ ☑ **b)** $\Theta(n^3)$ ☐ **c)** $\Theta(n \lg n)$ ☐ **d)** $\Theta(n^2)$

2. (*7 points*) Consider the following recurrence relation:

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 4T(\lceil n/2 \rceil) + 2 \qquad (n > 1). \end{aligned}$$

Mark the correct solution. $T(n) =$

☐ **a)** $\Theta(\sqrt{n})$ ☑ **b)** $\Theta(n^2)$ ☐ **c)** $\Theta(n \lg n)$ ☐ **d)** $\Theta(n)$

3. (*6 points*) Consider a STACK ADT with the following standard operations: *init*(), *push*(*x*:*int*), and *pop*():*int*. Assume an efficient implementation of this ADT (for example, using a linked list). Assume that $n \geq 1$ and consider the following algorithm:

```
DoSomething(n:int):int
 1   s:Stack
 2   s.init()
 3   s.push(1)
 4   i ← 1
 5   while i < n do
 6       t ← s.pop()
 7       s.push(i)
 8       for k ← i to t do s.push(2 · k)
 9       i ← 2 · i
10   return s.pop()
```

3.1. The running time of DoSomething is:

☑ **a)** $\Theta(\lg n)$ ☐ **b)** $\Theta(n)$ ☐ **c)** $\Theta(n^2)$ ☐ **d)** $\Theta(n \lg n)$

3.2. The amount of memory used by DoSomething for the stack s is:

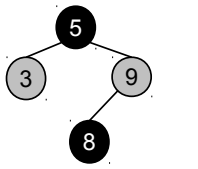
☐ **a)** $\Theta(n)$ ☐ **b)** $\Theta(1)$ ☑ **c)** $\Theta(\lg n)$ ☐ **d)** $\Theta(n^2)$

4. (*8 points*) Which of the four arrays can be made into a max-heap by a single call to Max-Heapify (as it is defined in CLRS) on one of the elements of the array?

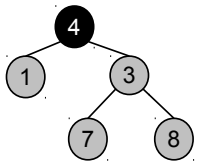
☐ **a)** `2, 7, 8, 4, 5, 9, 6, 1, 3`
☐ **b)** `9, 7, 6, 2, 5, 8, 1, 4, 3`
☑ **c)** `2, 7, 9, 5, 1, 8, 6, 4, 3`
☐ **d)** `9, 7, 6, 3, 5, 1, 8, 4, 2`

5. (*7 points*) Mark which of the following four trees is a red-black tree (the lighter nodes are red).

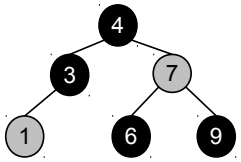
☐ **a)**



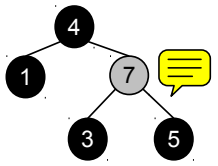
☐ **c)**



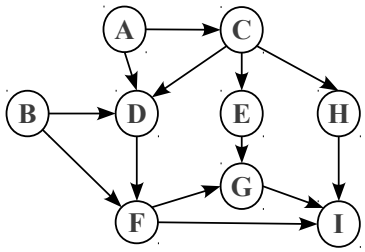
☑ **b)**



☐ **d)**



6. (*7 points*) Consider the following directed graph.

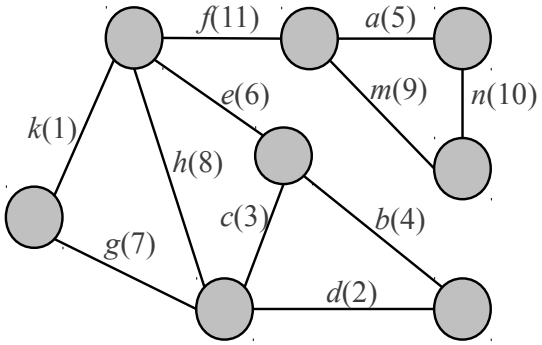


Which of the following four sequences of vertices is *not* topologically sorted?

☐ **a)** A, B, C, D, F, E, H, G, I ☐ **b)** B, A, C, H, E, D, F, G, I

☐ **c)** A, C, E, B, D, F, G, H, I ☑ **d)** B, A, C, D, F, G, H, E, I

7. (*7 points*) Consider the following weighted graph. Weights of edges are given in parentheses.



Which of the following sets of edges is a minimum spanning tree?

☑ **a)** k, d, c, e, a, m, f ☐ **b)** k, d, c, b, a, e, g

☐ **c)** k, d, c, g, a, m, f ☐ **d)** k, d, b, e, n, m, f

Exercise 2 [25 points]

A *completely balanced binary search tree* is a binary search tree that, in addition to the *binary-search-tree property*, satisfies the following *balance property*:

> Let $lsize(u)$ be the number of nodes in the left subtree of node u and $rsize(u)$ be the number of nodes in the right subtree of u. Then, for any node u of the tree, $|lsize(u) - rsize(u)| \leq 1$.

To remind, the *binary-search-tree property* governs the keys stored in the tree nodes:

> For any node u of the tree: if v is in the left subtree of u, then $v.key < u.key$; if v is in the right subtree of u, then $v.key > u.key$.

Note that for simplicity we assume that all keys are unique.

For this exercise, assume that you can freely access and modify the following fields in any tree node x: $x.key$, $x.left$, and $x.right$. A tree is represented by its root node. To create a new node, use $x = newNode()$.

1. (*5 points*) Draw a completely balanced binary search tree containing the following keys: 4, 6, 9, 2, 3, 1, 7, 5, 8.

2. (*8 points*) Write an algorithm, that given a *sorted* array $A[1..n]$, builds a completely balanced binary search tree containing all the keys from the array. Analyze the worst-case running time of your algorithm.

3. (*12 points*) Write an efficient algorithm that, given a binary tree t, checks if it is a completely balanced binary search tree. Analyze the worst-case running time of your algorithm.

Exercise 3 [25 points]

Consider a downtown road network with plenty of one-way streets. For each segment of a street between two intersections we record the average travel time (based on the observed average speed on that segment). Note that, if a street is a two-way street, the average travel time is recorded separately for each of the two travel directions. For each intersection, a single, average time to traverse that intersection is also recorded. This time includes, for example, waiting for traffic lights. Intersections have no turn restrictions, except for those implied by one-way streets.

1. (*5 points*) Suggest a mathematical model of the problem.

2. (*10 points*) Provide an algorithm that, given an intersection s in the road network, outputs all intersections which are not accessible from s, i.e., it is impossible to arrive to any of these intersections starting from s. Analyze the worst-case running time of your algorithm.

3. (*10 points*) Two intersections u and v in the road network are considered *two-way connected*, if the difference between the shortest time to get from u to v and the shortest time to get back from v to u is not more than d. (Note that the shortest times are computed assuming the observed average speeds of travel). Provide an efficient algorithm that, given an intersection s in the road network, outputs all intersections which are *two-way connected* with s. Analyze the worst-case running time of your algorithm.