Algorithms and Data Structures (DAT3/SW3/INF5/BAIT5)

Exam Assignments

Simonas Šaltenis

9 January 2013

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, mark the check-boxes on this paper. Remember also to put your name and your CPR number on any additional sheets of paper you will use for exercises 2 and 3.

- *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.

`See explanations in the pdf annotation comments`

Exercise 1 [50 points in total]

1. (*7 points*)

1.1. $n \lg^2 n + n \lg n^3 + 5n$ is:

☐ **a)** $\Theta(n^3)$ ☐ **b)** $\Theta(n \lg n)$ [✓ **c)** $\Theta(n \lg^2 n)$] ☐ **d)** $\Theta(n)$

1.2. $\sqrt{n} + 2^{\log_8 n}$ is:

☐ **a)** $\Theta(\sqrt[3]{n})$ [✓ **b)** $\Theta(\sqrt{n})$] ☐ **c)** $\Theta(\lg n)$ ☐ **d)** $\Theta(2^n)$

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

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

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

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

3. (*8 points*) Consider a Stack ADT with the following standard operations: *push*(x:*int*), and *pop*():*int*. Assume an efficient implementation of this ADT (for example, using a linked list). Assume that passing a stack as an argument to a function takes $O(1)$ time, i.e., it is passed "by reference" (without creating a new copy of the stack). Assume that $n \geq 1$ and consider the following algorithm:

```
PLAY(s:STACK, n:int)
1  PLAYAUX(s, 1, n)
PLAYAUX(s:STACK, c:int, n:int)
1  if $c = n$ then
2      for $i \leftarrow 1$ to $c - 1$ do $s.pop()$
3  else
4      for $i \leftarrow 1$ to $c$ do $s.push(i)$
5      PLAYAUX(s, $c + 1$, n)
```

3.1. For any s, the running time of Play(s, n) is:

☐ **a)** $\Theta(n \lg n)$ ☐ **b)** $\Theta(n)$ [✓ **c)** $\Theta(n^2)$] ☐ **d)** $\Theta(\lg n)$

3.2. Assume s starts as an empty stack. The size of s after calling Play(s, n) is:

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

4. (*7 points*) Here is the HEAPSORT algorithm, as presented in CLRS (*A.length* is the length of the array A and *A.heap-size* is the number of elements arranged into a max-heap in the array A):

```
HEAPSORT($A$)
1  BUILD-MAX-HEAP($A$)
2  for $i \leftarrow A.length$ downto 2 do
3      exchange $A[1] \leftrightarrow A[i]$
4      $A.heap\text{-}size \leftarrow A.heap\text{-}size - 1$
5      MAX-HEAPIFY($A, 1$)
```

After some number of iterations of the **for** loop, here is a possible state of the array $A[1..9]$ *right after* line 5 is executed:

`6, 3, 5, 2, 1, 4, 8, 9, 11.`

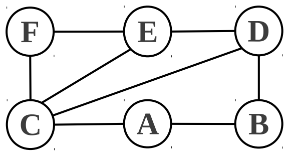
How many iterations of the loop have been completed?

☐ **a)** 2 ($i = 8$) ☐ **b)** 1 ($i = 9$) [✓ **c)** 3 ($i = 7$)] ☐ **d)** 4 ($i = 6$)

5. (*7 points*) Suppose we are searching for the number 17 in a binary search tree. Which one of the following four sequences could be a sequence of tree nodes examined? (Note that 17 may or may not be in the tree.)

[✓ **a)** 9, 12, 23, 16, 21, 19] ☐ **b)** 21, 23, 7, 9, 14, 17
☐ **c)** 21, 7, 18, 5, 3, 11 ☐ **d)** 9, 11, 15, 13, 16, 17

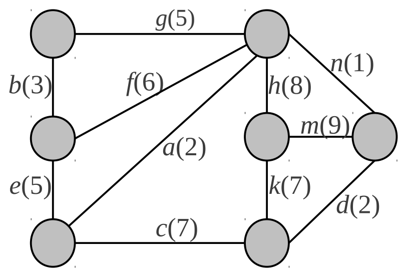
6. (*7 points*) Consider the following graph and consider the breadth-first search algorithm as described in CLRS.



Assume that adjacent vertices are always examined in the alphabetical order. What is the maximum number of vertices stored in the queue while performing the breadth-first search starting from A? (Note that the algorithm dequeues a vertex *before* examining its adjacent vertices.)

☐ **a)** 2 [✓ **b)** 3] ☐ **c)** 4 ☐ **d)** 5

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



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

[✓ **a)** a, b, d, g, k, n] ☐ **b)** a, b, d, e, g, n
☐ **c)** b, d, e, g, k, n ☐ **d)** a, b, d, e, f, n

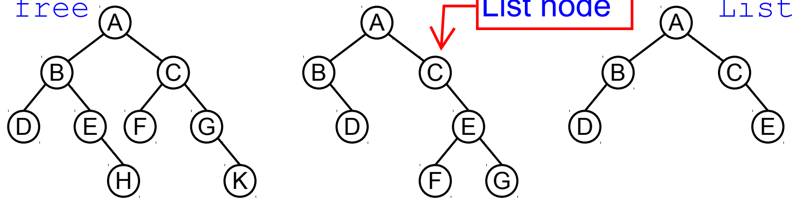
Exercise 2 [25 points]

Let a node in a binary tree be called a *list node* if that node has only one child *and* that child is *not* a leaf (a leaf is a node without children). A binary tree is called a *list-free* binary tree if it has no list nodes.

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.

1. (*5 points*) For each of the following three trees, determine if the tree is list free or not. To explain your answer, write which nodes are list nodes.

List free



List free

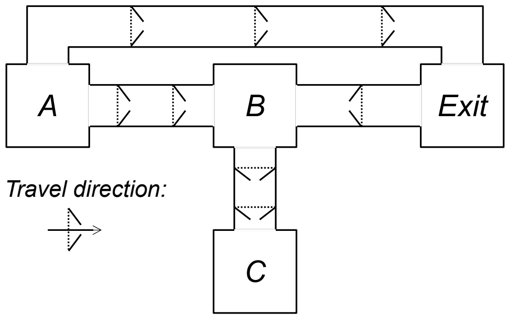
a) b) c)

2. (*9 points*) Write an efficient algorithm that, given a binary tree t, prints out all keys in the list nodes of t. (*Hint*: you may want to have a separate function that checks if a node is a list node. It will be useful in the next exercise too.) Analyze the worst-case running time of your algorithm.

3. (*11 points*) Write an efficient algorithm that, given a binary tree t, returns the largest list-free subtree in t. The algorithm should return a pair (*root*, *size*), where *root* is the root node of the returned subtree and *size* is the number of nodes in that subtree. Analyze the worst-case running time of your algorithm.

Exercise 3 [25 points]

Consider a maze consisting of rooms connected by hallways. Each hallway connects two rooms and has a number k ($k \geq 1$) of one-way doors, i.e., doors that can be opened only from one side and that close behind as soon as a person has passed through them. This means that each hallway allows walking only in one direction, from the beginning to the end of the hallway, but not backwards. Different hallways have different numbers of doors. A person can leave the maze only by finding the special *exit* room. Here is an example of a maze with four rooms and for hallways connecting them.



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

2. (*10 points*) A room is called a *dead-end* if it is not possible to exit the maze starting from that room. Provide an efficient algorithm that checks if there are no dead-ends in a given maze. The example above has dead-ends, as it is not possible to exit from rooms B and C. Analyze the worst-case running time of your algorithm.

3. (*10 points*) We say that a maze is *easy* if it contains *no* rooms such that leaving the room it is possible to return to it later. Given a maze and a room s in the maze, write an efficient algorithm that (i) checks whether the maze is easy and (ii) finds a path from s to the exit with the smallest number of doors along the way. If s is a dead-end, it should return a NIL path. Analyze the worst-case running time of your algorithm.