Algorithms and Data Structures (DAT1/SW3/BAIT5)

Re-exam Assignments

Simonas Šaltenis

11 February 2011

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 from exercise 1, write directly in the provided fields on this paper. Remember to put your name and CPR number also 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, "the course textbook" 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. (*6 points*)

1.1. $2n^{10} + \frac{1.5^n}{100} + 4n^9 \lg n$ is:

☐ **a)** $\Theta(n^{10})$ ☑ **b)** $\Theta(1.5^n)$ ☐ **c)** $\Theta(n^{10} \lg n)$ ☐ **d)** $\Theta(n^9 \lg n)$

1.2. $3^{n/3} + 3^{n/10} + 10^{\log_3 n}$ is:

☐ **a)** $\Theta(10^{\log_3 n})$ ☐ **b)** $\Theta(3^n)$ ☑ **c)** $\Theta(3^{n/3})$ ☐ **d)** $\Theta(3^{n/10})$

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

$$\begin{aligned} T(1) &= 10 \\ T(n) &= T(n-1) + 3 \qquad (n > 1). \end{aligned}$$

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

☐ **a)** $\Theta(n^{10})$ ☑ **b)** $\Theta(n)$ ☐ **c)** $\Theta(n^2)$ ☐ **d)** $\Theta(n^3)$

3. (*8 points*) Consider a STACK ADT with the following standard operations: *init*(), *push*(*x*:*int*), *pop*():*int*, and *top*():*int*. Here, *pop*() both removes an element and returns it as a result, while *top*() just returns the element at the top of the stack. 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   sk, st: Stack
 2   sk.init()
 3   st.init()
 4   for i ← 1 to n do sk.push(i)
 5   for i ← n downto 1 do st.push(i)
 6   i ← n
 7   while sk.top() > 1 do
 8       for j ← 1 to i do sk.push(st.pop())
 9       i ← i − 1
10       for j ← 1 to i do st.push(sk.pop())
11   return st.top()
```

3.1. DoSomething(10) =

☐ **a)** 10 ☐ **b)** 1 ☐ **c)** 9 ☑ **d)** 2

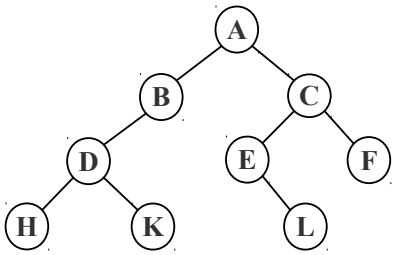
3.2. The running time of DoSomething is

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

4. (*7 points*) Which of the four arrays is *not* a max-heap?

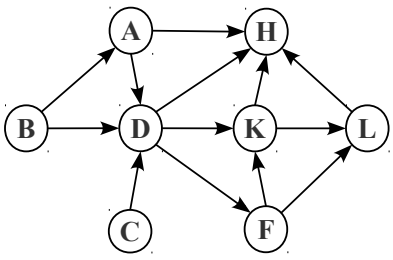
☐ **a)** `9, 8, 7, 6, 1, 3, 4, 2, 5`
☐ **b)** `9, 8, 3, 6, 7, 1, 2, 4, 5`
☐ **c)** `9, 7, 8, 6, 5, 4, 3, 2, 1`
☑ **d)** `9, 7, 6, 8, 4, 5, 1, 3, 2`

5. (*6 points*) Write the preorder traversal of the following binary tree.



A	B	D	H	K	C	E	L	F

6. (*8 points*) Consider the following directed graph.



In which order does the depth-first search visit the first six vertices if it starts from **A**? (In other words, in which order are the vertices colored gray according to the DFS algorithm from the textbook?) Assume that for each vertex its adjacent vertices are examined in alphabetical order.

☐ **a)** A, B, C, D, F, K ☐ **b)** A, D, H, F, K, L

☑ **c)** A, D, F, K, H, L ☐ **d)** A, D, F, H, K, L

7. (*8 points*) Consider the same graph from the previous assignment. Which of the following four sequences of vertices is topologically sorted?

☐ **a)** C, B, A, D, L, F, K, H ☐ **b)** B, C, D, A, F, K, L, H

☐ **c)** B, A, C, F, D, K, L, H ☑ **d)** B, C, A, D, F, K, L, H

Exercise 2 [25 points]

Consider a *sorted* array of integers $A[1..n]$. If an integer x appears i times in the array, we say that there are $i-1$ duplicates of x in the array. The *overhead* of A is the total number of duplicates in A.

1. (*3 points*) What is the *overhead* of $A = [3, 3, 5, 8, 8, 8, 8, 9, 9]$?

2. (*8 points*) Write a non-recursive algorithm to find the *overhead* of a sorted array $A[1..n]$ ($n \geq 1$). Count exactly how many comparisons of array elements your algorithm performs for an array of n integers.

3. (*14 points*) Write a divide-and-conquer algorithm to find the *overhead* of a sorted array $A[1..n]$ ($n \geq 1$). Count exactly how many comparisons of array elements your algorithm performs for an array of n integers. If it helps, you can assume that $n = 2^k$ (for some $k \geq 0$).

Exercise 3 [25 points]

Consider a *thesaurus* (a dictionary of synonyms). It is defined by the function $synonym(w_1, w_2)$ that, given two words w_1 and w_2, returns *true* if the words are synonyms and *false* otherwise. The function is symmetric, i.e., $synonym(w_1, w_2) = synonym(w_2, w_1)$. Assume that the complexity of *synonym* is $O(1)$.

Two words x and y are said to be *k-synonyms* ($k \geq 1$) if there is a sequence $x = w_0, w_1, \ldots, w_k = y$, where for each i ($0 \leq i < k$), $synonym(w_i, w_{i+1})$ returns *true*.

In this exercise, you can use the following two functions to construct a graph. Given an array of words A, $G.init(A)$ initializes G so that each word from A is a vertex and there are no edges. Then, given two words x and y, $G.makeEdge(x, y)$ creates a new, undirected edge between vertices corresponding to words x and y. A variant of this function $G.makeEdge(x, y, w)$ creates a weighted edge with the

weight w. Assume that the complexity of $G.makeEdge$ is $O(1)$ and the complexity of $G.init(\mathrm{A})$ is $O(n)$, where n is the number of words in A.

1. (*10 points*) Given a large array of words $A[1..n]$, it is expensive to find all synonyms of a given word just by using the function *synonym*. To enable more efficient search for synonyms, write an algorithm that transforms the array A into a synonym graph. The constructed graph G should be such that, a word x is a synonym of a word y if and only if there is an edge (x, y) in G. What is the worst-case complexity of your algorithm?

2. (*15 points*) Given a parameter k_{max} and a synonym graph G constructed by your algorithm from the previous question, provide an algorithm that transforms G into a weighted graph G^*. This graph should be such that, for any $k \leq k_{max}$, if a word x is a k-synonym of a word y, then there is an edge (x, y) with the weight k in G^*. What is the worst-case complexity of your algorithm?