Algorithms and Data Structures (DAT1/SW3/BAIT5)

Exam Assignments

Simonas Šaltenis

7 January 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 2, "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. $2^{\lg(2n)} + 10\lg(9n) + 3n\lg n$ is:

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

1.2. $\sqrt{n} + 7\lg n^4$ is:

☑ **a)** $\Theta(\sqrt{n})$ ☐ **b)** $\Theta(n^4)$ ☐ **c)** $\Theta(n \lg n)$ ☐ **d)** $\Theta(\lg n)$

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

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

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

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

3. (*7 points*)
Consider a QUEUE ADT with the following standard operations: *init*(), *enqueue*(*x*:*int*), and *dequeue*():*int*. Here, *dequeue*() both removes an element and returns it as a result. 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):Queue
 1  q:Queue
 2  q.init()
 3  for i ← 1 to n do q.enqueue(i)
 4  i ← n
 5  while i > 1 do
 6      for j ← 1 to n − 1 do q.enqueue(q.dequeue())
 7      i ← ⌈i/2⌉
 8  return q
```

3.1. DoSomething(8).*dequeue*() =

☐ **a)** 1 ☐ **b)** 3 ☑ **c)** 6 ☐ **d)** 8

3.2. The running time of DoSomething is

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

4. (*8 points*)
Consider a double hashing method given by this hash function:

$$h(k,i) = (k + ih'(k)) \mod 11, \quad \text{where} \quad h'(k) = 1 + (k \mod 10).$$

Here k is the key to be inserted and i is the probe number ($i = 0, 1, \ldots, 10$). Insert keys $21, 32, 43, 8, 14$ (in this order) into the hash table $A[0..10]$ below:

0	1	2	3	4	5	6	7	8	9	10
		32	43				14	8		21

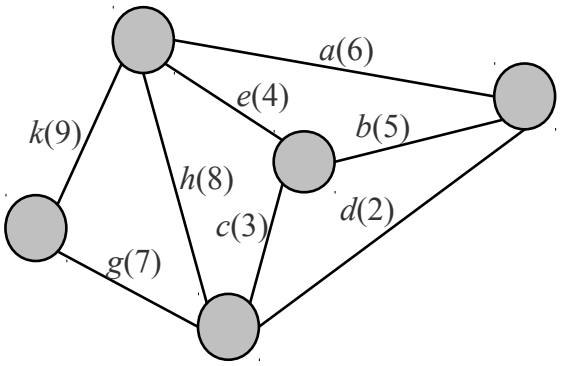
5. (*6 points*)
Consider the Quicksort(A, p, r) algorithm and the last two lines in it: the recursive calls Quicksort$(A, p, q-1)$ and Quicksort$(A, q+1, r)$. Let $p = 1$ and $r = 9$. Assume that the array $A[p..r]$ looks as follows after the call to Partition(A, p, r) and just before the two recursive calls:

`5, 1, 3, 6, 8, 7, 9, 12, 11.`

Which of the following four values is a possible value of q returned by Partition?

☐ **a)** 3 ☑ **b)** 4 ☐ **c)** 6 ☐ **d)** 8

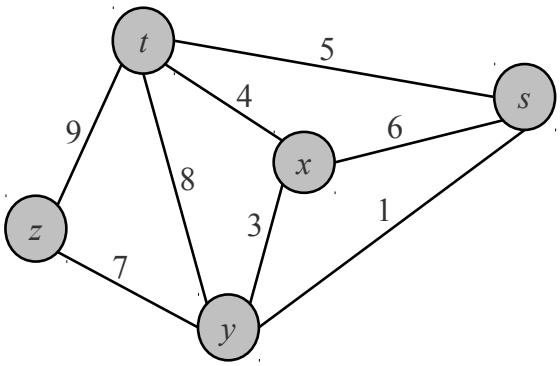
6. (*8 points*)
Consider the following weighted graph. Weights of edges are given in parentheses.



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

☐ **a)** a, c, d, g ☑ **b)** c, d, e, g ☐ **c)** b, c, d, e ☐ **d)** b, c, e, g

7. (*8 points*)
Consider the following weighted graph.



Consider Dijkstra's algorithm on this graph to find the shortest paths with s as a starting vertex. Which are the first four vertices extracted from the priority queue by the algorithm (listed in the order they are extracted)?

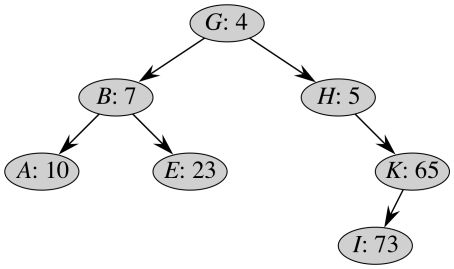
☐ **a)** s, y, t, x ☐ **b)** s, y, x, z ☐ **c)** s, t, y, x ☑ **d)** s, y, x, t

Exercise 2 [25 points]

Consider a ***treap*** data structure. It is a binary search tree, except that each tree node, in addition to a *key* value, has a *priority* value. Nodes of the treap are arranged so that keys obey the binary-search-tree property: *if v is in a left subtree of u, then $v.key < u.key$; if v is in a right subtree of u, then $v.key > u.key$.* The priorities obey the min-heap order property: *if v is a child of u, then $v.priority < u.priority$.* Note that for simplicity we assume that all keys and priorities are unique.

The figure below shows an example treap (it is Figure 13.9 from the textbook). Each node x is labeled with $x.key$: $x.priority$. For example, the root has key G and priority 4. Alphabetical order of keys is assumed.

For this exercise, assume that you can freely access and modify the following fields in any treap node x: $x.key$, $x.priority$, $x.left$, $x.right$, and $x.p$. The last three fields point to the left child, the right child, and the parent of x. Given

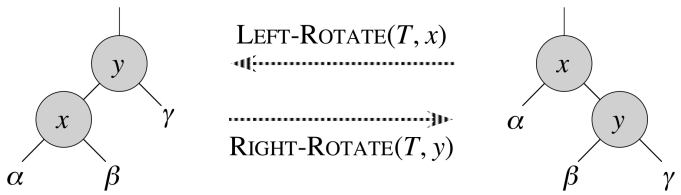


a treap T, use $T.root$ to access its root node. Although not necessary to solve this exercise, more information about treaps can be found in Exercise 13-4 in the textbook (including a few examples of node insertions). Note that, while that exercise talks about random priorities, in this exercise we are not concerned about how a priority is assigned to a node, it is just given as a parameter to the insertion algorithm.

1. (*5 points*) Draw a treap containing the following nodes, given as (*key*: *priority*) pairs: (L:4), (N:6), (C:9), (A:7), (F:3), (D:1), (K:8), (P:5).

2. (*10 points*) Using the treap properties, write an efficient algorithm that, given a treap T and a priority P, prints all the keys in T that are stored in the nodes with priorities smaller than P. The keys should be printed in the sorted (ascending) order. Analyze the worst-case complexity of your algorithm.

3. (*10 points*) Write a TREAP-INSERT(T, z) algorithm that inserts a node z into a treap T. Here, $z.key$ and $z.priority$ is the pair that we want to insert and $z.left$ as well as $z.right$ are set to *nil*. (*Hint:* Execute the usual binary-search-tree insertion procedure and then perform rotations to restore the min-heap order property.) In your pseudocode, you can call the binary-search-tree insertion algorithm (TREE-INSERT(T, z) as defined in Section 12.3 in the textbook) and LEFT-ROTATE(T, x) as well as RIGHT-ROTATE(T, y) algorithms as defined in the figure below (and Section 13.2 in the textbook). In the figure, α, β, and γ are arbitrary subtrees, possibly *nil*.



Exercise 3 [25 points]

Consider a model of a complex mechanical system, for example, an internal combustion engine. It contains a number of subsystems. Each of these subsystems may in turn contain a number of smaller subsystems and so on, until we get to the smallest, indivisible parts.

For example, let us assume for simplicity that an engine contains two subsystems—an engine block and a fuel pump. Note, that we are not interested in the specific instances of subsystems or their number, just the different types of subsystems. In this example, an engine block may contain ten 11-mm bolts while a fuel pump may contain additional four 11-mm bolts, but we just want to model that both an engine block and a fuel pump share the same type of 11-mm bolt as a subpart. Thus, 11-mm bolt appears only once in the model of the system.

We say that a system S *contains* a subsystem s if and only if a model explicitly says that s is a direct subsystem of S. To generalize this relation, we say that S *includes* s, if and only if there is a sequence $S = s_0, s_1, \ldots, s_k = s$, where $k \geq 1$ and s_i *contains* $s_{i+1} (0 \leq i < k)$. In the above example, an engine *contains* a fuel pump, but it does not *contain*, in a direct way, an 11-mm bolt. Instead, we say that an engine *includes* an 11-mm bolt (as well as a fuel pump and an engine block).

1. (*5 points*) Suggest a mathematical formalization of the problem. Show, how the example above is represented in your model. A drawing is sufficient.

2. (*10 points*) Provide an algorithm that, given a system model, computes an order in which all the subsystem types in the model have to be assembled (or produced). A subsystem can be assembled only after all the subsystems that it contains are assembled. First, an algorithm should check if a given system model is *valid*. We say that a model is invalid, if there is a subsystem in the model that *includes* itself, otherwise the model is valid. If the given model is not valid, the algorithm should abort, otherwise it should output subsystems in an assembly order.

   Specify clearly what is the input of your algorithm. Analyze the worst-case complexity of your algorithm.

3. (*10 points*) Provide an algorithm that, given a system model and a subsystem S within this model, outputs all the *shared* subsystems within S. A subsystem s is *shared* within S, if S *includes* at least two subsystems s_1 and s_2 that both *contain* s.

   Analyze the worst-case complexity of your algorithm.