Algorithms and Data Structures (DAT1/SW3)

Re-Exam Assignments

Simonas Šaltenis

30 August 2004

Full name:	
CPR-number:	

This exam consists of three exercises. When answering the questions from exercise 1, write in the provided fields on this paper.

- *Read carefully the text of the exercises before solving them.*
- *For exercises 2 and 3, it is important that you argue for your answers and that your solutions are presented in a readable form. In particular, you should provide precise descriptions of your algorithms using pseudo-code.*
- *Make an effort to use a readable handwriting and to present your solutions neatly. This will make it easier to understand your answers, and thus to mark your exams.*

In exercise 1, "the course textbook" refers to T.H. Cormen, Ch. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*.

During the exam you are allowed to consult books and notes. The use of pocket calculators is also permitted.

Exercise 1 [50 points in total]

1. (*6 points*)
$2n + \log(9n^3)$ is:

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

2. (*8 points*)
Consider the following recurrence relation:

$$\begin{aligned} T(1) &= 2 \\ T(n) &= 2T(n-1) + 3 \qquad (n > 1) \ . \end{aligned}$$

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

[✓ **a)** $5 \cdot 2^{n-1} - 3$] ☐ **b)** $3 \log_2 n + 2$ ☐ **c)** $3 \cdot 2^n - 4$ ☐ **d)** $5n^2 - 3$

3. (*6 points*)
Let A be the following array of numbers:

`7, 1, 5, 3, 2, 6, 9, 4`

It is a min-heap, except that the value at its top ($A[1] = 7$) is in the wrong place. Write the array A after calling MIN-HEAPIFY$(A, 1)$, which moves $A[1]$ into the right place in the min-heap A. MIN-HEAPIFY is analogous to the MAX-HEAPIFY presented in the course book.

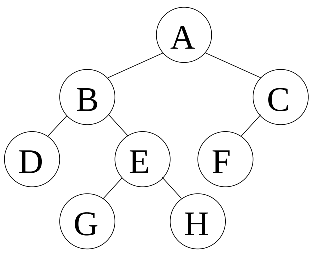
1 | 2 | 5 | 3 | 7 | 6 | 9 | 4

4. (*6 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. Consider also a STACK ADT with the following standard operations: *init*(), *push*(x:*int*), and *pop*():*int*. Here, *pop*() both removes the top element of the stack and returns it as a result. Write the sequence of values stored in the queue Q after PLAY(Q) has been executed. The front (head) of the queue is on the left.

```
PLAY($Q$ : QUEUE)
1   $S$ : STACK
2   $S.init()$
3   $Q.init()$
4   $Q.enqueue(2)$
5   $Q.enqueue(3)$
6   $Q.enqueue(1)$
7   $Q.enqueue(4)$
8   $S.push(Q.dequeue())$
9   $S.push(Q.dequeue())$
10  $Q.enqueue(S.pop())$
11  $Q.enqueue(S.pop())$
12  $Q.enqueue(Q.dequeue())$
```

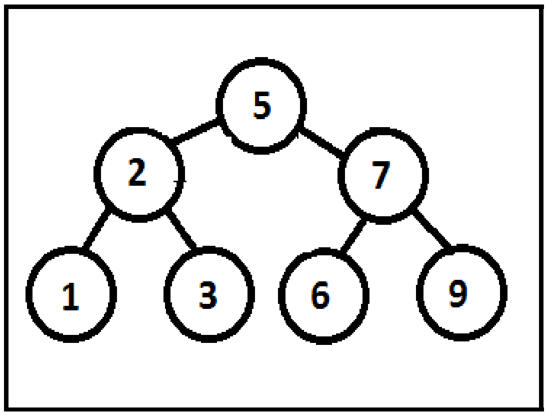
4 | 3 | 2 | 1

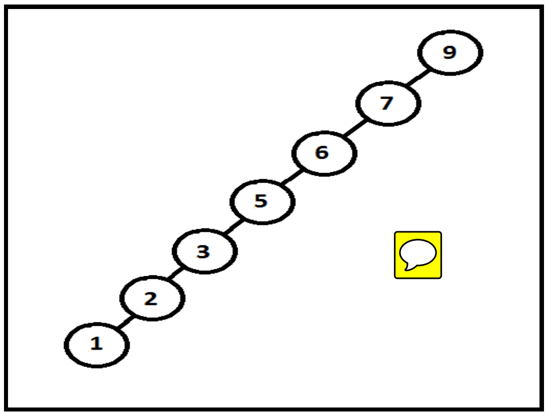
5. (*5 points*)
Write the sequence of visited nodes in a preorder traversal of the following binary tree.



A | B | D | E | G | H | C | F

6. (*6 points*)
Consider a set of integers: 3, 9, 1, 7, 5, 6, 2. In the left box, draw a binary search tree of the smallest possible height storing all these integers. In the right box, draw a binary search tree of the largest possible height storing all these integers.



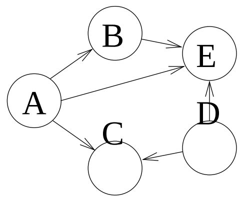


7. (*6 points*)
Consider the following two sequences of graph vertices:

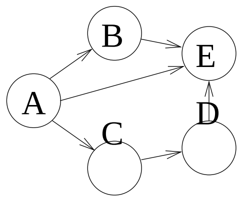
```
A, B, C, D, E
A, C, D, B, E
```

Of the following four graphs, mark only the graph for which the two given sequences of vertices are topologically sorted.

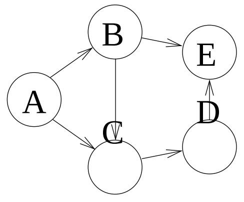
☐ **a)**



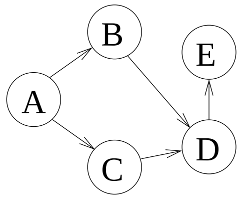
[✓ **b)**]



☐ **c)**



☐ **d)**



8. (*7 points*)
Consider the following two statements about weighted graphs.

1. A shortest path between two vertices in a graph may include edges that do not belong to any minimum spanning tree of the graph.

2. An edge with the highest cost is never included in a minimum spanning tree.

Are these statements true or false?

☐ **a)** both statements true
[✓ **b)** statement 1 true, statement 2 false]
☐ **c)** statement 1 false, statement 2 true
☐ **d)** both statements false

Exercise 2 [25 points]

Consider a PriorityQueue abstract data type with the following operations:

insert(v:*int*)
: *q.insert*(v) inserts the integer v into the priority queue q.

extractMax():*int*
: *q.extractMax*() returns the largest integer in q and removes it from q. If q contains more than one integer equal to the largest one, only one of them is removed from q.

Both operations are implemented so that they have the $O(\lg n)$ worst-case complexity, where n is the number of integers in the priority queue.

Consider also a BinTree ADT with the following operations:

root():*int*
: *a.root*() returns the integer stored in the root of the binary tree a.

setRoot(v:*int*)
: *a.setRoot*(v) stores the integer v in the root of the binary tree a.

leftSubtree():*BinTree*
: *a.leftSubtree*() returns the left subtree of a. It returns *nil*, if a has no left subtree.

rightSubtree():*BinTree*
: *a.rightSubtree*() returns the right subtree of a. It returns *nil*, if a has no right subtree.

parent():*BinTree*
: *a.parent*() returns the parent node of a. (More formally, the tree rooted at the parent node is returned). It returns *nil*, if a is not a subtree in a larger tree.

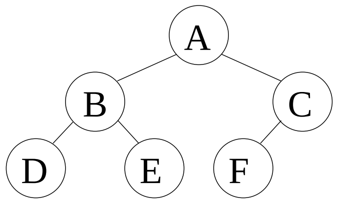
addNewChild(v:*int*):*BinTree*
: If *a.leftSubtree*() is *nil*, *a.addNewChild*(v) creates the new left subtree of a and returns it, else if *a.rightSubtree*() is *nil*, *a.addNewChild*(v) creates the new right subtree of a and returns it. Let the created subtree be t, then *t.root*()= v, both *t.leftSubtree*() and *t.rightSubtree*() are *nil*, and *t.parent*()= a. If both *a.leftSubtree*() and *a.rightSubtree*() are not *nil*, this operation does nothing and returns *nil*.

Note that, instead of saying that an instance of a BinTree ADT represents a tree or a subtree, we may say that it represents a node, in which case we have in mind the root node of the corresponding tree or subtree (see, for example, the description of the *parent*() operation).

The operations listed above are the only ones that you are allowed to use to access or modify the information in the instances of the two presented ADTs.

1. Using the PriorityQueue ADT, write an algorithm that, given two priority queues, determines if they contain the same set of integers. For example, if one priority queue contains integers $3, 4, 4, 7, 7, 7$ and another priority queue contains integers $3, 3, 4, 7$, they contain the same set of integers (i.e., the set $\{3, 4, 7\}$). The queues can be destroyed by your algorithm. What is the worst-case complexity of your algorithm?

2. Using the BinTree ADT, write an algorithm *findPlace*(*a*:*BinTree*):*BinTree*, which returns a node p in a such that *p.leftSubtree*() or *p.rightSubtree*() is *nil*. If several such nodes exist (which is usually the case), a node on the highest level is returned. A child node is considered to be on the lower level than its parent node. Consider the following example tree. Nodes D, E, F, and C all have at least one *nil* subtree, but C is on the higher level than D, E, or F. Thus, for this tree, *findPlace*(A)$= C$. In this assignment, ignore the integers stored in the nodes of a binary tree.



3. Assume that a max-heap is used to implement the PriorityQueue ADT operations. Write an algorithm implementing the *insert* operation from the PriorityQueue ADT, assuming that the max-heap is represented as an instance of the BinTree ADT. In your algorithm, use variable *h*:*BinTree* to access the max-heap corresponding to the given priority queue. (*Hint*: you can use the algorithm *findPlace* from question 2, even if you did not solve question 2.)

Exercise 3 [25 points]

Let us consider a parking place with a finite number of cars. Each car has a unique color, however, this color cannot be observed directly by a robot camera guarding the parking place. The only thing the camera can see is whether any two given cars share the same color or not (but it cannot tell which color it is).

One of the tasks of the camera is to keep track of the number of different colors that can be seen on the parking place.

Consider a concrete example. Assume that on a parking place there are six cars called c_1, c_2, c_3, c_4, c_5, and c_6. The camera sees that c_1 and c_2 have the same color, c_3 and c_4 have the same color, c_4 and c_5 have the same color, and c_5 and c_3 have the same color. No other two cars, apart from the ones mentioned above, have the same color.

Answer the following questions:

1. In the given example, how many different colors are there on the parking place?

2. Suggest a mathematical formalization of the problem. Show, how the example above is represented in your model. A drawing is sufficient.

3. Give an algorithm NumberOfColors that computes the number of different colors that can be seen on the parking place. The algorithm should work for the general situation (not only for the concrete example mentioned above!). Specify clearly what is the input of your algorithm.

   If, and only if, you are not able to construct the algorithm, write an algorithm which computes whether all cars on the parking place are of the same color.

4. Analyze the worst-case complexity of your algorithm from question 3.