

Self-Study Assignments 5

Name:	Niklas Kirk Mouritzsen
Grouproom:	2.1.10

Exercise 1 [50 points in total]

Assignment 1.1. as well as 3.-6. are from 2012 exam and 7. is from 2003 re-exam.

1. (6 points)

1.1. $\sqrt{n} + 2^{\lg n}$ is:

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

1.2 $2\sqrt{n} \lg n$ is:

- ☒ a) $O(n \lg n)$ ☐ b) $O(n^{\frac{1}{4}} \lg n)$ ☐ c) $O(\lg n)$ ☐ d) $O(\sqrt{n})$

2. (8 points)

Consider the following recurrence relation:

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

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

- ☒ a) $\Theta(n^3)$ ☐ b) $\Theta(n)$ ☐ c) $\Theta(3^n)$ ☐ d) $\Theta(2^n)$

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 ← 1 to c − 1 do s.pop()
3  else
4      for i ← 1 to c do s.push(i)
5      PLAYAUX(s, c + 1, n)

```

3.1. For any s , the running time of $\text{PLAY}(s, n)$ 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 $\text{PLAY}(s, n)$ 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 $A.heap\text{-}size$ is the number of elements arranged into a max-heap in the array A):

```

HEAPSORT(A)
1  BUILD-MAX-HEAP(A)
2  for i ← 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.

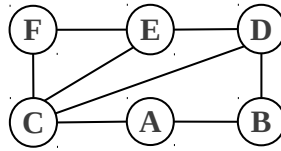
How many iterations of the loop have been completed?

- ☐ 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

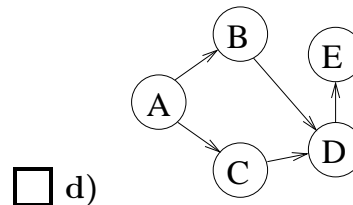
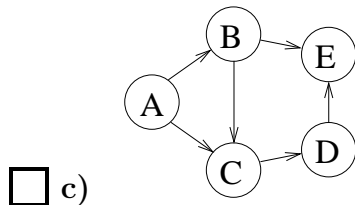
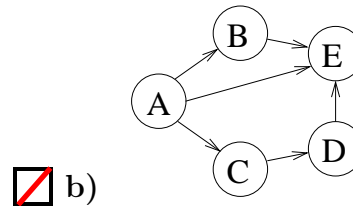
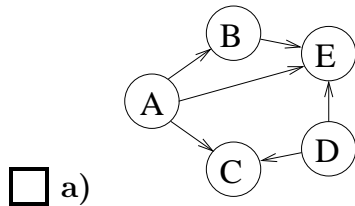
7. (7 points)

Consider the following 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.

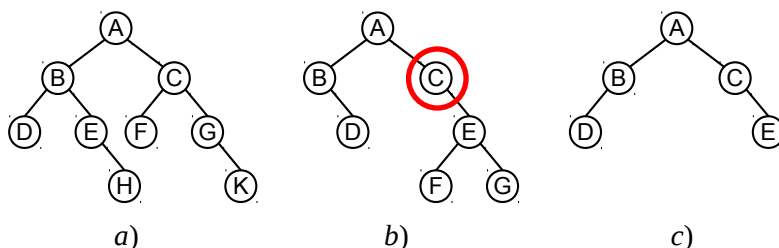


Exercise 2 [25 points] (from 2012 exam)

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.



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] (from 2010 exam)

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 $S = s_0, s_1, \dots, s_k = s$, 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 how the problem is modeled as a graph. 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 running time 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 running time of your algorithm.