Data Structures and Algorithms – Sample Exam Subject

Notes: This is a closed book exam. You are not allowed to consult any notes, books, online resources or use any electronic devices or receive assistance from others during the exam. Any cheating attempt will result in elimination from the exam.

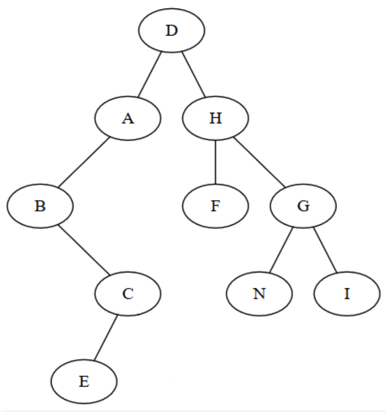
The problems from the exam are divided into 3 parts (A, B and C). Please write the solution for each part on a **different** sheet of paper. You can use several sheets of paper for the solutions of one part. For every part, on the first page of the solutions, you should write in the **top right corner** your **name, group, subject number** and the letter of the exam part you are solving (**A, B or C**). Please turn in at least one paper for every part (even if it is an empty one)

You have **90 minutes** to complete the exam.

PART A. Choose the correct answer(s) for the following questions. Do not forget to justify your answer(s). (4 * 0.4p)

1. Consider a hash table with *m* positions, collision resolution with open addressing, which currently contains *n* elements. If we do a search for a given element e, is it possible that during the search, more than n positions are checked?
   a. No, it is not possible.
   b. It is possible only if *e* is not in the hash table.
   c. It is possible, no matter whether *e* is in the hash table or not.
   d. For a search it is not possible, but if we had to count how many times *e* occurs in the hash table, then we could check more than *n* positions.
2. Starting from an initially empty stack, we push onto it, in the given order, the following elements: 1, 2, 3, 4, 5, 6. Then we pop an element and push it into another, initially empty stack. We do this three more times (so we pop a total of 4 times). After this, we pop and element from the second stack. What value is now on the top of the second stack?
   a. 1 c. 3 e. 5
   b. 2 d. 4 f. 6
3. What is the result of evaluating the following expression made of single digit numbers and regular operators:
   4 6 7 + 1 – 2 5 * + +
   a. A value below -15
   b. A value between -15 and -5
   c. A value between -5 and 5
   d. A value between 5 and 15
   e. A value above 15
4. What complexity class does the following expression belong to: $12n^3 + n*\log_2 n + 52$?
   a. $O(n^4)$ c. $\Theta(n*\log_2 n)$ e. $\Omega(1)$
   b. $O(n^2)$ d. $\Omega(n^3)$ f. $\Theta(n^4)$

PART B. Answer the following questions. Do not forget to justify your answer. (3 *1p)

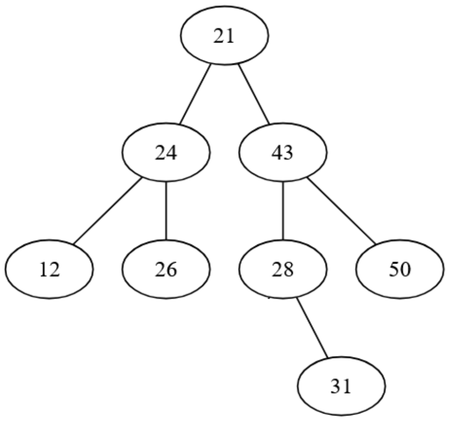


1. For the binary tree on the right, specify the preorder, postorder, inorder and level order traversals.
2. Consider a hash table with m=8 positions, in which the division method is used as hash function and separate chaining is used as collision resolution method, but every position of the table contains the root of a BST. Show how the following elements can be added into this hash table (which is initially empty): 8, 99, 40, 19, 56, 16, 17, 28, 62. It is enough to draw the final hash table, but show how you computed the position for every element. Specify the load factor of the table.
3. Is the following array a binary heap? If not transform it into a binary heap by swapping two elements: [41, 54, 13, 73, 68, 98, 63, 100, 76]. In the (possibly modified) heap add the following elements (in this order): 19, 18, 59. After adding these 3 elements, remove an element from the heap. Draw the heap in the tree form after every operation (4 drawings in total).

PART C. From the next problems, you need to solve C1 and EITHER C2 OR C3. Please, specify clearly which problem you are solving (C2 or C3). (C1 – 1p, C2 – 1.8p, C3 – 2.9p)

1. Write a subalgorithm to reverse a singly linked list in linear time and Θ(1) extra space complexity. Provide the representation of the singly linked list, implement the algorithm and justify why it respects the complexity requirements.

2. Write a subalgorithm that finds the maximum value in a binary tree containing integer numbers. Give the representation of the binary tree (do not memorize the parent of a node), implement the algorithm and specify and explain the complexity of the operation. If you use auxiliary containers, specify the used operations. For example, for the following binary tree, the maximum value is 50.



3. Consider ADT Quartiler, which contains integer numbers and has the following operations (with the provided complexity requirements):

- init(q) – creates a new, empty Quartiler – Θ(1) total complexity
- add(q, elem) – add a new element to the Quartiler q – O($\log_2 n$) – amortized
- getTopQuartile(q) – returns the element closest to the 75th percentile (see explanation below). If there is no such element, the operation throws and exception – Θ(1) – total complexity
- deleteTopQuartile(q) – removes the element closest to the 75th percentile. If there is no such element, it throws an exception – O($\log_2 n$) – total complexity.

Explanation: the 75th percentile (called 3rd quartile as well) of a sequence is a value from the sequence, the one below which 75% of the sequence's values are found, if the sequence is sorted. So if you have the values from 1 to 100 (in any order) the 75th percentile is the value 75. If you have the values 3, 1, 2, 4, the 75th percentile is 3. In case of a tie, any value can be returned, for example, if you have the values 1,2,3,4,5,6, either 5 or 6 can be returned.

1. Which data structure (out of the ones discussed during the lectures) would you use as a representation for the Quartiler and how?
2. Provide a drawing of how the representation of the Quartiler looks like with a few elements added (somewhere between 5- 10 elements).
3. Explain in short how would you implement each operation of the Quartiler and why the implementation fits the complexity requirement.
4. Give the representation of the Quartiler and implement in pseudocode the deleteTopQuartile operation.

Obs: (1) You can ignore memory deallocation altogether when computing the complexity of an operation. You can assume that you can deallocate anything (including a linked list with dynamic allocation, for example) in Θ(1) complexity. (2) You do not have to implement resize. (3) You need to implement every function you need for the last question, you cannot assume that you already have operations implemented (unless specified differently). But you can define (and implement) auxiliary functions.