Exam for DAT038 and DAT525

Re-exam for TDA417, DAT495, DIT182
and earlier versions of those courses

Datastrukturer och algoritmer

Thursday, 2024-01-11, 14:00–18:00

Teacher	Peter Ljunglöf, tel. 0766–075561. (Will visit exam rooms around 15:00 and 16:30)
Notes	*Answer directly on the question sheets unless told otherwise.* If you need additional space, you can use extra papers. Please refer to the extra pages from the question sheet! Write your anonymous code (not your name) on the first page of the question sheet and on every extra page you hand in. You may answer in English or Swedish. Excessively complicated answers might be rejected. Write legibly — we need to be able to read your answer! You can take this cover page home after the exam. Questions and solutions will be published afterwards here: https://github.com/ChalmersGU-data-structure-courses/past-exams
Allowed aids	None.
Exam review	When the exams have been graded, they are available for review in the CSE student office at Johanneberg. There will be an exam review Wednesday 31 January 10–12, in room 6217. After the exam review the exams for DAT495 and DIT182 will be available to collect in the CSE student office at Lindholmen.
Grading	See opposite side.

Good luck!

Grading

There are **8 basic questions** and **4 advanced questions**.
The answer to each question is graded either **correct** or **incorrect**.

- To pass the exam, you must pass **6** of the 8 basic questions.

If your course uses the U/3/4/5 grading scale:

- To get a four, you must **also** pass **2** of the 4 advanced questions.
- To get a five, you must **also** pass **3** of the 4 advanced questions.

If your course uses the G/VG grading scale:

- To get a VG, you must **also** pass **3** of the 4 advanced questions.

Grade	**Correct answers needed**
3/G	6 basic
4	6 basic + 2 advanced
5/VG	6 basic + 3 advanced

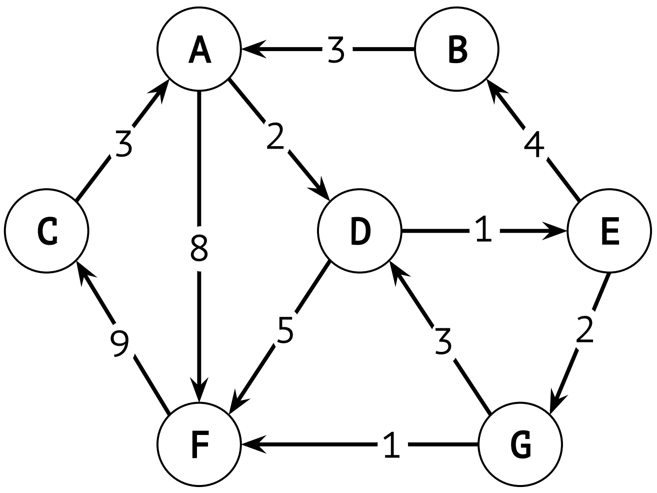
Write your anonymous code (*not* your name):

DAT038/DAT525, LP2 2024 only:

Check this box if you passed the exam 2024-12-15: ☐

Then you can skip questions 1–8. If you don't know what this is about you can safely ignore it.

Basic question 1: Graphs



You are given the directed weighted graph to the right.

Perform uniform-cost search (also known as Dijkstra's algorithm). In which order does the algorithm visit the vertices, and what is the computed distance to each of them?

If you start searching from node A?

	first visited						*last visited*
vertex	A						
distance from A	0						

If you start searching from node G?

	first visited						*last visited*
vertex	G						
distance from G	0						

Basic question 2: Sorting complexity

The following algorithm should sort an array A of *n* distinct numbers in descending order, in-place. However, the data structure to use for X remains to be specified:

```
X = new  ???  (initially empty)
for j in A:
    X.add(j)

for i in 0 to A.size()-1:
     x = get largest element from X
    remove largest element from X
    A[i] = x
```

What data structure(s) can we use to get a sorting algorithm with runtime O(*n* log(*n*))?

linked list	dynamic array	binary search tree	AVL tree
hash table	red-black tree	max-heap	min-heap

Circle **all** correct answers – there may be several.

Brief explanation of why the data structure(s) you chose are suitable:
(You do not need to explain why the other ones are unsuitable.)

Basic question 3: Ordered collections

Here is a program interacting with a collection data type:

```
S = new empty collection
S.add(3)
S.add(5)
print(S.remove())
S.add(4)
S.add(2)
S.add(8)
print(S.remove())
S.add(6)
print(S.remove())
```

Which numbers will be printed, and in which order, assuming that:

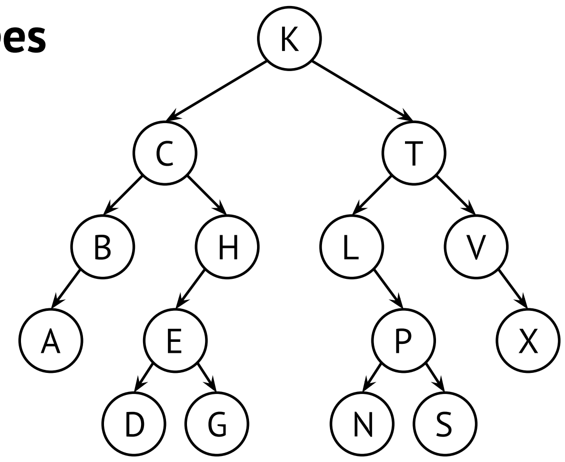
a) add/remove are enqueue/dequeue for a queue?

b) add/remove are push/pop for a stack?

c) add/remove are add/removeMin for a priority queue?

Basic question 4: Binary search trees

Given the following BST (binary search tree):

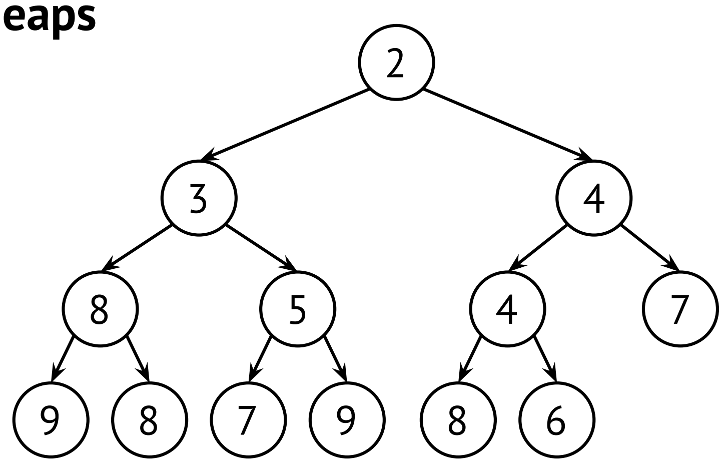


First, delete the node **K** from the tree (using the standard BST deletion procedure).
What does it look like after that?

Afterwards, reinsert **K** into the tree (using the standard BST insertion procedure).
What does it look like now?

Basic question 5: Binary heaps

Given the following binary heap:

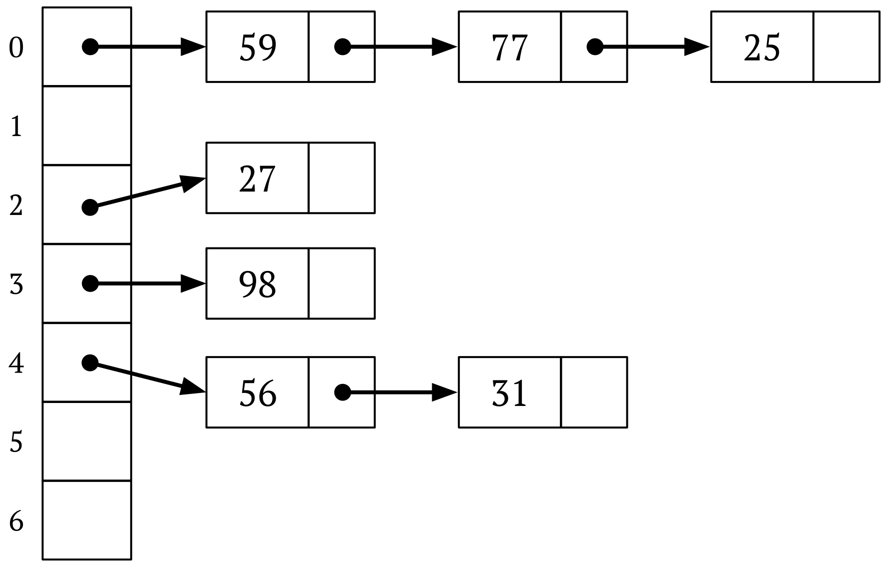


First, delete the minimum element from the heap – what does it look like after that?

Afterwards, reinsert the deleted number into the heap – how does it look like now?

Basic question 6: Hash tables

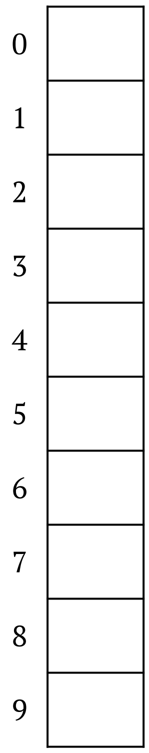
Here is a separate chaining hash table:



The hash function is the digit sum, so that e.g.:

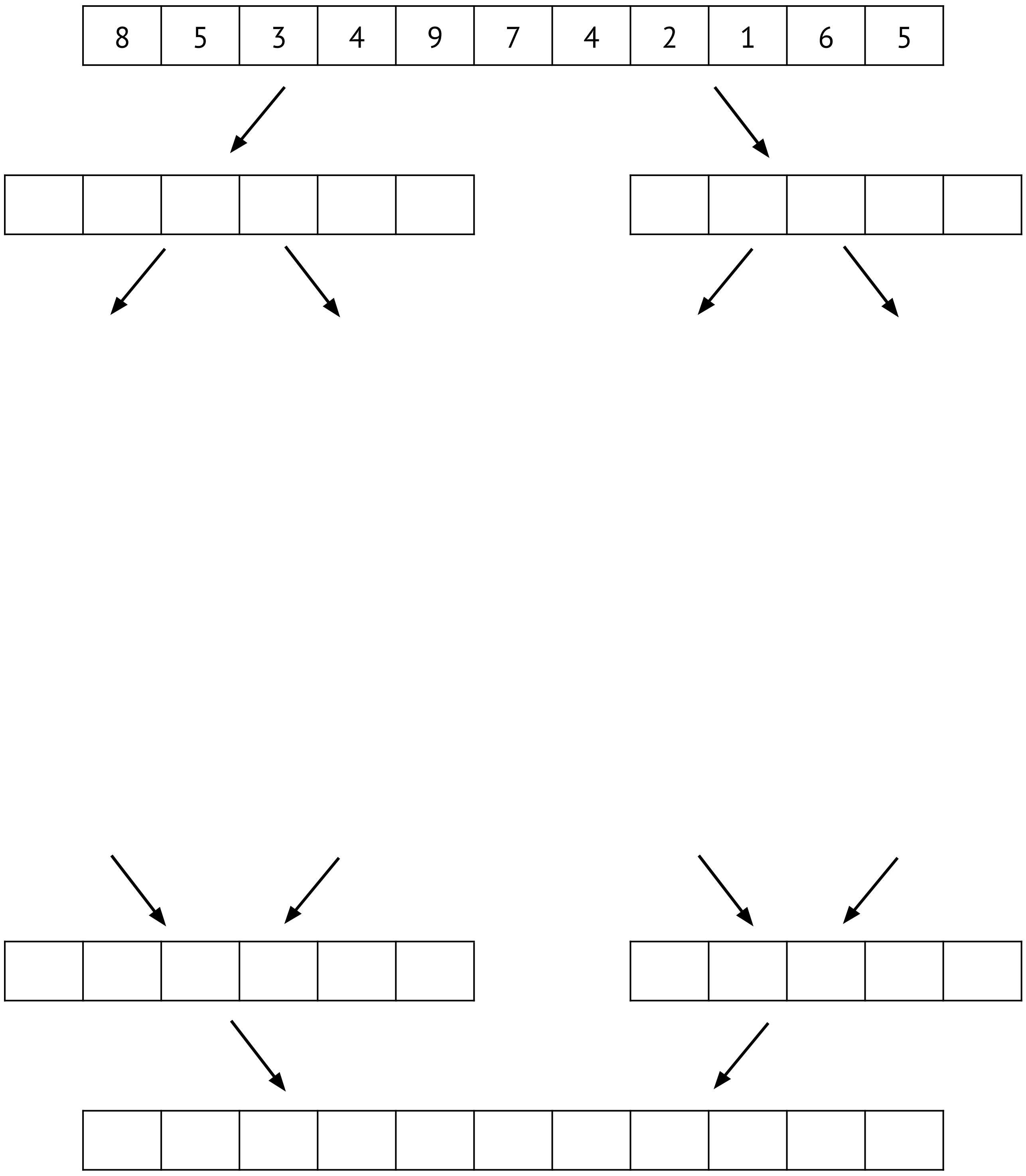
$h(358) = 3+5+8 = 16$

Resize the table to an array of 10 elements. What does the resulting hash table look like?
Make sure you use a consistent algorithm for deciding in which order elements should be inserted.



Basic question 7: Merge sort

Perform merge sort on the following list. Show each step of splitting and merging.



Basic question 8: Complexity of graph search

Here is an implementation of the general graph search algorithm.

```
graph_search(start, goal):
    visited = new set of vertices                    ______________O(1)______
    agenda = new priority queue of vertices          ______________O(1)______
    add start to agenda                              ______________O(1)______
    while agenda is not empty:                       ________________________
        current = remove a vertex from agenda        ________________________
        if current is not in visited:                ________________________
            add current to visited                   ________________________
            if current == goal:                      ______________O(1)______
                return true                          ______________O(1)______
            for next in neighbourVertices(current):  ________________________
                add next to agenda                   ________________________
    return false                                     ______________O(1)______
```

Answer the following questions asymptotically in terms of *N*, the number of vertices in the graph:

- How many times will the while-loop be iterated? ____________________
- What is the asymptotic complexity of the algorithm? ____________________

Write your answers in O-notation, and be as exact and simple as possible.
Justify your answer by annotating each line in the code. Some lines are already done for you.

You can assume the following:

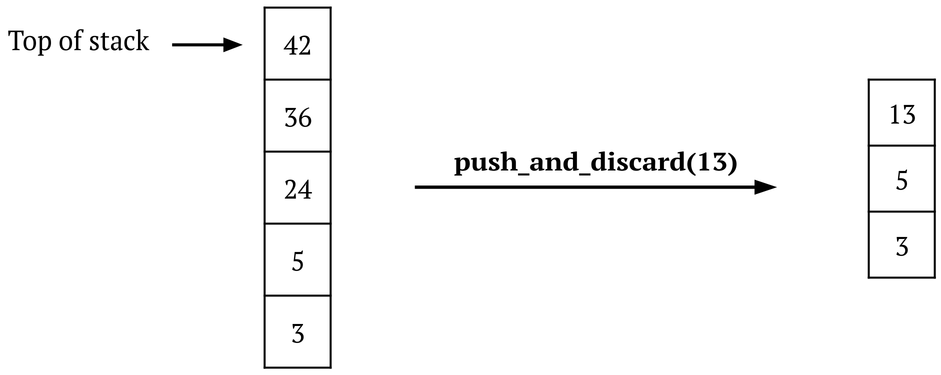
- the set `visited` is implemented as a *self-balancing binary search tree*
- the `agenda` is implemented as a *binary heap*
- there are at most a *constant number* of `neighbourVertices` per vertex
  (in other words, the graph is sparse)

Advanced question 9: Complexity of ordered stacks

An ordered stack is a stack where the elements appear in increasing order (from bottom to top). We implement the ordered stack as a linked list, maintaining a pointer to the top element. An ordered stack supports the operations:

- **init()**: creates an empty stack
- **pop()**: returns and deletes element on top of the stack
- **push_and_discard(*x*)**: pushes elements on top of the stack, maintaining the increasing order, i.e., elements larger than *x* are popped until *x* is the largest element in the stack

Below you see an example of an ordered stack and the resulting stack after performing a **push_and_discard** operation for the new element 13:



Answer the following questions on a separate sheet of paper.

A. Starting with an empty ordered stack, perform the following operations in the given order. Show the resulting stack after every operation.

push_and_discard(4)
push_and_discard(9)
push_and_discard(15)
push_and_discard(5)
push_and_discard(23)
push_and_discard(3)

B. What is the worst-case time complexity of **init**, **pop**, and **push_and_discard** in the size of the stack *N*? Write your answers in O-notation, and be as exact as possible.

C. If we start with an empty ordered stack, what is the amortized complexity of **push_and_discard**? Write your answer in O-notation, and be as exact as possible! Provide an explanation for your answer!

Hint: What is the worst-case complexity of performing *N* operations in sequence?

Advanced question 10: Diameter in a graph

Here are three definitions related to distance measures, copied from Wikipedia (https://en.wikipedia.org/wiki/Distance_(graph_theory)#Related_concepts):

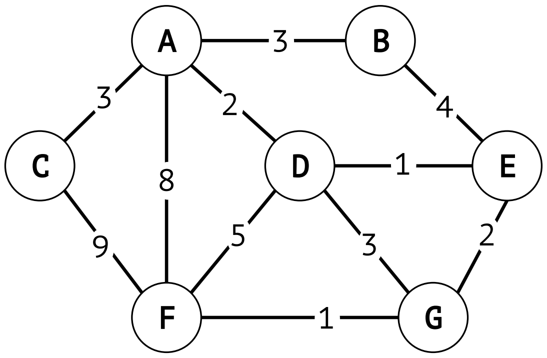
- The *distance* $d(v, u)$ between two vertices v and u is the length of a shortest path.
- The *eccentricity* $\epsilon(v)$ of a vertex v is the greatest distance between v and any other vertex:

  $$\epsilon(v) = \max_{u \in V} d(v, u)$$

  It can be thought of as how far a node is from the node most distant from it in the graph.
- The *diameter* d of a graph is the maximum eccentricity of any vertex in the graph.
  That is, d is the greatest distance between any pair of vertices or, alternatively:

  $$d = \max_{u \in V} \epsilon(u) = \max_{v \in V} \max_{u \in V} d(v, u)$$

A: What is the diameter of the graph on the right?



B: How can you find out the diameter of a graph?

Describe your algorithm as pseudocode, or as a detailed description. If you use data structures or algorithms from the course you don't have to describe how they work.

Write your answer on a separate sheet of paper.

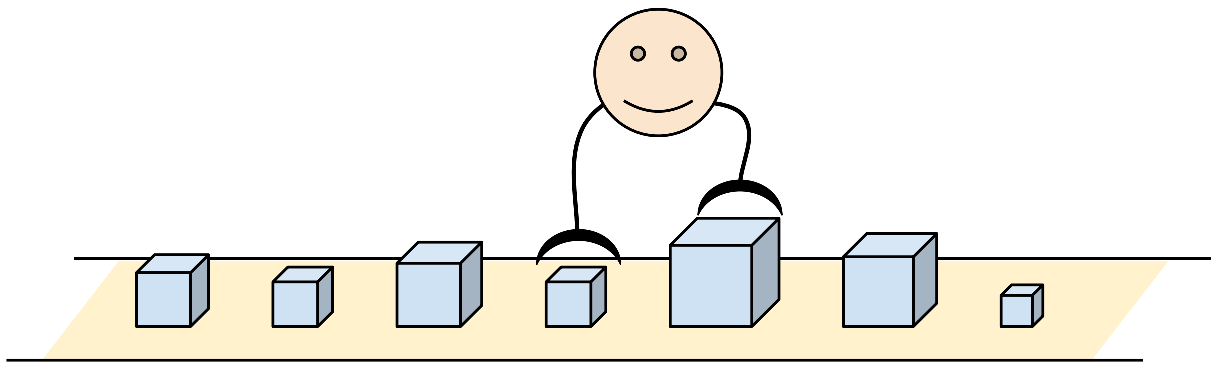
C: What is the asymptotic complexity of your algorithm?

Describe the complexity in terms of the number of vertices $N = |V|$.
You can assume that the graph is sparse, meaning that the number of edges is proportional to the number of vertices, or $|E| \in O(|V|)$.

Justify your answer.

Advanced question 11: Sorting robot

Assume you have a long line of *N* objects of different sizes, and a robot:



You want to instruct the robot to sort the long line (with the smaller objects to the left), but it has just a limited instruction set. The robot is always in front of two adjacent objects (as in the picture). The number of objects is known (called N) and the robot always starts in the leftmost position. The robot understands the following instructions:

- LEFT: move one step left – return true if it succeeded, otherwise false
- RIGHT: move one step right – return true if it succeeded, otherwise false
- COMPARE: compare the two objects in front – return true if the left object is smaller than the right object
- SWAP: swap places of the two objects in front – this always succeeds

A: Implement an algorithm that makes the robot sort the objects.

Write your answer on a separate sheet of paper.
You can use pseudocode, Java, Python, or some other programming language.

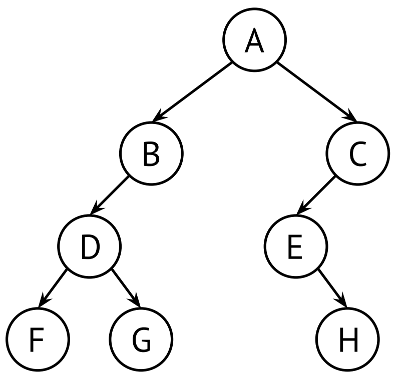
B: What is the asymptotic complexity of your algorithm in terms of *N*?

Justify your answer.

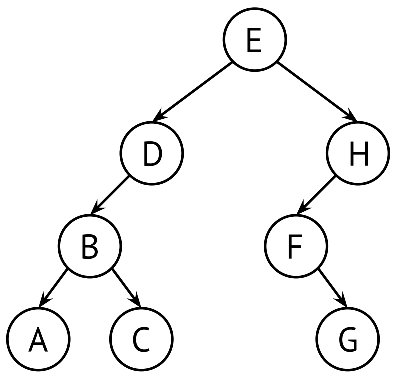
Advanced question 12: Convert a binary tree to a BST

Implement an algorithm that converts an unordered binary tree into a binary search tree (BST), without changing its skeleton, i.e., by retaining the structure of the tree. This means that the "skeleton" of the tree should remain the same, but the values of the nodes should move around so that the tree becomes a BST. Here's an example of how it should work:

The following unordered binary tree...



...should be transformed into this BST:



You can assume the following class definition for trees and nodes, where the values are strings. You can also assume that the node values can be compared in constant time.

```
class Node:
    left, right : Node
    value : String

class BinaryTree:
    root : Node
    size : int

    method convertToBST():
        ...this should be implemented...

    (...possible auxiliary private methods that you want to use...)
```

You can use any standard data structures and algorithms from the course without explaining how they work. You are allowed to create any kind of intermediate structure, but the final tree should maintain the same structure as the original tree.

Your solution must have worst-case complexity O(N log(N)) in the number of tree nodes N.

Hint: you can, e.g., use an array with all node values as an intermediate structure.

Write your answer on a separate sheet of paper.
You can use pseudocode, Java, Python, or some other programming language.