Re-exam – Datastrukturer

DIT961, VT-22
Göteborgs Universitet, CSE

Day: 2022-08-18, *Time:* 8:30-12.30, *Place:* Johanneberg

Course responsible

Alex Gerdes will visit at around 9:30 and is available via telephone 031-772 6154 during the entire exam.

Allowed aids

You may bring a dictionary.

Grading

The exam consists of *six questions*. For each question you can get a U, a G or a VG. To get a G on the exam, you need to answer at least *five* questions to G or VG standard. To get a VG on the exam, you need to answer at least three questions to a VG standard and all remaining questions to a G standard.

A fully correct answer for a question, including the parts labelled "For a VG", will get a VG. A correct answer, without the "For a VG" parts, will get a G. An answer with minor mistakes might be accepted, but this is at the discretion of the marker. A correct answer on the VG part may compensate a mistake in the G part, also at the discretion of the marker. An answer with large mistakes will get an U.

Inspection

When the exams have been graded they are available for review in the student office on floor 4 in the EDIT building. If you want to discuss the grading, please contact the course responsible and book a meeting. In that case, you should leave the exam in the student office until after the meeting.

Note

- Begin each question on a new page and write your anonymous code (not your name) on every page.
- When a question asks for pseudocode, you don't have to write precise code, such as Java. But your answer should be well structured. Indenting and/or using brackets is a good idea. Apart from being readable your pseudocode should give enough detail that a competent programmer could easily implement the solution, and that it's possible to analyse the time complexity.
- Excessively complicated answers might be rejected.
- *Write legibly!* Solutions that are difficult to read are not evaluated!

Exercise 1 (complexity)

The following function produces a copy of an array of values sorted according to a separately specified array of keys. We assume that the keys and values arrays have the same length n. We are interested in the asymptotic complexity of `sort_values` in terms of n. We assume that comparing two keys takes constant time.

```
function sort_values(keys : Array, values : Array) -> Array:
        map = new SortedMap()
        i = 0
        while i < length(keys):
                map.put(keys[i], values[i])
                i = i + 1
        result = new Array of size length(keys)
        i = 0
        for each key, value in the entries of map:
                result[i] = value
                i = i + 1
        return result
```

a) What is the asymptotic complexity if `SortedMap` is a Binary Search Tree (BST)?

b) What is the asymptotic complexity if `SortedMap` is an AVL Tree?

Write your answer in O-notation. Be as exact and simple as possible. Justify why the complexity of the function has this order of growth.

For a VG only:

Three more questions about the function `sort_values`:

c) Give an example that displays the worst-case complexity when `SortedMap` is a BST. Answer with example arrays `keys` and `values` that make `sort_values` perform at the worst-case complexity you answered in 1*a*.

Note that since the complexity depends on the size n of the input, your example should be generic in n, that is, you should make it clear how the `keys` and `values` arrays would look for any value of n.

d) The function will not work as intended if `SortedMap` is a hash table. Explain why.

e) However, if we used a hash table, the function would still be able to run. What would the worst-case complexity of `sort_values` be in that case (assuming a perfect hash function)?

Exercise 2 (sorting)

Perform one level of split+sort+merge when *merge sorting* the following list:

```
[42,12,54,72,36,82,99,66,6,27]
```

Show the following:

a) The result of splitting the input list.

b) The result of calling the merge sort function on each of the splits.

c) The result of merging the sorted sublists. Explain how you do it.

For a VG only:

Your tasks are:

d) Define a `merge` function in Haskell that merges two sorted lists into a new sorted list:

```haskell
merge :: Ord a => [a] -> [a] -> [a]
```

The function should have a linear worst-case time complexity.

e) Consider the following alternative implementation of merge sort:

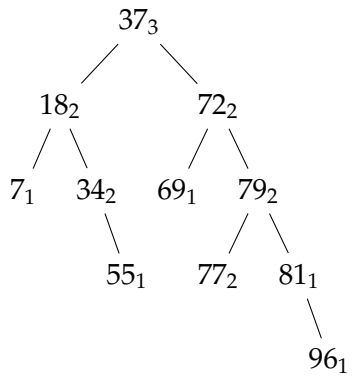
```haskell
mergeSort :: Ord a => [a] -> [a]
mergeSort = go []
 where
  go acc []     = acc
  go acc (x:xs) = go (merge [x] acc) xs
```

What is the worst-case time complexity of this function? Motivate your answer!

f) What is the best- and worst-case input for the `mergeSort` function?

Exercise 3 (search trees)

The tree below is neither a valid AVL tree nor is it a valid AA tree (the sub-scripts denote the AA tree level of a node).



You must answer the following questions:

a) Why is the tree not a valid AA tree? Explain how and where (in the tree) the invariant is invalidated.

b) Is it possible to change the level of the nodes such that the tree becomes a valid AA tree? Either give a new tree with updated levels or explain why the tree can't be made into a valid AA tree.

c) Why is the tree not a valid AVL tree? Explain how and where (in the tree) the invariant is invalidated. You can ignore the AA tree level annotations.

d) Perform one tree rotation to restore the AVL invariant and draw the resulting tree. Explain which rotation you performed and where it was done in the tree.

For a VG only:

Now consider the above tree as a plain BST.

e) Delete the root node (67) from the tree, and show the resulting tree.

f) There are two possible ways to delete a node which has two children. Now show the resulting tree after deleting 67 from the initial tree in the alternative way.

g) The following data type models a binary search tree in Haskell:

```
data Tree a
  = Empty
  | Node (Tree a) a (Tree a)
```

Implement the function

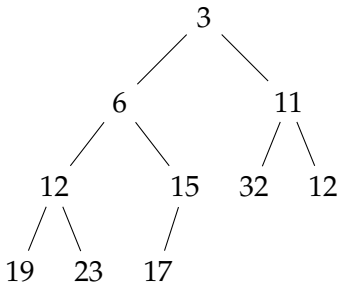
```
delete :: Ord a => a -> Tree a -> Tree a
```

which removes the given value from the tree (if it exists). You can pick any of the two possible ways to delete nodes with two children.

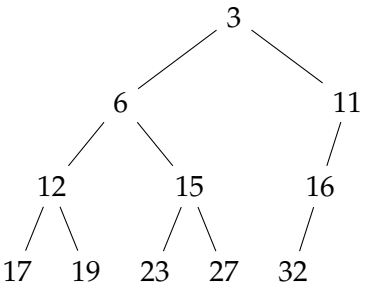
Exercise 4 (heaps)

Which of the following trees represent a binary min-heap?

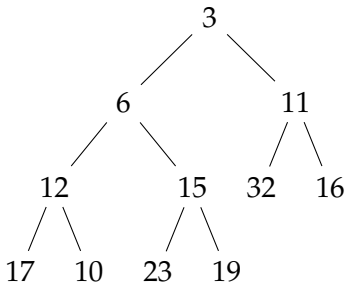
A)



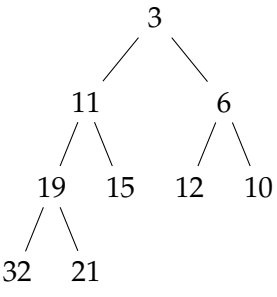
C)



B)



D)



For each of the trees:

- if it is not a binary heap, explain why not – i.e. point out a concrete specific problem with it,
- if it is a binary heap, write it in array notation.

For a VG only:

a) Explain how to represent a binary min-heap as an array. Make sure to describe all invariant(s).

b) Give pseudocode for inserting a new element into a heap represented as a dynamic array.

Exercise 5 (basic data structures: hash table and queue)

For each of the following questions, select the correct answer (1, 2 or 3).

Assume that you don't know anything at all about the hash function except that it takes constant time to calculate hash values.

a) What is the worst-case asymptotic time complexity, in terms of n, of adding one element to a separate-chaining hash table containing n elements?

(a) Constant, $O(1)$

(b) Logarithmic, $O(\log n)$

(c) Linear, $O(n)$

b) What is the most efficient way of resizing an open-addressing hash table that is close to full?

(a) Create a new table, roughly twice as large. Add each element from the old table to the new table, using normal hash table insertion.

(b) Create a new table, roughly twice as large. Add each element from the old table to the cell that is indicated by the new hash value.

(c) Create a new table, roughly twice as large. Since the hash values are stable, you can simply put the elements in the same positions as they were in the old table.

c) Which expression most accurately describes the worst-case time complexity of searching for an element in a separate-chaining hash table, with table size m, containing n elements, and buckets as linked lists of size at most k?

(a) Linear in the table size, $O(m)$

(b) Linear in the number of elements, $O(n)$

(c) Linear in the bucket size, $O(k)$

d) How can you delete an element `d` in an open-addressing hash table using linear probing?

(a) Find the element `d`, and clear the cell.

(b) Find the element `d`, and set the value of the cell to a special "deleted" value.

(c) Find the element `d`, then continue searching for the last element `x` in the cluster. Move `x` to the cell where `d` is (thus deleting `d`), and then clear the old `x` cell.

For a VG only:

Assume we have the following interface for queues in Java:

```
interface Queue<E> {
  void enqueue(E elem);
  E dequeue();
  boolean isEmpty();
}
```

a) What is the worst-case time complexity (in terms of n, the number of elements in the queue) for the different methods if you implement the interface with a singly linked list?

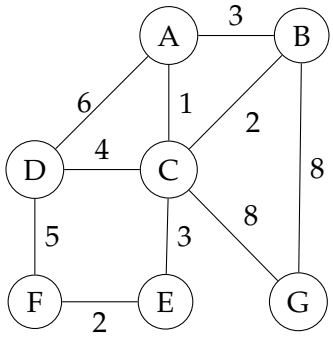
Hint: we can have multiple pointers to different nodes in the queue.

b) Give an implementation for the above `Queue` interface that uses a singly linked list.

c) Add a `size` method, which returns the number of elements in the queue, to the implementation. The method must have a constant worst-case time complexity $O(1)$.

Exercise 6 (graphs)

You are given the following undirected weighted graph:



Suppose we perform Dijkstra's algorithm starting from node A. In which order does the algorithm visit the nodes, and what is the computed distance to each of them? In addition, show the contents of the priority queue after each visited node. Explain your answer.

For a VG only:

Describe an algorithm (in pseudocode or your favorite programming language) that detects if a directed graph contains a cycle.

Assume that the graph is given in adjacency list representation:

- you can iterate over each vertex (e.g. `for each vertex v`),
- for each vertex `v`, you can iterate over each outgoing edge (e.g. `for each edge e from v`).

Your algorithm should be reasonably efficient, close to $O(V + E)$ in terms of V vertices and E edges. You can use any data structure from the course.