Exam
Data structures DIT960

Time	Friday 5h June 2015, 14:00–18:00
Place	Väg och vatten
Course responsible	Nick Smallbone, tel. 0707 183062

The exam consists of **six questions**.

For a **G**, you need to answer **three** questions correctly.
You can ignore any parts labelled **"VG"**.

For a **VG**, you need to answer **five** questions correctly.
You must also answer all parts labelled **"For a VG"** in those questions.

For an answer to be considered correct, it must contain no major mistakes.
Minor mistakes might be accepted, but this is at the discretion of the marker.

When a question asks for **pseudocode**, you can use a mixture of English and programming notation to describe your solution, and should give enough detail that a competent programmer could easily implement your solution.

Allowed aids One A4 piece of paper of hand-written notes, which should be handed in after the exam. You may write on both sides.

You may also bring a dictionary.

Note Begin each question on a new page.

Write your anonymous code (*not* your name) on every page.

Good luck!

1. The following algorithm takes as input an array which may contain duplicates. It returns `true` if all elements of the array occur an odd number of times. Otherwise it returns `false`.

   For example, on the array `{1, 3, 2, 2, 5, 2}` it returns `true`, but on the array `{1, 3, 2, 2, 5, 2, 5}` it returns `false` because 5 occurs an even number of times.

```
S = new AVL tree
for every element x in input array
  if S.member(x)
    S.delete(x)
  else
    S.insert(x)
// At this point, S contains those elements that
// occur an odd number of times

for every element x in input array
  if not S.member(x)
    return false
return true
```

   Assuming that n is the length of the input array, what is the big-O complexity of this algorithm?

2. Which array out of A, B and C represents a binary heap? Only one answer is right.

A =

0	1	2	3	4	5	6	7	8	9	10	11
1	12	23	10	15	38	45	15	18	20	21	

B =

0	1	2	3	4	5	6	7	8	9	10	11
1	8	27	10	45	83	91	31	12	52	51	

C =

0	1	2	3	4	5	6	7	8	9	10	11
1	13	20	21	65	54	67	41	30	83	52	

a) Write the heap out as a binary tree.

b) Add 23 to the heap, making sure to restore the heap invariant. How does the array look now?

3. Design an algorithm that takes:

   - An array containing n distinct natural numbers
   - A number $k \leq n$

   and calculates the sum of the k largest numbers in the array.

   For example, if the array is {3, 7, 5, 12, 6} and $k = 3$, then the algorithm should return 25 (12+7+6).

   You may freely use standard data structures and algorithms from the course in your solution, without explaining how they are implemented.

   Write down your algorithm as **pseudocode** – you don't need to write fully detailed Java code.

   For a G: your algorithm should take O(n log n) time.

   For a VG: your algorithm should take O(n log k) time.

4. Design a data structure for storing a set of integers. It should support the following operations:
    - `new()`: create a new, empty set
    - `insert(x)`: add an integer `x` to the set
    - `member(x)`: test if a given integer `x` is in the set
    - `increaseBy(x)`: add `x` to all the integers in the set

    For example, calling `increaseBy(2)` on a set containing the values 1,2,3,4,5 should give a set containing the values 3,4,5,6,7.

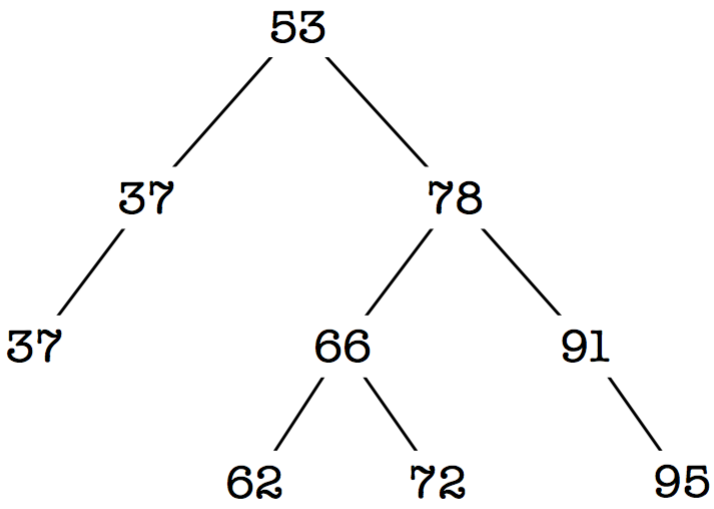
    You may freely use standard data structures and algorithms from the course in your solution, without explaining how they are implemented.

    You should say what design or existing data structure you have chosen, and give **pseudocode** for each of the operations – you don't need to write fully detailed Java code.

    The operations must have the following time complexities:

    - **For a G:**
      O(1) for `new`,
      O(log n) for `insert/member`,
      O(n) for `increaseBy`
      (where n is the number of elements in the set)
    - **For a VG:**
      as for G but the complexity of `increaseBy` must be O(1).

5. You are given the following binary search tree.



a) Colour the nodes of the tree red and black so that it becomes a valid red-black tree. If you don't have a coloured pen, you could e.g. draw a circle for red nodes and a square for black nodes.

b) Insert 60 into the tree using the red-black insertion algorithm. Write down the final tree.

6. Suppose we are given the following type of binary search trees in Haskell:

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

a) Implement a Haskell function

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

that returns the greatest element in a non-empty binary search tree (in an empty binary search tree it is allowed to crash).

The complexity of your function should be O(height of tree), i.e., O(log n) for balanced trees, O(n) for unbalanced trees.

b) **For VG only:**
Write a Haskell function to delete an element. It should take two parameters, which are the element to delete and the tree, and have the following type:

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

The complexity of your function should be O(height of tree), i.e., O(log n) for balanced trees, O(n) for unbalanced trees.

Hint: it will help to use `greatest` when implementing `delete`.