Re-exam
Data structures DIT960

Time	Tuesday 27th August 2013, 08:30–12:30
Place	V-huset
Course responsible	Nick Smallbone, tel. 0707 183062

The exam consists of **six easy questions** (**nos. 1–6**) and **three hard questions** (**nos. 7–9**).

For *Godkänd* you need to answer at least **four easy questions** correctly.

For *Väl Godkänd* you *also* need to answer **two hard questions** correctly.

Allowed aids One A4 piece of paper of notes, which may be hand-written or typed. 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.

Excessively complicated answers might be rejected.

Write legibly!

1. 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
3	5	15	7	18	22	35	30	9	17		

☐ B =

0	1	2	3	4	5	6	7	8	9	10	11
3	5	18	15	7	22	35	30	9	17		

☐ C =

0	1	2	3	4	5	6	7	8	9	10	11
3	5	18	7	15	22	35	30	9	17		

a) Write the heap out as a binary tree.

Remove the smallest element from the heap, making sure to restore the heap invariant. How does the array look now?

0	1	2	3	4	5	6	7	8	9	10	11

2. Perform an *insertion sort* on the following array. Your sort must be in-place, so you must not use a temporary array.

0	1	2	3	4	5	6	7	8
53	71	32	67	90	23	44	88	42

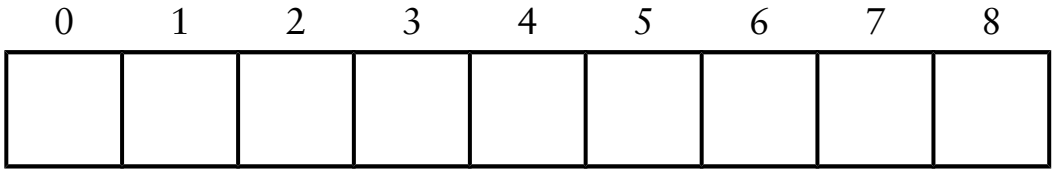
Show how the array looks after each execution of the outer loop. Also show which part of the array is sorted, for example by underlining it.

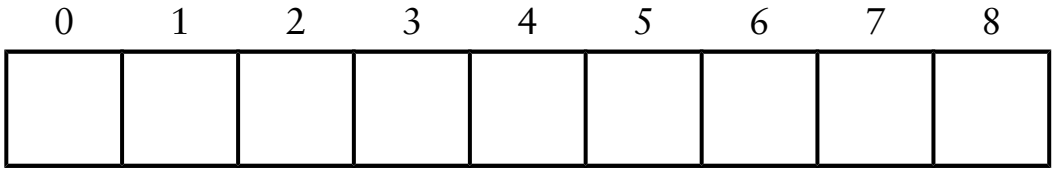
0	1	2	3	4	5	6	7	8

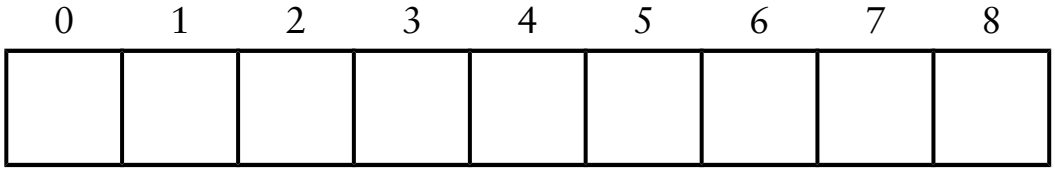
0	1	2	3	4	5	6	7	8

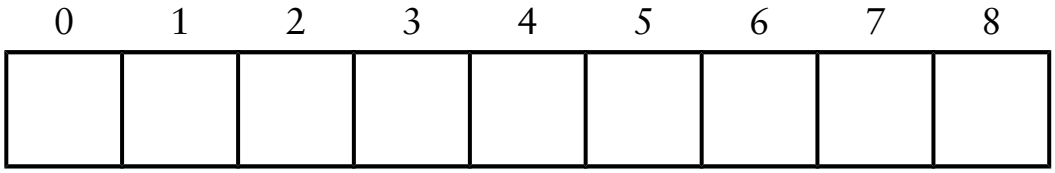
0	1	2	3	4	5	6	7	8

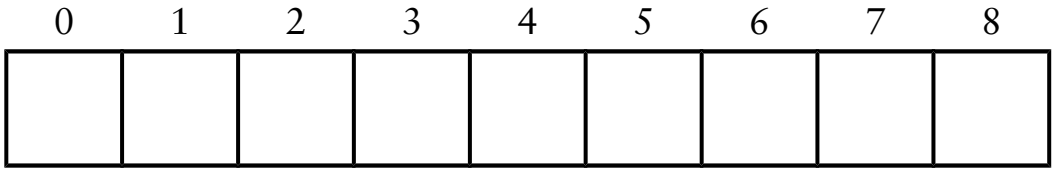
0	1	2	3	4	5	6	7	8


0 1 2 3 4 5 6 7 8

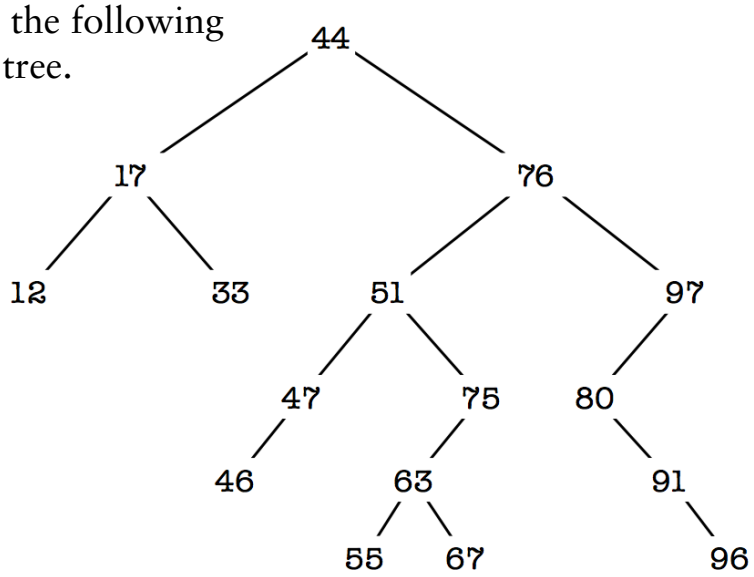

0 1 2 3 4 5 6 7 8


0 1 2 3 4 5 6 7 8


0 1 2 3 4 5 6 7 8


0 1 2 3 4 5 6 7 8

3. You are given the following binary search tree.



a) In which order could the elements 17, 33, 47, 63, 67 and 76 have been added to the tree? There are several correct answers, and you should choose them all.

- ☐ **A** = 17, 33, 47, 63, 67, 76
- ☐ **B** = 17, 76, 33, 63, 47, 67
- ☐ **C** = 17, 76, 47, 33, 67, 63
- ☐ **D** = 76, 63, 67, 47, 17, 33
- ☐ **E** = 76, 67, 17, 33, 47, 63

b) Remove 76 from the tree, then insert it again. How does the tree look?

4. In this question you will show how *partitioning* works in Quicksort.

0	1	2	3	4	5	6	7	8
53	88	32	44	23	90	67	71	42

Below, you are given several choices of *pivot element*. For each pivot, show the result of partitioning the above array.

You should also mark which parts of the array need to be recursively quicksorted after the partitioning, for example by underlining them.

a) Pivot = first element

0	1	2	3	4	5	6	7	8

b) Pivot = middle element

0	1	2	3	4	5	6	7	8

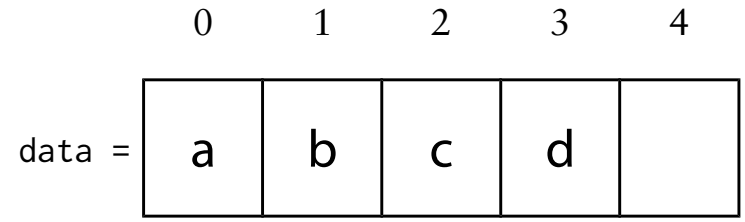
c) Pivot = last element

0	1	2	3	4	5	6	7	8

5. Suppose we have a class `ArrayQueue` which implements a queue as a circular array with the following private variables and public methods:

```
class ArrayQueue<E> implements Queue<E> {
  private E[] data;
  private int front, rear;
  public ArrayQueue();
  // Add an item to the queue.
  public boolean offer(E item);
  // Remove an item from the queue.
  public E poll();
}
```

If we create a queue q, insert four elements and then remove one, then q's instance variables will be as follows afterwards:



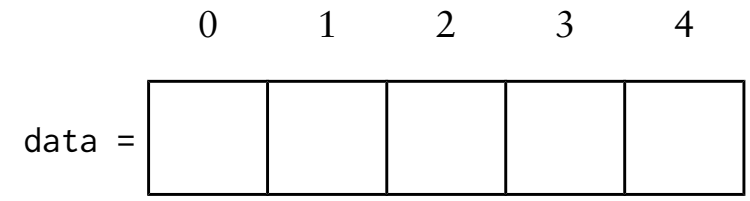
`front` = 1

`rear` = 3

Suppose that we now execute the following sequence:

```
q.offer("e");
q.offer("f");
q.poll();
q.poll();
q.offer("g");
q.poll();
```

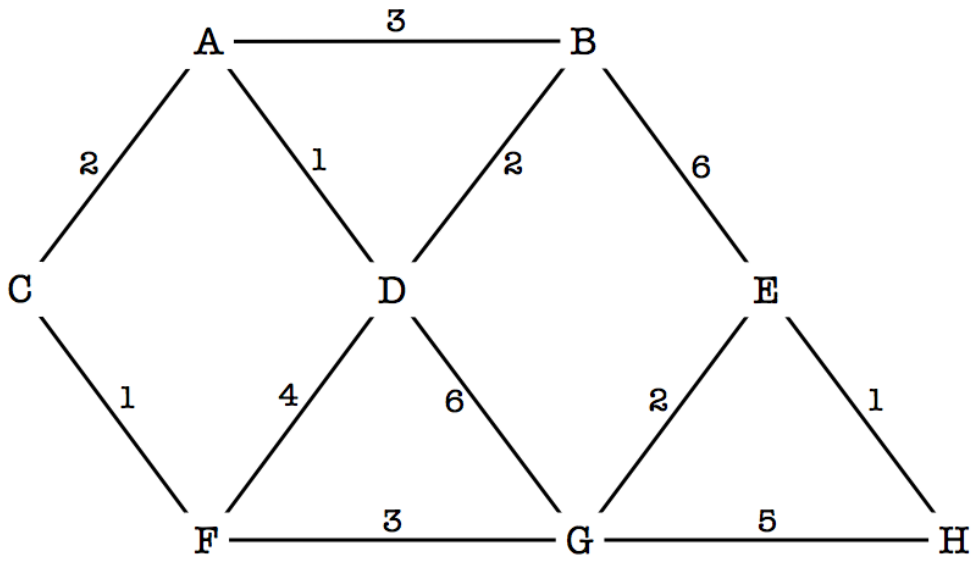
What will q's instance variables contain afterwards?



front =

rear =

6. You are given the following weighted graph:



a) Do a breadth-first search of the graph, starting from node **A**. Which order do you visit the nodes in? There are several possible orders, and you should choose any **two** of them.

Node	A							
Node	A							

b) Perform Dijkstra's algorithm starting from node **A**. At each step the algorithm visits a new node. In which order are the nodes visited, and what is the distance to each of them? There are several possible orders to visit the nodes in – you may choose any of them.

Node	A							
Distance	0							

c) What is the shortest path to node **H**? List all the nodes along the way.

7. (**hard**) Define a method `sort` that sorts an array of **booleans**. (When comparing booleans, `false < true`.) It should have the following signature:

   ```
   void sort(boolean[] array);
   ```

   Your method should take **O(n)** time and **O(1)** memory. You may not use the Java `new` operator.

   Hint: the idea is the same as one of the sorting algorithms from the course. You can solve the problem in less than 10 lines of code.

8. (**hard**) Consider the following implementation of binary search.

```
 1 int search(Object[] array, Object key) {
 2    int low = 0;
 3    int high = array.length - 1;
 4    while(low <= high) {
 5       int mid = (low+high)/2;
 6       if (array[mid].compare(key) < 0)
 7          low = mid+1;
 8       else if (array[mid].compare(key) > 0)
 9          high = mid-1;
10       else
11          return mid;
12       }
13    }
14    return -1;
15 }
```

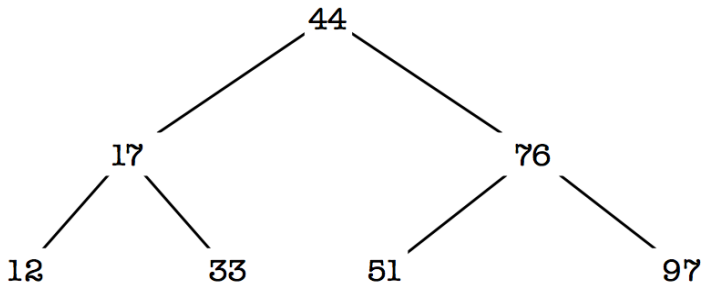
Suppose we change the code in the following ways. For each change, say whether the code will still work as intended. If not, explain what can go wrong.

a) Change the test in line **4** to **`low < high`**.

b) Change line **5** to **`int mid = low;`**

c) Change line **7** to **`low = mid;`**

9. (**hard**) Here is an example of a *perfect binary tree*.



A binary tree is *perfect* if all of its levels are full.

Your job is to write a Haskell function `perfect :: Tree a → Bool` that takes a binary tree and finds out whether it is perfect or not. **It should run in O(n) time,** where n is the size of the tree. `Tree a` is the usual datatype of binary trees in Haskell:

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

Hint: If you try to define `perfect` directly, it will take longer than O(n) time. Instead, you will need to define a more general function `perfect'` that returns both:

- whether the tree is perfect
- the height of the tree, when the tree is perfect

With the help of this function, you can define `perfect`.