Re-exam – Datastrukturer

DIT961, VT-19
Göteborgs Universitet, CSE

Day: 2019-10-11, *Time:* 8:30-12.30, *Place:* M

Course responsible

Alex Gerdes, tel. 031-772 6154. Will visit at around 9:30 and 11:00.

Allowed aids

One hand-written sheet of A4 paper (both sides). You may also 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 five questions to VG standard (and the remaining question 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.
- 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)

Your task is to describe the complexity of the following three code fragments. The complexity should be expressed in terms of n, the size of the input. You should express the complexity in the simplest form possible. All answers need to be well motivated!

a)
```
for (int count = 0; count < n; count++) {
  /* statement with O(1) time complexity */
}
```

b)
```
for (int count = 0; count < n; count ++) {
  for (int count2 = 1; count2 < n; count2 = count2 * 2) {
    /* statement with O(n) time complexity */
  }
}
```

c)
```
for (int i = 0; i < n; ++i) {
  for (int j = i; j < n; ++j) {
    /* statement with O(n) time complexity */
  }
}
```

For a VG only:

Which of the following statements are correct?

1. A stack can be implemented efficiently with a linked list.
2. An algorithm with $O(n^2)$ complexity is always slower than an algorithm with $O(n \log n)$ complexity.
3. If an array is sorted then it is better to use merge sort instead of insertion sort.
4. The complexity of finding an element in a binary tree is $O(\log n)$.

Motivate your answer!

Exercise 2 (sorting)

Perform a quicksort-partitioning of the following array:

46	14	52	58	64	3	8	24	32
0	1	2	3	4	5	6	7	8

Show the resulting array after partitioning it with the following pivot elements:

a) first element

b) middle element (swap the first and middle element before partitioning)

c) last element (swap the last and first element before partitioning)

Also, highlight which subarrays that need to be sorted using a recursive call. (for example by drawing a line under each subarray)

For a VG only:

Define a Haskell function `sort` that sorts a list of elements using the insertion sort algorithm. It should have the following type signature:

```
sort :: Ord a => [a] -> [a]
```

Hint: define a help function `insert`. You can solve the problem in less than 10 lines of code. Write a recurrence relation for each function you have defined and use these to give the worst-case time complexity for the `sort` function.

Exercise 3 (basic data structures: hash table)

Suppose you have the following hash table, implemented using *linear probing*. For the hash function we are using the identity function, $h(x) = x \bmod 9$, where mod is the modulo operator that calculates the remainder of a division.

35	71			4	68	60	70	44
0	1	2	3	4	5	6	7	8

a) In which order could the elements have been added to the hash table? There are several correct answers, and you should choose them all.

A) 44, 35, 68, 71, 60, 70, 4

B) 35, 68, 71, 44, 60, 4, 70

C) 68, 70, 44, 35, 71, 60, 4

D) 68, 71, 44, 35, 60, 70, 4

E) 4, 35, 68, 71, 70, 60, 44

b) **For a VG only:** Add 79 to the hash table (assume there is no rehashing). What does the hash table look like now?

c) **For a VG only:** Remove 35 from the hash table. What does it look like now?

Exercise 4 (heaps)

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

A

15	49	39	23	59	63	50	71	70	66
0	1	2	3	4	5	6	7	8	9

B

3	17	5	67	12	62	56	74	72	79
0	1	2	3	4	5	6	7	8	9

C

5	13	11	41	23	54	35	74	60	42
0	1	2	3	4	5	6	7	8	9

a) Write out that heap as a binary tree.

b) **For a VG only:** Add 12 to the heap, making sure to restore the heap invariant. How does the array look now?

c) **For a VG only:** If we want a hash table that stores a set of strings, one possible hash function is the string's length, $h(x) = x$.length. Is this a good hash function? Explain your answer.

Exercise 5 (search trees)

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

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

a) Implement a Haskell function:

```
member :: Ord a => a -> Tree a -> Bool
```

that checks if a given value is an element in the binary search tree. The complexity of your function should be $O(\text{height of tree})$, i.e., $O(\log n)$ for balanced trees, $O(n)$ for unbalanced trees.

b) Write a function that adds a value to a binary tree with the following type signature:

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

while preserving the binary search tree invariant.

c) **For a VG only:**

An inorder traversal of a binary tree goes through the left subtree first, then visits the node, and continues with the right subtree. Implement a function that performs an inorder traversal of a binary search tree:

```
inorder :: Tree a -> [a]
```

and returns the visited elements in a list.

d) **For a VG only:**

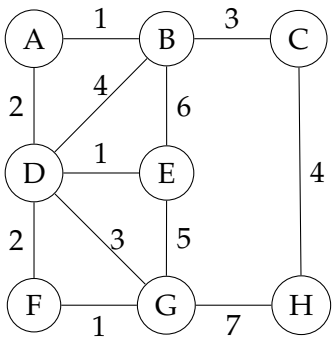
A breadth first traversal goes through a tree level by level. Write a function that traverses a binary search tree in a breadth first manner:

```
bfs :: Tree a -> [a]
```

and collects the elements in a list. *Hint:* use a queue to remember which nodes to visit.

Exercise 6 (graphs)

You are given the following undirected weighted graph:



For a G you may choose one of the following questions, for a VG you need to answer both.

a) Compute a minimal spanning tree for the following graph by manually performing Prim's algorithm using H as starting node.

Your answer should be the set of edges which are members of the spanning tree you have computed. The edges should be listed in the order they are added as Prim's algorithm is executed. Refer to each edge by the labels of the two nodes that it connects, e.g. DF for the edge between nodes D and F.

b) 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? (explain your answer)