Exam – Datastrukturer

DIT961, VT-19
Göteborgs Universitet, CSE

Day: 2019-06-05, *Time:* 8:30-12.30, *Place:* SB

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. You may use 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.

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. 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)

The following three methods perform the same task: given an integer array (which can contain positive and negative numbers) calculate the maximum sum of all elements in a subsequence (without gaps). Your task is to describe the complexity of the three methods. The complexity should be expressed in terms of n, the size of the input array. You should express the complexity in the simplest form possible. All answers need to be well motivated!

a)
```
public static int maxSubsequenceSumA(int[] a) {
  int maxSum = 0, thisSum = 0, start = 0, end = 0;
  for (int i = 0, j = 0; j < a.length; j++) {
    thisSum += a[j];
    if (thisSum > maxSum) {
      maxSum = thisSum; start = i; end = j;
    } else if (thisSum < 0) {
      i = j + 1; thisSum = 0;
    }
  }
  return maxSum;
}
```

b)
```
public static int maxSubsequenceSumB(int[] a) {
  int maxSum = 0, start = 0, end = 0;
  for (int i = 0; i < a.length; i++) {
    int thisSum = 0;
    for (int j = i; j < a.length; j++) {
      thisSum += a[j];
      if (thisSum > maxSum) {
        maxSum = thisSum; start = i; end = j;
      }
    }
  }
  return maxSum;
}
```

c)
```
public static int maxSubsequenceSumC(int[] a) {
  int maxSum = 0, start = 0, end = 0;
  for (int i = 0; i < a.length; i++) {
    for (int j = i; j < a.length; j++) {
      int thisSum = 0;
      for (int k = i; k <= j; k++)
        thisSum += a[k];
      if (thisSum > maxSum) {
        maxSum = thisSum; start = i; end = j;
      }
    }
  }
  return maxSum;
}
```

For a VG only:

The following Haskell code also calculates the maximum subsequence sum. Your task is to give the worst-case complexity (again in terms of n, the size of the input list) of the `powerslice` function. Begin by writing recurrence relations for the `tails` and `inits` functions and then continue with reasoning about the `powerslice` function. You don't need to analyse the `maxsubseq` function. Motivate your answer!

```
tails :: [a] -> [[a]]
tails [] = [[]]
tails xs = xs : tails (tail xs)

inits :: [a] -> [[a]]
inits [] = [[]]
inits xs = xs : inits (init xs)

powerslice :: [a] -> [[a]]
powerslice = concat . map inits . tails

maxsubseq :: [Integer] -> Integer
maxsubseq = maximum . map sum . powerslice
```

Note that `concat` has $O(m)$ worst-case time complexity, where m is the sum of the number of elements in each sub-list.

Exercise 2 (sorting)

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

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

Your function may have $O(n^2)$ worst-case time complexity. *Hint*: define a help function `bubble`. You can solve the problem in less than 10 lines of code.

For a VG only:

Perform a quicksort-partitioning of the following array:

15	58	5	30	79	29	4	14	22
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)

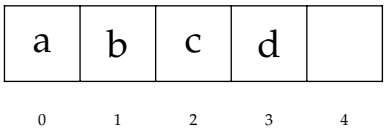
Also, highlight which subarrays that need to be sorted using a recursive call. (for example by drawing a line under each subarray)

Exercise 3 (basic data structures)

Assume we have a class `BoundedQueue` that implements a fixed-size queue using a circular buffer. It has the following instance variables and methods:

```
class BoundedQueue<E> {
  private Object[] data;
  private int front, back, size;
  public BoundedQueue(int capacity);
  public void enqueue(E item);
  public E dequeue();
}
```

If we create a queue `q` with a capacity for 5 elements and enqueue four elements (`a` to `d`) and then dequeue once, then the instance variable `data` contains the following:



and `front` is 1, `back` is 4, and `size` contains 3. That is, `front` points to the first element in the circular buffer and `back` points to the index after the last element in the circular buffer. Now assume we run the following sequence of statements:

```
q.enqueue("e");
q.enqueue("f");
q.dequeue();
q.dequeue();
q.enqueue("g");
q.dequeue();
```

What do `q`'s instance variables (`data`, `front`, `back`, `size`) contain?

For a VG only:

Give an implementation for the public `dequeue` method. Remember that the `BoundedQueue` class implements a *circular* buffer.

Exercise 4 (heaps)

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	8	17	42	9	39	16	72	48	32	24	

B

0	1	2	3	4	5	6	7	8	9	10	11
2	8	17	21	19	22	41	76	32	26	20	

C

0	1	2	3	4	5	6	7	8	9	10	11
0	10	16	33	19	55	24	80	11	74	54	

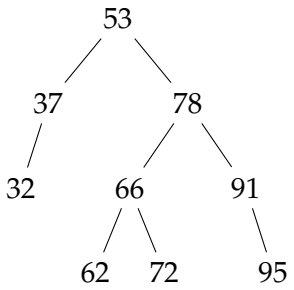
a) Write out that heap as a binary tree.

b) **For a VG only:** Add 13 to the heap, making sure to restore the heap invariant. How does the array look now?

c) **For a VG only:** Describe the procedure how we can build a heap from an arbitrary array. Answer either in pseudocode or English (or Swedish).

Exercise 5 (search trees)

You are given the following AVL tree:

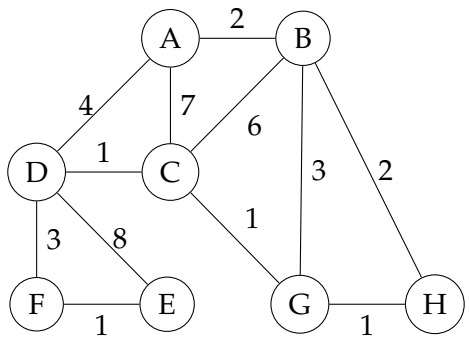


a) Mark each node with its AVL balance (right height minus left height).

b) **For a VG only:** Insert 60 into the tree and balance it to restore the AVL invariant. Write down the final tree.

Exercise 6 (graphs)

You are given the following undirected weighted graph:



a) Compute a minimal spanning tree for the following graph by manually performing Prim's algorithm using A 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) **For a VG only:** 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)