# Suggested solutions – Exam – Datastrukturer

DIT961, VT-19
Göteborgs Universitet, CSE

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

## Exercise 1 (complexity)

*a*) The algorithm has $O(n)$ worst-case time complexity. The `for`-loop executes $n$ times and every execution is $O(1)$. The initialisation and return of the method are $O(1)$ as well.

*b*) The algorithm has $O(n^2)$ worst-case time complexity. The out `for`-loop executes $n$ times and the inner `for`-loop executes $n + (n-1) + ... + 2 + 1$ times. The sum is $\frac{n(n+1)}{2}$, which is $O(n^2)$. The initialisation, single step, and return are $O(1)$.

Alternatively, we can say that the both loops are $O(n)$ and use the multiplication rule to conclude that the complexity is $O(n^2)$.

*c*) The algorithm is $O(n^3)$. All three nested loops run from 0 to $n$.

**For a VG only:**

For the `tails` function we can write the following recurrence relation:

$$T(n) = 1 + T(n-1)$$

which has $O(n)$ worst-case time complexity, that is, it is linear in the size of the input list. For the `inits` function we can write the following recurrence relation:

$$T(n) = n + T(n-1)$$

which has $O(n^2)$ worst-case time complexity, that is, it is quadratic in the size of the input list. The nexy part was quite hard and is not considered during grading. So, to get VG for this question you only need to have the previous recurrence relations correct.

The `powerslice` function composes three functions, so the complexity is the addition of each these functions. We calculated the complexity of `tails`, namely $O(n)$. The complexity of `map inits` is $O(n^3)$ since `map` applies a function (in our case `inits`, $O(n^2)$) to every element in the input list (which is of size $O(n)$). The last of the three functions is `concat`, which goes linear through the list appends all elements of the sublists, which contains $O(n^2)$ lists (with roughly $n$ elements). So, `powerslice` has $O(n^2) + O(n^3) + O(n)$ complexity, which we can reduce to $O(n^3)$ using the hierarchy rule.

## Exercise 2 (sorting)

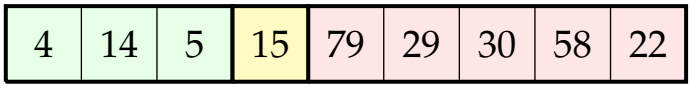
```
bubble :: Ord a => [a] -> [a]
bubble []  = []
bubble [x] = [x]
bubble (x:y:ys) | x < y     = x : bubble (y:ys)
                | otherwise = y : bubble (x:ys)

bubblesort :: Ord a => [a] -> [a]
bubblesort xs = iterate bubble xs !! length xs
```

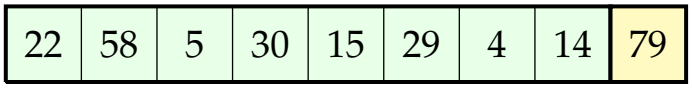
**For a VG only:**

It is important that all elements to the left of the pivot are *smaller* and to the right are *larger*, that is, the pivot should be in the right place. The elements in the to be sorted arrays should be in the correct order, reflecting how the quicksort algorithm works. The subarrays next to the pivot should *not* necessarily be sorted.

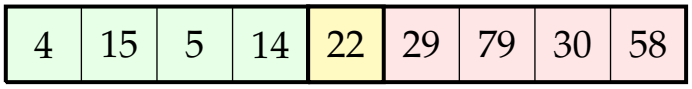
*a*)




*b*)




*c*)




## Exercise 3 (basic data structures)

The instance variable `data` contains the following:

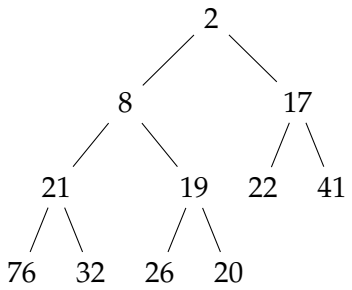

and `front` is 4, `back` is 2, and `size` contains 3. The elements between parentheses don't necessarily have to be in your answer.

```
public E dequeue() {
  if (size > 0) {
    E x = (E) data[front];
    front = (front + 1) % data.length;
    size--;
    return x;
  } else
    throw new RuntimeException("Empty!");
}
```

## Exercise 4 (heaps)

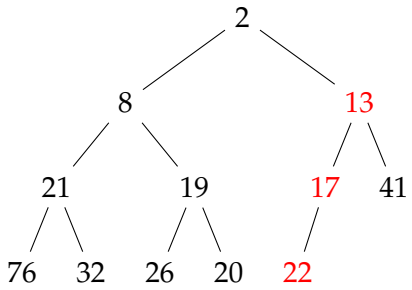
B is the only array that represents a binary heap, which can be drawn as a tree as follows:

### A




### B

The moved bits and new value are drawn in red.



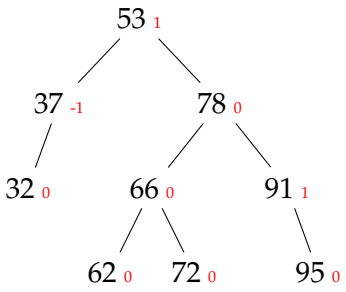

As array: `[2, 8, 13, 21, 19, 17, 41, 76, 32, 26, 20, 22]`

### C

The heap property states that each element must be less than its children. If we have an element that violates the heap property, we can restore it by 'sifting down' the element (also called 'sink'). So, we loop through the array and sift down each element in turn. However, when sifting down the children must already be in heap order. To make sure that the children have the heap property we loop through the array in *reverse* order.

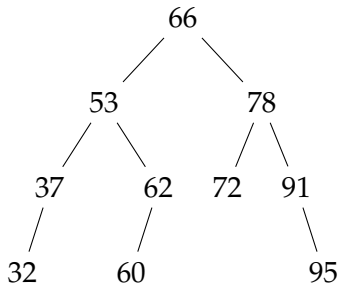
## Exercise 5 (search trees)

### A




### B

60 becomes the left child of 62, and the root node now has a balance of 2. It is a right-left tree, and a pair of rotations fixes the invariant.




## Exercise 6 (graphs)

*a*) $\{AB, BH, HG, GC, CD, DF, FE\}$

*b*) A : 0, B : 2, D : 4, H : 4, C : 5, G : 5, F : 7, E : 8

Depending on the priority queue implementation nodes with equal shortest distances (such as D and H) may be visited in a different order.