Suggested solutions – Exam – Datastrukturer

DIT961, VT-19
Göteborgs Universitet, CSE

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

Exercise 1 (complexity)

a) The complexity is $O(n)$, the statement with $O(1)$ is executed n times.

b) The complexity is $O(n^2 \log n)$, the inner for-loop has complexity $O(n \log n)$, because `count2` is doubled at each iteration, and each iteration takes $O(n)$. The outer for-loop runs n times.

c) The complexity is $O(n^3)$, both loops have $O(n)$ complexity and the body has that complexity as well.

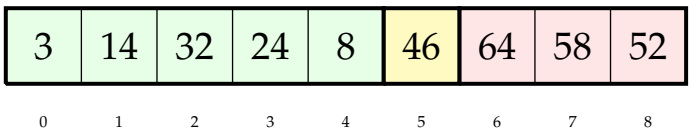
For a VG only:

1. Yes, the most important stack operations, such as pop and push, only manipulate the top of the stack. These are $O(1)$ when implemented with a linked list.
2. No, this depends on the constant factors and the size of the input data. For example, insertion sort is faster than merge sort for small arrays.
3. No, insertion sort is $O(n)$ for (almost) sorted arrays.
4. No, the complexity is only $O(\log n)$ if the binary is balanced.

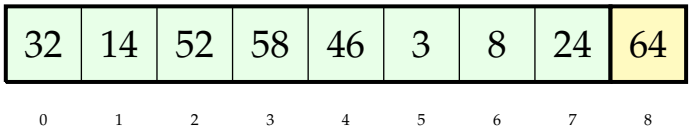
Exercise 2 (sorting)

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.

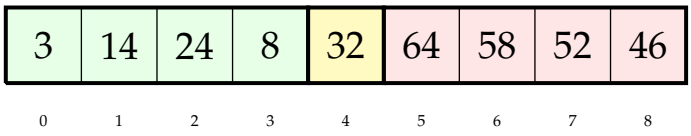
a)



b)



c)



For a VG only:

```
sort :: Ord a => [a] -> [a]
sort []     = []
sort (x:xs) = insert x (sort xs)

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

The `insert` function only uses constant time functions in each call (such as `(:)` and `(<)`) and goes in recursion once (in the worst case). So, the recurrence relation for this function is:

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

where n is the size of the input list. This relation is one of the standard recurrence relations and is $O(n)$. The recurrence relation for `sort` is:

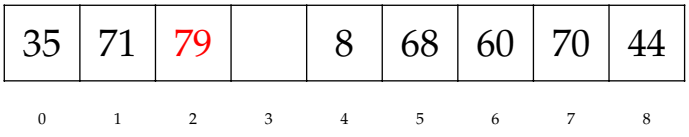
$$T(n) = O(n) + T(n-1)$$

which is $O(n^2)$, because we call `insert` for every element in the list.

Exercise 3 (basic data structures)

a) The was a small mistake in the question: the 8 should have been a 4. So, we skip this question. (The correct answers would have been A and C).

b)



c)

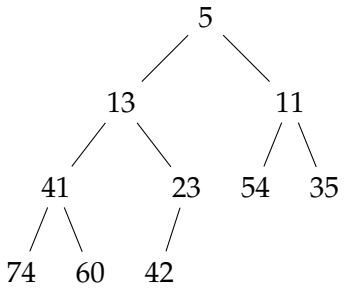
XX	71			8	68	60	70	44
0	1	2	3	4	5	6	7	8

Answer: 35 is replaced by a 'deleted' marker.

Exercise 4 (heaps)

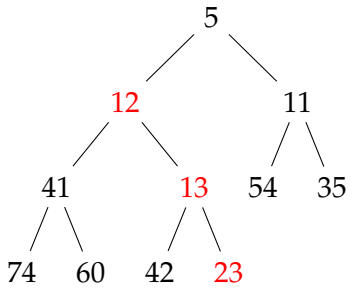
C is the only array that represents a binary heap, which can be drawn as a tree as follows:

A



B

The moved bits and new value are drawn in red.



As array: `[5, 12, 11, 41, 13, 54, 35, 74, 60, 42, 23]`

C

Answer: no. Strings with the same length will have the same hash code. If we insert lots of strings with the same length, then lookup will take $O(n)$ time instead of $O(1)$.

Exercise 5 (search trees)

```haskell
module Tree where

data Tree a = Nil | Node (Tree a) a (Tree a) deriving (Eq, Show)

member :: Ord a => a -> Tree a -> Bool
member x Nil = False
member x (Node l y r)
  | x < y     = member x l
  | x > y     = member x r
  | otherwise = True

insert :: Ord a => a -> Tree a -> Tree a
insert x Nil = Node Nil x Nil
insert x t@(Node l y r)
  | x < y     = Node (insert x l) y r
  | x > y     = Node l y (insert x r)
  | otherwise = t

inorder :: Tree a -> [a]
inorder Nil = []
inorder (Node l x r) = inorder l ++ x : inorder r

bfs :: Tree a -> [a]
bfs t = reverse $ go [t] []
 where
  go []     xs = xs
  go (t:ts) xs = case t of
    Nil        -> go ts xs
    Node l x r -> go (ts ++ [l, r]) (x:xs)
```

Exercise 6 (graphs)

a) $\{HC, CB, BA, AD, DE, DF, FG\}$

b) A: 0, B: 1, D: 2, E: 3, F: 4, C: 4, G: 5, H: 8

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