Data Structures 2018
Exercise 13, solutions (Week 49)

1. About trees in general:

    - A hierarchical structure, with a root value and subtrees of children with a parent node, represented as a set of linked nodes (Tree (data structure) / Wikipedia)
    - Some useful operations: root(), parent(v), children(v), isInternal(v), isExternal(v), isRoot(v), size(), isEmpty()

    Binary tree:

    - Binary tree is a tree where each node has 0 or 2 children: left and right child.
    - This is also referred as a proper binary tree.
    - Useful operations: leftChild(), rightChild()

    Binary search tree:

    - BST is a binary tree where the leaf nodes do not contain any data, they are only placeholders (if you don't want to use placeholders you can say that BST is a *tree* where nodes can have 0, 1 or 2 children)
    - The following property also applies: if node X contains a key-value pair (k,e): the left sub tree of X contains only keys that are smaller than k and the right sub tree contains only keys that are bigger than k.

2. a) Selection sort

- Each round one selects the smallest element that is left in the unsorted sequence and takes it to the end of the sorted sequence.
- The selection of the smallest element is the most time consuming part of the algorithm
- $O(n^2)$
- You can use unsorted priority queue to implement selection sort: take all the elements from the unsorted sequence S to the priority queue P, then use the *removeMinElement*-operation and insert the return value back to the sequence S; repeat this until P in empty.

$$S = (4,1,6,7,2),\ P = ()$$
$$\dots$$
$$S = (),\ P = (4,1,6,7,2)$$
$$S = (1),\ P = (4,6,7,2)$$
$$S = (1,2),\ P = (4,6,7)$$
$$S = (1,2,4),\ P = (6,7)$$
$$S = (1,2,4,6),\ P = (7)$$
$$S = (1,2,4,6,7),\ P = ()$$

Insertion sort

- Each round one removes the first element from the unsorted sequence and adds it to its' right place in the sorted sequence.
- Finding the right place for each element is the most time consuming part of the algorithm.
- O(n^2)
- You can use sorted priority queue to implement insertion sort: remove the first element from the unsorted sequence S and add it to its' right place to the priority queue P; repeat this until S is empty and P has all the elements. Then use the *removeMinElement*-operation and move all the elements back to the sequence S.

$$S = (4, 1, 6, 7, 2),\ P = ()$$
$$S = (1, 6, 7, 2),\ P = (4)$$
$$S = (6, 7, 2),\ P = (1, 4)$$
$$S = (7, 2),\ P = (1, 4, 6)$$
$$S = (2),\ P = (1, 4, 6, 7)$$
$$S = (),\ P = (1, 2, 4, 6, 7)$$
$$...$$
$$S = (1, 2, 4, 6, 7),\ P = ()$$

b) Merge sort
   - Based on divide and conquer -method
   - Commonly implemented as a recursive algorithm
   - If the unsorted sequence S has at least 2 elements, one will divide it to two sequences S_1 and S_2
   - Merge sort S_1 and S_2 recursively
   - Merge the sorted sequences S_1 and S_2 back to sequence S
   - O(n log(n))
   - Example: Exercise 9 assignment 7

   Quick sort
   - Based on divide and conquer -method
   - Commonly implemented as a recursive algorithm
   - If the unsorted sequence S has at least 2 elements, one will select a pivot element x and divide the sequence S based on it to three sequences: L (smaller elements than x), E (equal elements) and G (greater elements).
   - Quick sort L and G recursively
   - Place the sequences L, E and G back to sequence S in the following order: L, E, G
   - Average O(n log(n)) / worst case O(n^2)
   - Example: Exercise 9 assignment 7

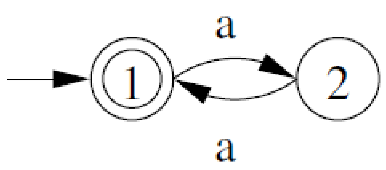
3. a) $(aa)^*$

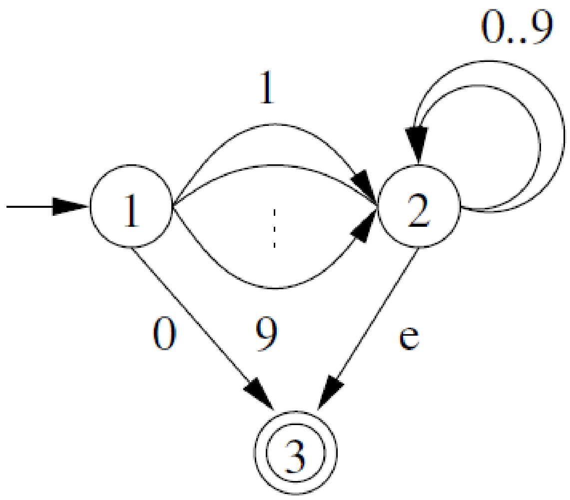
b) $0 + (1 + 2 + \cdots + 9)(0 + 1 + \cdots + 9)^*$

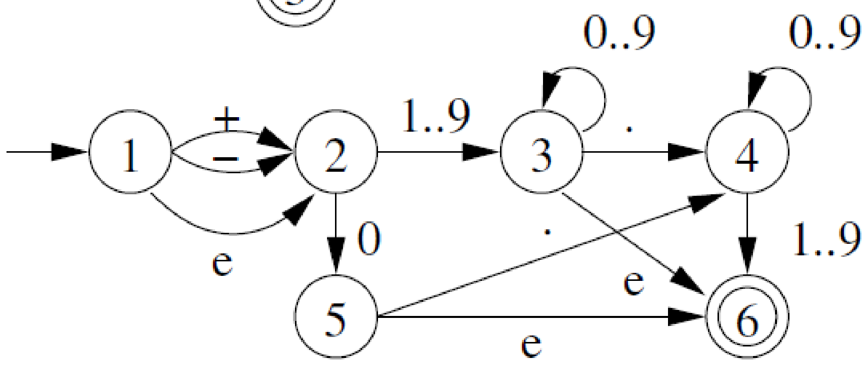
c)

$$
\begin{aligned}
0 + (\text{'+'} + \text{'-'})0 + (\text{'+'} + \text{'-'})(0.(0 + \cdots + 9)^*(1 + \cdots + 9) \\
+ (1 + \cdots + 9)(0 + \cdots + 9)^* \\
+ (1 + \cdots + 9)(0 + \cdots + 9)^* \\
.(0 + \cdots + 9)^*(1 + \cdots + 9)) \\
+(0.(0 + \cdots + 9)^*(1 + \cdots + 9) \\
+ (1 + \cdots + 9)(0 + \cdots + 9)^* \\
+ (1 + \cdots + 9)(0 + \cdots + 9)^* \\
.(0 + \cdots + 9)^*(1 + \cdots + 9))
\end{aligned}
$$

4. See picture below.



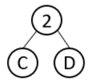




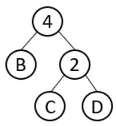
5. DATA STRUCTURES AND ALGORITHMS IN JAVA
6. Answers may vary due to selections made with elements having same key values in the priority queue. Frequency table:

A	B	C	D	R
5	2	1	1	2

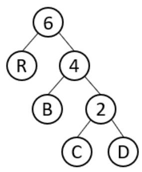
Step 1. Priority queue $Q = ((A, 5), (B, 2), (C, 1), (D, 1), (R, 2))$. Remove the two elements having smallest keys in the queue: they will form a tree that will be added back to the priority queue. The characters C and D will form the following tree:



Step 2. Priority queue $Q = ((A, 5), (B, 2), (\text{"C, D"}, 2), (R, 2))$. Now the tree formed in the previous step and the character B will form a tree: the character B is the left child and the tree is the right child.



Step 3. Priority queue $Q = ((A, 5), (\text{"B, C, D"}, 4), (R, 2))$. Now the tree formed in the previous step and the character R will form a tree: the character R is the left child and the tree is the right child.



Step 4. Priority queue $Q = ((A, 5), (\text{"R, B, C, D"}, 6))$. Now the tree formed in the previous step and the character A will form a tree: the character A is the left child and the tree is the right child.