INF102 18H
Algoritmar, datastrukturar og programmering
Mandatory assignment 1

Student – Andrey Belinskiy (zur008)

Quicksort

a) Show a trace of how the partition function partitions the array `[11, 12, 4, 13, 2, 5, 11, 5, 16, 14]` **(lb=0, ub=10)**

	i	j	v	a[] 0	1	2	3	4	5	6	7	8	9
Initial values	0	9		11	12	4	13	2	5	11	5	16	14
Scan left, scan right	1	7		11	12	4	13	2	5	11	5	16	14
Exchange	1	7		11	5	4	13	2	5	11	12	16	14
Scan left, scan right	3	6		11	5	4	13	2	5	11	12	16	14
Exchange	3	6		11	5	4	11	2	5	13	12	16	14
Scan left, scan right	6	5		11	5	4	11	2	5	13	12	16	14
Final exchange	6	5		5	5	4	11	2	11	13	12	16	14
Result		5		5	5	4	11	2	11	13	12	16	14

b) Show a trace of how quicksort sorts the array `[11, 12, 4, 13, 2, 5, 11, 5, 16, 14]` **, assuming that the initial shuffle is omitted**

	lo	j	hi	0	1	2	3	4	5	6	7	8	9
Initial values				11	12	4	13	2	5	11	5	16	14
	0	5	9	5	5	4	11	2	11	13	12	16	14
	0	2	4	4	2	5	11	5	11	13	12	16	14
	0	0	2	2	4	5	11	5	11	13	12	16	14
	1		1	2	4	5	11	5	11	13	12	16	14
	3	3	4	2	4	5	5	11	11	13	12	16	14
	4		4	2	4	5	5	11	11	13	12	16	14
	6	6	9	2	4	5	5	11	11	12	13	16	14
	7	7	9	2	4	5	5	11	11	12	13	16	14
	8	8	9	2	4	5	5	11	11	12	13	14	16
	9		9	2	4	5	5	11	11	12	13	14	16
Result				2	4	5	5	11	11	12	13	14	16

c) What happens if you require `i` to point at an element which is *strictly greater* than the pivot, and `j` to point at an element which is *strictly less* than the pivot?

If we skip the elements that are equal to the pivot when comparing them, then we risk reaching the quadruple runtime for quicksort in some situations.

For example, if we need to sort an array that contains many duplicate items, then the partitioning process will put elements equal to the pivot on one side, which leads to increased runtime for the following sorting process (to the point of $O(n^2)$).

Priority Queues

a) The array `[null, T, P, R, N, H, O, A, E, I, G]` represents a 1-indexed max heap in a priority queue of type `Character`. What does the array look like after `S` is added to the priority queue?

First, we add the new key to the end of the tree. Then, swim it up until we reach a node with larger key or the root.

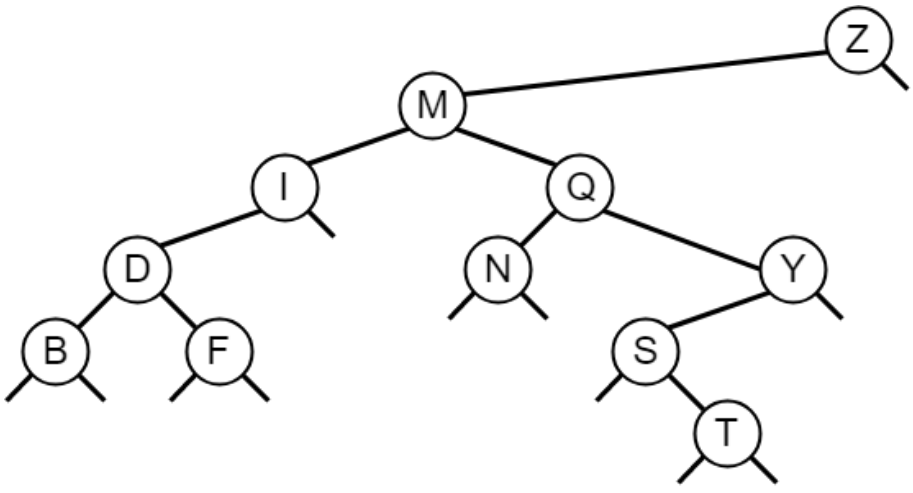
Result - [null, T, S, R, N, P, O, A, E, I, G, H].

b) To implement `peek()` (find the maximum/minimum) in constant time, why not use a stack or a queue, but keep track of the maximum/minimum value inserted so far, then return that value for `peek()`?

Suppose we're using a stack or a queue to keep the values. Then, when we perform the remove max/min value, we'll have to find the new max/min value in the stack/queue (which will take linear time). By keeping track of the max/min value we can just replace it in constant time with some new value.

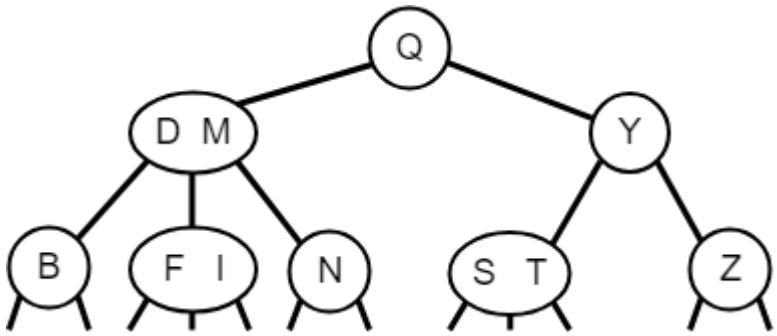
Binary Search Tree

a) Draw the BST that results after you insert the keys `Z M Q N Y I D S B F T` in that order into an initially empty tree



Balanced Binary Search Trees

a) Draw the 2-3 tree that results when you insert the keys `Z M Q N Y I D S B F T` in that order into an initially empty tree

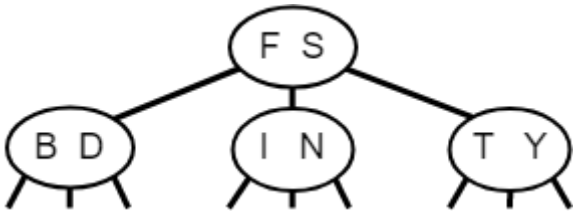


b) Find an insertion order for the keys `T F B S D I Y N` that leads to a 2-3 tree of height 1

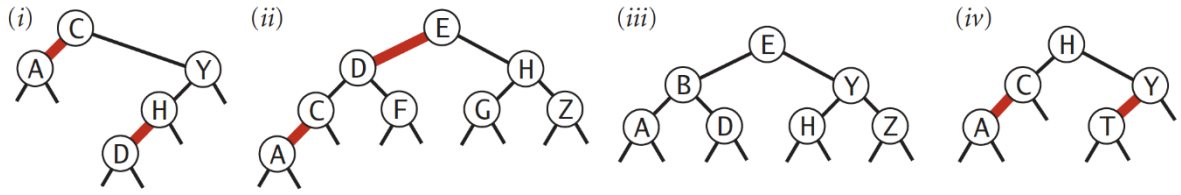
To get a 2-3 tree of h=1 using given keys we need to insert them in the following order:

B I F D S T N Y

The resulting 2-3 tree of h=1:



c) Which of the following are red-black BSTs?



To understand if presented BSTs are red-black BSTs, they need to satisfy the following restrictions:

1) Red links should always lean left. All BSTs satisfy this condition;
2) No node should have two red links connected to it. Again, this condition is satisfied by all BSTs;
3) BST should have a perfect black balance, i.e. all paths to null links should have the same amount of black links in them.

Let's check the BSTs one by one to find out if they satisfy third restriction *(for convenience purposes, let's count the null link as a black link)*.

The first tree doesn't satisfy this condition, because the path from node C to a null link at node A contains one black link, but the path from node C to a null link at node Y contains two black links.

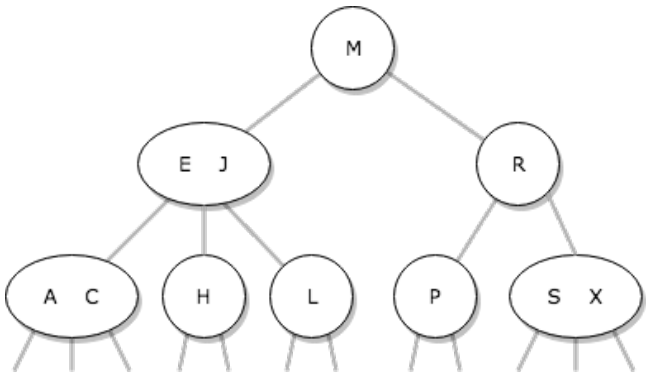
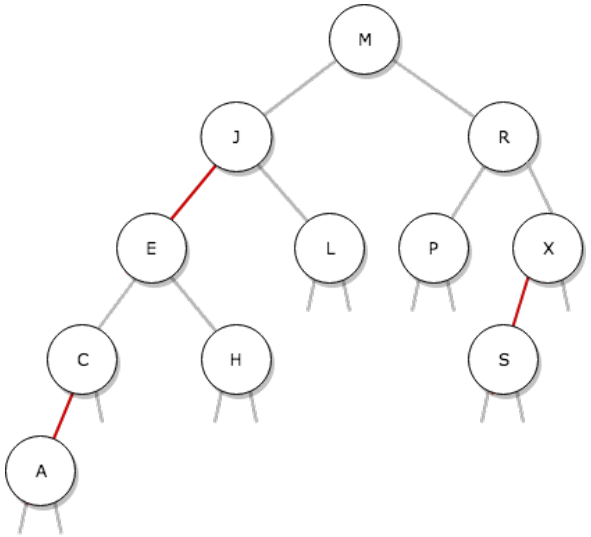
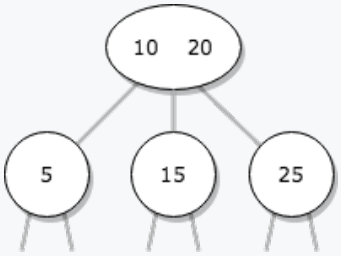
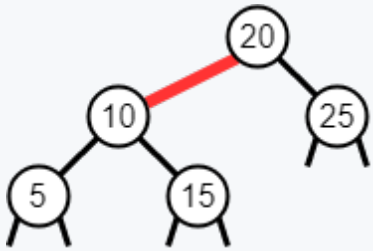
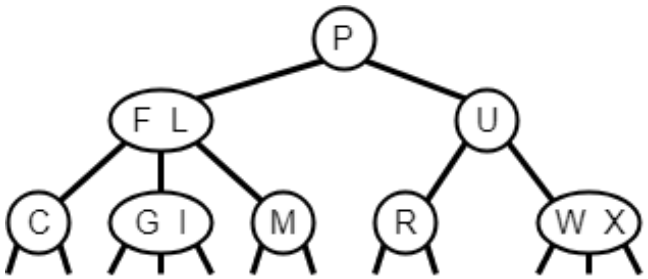
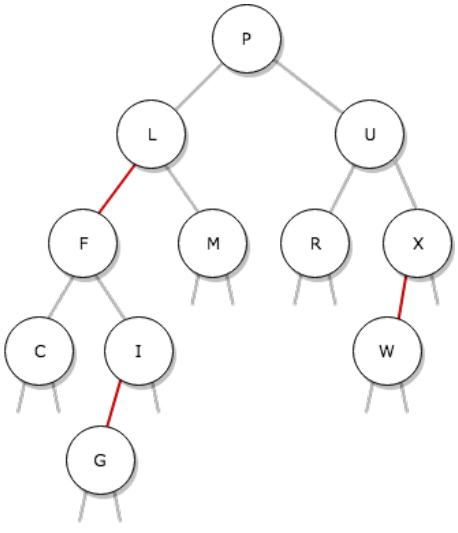
The second tree doesn't qualify either. The path from node E to a null link at node Z contains three black links, whereas the path from node E to a null link at node A has only two black links.

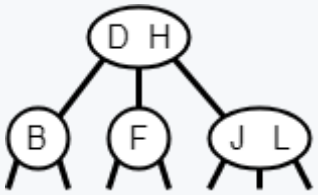
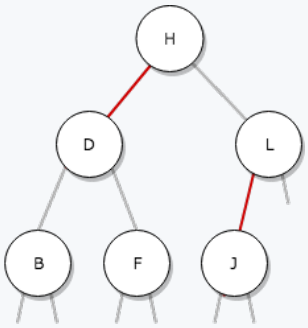
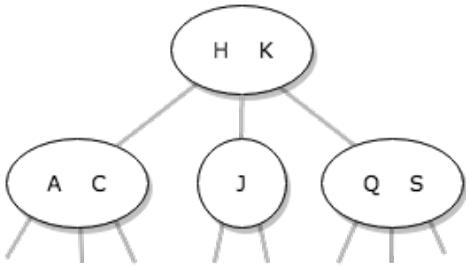
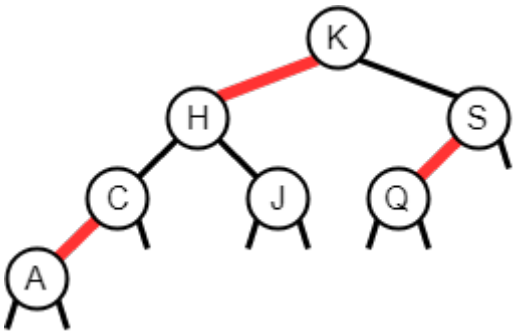
While the third tree doesn't have any red links, it still satisfies the condition of having the same amount of black links in the path leading to a null link (3 black links).

The fourth tree also satisfies this condition. All paths to null links have the same amount of black links.

Thus, third and fourth trees are red-black BSTs.

d) Fill in the table below with the missing pictures

№	2-3 BST	Red-black BST
i		
ii		
iii		

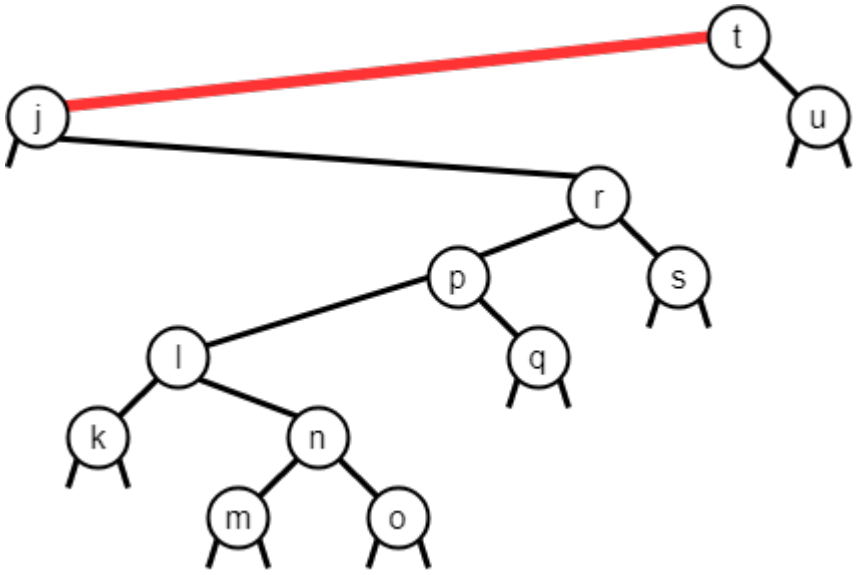
iv		
v		

e) Give the order that operations are applied when the character `n` is inserted. Give your trace as a single string with the characters L, R and F

The following operations will be performed when the character *n* is inserted:

LRFFRFRFRFR

Resulting tree (subtrees are omitted, node *m* has null children, node *o*'s left child is null):



f) What is the maximum depth of a left-leaning red-black tree with `n` nodes?

The average depth of the left-leaning red-black tree with n nodes is $log_2(n)$. The maximum depth, however, is $2log_2(n)$.

Let's consider the rules that define a valid left-leaning RBT. According to the second rule, the node can't have two red links connected to it, so red and black links in the worst case should alternate. That means that in the worst case the path to a node should have at least the same amount of black links as there are red links. Furthermore, the third rule implies that BST should have a perfect black balance, so the amount of black links is the same in any chosen path. So, assuming the path which consists of only black links is of length $log_2(n)$, then in the worst case this path will have the same amount of red links, thus it will be twice as long.

Therefore, in the worst case, the depth of the left-leaning RBT is $2log_2(n)$.