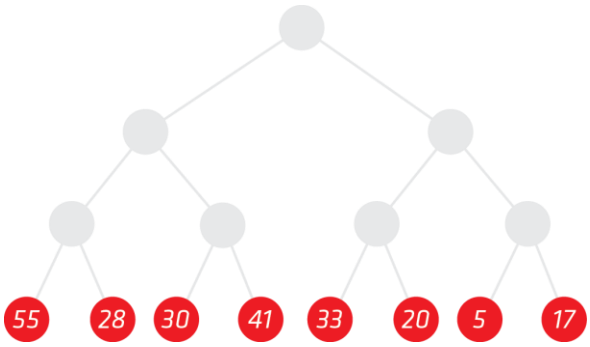
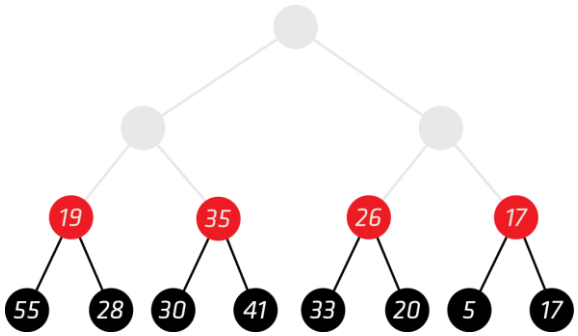
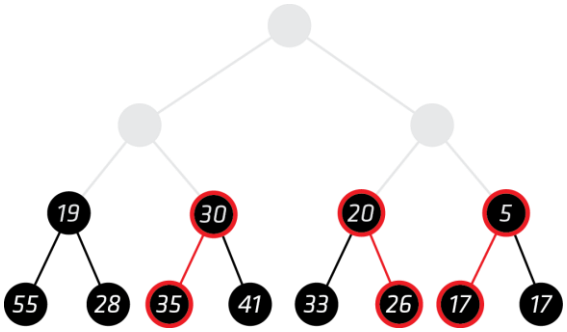
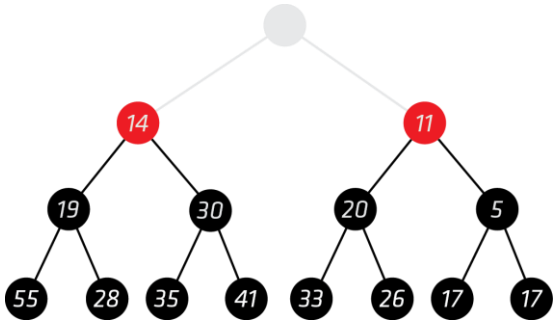
Q.1

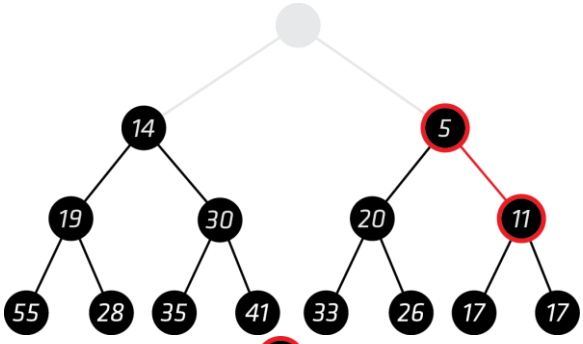
a

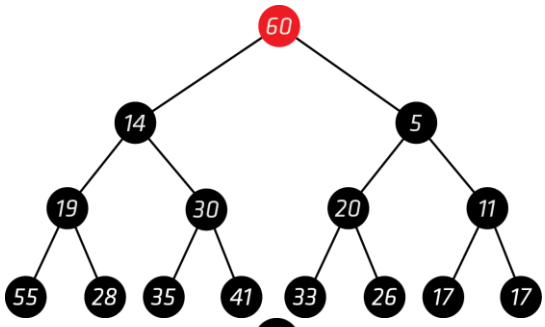

55 28 30 41 33 20 5 17

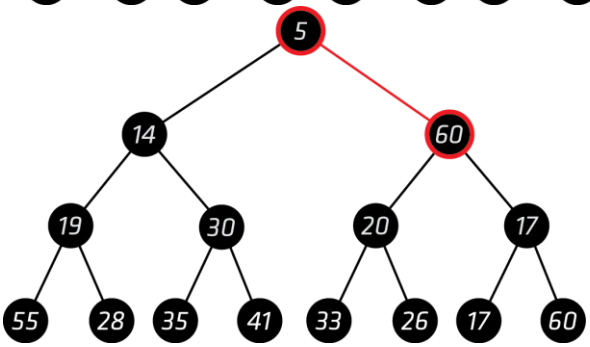

19 35 26 17
55 28 30 41 33 20 5 17

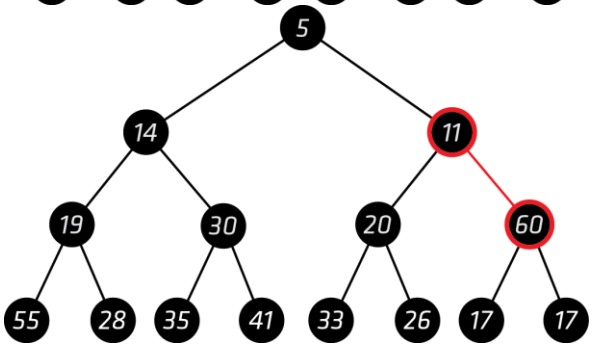

19 30 20 5
55 28 35 41 33 26 17 17

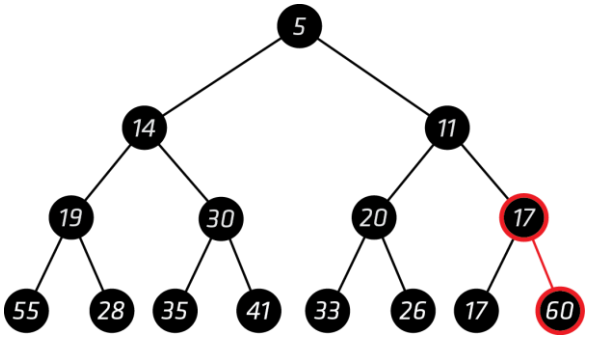

14 11
19 30 20 5
55 28 35 41 33 26 17 17

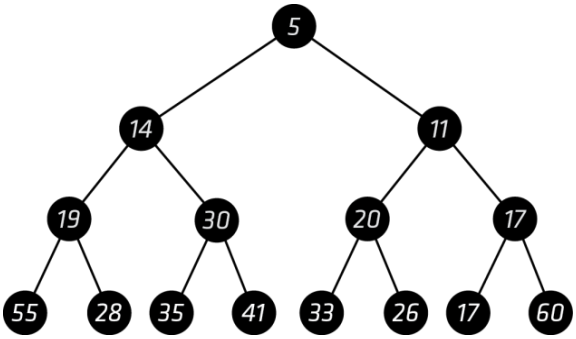

14 5
19 30 20 11
55 28 35 41 33 26 17 17


60
14 5
19 30 20 11
55 28 35 41 33 26 17 17

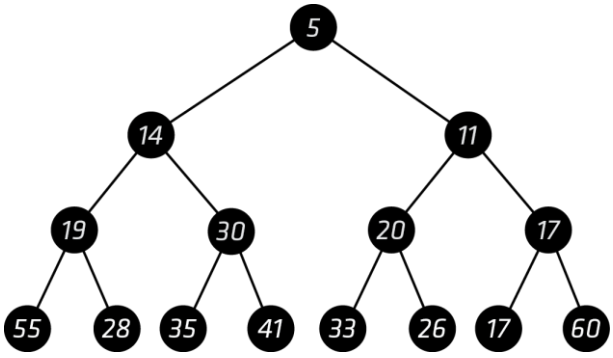

5
14 60
19 30 20 17
55 28 35 41 33 26 17 60

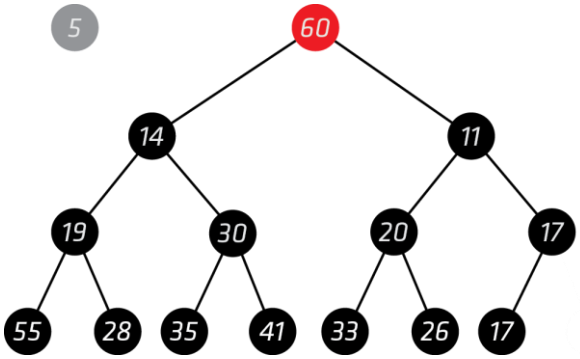

5
14 11
19 30 20 60
55 28 35 41 33 26 17 17

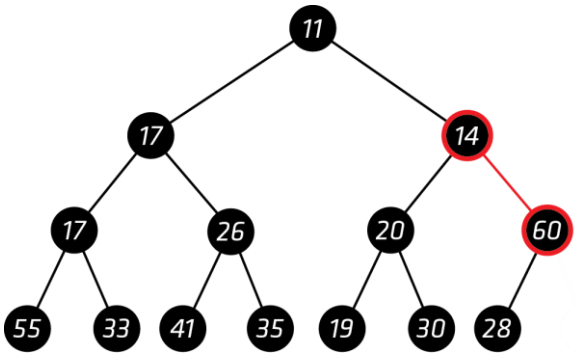

5
14
11
19
30
20
17
55
28
35
41
33
26
17
60

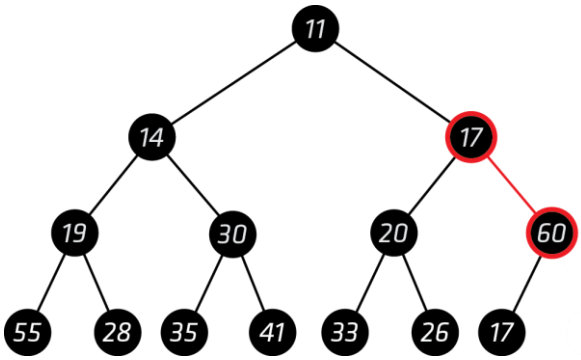

5
14
11
19
30
20
17
55
28
35
41
33
26
17
60

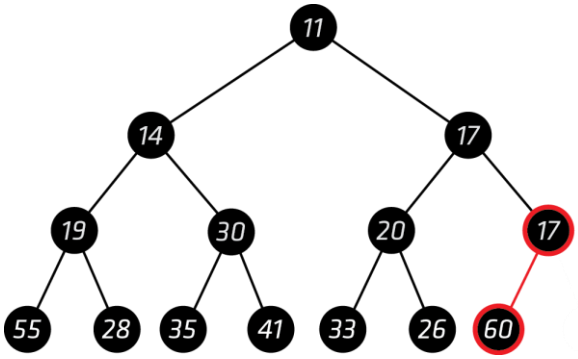
b

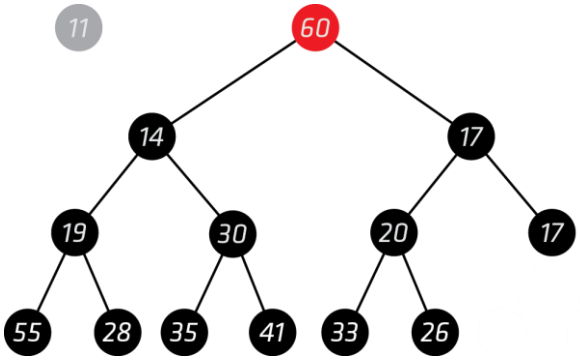

5
14
11
19
30
20
17
55
28
35
41
33
26
17
60

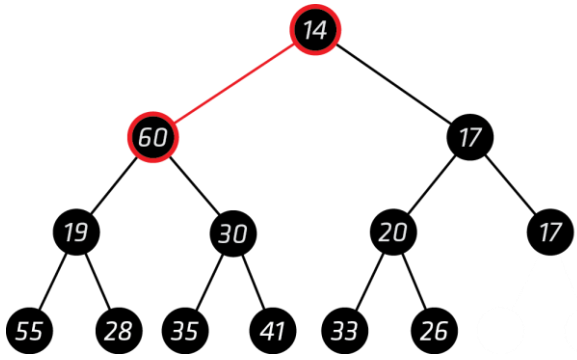

5
60
14
11
19
30
20
17
55
28
35
41
33
26
17

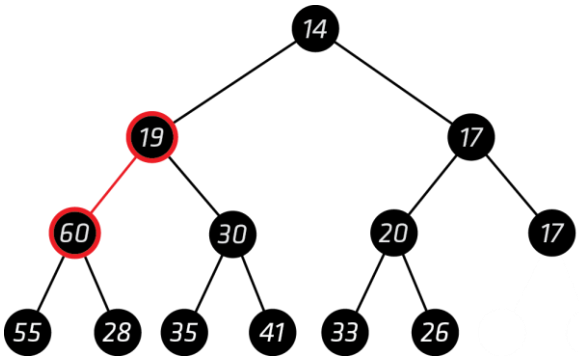

11
17
14
17
26
20
60
55
33
41
35
19
30
28

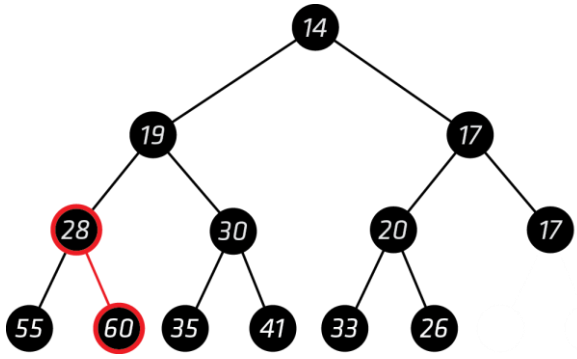

11
14
17
19
30
20
60
55
28
35
41
33
26
17

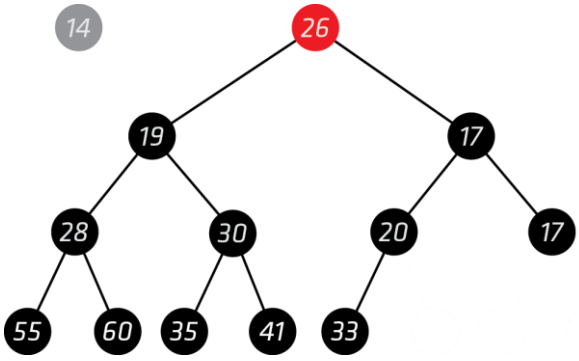

11
14
17
19
30
20
17
55
28
35
41
33
26
60

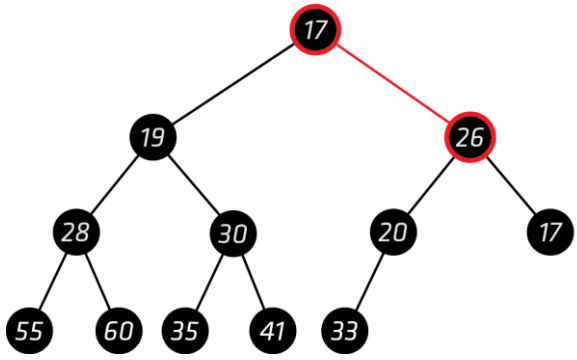

11
60
14
17
19
30
20
17
55
28
35
41
33
26

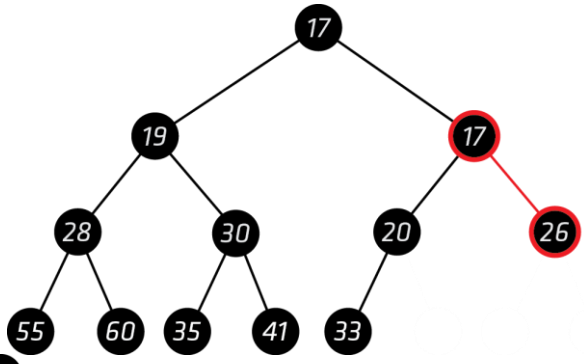

14
60
17
19
30
20
17
55
28
35
41
33
26

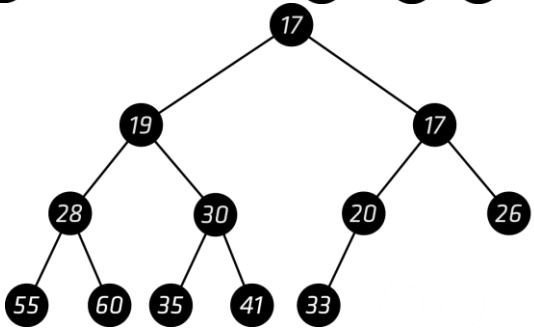

14
19
17
60
30
20
17
55
28
35
41
33
26


14
19
17
28
30
20
17
55
60
35
41
33
26

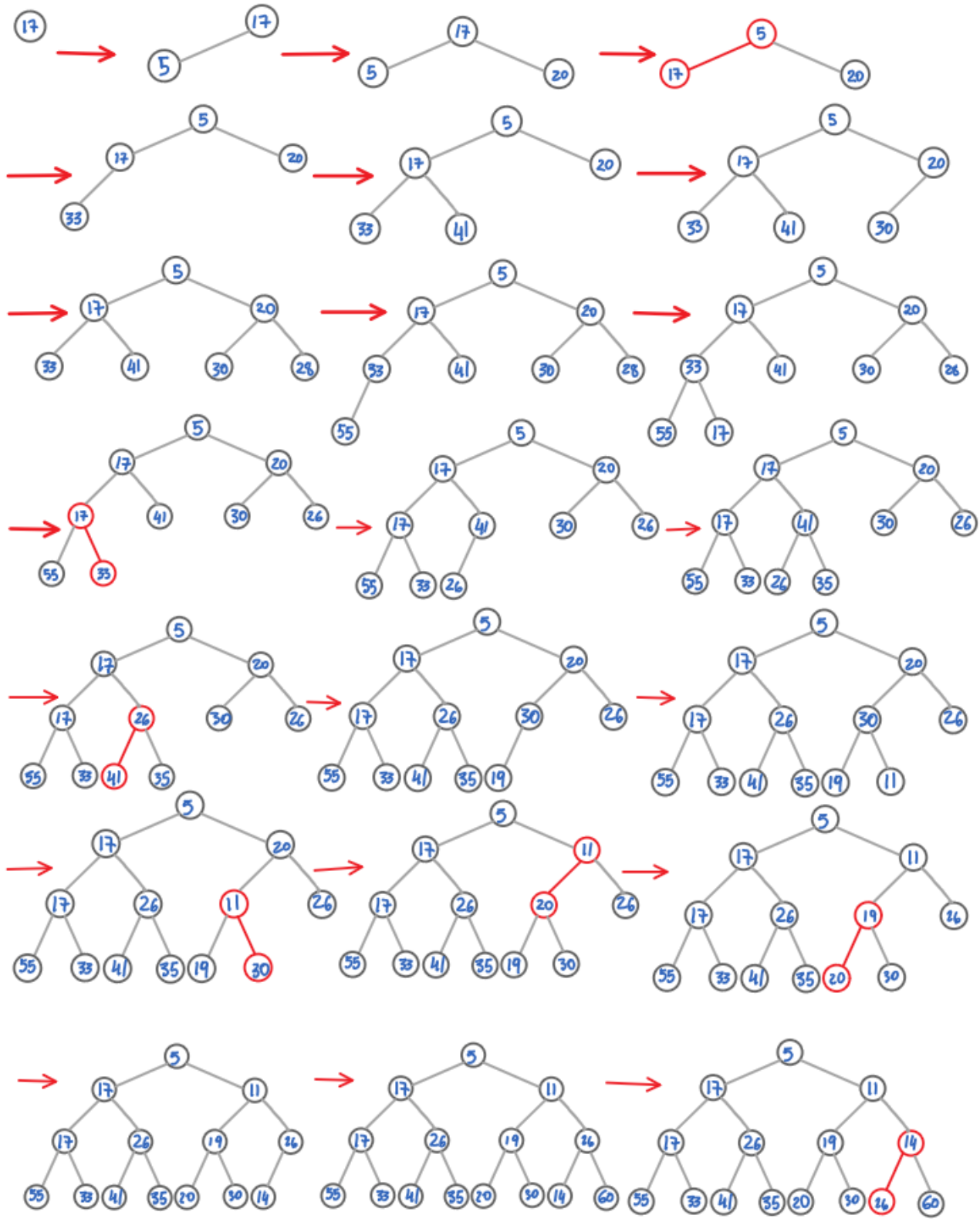

14
26
19
17
28
30
20
17
55
60
35
41
33


17
19
26
28
30
20
17
55
60
35
41
33

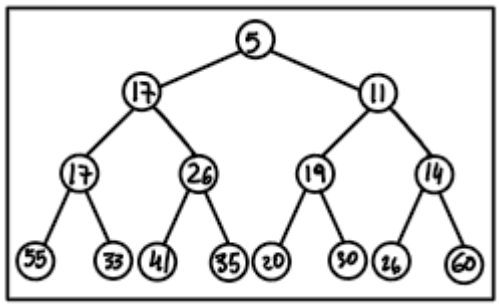

17
19
17
28
30
20
26
55
60
35
41
33


17
19
17
28
30
20
26
55
60
35
41
33

C

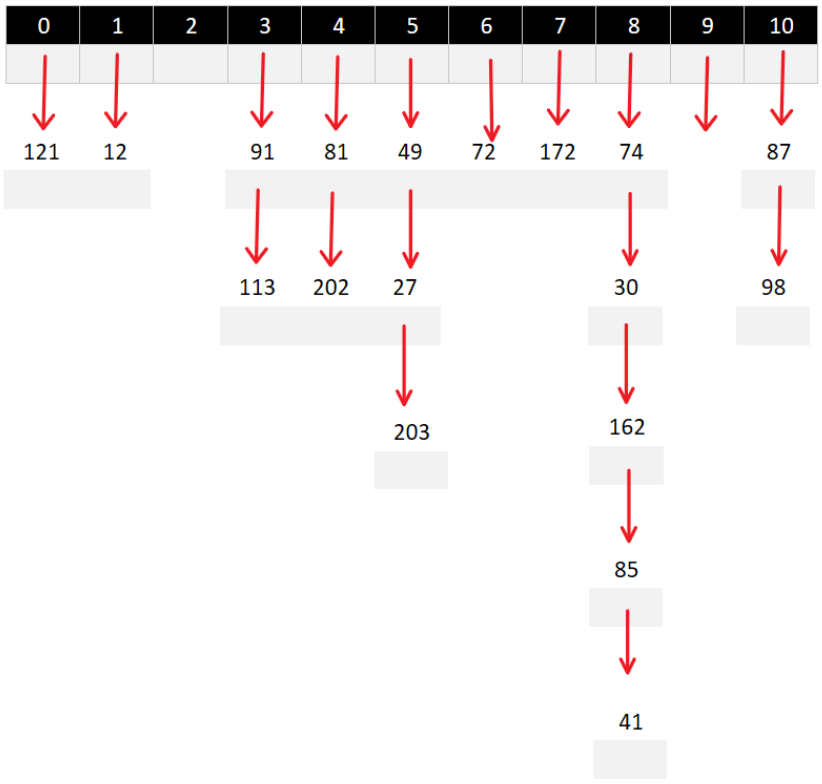


Final tree:



Q.2

a

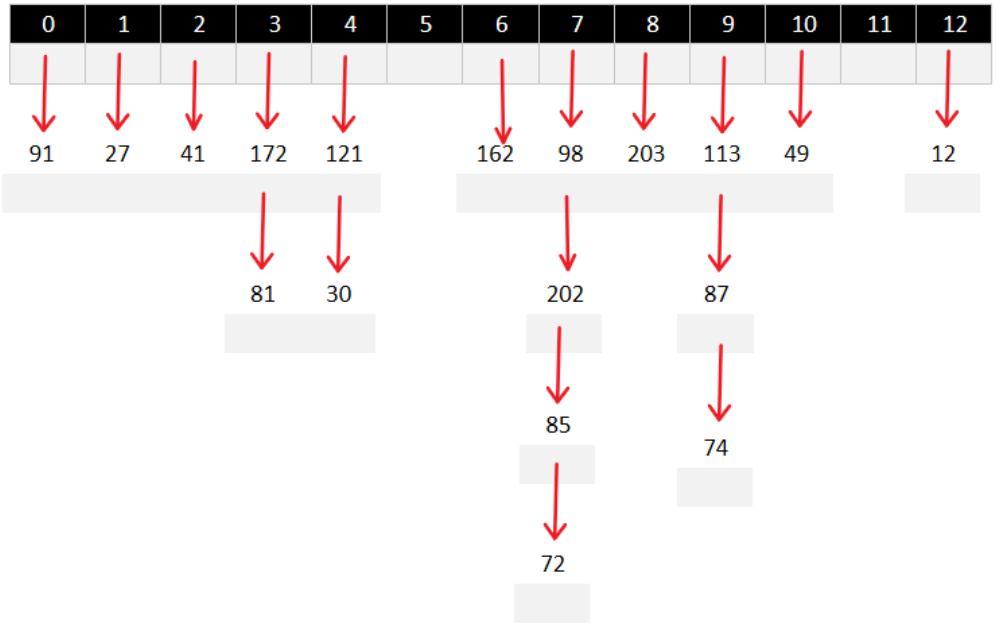


b

the maximum number of collision is 10.

Q3

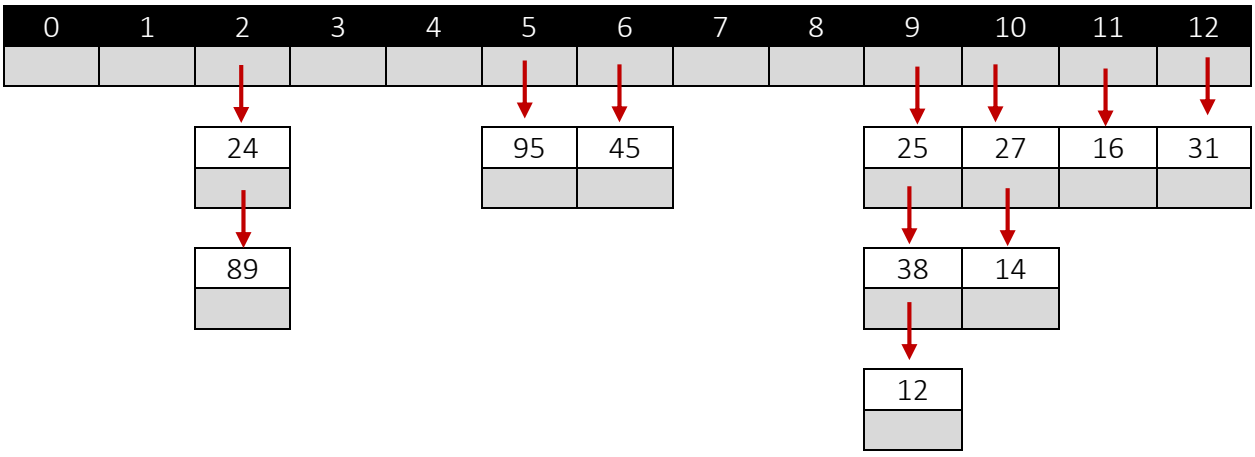
The proposed solution with a larger array of 13 yields the following table:



This method reduces the load factor in this particular example with the given keys of the list since the maximum number of collision this time is 7 compared to 10 which was obtained previously with the array of 11 elements. However, it (the method) could potentially fail to reduce the number of collision if we would be given a different set of keys and therefore making this proposal senseless.

Q.4

a



b

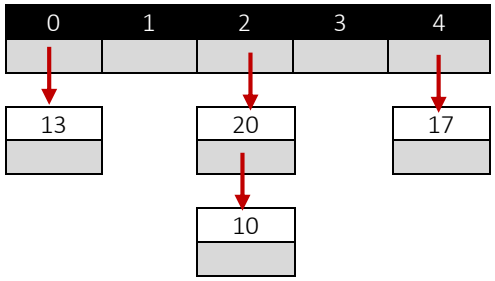
0	27
1	16
2	89
3	24
4	25
5	95
6	45
7	
8	
9	12
10	14
11	38
12	31

c

0			
1	27		
2	89		
3	25		
4	24	38	COLLISION
5	95		
6	45		
7			
8			
9	12		
10	14		
11	16		
12	31		

Q5

a



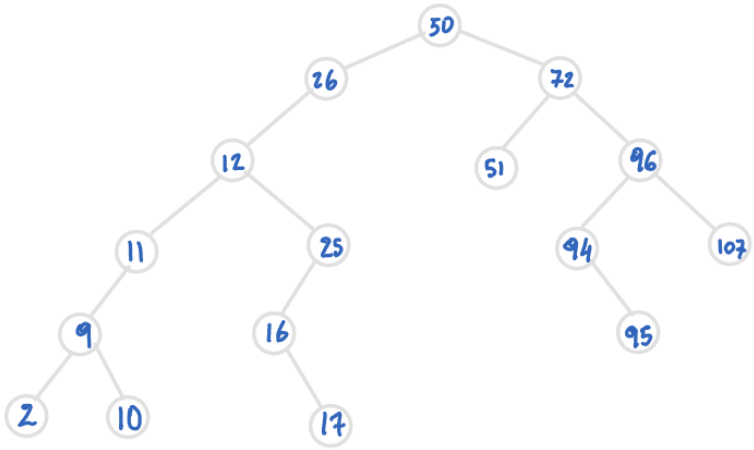
b

0	13
1	
2	10
3	20
4	17

c

0	13
1	20
2	10
3	
4	17

Q6

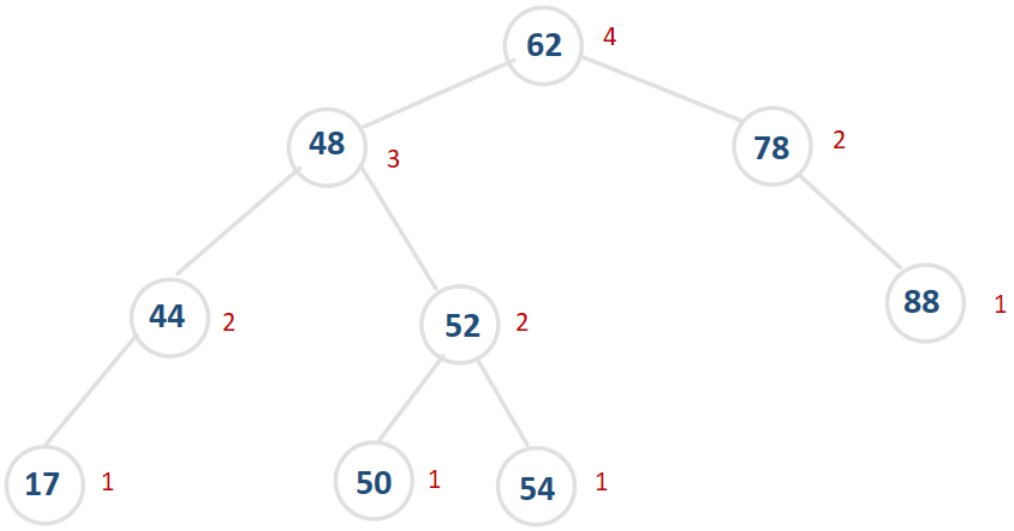


Q7

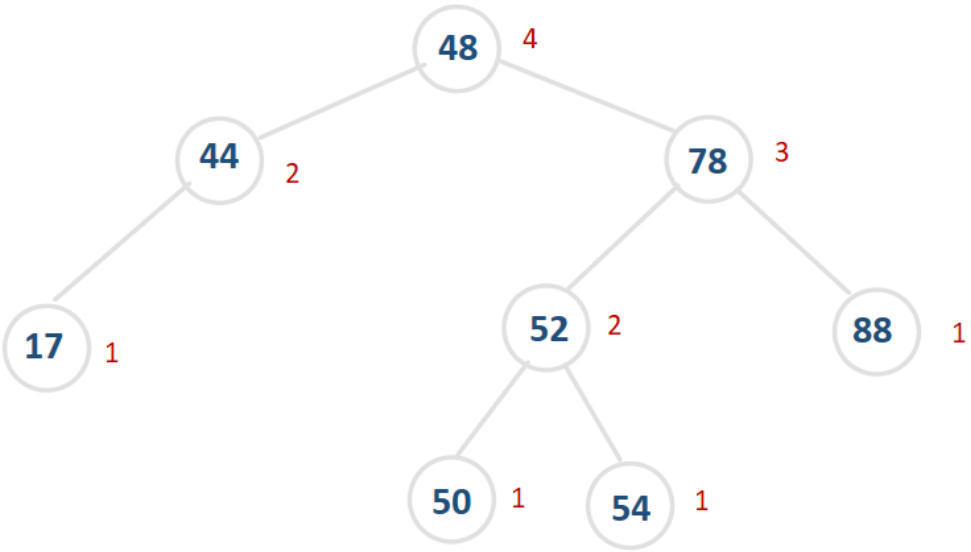
1) **Sorted linked list:** When searching for an element, in the worst case where the element is not found the time efficiency for searching is **O(n)** since the search was done on the size n of that list.
2) **Array based linked list:** The same operation will also result in **O(n)** and this also happens in the worst case where the element is not found
3) **Binary Search Tree:** As the name suggests a binary tree is used for searching because it is more
4) efficient then above abstract data structures, its efficiency is **O(log n)**.

Q9

a


62
4
48
3
78
2
44
2
52
2
88
1
17
1
50
1
54
1

b


48
4
44
2
78
3
17
1
52
2
88
1
50
1
54
1

Q9

Algorithm findHeight(h)
Input: h is the current height of the AVL tree, it is intialy equal to zero and only used reclusively.
Output: height of the AVL tree.

```
        IF(T.parent = null)
                return 0;
        END IF
        ELSE IF (T.balance < 1 and T.balance > -1)
                return max(findHeight(T.left), findHeight(T.right)) + 1

        // alternatively we can replace the return statement with the code bellow

                left <-- findHeight(T.left)
                right <-- findRight(T.right)
        END ELSE IF
END
```

Q10

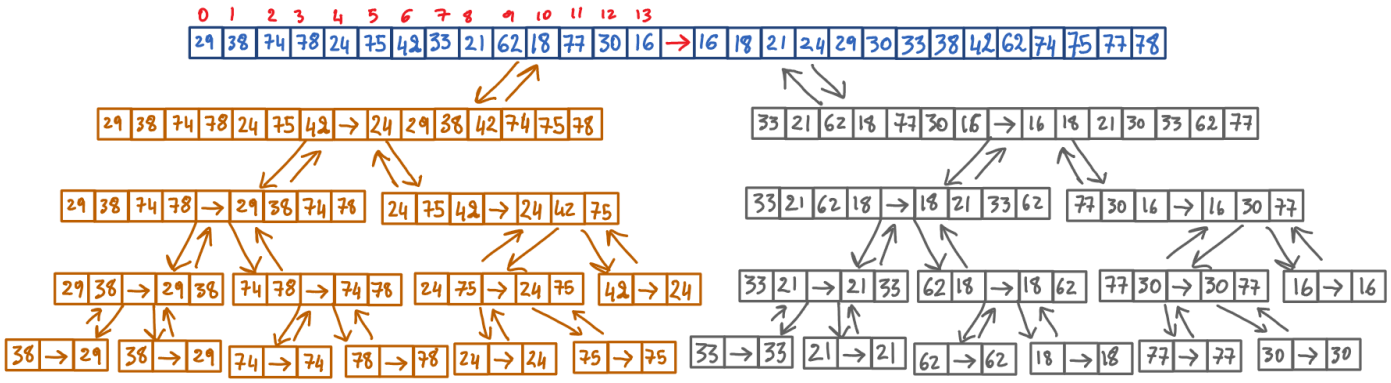
count = 0
Algorithm equalElements(S[], i)
Input: S[] is an ordered array or sequence of n element, i is the current index of an array.
Output: returns true if there are two equal elements in the sequence or false otherwise.

```
        IF (count = 0 AND S.length > 2 )
           count ++
           i = 1        { since this is the first time the method runs we want i to be after index 0}
        END IF
        IF(S.length =< i)
           return false
        ELSE IF (S.length = 2 OR (i+2) = (S.length -1))
           IF(S[S.length -2] != S[S.length -1])
                return false
           ELSE
                return true
           END IF
        ELSE
           IF(S[i-1] = S[i] or S[i+1] = S[i])
                return true
           ELSE
                equalElements(S, i+2)
           END IF
        END IF
END
```

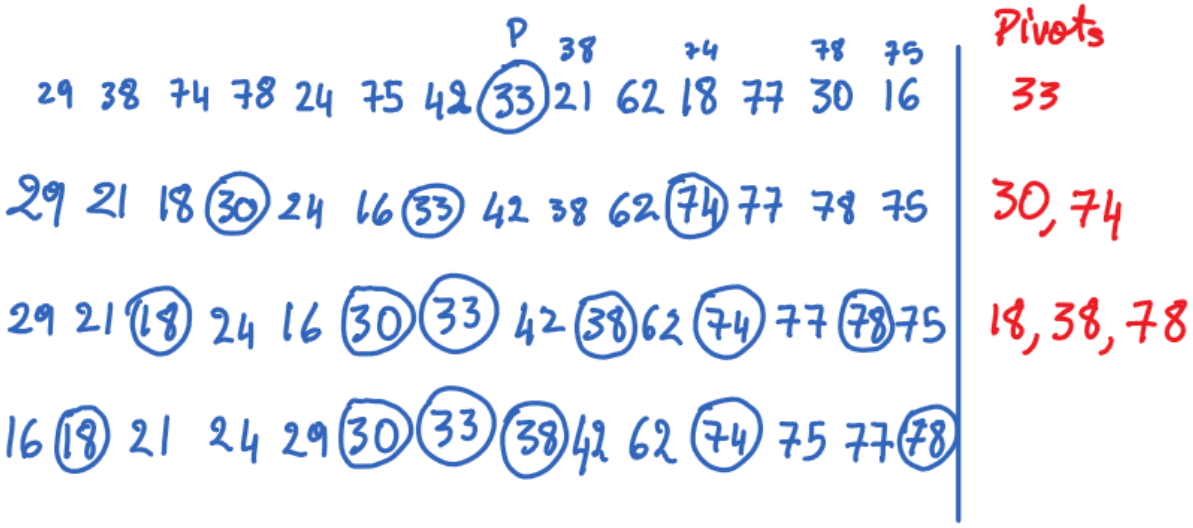
The running time of this algorithm is O(n).

Q11

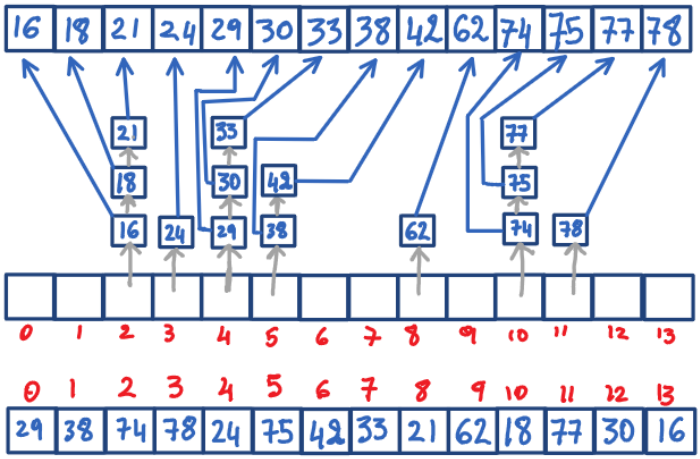
a



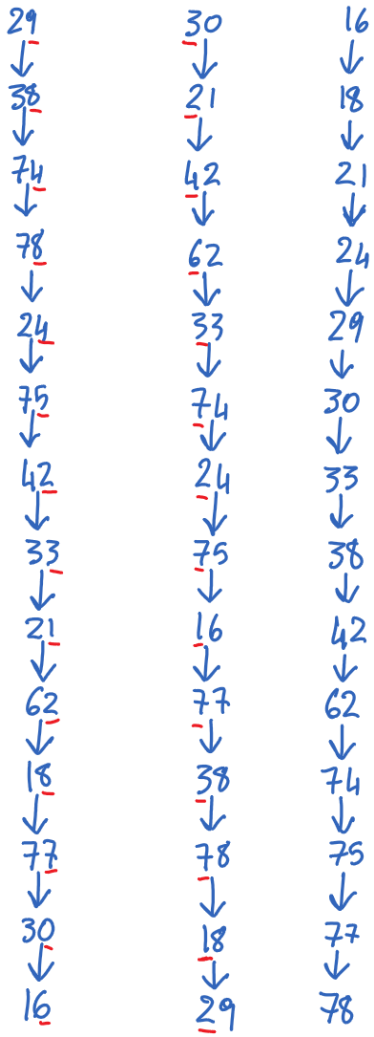
b


P
38
74
78
75
Pivots
29 38 74 78 24 75 42 33 21 62 18 77 30 16
33
29 21 18 30 24 16 33 42 38 62 74 77 78 75
30, 74
29 21 18 24 16 30 33 42 38 62 74 77 78 75
18, 38, 78
16 18 21 24 29 30 33 38 42 62 74 75 77 78

c


16 18 21 24 29 30 33 38 42 62 74 75 77 78
0 1 2 3 4 5 6 7 8 9 10 11 12 13
0 1 2 3 4 5 6 7 8 9 10 11 12 13
29 38 74 78 24 75 42 33 21 62 18 77 30 16

d


29 38 74 78 24 75 42 33 21 62 18 77 30 16
30 21 42 62 33 74 24 75 16 77 38 78 18 29
16 18 21 24 29 30 33 38 42 62 74 75 77 78