4/1)

i) (ii)

4,5

1,2

3

3,4,5

1

2

(iii)

5

1,2,3

4

4/3)Let us consider the worst case of AVL-Tree with height 5. If that can be colored as a red-black tree then every AVL-Tree can be colored as a red-black tree.

Let us consider the following AVL tree:-

17

32

10

35

27

15

5

38

33

30

25

12

7

3

41

37

31

2

28

26

24

4

23

21

Lets us try to colorize the above AVL tree according to Red-Black Tree format.

17

32

10

5

35

27

15

38

30

25

33

12

7

3

3

41

37

31

24

28

26

4

2

23

21

The above tree satisfies all the Red-Black Tree properties. Hence we can say that an Avl tree can be colored as a red-black tree.

No, all red-black trees do not satisfy the AVL Tree property and the following case will illustrate it clearly.

50

76

25

35

15

10

4/4)

#lO is the lower limit in the internal query, uP is the upper limit

(a) def find(root,lO,uP,list):

if root!=None:

if root.data<=uP and root.data>=lO:

li.append(root.data)

find(root.left,lO,uP,list)

find(root.right,lO,uP,list)

elif root.data<=uP:

find(root.right,lO,uP,list)

elif root.data>=lO:

find(root.left,lO,uP,list)

(b),(c),(d) I think RB-Trees, splay trees and AVL Trees all are modified versions of BST. All trees follow the basic condition and they are also the binary trees. So the algorithm for the internal query will remain same as it is for Binary Search Trees but the time complexity may differ which don’t have to consider in this part.

(e)

def find(node,lO,uP,list):

for i in range(0,no\_of\_keys(node)):

if node.key[i]>lO:

find(node.child[i],lO,uP,list)

if node.key[i]>=lO and node.key[i]<=uP:

li.append(node.key[i])

i=no\_of\_key(node)

if node[i].key<uP:

find(node.child[i+1],lO,uP,list)

(f)

(g)Upper bound for each of the case will be same and that will be n because in the worst case the user will give lower limit the value of lowest value in tree and upper limit the value of maximum value in the tree.