3/1) (a)

34

34

23

34

27

33

50

23

45

34

28

27

33

15

7

23

20

(c) pseudo code of the function atRank(nod,i) is as follows:

if x!=None:

y=x

while y!=None:

y=y.next

if x.rank<i:

while x.rank<i and x.next!=None:

x=x.next

if x.rank==i:

print x.da

elif x.up!=None:

atRank(x.up,i)

elif x.rank==i:

print x.da

else:

while x.rank>i and x.pre!=None:

x=x.pre

if x.rank==None:

print x.da

elif x.up!=None:

atRank(x.up,i)

3/2)

Binary Tree of height 2:

10

4

17

16

5

1

21

Binary Search Tree of height 3:

16

21

17

5

1

4

10

Binary Search Tree of height 4:

21

5

1

4

10

16

17

Binary Search Tree of height 5:

21

5

1

4

10

16

17

Binary Search Tree of height 6:

21

1

4

5

10

16

17

3/3) c.) 925,202,911,240,912,245,363

e.) 935,278,347,621,299,392,358,363

3/4) Let us suppose that there are total of n number of entries and left and right pointers of every leaf and also the parent of root will point to the sentinel whose id is set to 0. Initially id’s of every node except sentinel are set to 1.

X=T.root

While n>0

if x.id==1 and x.left.id==1

X=x.left

Elsif x.id==1 and x.left.id==0

x.id=0

print x.value

n=n-1

if x.right.id==1

x=x.right

else

x=x.parent

elsif x.id==0

x=x.parent

3/5)In **worst case** the tree will be skewed one and hence the worst case running time will be O(n2)

In **best case ,**the tree will be full binary tree and hence running time will be Ω(nlgn).

3/6).

a)Yes, it an AVL tree.

b)

insert(1)

8

12

2

4

16

10

4

1

Insert(15)

16

14

15

10

12

4

1

2

8

8

Delete(4)

16

14

15

10

12

1

2

Delete(8)

16

14

15

4

10

1

2

12

17

3/7)

32

10

35

27

15

5

38

33

30

25

12

7

4

41

37

31

2

28

26

24

3

23

21

Let us apply delete operation on 15.

**Applying right rotation**

17

32

10

35

27

12

5

38

33

30

25

15

7

4

31

41

37

28

26

24

3

2

23

21

**Delete(15)**

17

32

10

38

33

24

30

25

3

2

7

4

35

27

12

5

41

37

31

28

26

23

21

**Applying left rotation on 17 and 32**

32

35

17

38

33

27

10

41

37

30

25

12

5

31

28

26

24

7

4

23

21

3

2

**Applying right rotation on 5 and 10**

32

27

5

41

37

38

33

35

17

30

25

10

4

31

28

26

24

12

7

3

2

23

21

**Applying left rotation on 17 and 27**

32

35

27

38

30

17

33

41

37

25

31

28

5

26

24

10

4

23

21

12

7

3

2

**Applying right rotation on 27 and 32**

27

5

32

17

35

30

25

38

31

28

33

26

24

10

4

41

37

23

21

12

7

3

2

3/8)My algorithm has upper and lower bounds both 2n.

3/9) Let us consider an example to prove that left and right rotations are inverse of each other.

α

¥

β

x

y

Right Ro.

α

¥

y

β

x

Left Ro.

α

¥

β

x

y

From the above three figures, it is clear that if we consider two nodes x and y and first apply left rotation on x then we will get figure 2. After that if we apply right rotation on y, then we will get figure 3. But, wait a minute, figure 1 and 3 are exactly same. Therefore left and right rotations are inverse of each other.

No, double rotations are not inverse. Because in splay tress, most of the time we use double rotations but every time we yield a different tree structure.

3/10)

(b) I am pointing the left pointer of a leaf to its predecessor in an inorder tree traversal.

3/11)

a)insert(3)

3

insert(1)

3

1

Zig case

1

3

Insert(4)

3

1

4

Zig zig case

4

3

1

Insert(5)

5

3

1

4

Zig case

5

3

1

4

Insert(2)

3

4

5

1

2

Zig zig

3

1

2

4

5

Zig zag

2

3

1

4

5

Insert(9)

9

2

1

2

Zig zig

9

3

4

5

1

5

3

4

Insert(6)

3

4

5

1

2

9

6

zig

3

4

5

1

2

6

9

Zig zig

3

6

4

5

1

2

9

Insert(8)

3

4

5

1

2

6

9

8

Zig zag

3

4

8

5

9

1

2

6

Delete(3)

8

8

4

5

6

9

1

2

8

Delete(3)

9

4

5

6

8

1

2

3

zig

9

4

5

6

1

2

3

Zig zag

4

5

3

1

2

9

6

Zig zig

9

3

4

5

1

2

6

8