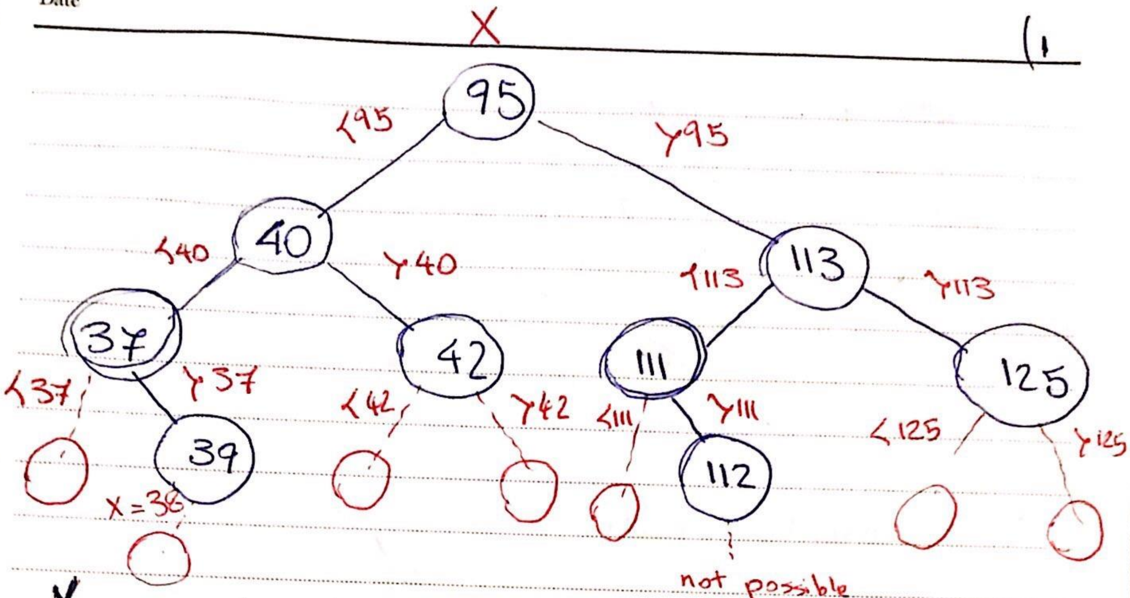
Bardia Ardakanian

9831072

HW3

Subject:
Date



X = node to be inserted

X	h	
<37	h ثابت	→ $x = [-\infty, 36]$
37<X<39	h=h+1	→ X=38
40<X<42	h	→ X = 41
42<x<95	h	→ $x = [43, 94]$
95<X<111	h	→ $x = [96, 110]$
113<X<125	h	→ $x = [114, 124]$
125<X	h	→ $x = [125, +\infty)$

Q2.

Let's try to prove the claim from hint: every binary search tree with n nodes can be transformed into a right-going chain by using at most $n-1$ right rotations.

If a given BST is already a right-going chain, we are finished. Otherwise, there is at least one left-going edge in the tree. On this edge we can perform right rotation. After the rotation, the child node moves to the right of its parent node, which means that the number of nodes in the right subtree is larger by 2. Thus the number of nodes in right subtree grows every time we do the right rotation. Consequently, after at most $n-1$ right rotations, the whole tree gets transformed into right-going chain.

Now our aim is to prove the original claim using this fact. Assume we want to transform binary-search tree T_1 with n nodes to T_2. First we transform T_1 to right-going chain, using the above procedure, which takes at most $n-1$ rotations. Now we reason as follows: we can transform T_2 to the same right-going tree using at most $n-1$ rotations. Since left rotations are **inverses** of right transformations, we use the left-rotations on right-going tree to obtain tree T_2. This takes at most $n-1$ transformations, i.e. in total there are $O(n)$ rotations.

Let

LEFT = an entire left linked list binary tree

RIGHT = an entire right linked list binary tree.

You can easily rotate LEFT to RIGHT in (n-1) rotations.

e.g: n = 3

```
   3           2          1
 2    to     1  3   to      2
1                             3
```

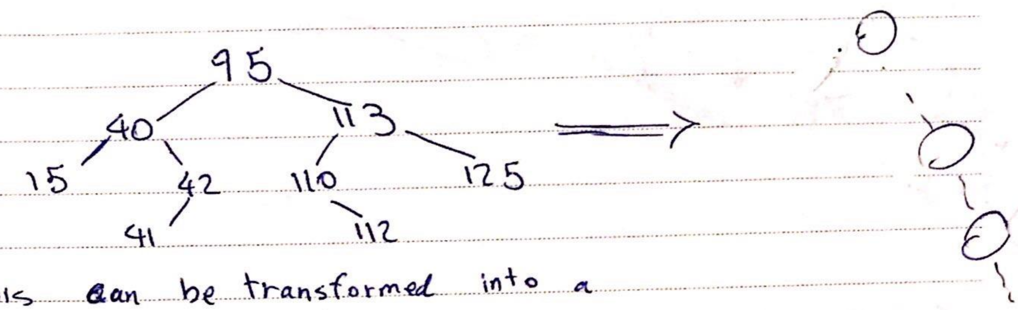
Proof: Since by definition, each right rotation will increase the length of the right most path by at least 1. Therefore, starting from right most path with length 1 (worst case), you need at most (n-1) rotations performed to make it into RIGHT.

Thus, you can easily conclude that any arbitrary shape of binary tree with n nodes can rotate into RIGHT within (n-1) rotations. Let T_1 be node you begin with Let T_2 be node you end with.

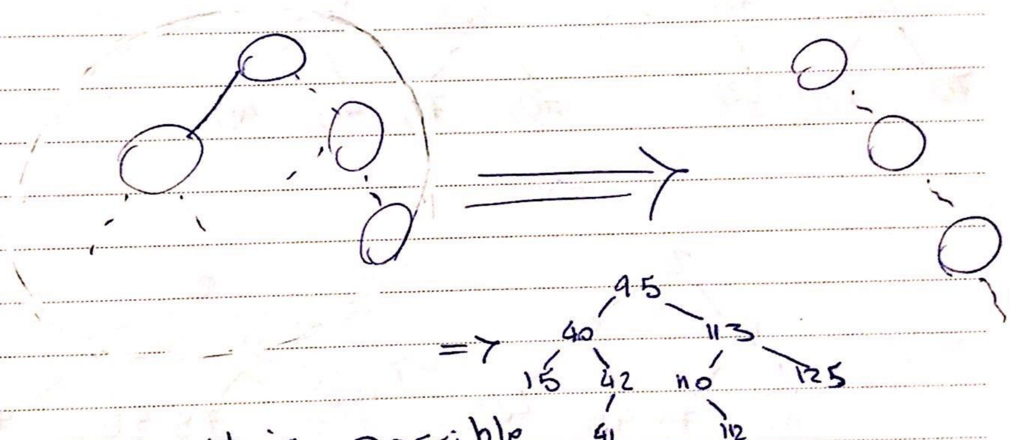
You can rotate T_1 to RIGHT within (n-1) rotations. Similarly, You can rotate T_2 to RIGHT within (n-1) rotations.

Therefore, To rotate T_1 into T_2, simply rotate T_1 into RIGHT , then do the inverse rotation to rotate from RIGHT into T_2.

Therefore, you can do this in (n-1)+(n-1) = 2n-2 rotations in upper bound.



this can be transformed into a right going tree using rotations same as all other BST with n = 9 and this attributes



so yes it is possible

Ps. The idea isn't mine, its from stackoverflow, so yeah.

We assume this struct as node object for the following questions

```
node
{
    data
    Node left
    node right
}

node root
```

Q3.

Inorder:

```
func inorder()
{
    if(root == NIL) return

    stack s
    node curr = root

    while(curr != NIL || size(s) > 0) //n - all nodes
    {
        // Reach most left node of curr
        while(curr != NIL)
        {
            s.push(curr)
            curr = curr.left
        }

        curr = s.pop()
        print(curr.data)
        //right subtree
        curr = curr.right
    }
}

/*
 * its actuall a*n - n*(subtree of curr)
 * 1 + n(1 + 1 + 1 + 1) = 1 + 4*n
 * O(n)
 */
```

Postorder:

```
func postOrder()
{
    stack s

    if (root == NIL) return

    s.push(root);
    Node prev = null;
    while (size(s) != 0) //n - each element get inserted
    {
        Node curr = s.peek();

        if (prev == NIL || prev.left == current || prev.right == current)
        {
            if (current.left != NIL)
                s.push(current.left);
            else if (current.right != NIL)
                s.push(current.right);
            else
                print(s.pop().data)
        }
        else if (current.left == prev)
        {
            if (current.right != NIL)
                s.push(current.right);
            else
                print(s.pop().data)
        }
        else if (current.right == prev)
            print(s.pop().data)

        prev = current;
    }
}

/*
 * the loop gets called untill all elements are pushed then popped from stack
 * 1 + n*(1 + 3 + 2 + 2 + 1 + 1 + 2 + 1 + 1 + 1 + 1) = 1 + 16n
 * O(n)
 */
```

Prerder:

```
func preorder()
{
    if (root == NIL) return

    stack s
    s.push(root)

    while (size(s) != 0) // n - each element get inserted to stack and get poped once
    {
        node newNode = s.peek();
        print(newNode.data)
        s.pop();

        if (newNode.right != NIL) {
            s.push(newNode.right);
        }
        if (newNode.left != NIL) {
            s.push(newNode.left);
        }
    }
}

/*
 * 1 + n*(1 + 1 + 1 + 1 + 1 + 1 + 1 + 1) = 1 + 8n
 * O(n)
 */
```

Q4.

a)

b)

```
//Find inorder and preorder traversals of tree, store them in
two auxiliary arrays inT[] and preT[].
//Find inorder and preorder traversals of subtree, store them
in two auxiliary arrays inS[] and preS[].
//If inS[] is a subarray of inT[] and preS[] is a subarray
preT[], then S is a subtree of T. Else not.
func is_subtree(node tree, node subtree)
{
    int inT[] = tree.inorder()
    int preT[] = tree.preOrder()

    int inS[] = subtree.inorder()
    int preS[] = subtree.preOrder()

    if (inS.subArray(inT) && preS.subArray(preS)) {
        return true
    }

    return false
}

//inorder and preOrder of a tree are O(n)
//is_subtree calls each function twice for two different trees
//we assume size of tree & subtree are n & m
//so we have order of growth n + n + m + m = 2(m + n)
//the function is from O(n+m) order of growth
```

Q5.

```
//initialize min and max based on your code
//or use a function to find it whatever
boolean isBST(node, min, max)
{
    if (root == NIL) return tree

    if (node.data < min || node.data > max) return false

    return isBST(node.next, min, node.data-1)
            && isBST(node.right, node.data+1)
}

//this function is on O(n) order of growth
//official call uses root as node, maximum input as max, minimum input as min
```

Subject:
Date:

In order: D, H, B, E, A, F, C, G (6

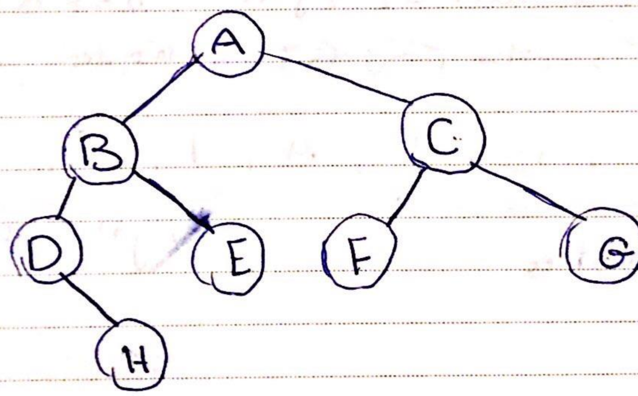
post order: H, D, E, B, F, G, C, A

اخری در post همان root است پس root = A

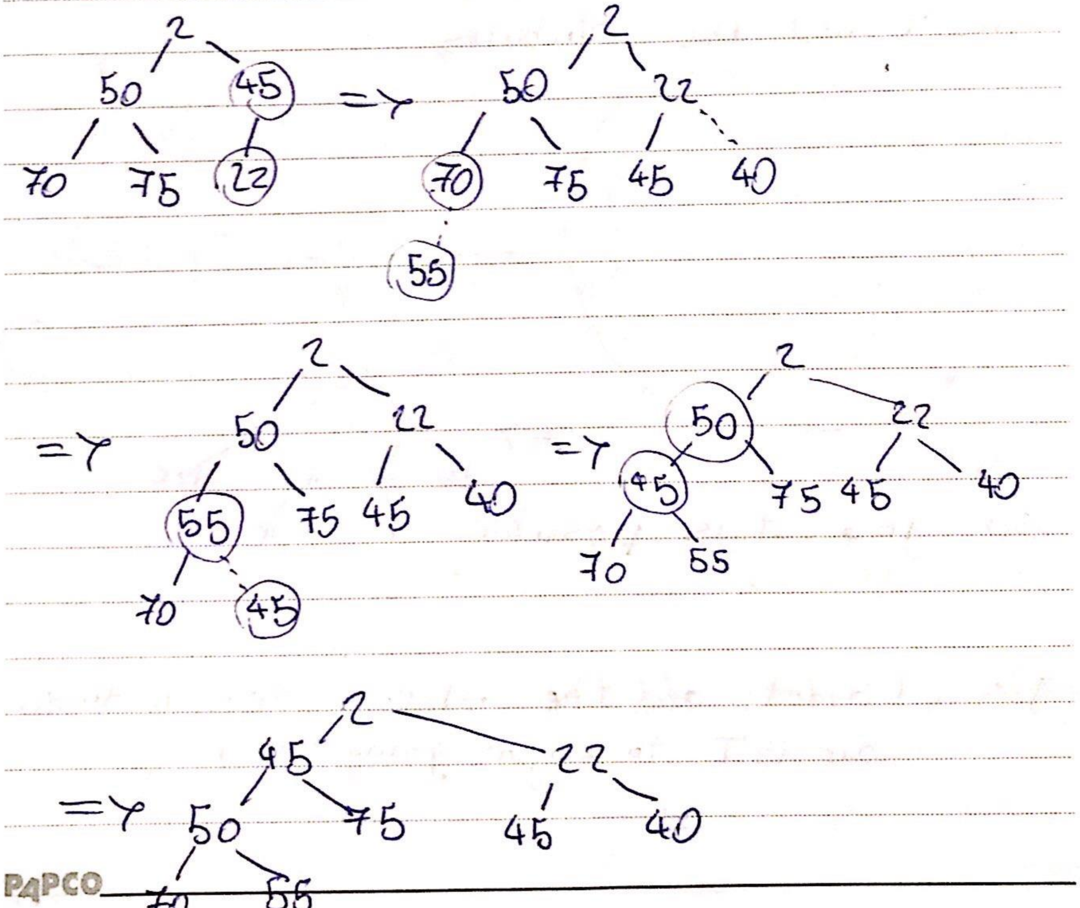
$\underbrace{D, H, B, E}_{\text{right subtree}}, A, \underbrace{F, G, G}_{\text{left subtree}}$

A.right = C   A.left = B

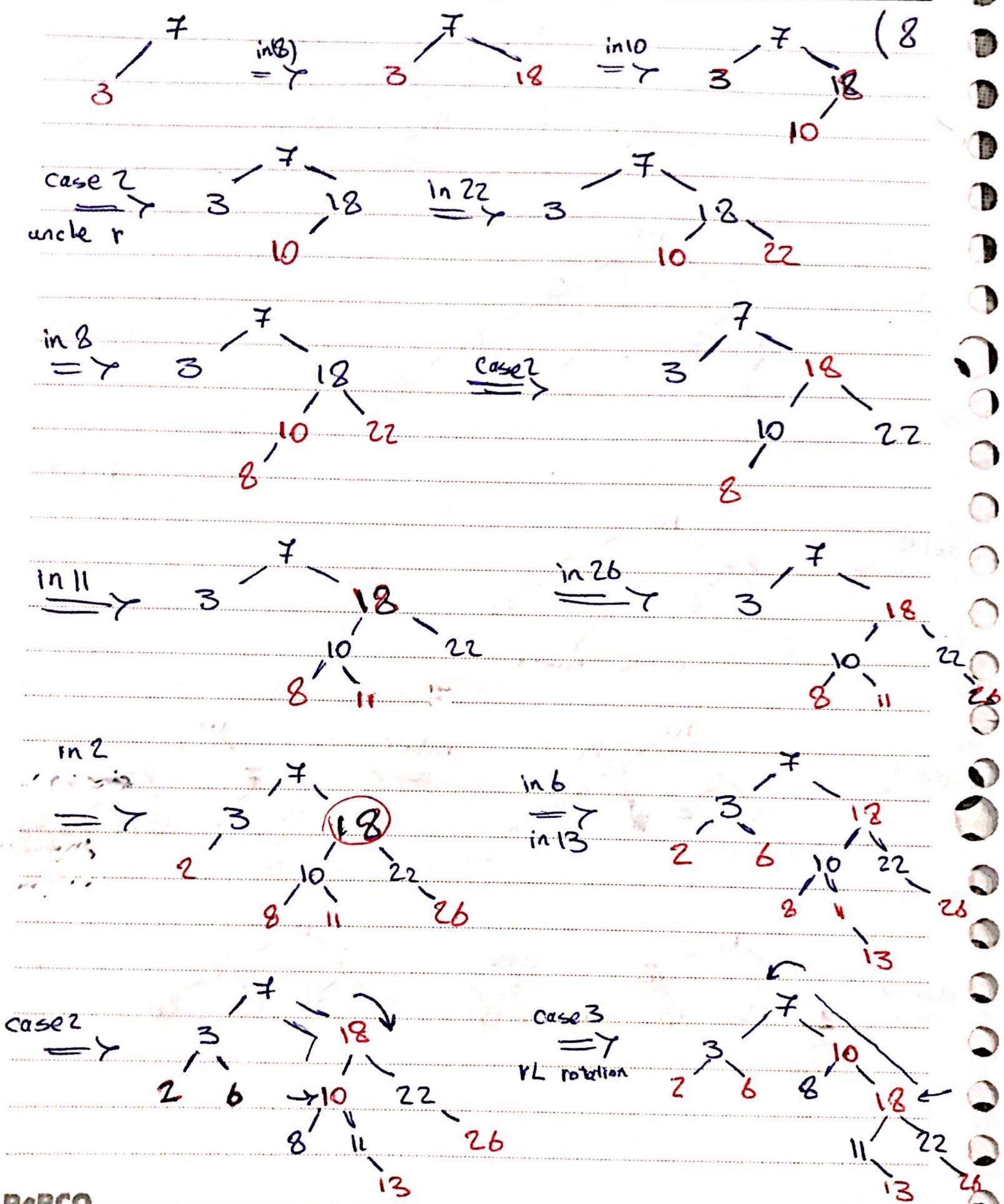
چون در inorder ، H بعد D چاپ شده پس H باید راست D باشد



2, 50, 45, 70, 75, 22, 40, 55, 45 (7



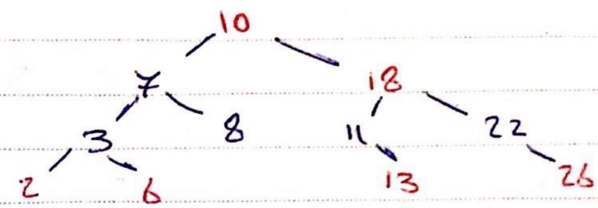
Subject:
Date



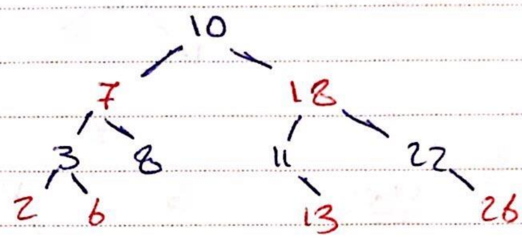
case 4 =>

Subject:
Date

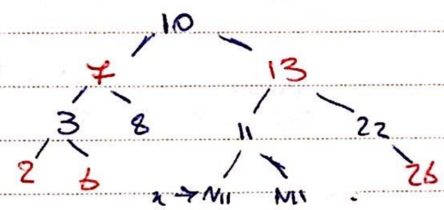
case 4 ⇒



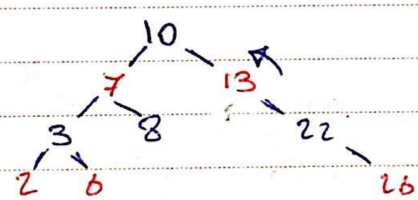
change ⇒ color



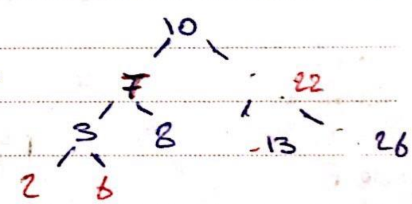
del 18 ⇒



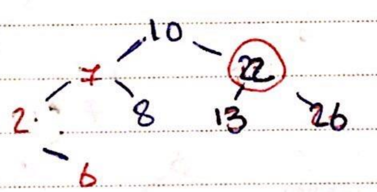
del 11 ⇒



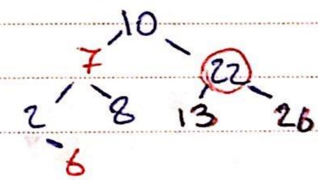
rotate ⇒



del 3 ⇒

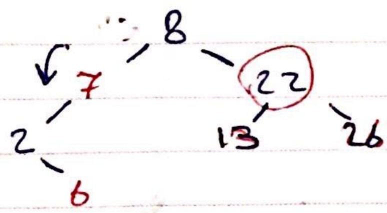


color ⇒ 2B

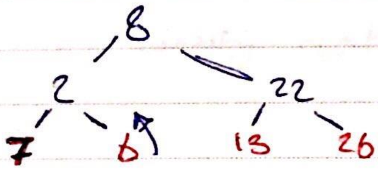


Subject :
Date

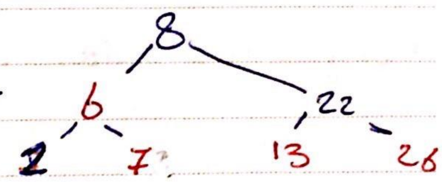
del 10 ⇒



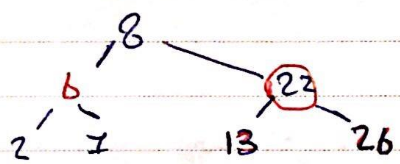
⇒



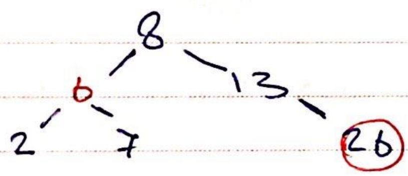
⇒



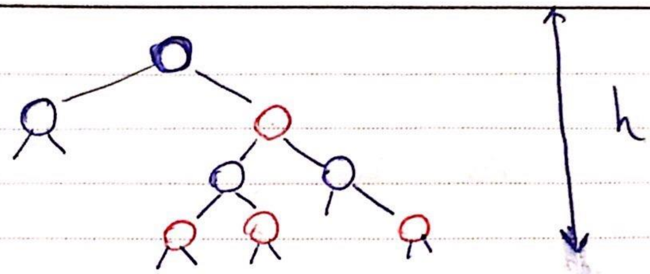
⇒



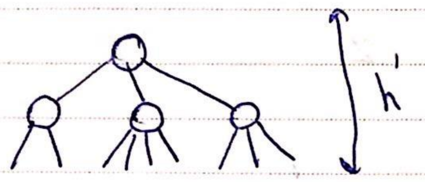
del 22 ⇒



Subject :
Date



merg red nodes into black parents



each node have 2-3-4 child

$h' \geq h/2 \longrightarrow$ most half of leaves on any Path are Red.

the number of leaves in each tree is "$n+1$"

$\Rightarrow n+1 \geq 2^{h'} \Rightarrow \log n+1 \geq h' \geq h/2$

$\Rightarrow h \leq 2\log^{n+1}$