15/08/2024

Day 18 OF DSA

Topics

CheckBox Tree

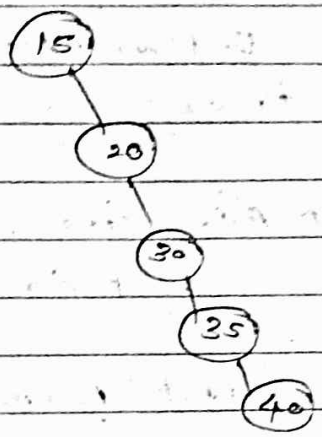
- [x] AVL Tree: Understanding and balancing AVL trees
- [x] Implement AVL tree operations: balancing rotation
- [x] kth smallest Number
- [x] is Binary Tree Heap?
- [x] Merge 2 Binary Max Heap
- [x] min Cost OF ropes

AVL Tree :

Eg: 15, 20, 30, 25, 40

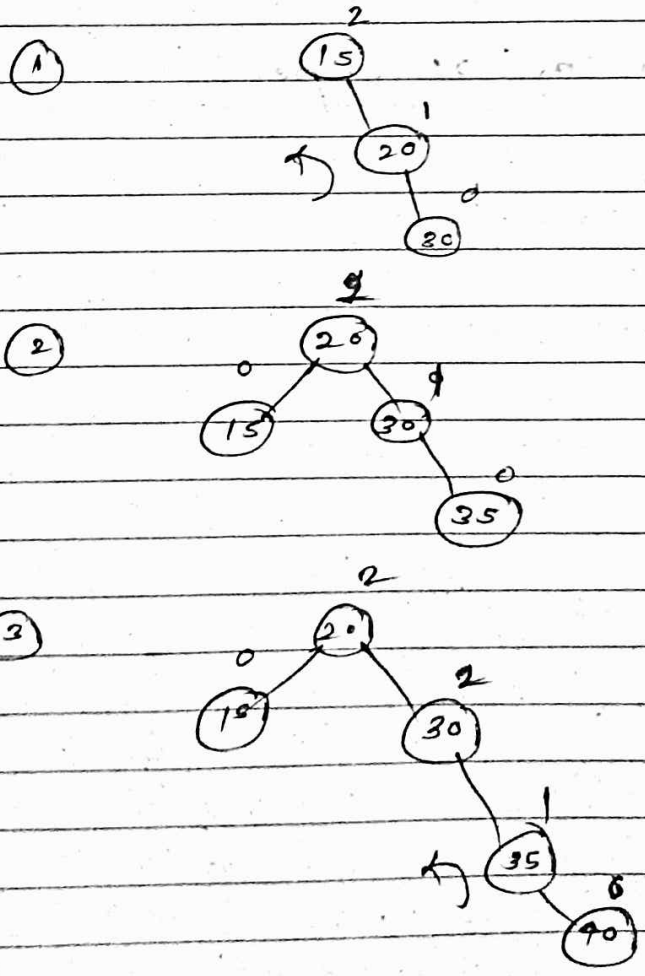
Search

Binary Tree

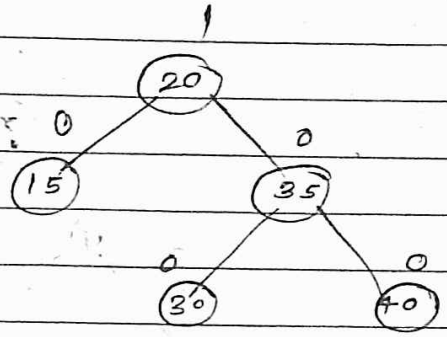


AVL - Tree

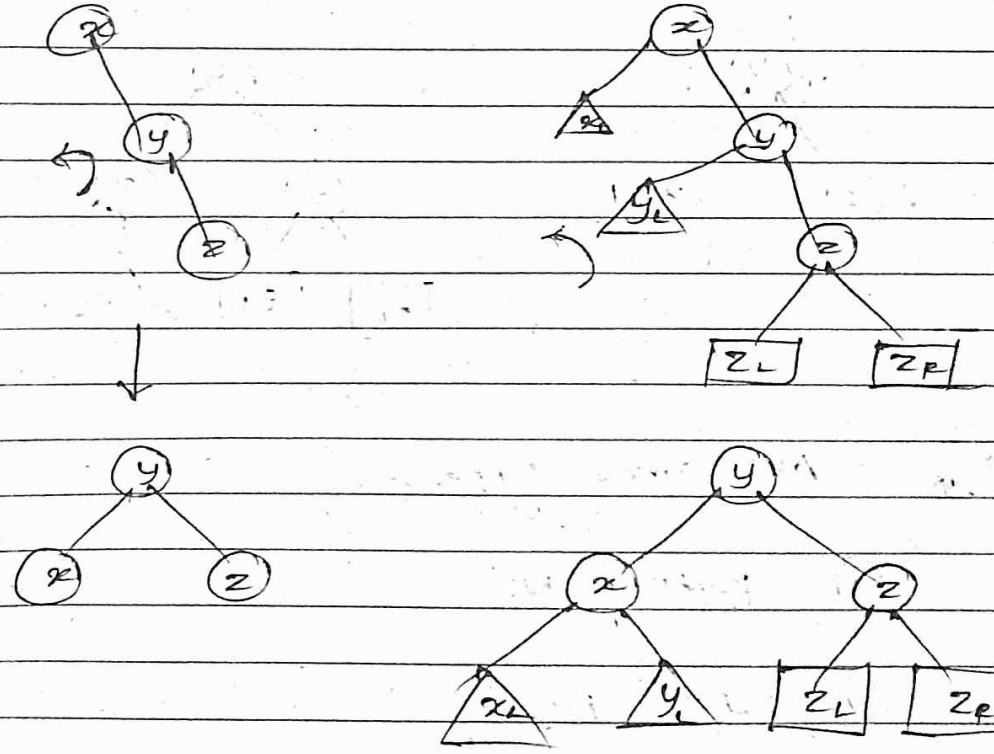
* height Balance = $|h_l - h_r| < 2$



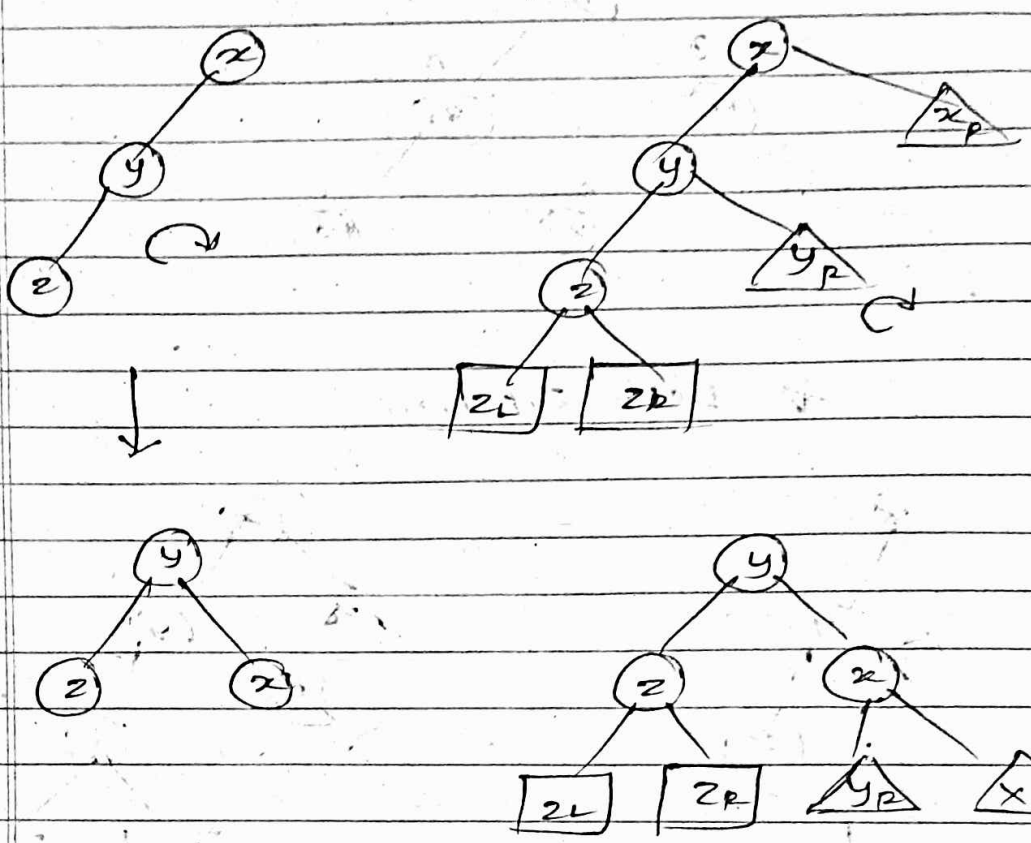
4



Left Rotation



* Right Rotation

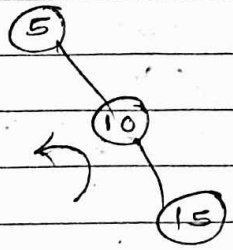


* AVL Tree Types :-

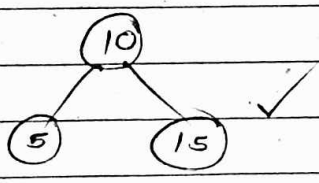
1. Right Right
2. Left Left
3. Right Left
4. Left Right

{5, 10, 15}

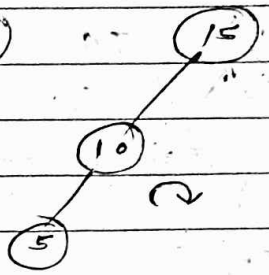
1



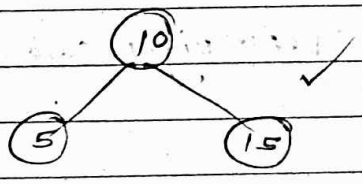
RR



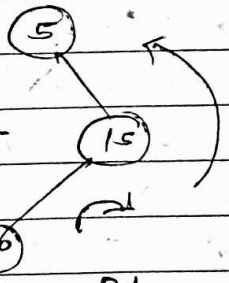
2



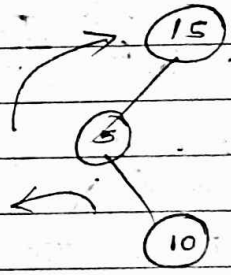
LL



3

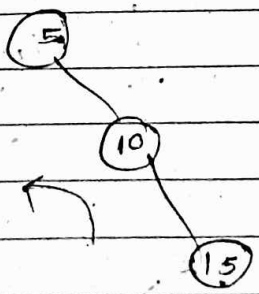


RL

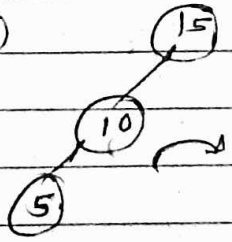


LR

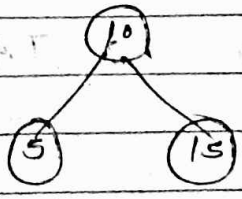
I



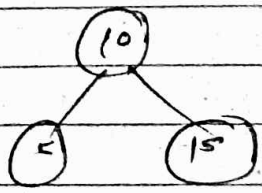
I



II

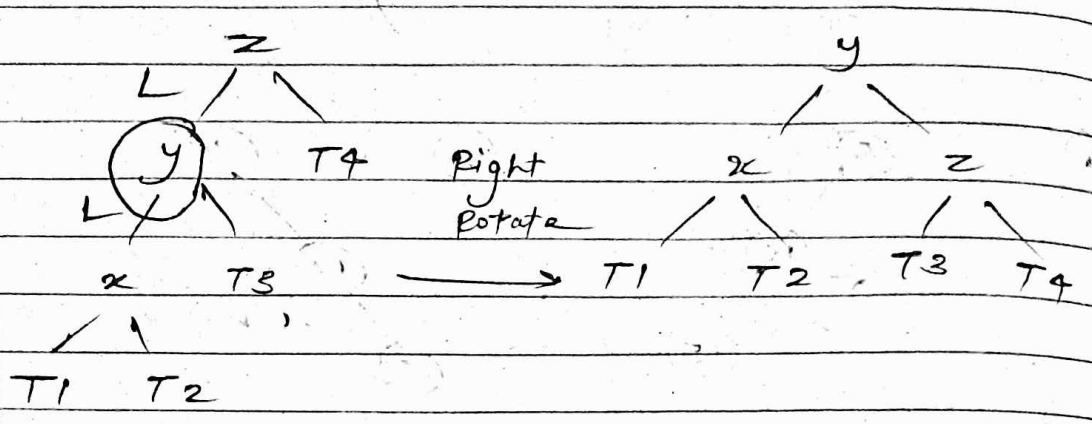


II

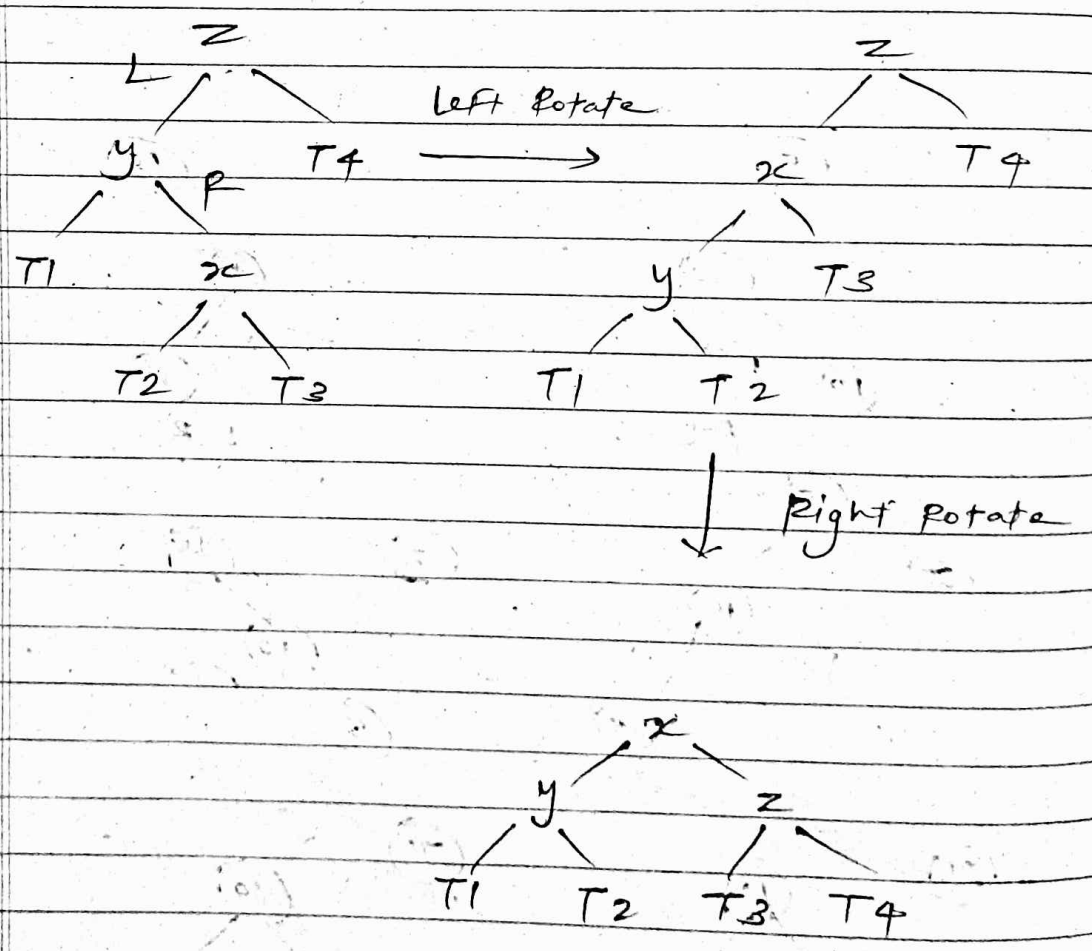


For Implementation purpose

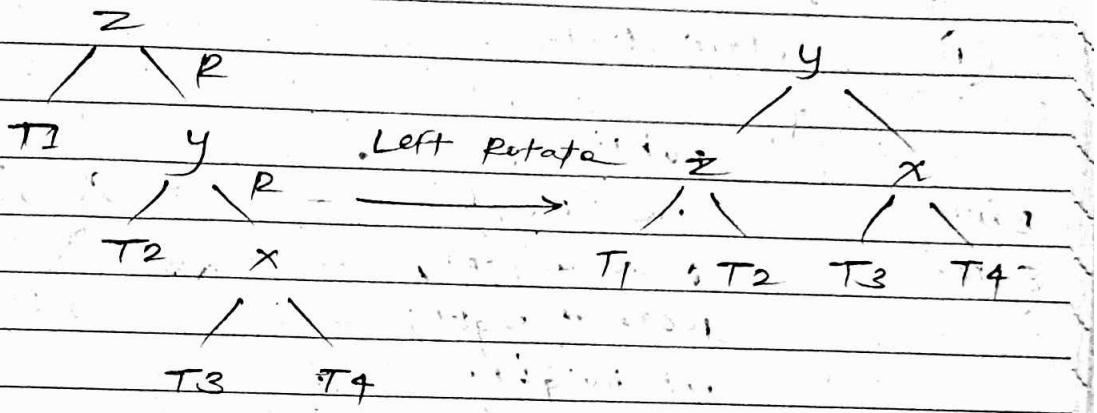
(1) Left Left case :-



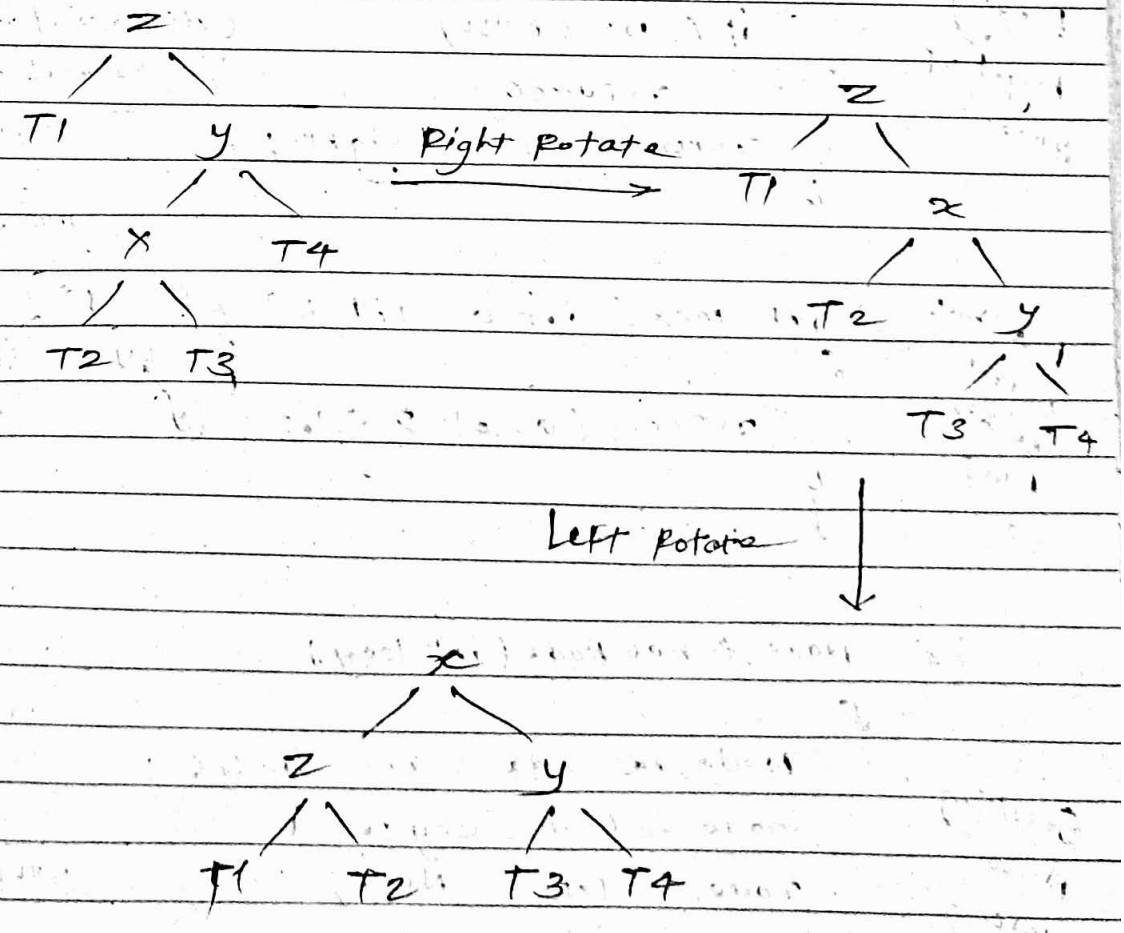
(2) Left Right case :-



③ Right Right case :-



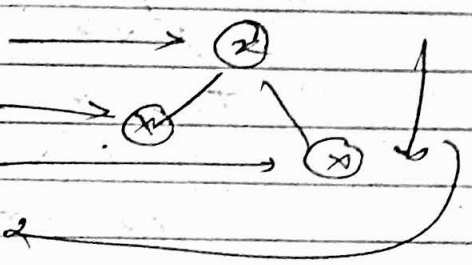
④ Right - left case :-



Implementation:-

(1) Basic Syntax

```
class Node
{
    public:
    int key;
    Node * left;
    Node * right;
    int height;
};
```



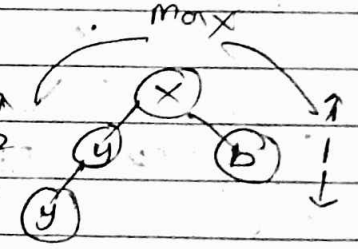
(2) Finding height of Node

```
int height (Node * N)
{
    if ( N == NULL)
        return 0;
    return N -> height;
}
```

Calculate it while Inserting

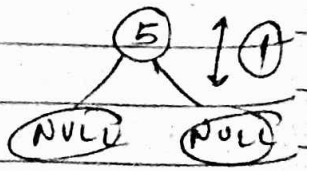
(3) Maximum height checking funct^n

```
int max( int a, int b)
{
    return (a > b) ? a : b;
}
```



(4) Creating New Node

```
Node * newNode( int key)
{
    Node * node = new Node();
    node -> key = key;
    node -> left = NULL;
    node -> right = NULL
    node -> height = 1;
    return (node);
}
```



(5) Node * rightRotate (Node * y)

{

(1) Node * x = y -> left ;

(2) Node * T2 = x -> right ;

(3) x -> right = y ;

(4) y -> left = T2 ;

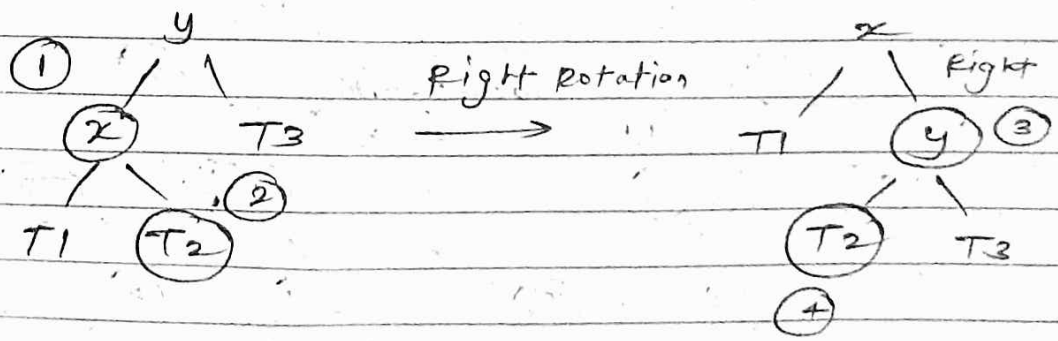
right rotation

y -> height = max (height (y -> left),
height (y -> right)) + 1 ;

x -> height = max (height (x -> left),
height (x -> right)) + 1 ;

return x ;

}



(6) Node * leftRotate (Node *x)

{

(1) Node *y = x → right;

(2) Node *T2 = y → left;

Left Rotation

(3) y → left = x;

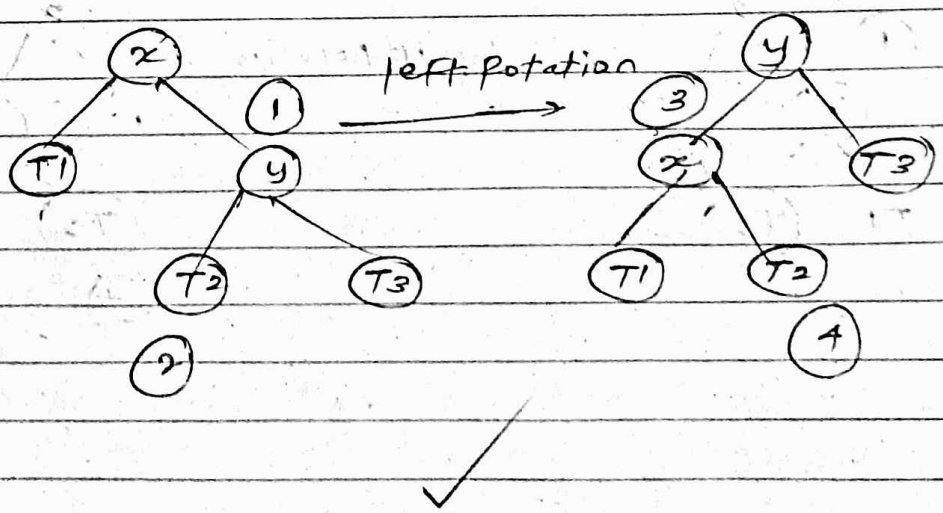
(4) x → right = T2;

x → height = (max(height(x → left),
height(x → right)) + 1;

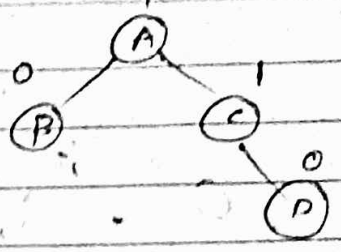
y → height = max(height (y → left),
height (y → right)) + 1;

return y;

}



$|h_L - h_R| < 2$ ✓



(7) int getBalance (Node *N)

{

if (N == NULL)

return 0;

return height (N → left) -

height (N → right);

}

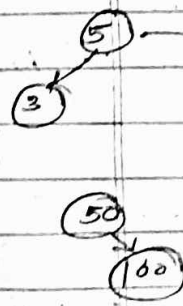
Checking balance

(8) Node * insert (Node * node, int key)

{

if (node == NULL)

return (newNode (key));



if (key < node->key)

node->left = insert (node->left, key);

else if (key > node->key)

node->right = insert (node->right,

key);

else

return node;

Updating height

node->height = 1 + (max (height (node->left),

height (node->right));

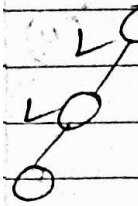
Checking for unbalancy

int balance = getBalance (node);

Page No.
Date

Left Left case

```
if (balance > 1 && key < node->left->key)
    return RightRotate(node);
```

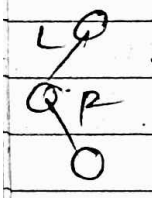


Right Right case

```
if (balance < -1 && key > node->right->key)
    return leftRotate(node);
```

L - R

```
if (balance > 1 && key > node->left->key)
{
    node->left = leftRotate(node->left);
    return RightRotate(node);
}
```



R - L

```
if( balance < -1 && key < node->right->key
{
    node->right = rightRotate(node->right);
    return leftRotate(node);
}

return node;
}
```

(9)

```
void preorder(Node * root)
{
    if(root != NULL)
    {
        cout << root->key << " ";
        preorder(root->left);
        preorder(root->right);
    }
}
```

printing purpose

✓

* Heap *

(1) kth Smallest Number:-

[7, 10, 4, 20, 15]

Any

ye alse pi kiya tha

K = 4

Approach:-

(i) For First K element, make a max heap

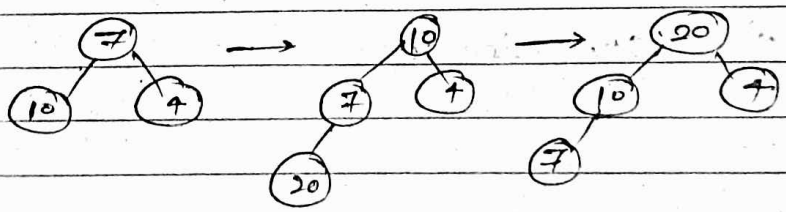
(2) For rest of elements,

if element < Heap.top()

→ heap.pop()

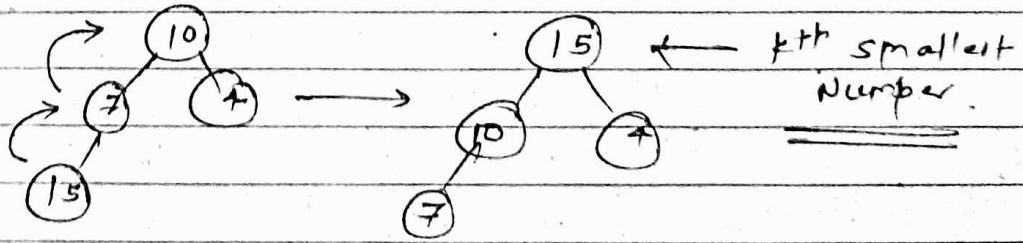
→ heap.push(element).

step 1:-



step 2:- Remaining 15

15 < 20 ✓



Time Complexity: O((n-k) * log(k))

Space Complexity: O(k)

(2) is Binary Tree Heap?

Approach:-

1) Create two Functions

CBT()
&
maxorder()

2) CBT - Complete Binary tree
maxorder - max heap

3) IF this both Functions are true then
Our Binary tree is heap

4) Otherwise Not

✓

③ Merge 2 Binary max heap

Approach:-

① Simple take two Binary max heap

② store it in new vector

③ Use Heapify Algorithm on it

④ It will make that vector new max heap

⑤ Finally return it.

Notes-

leaf = $\geq$ 0 - Base Indexing → $n/2 - 1$ → $i >= 0$ → Non leaf Nodes

$2*i+1$

right = $2i+2$

Left = $2i+1$ - Base indexing → $n/2$ → >0 - non-leaf Nodes

right = $2i+1$

④ Min Cost of ropes :-

Approach :-

① pick up two min elements From heap.

② sum it up & push in heap

③ Do this till heap.size > 1. And At the end sum all that elements