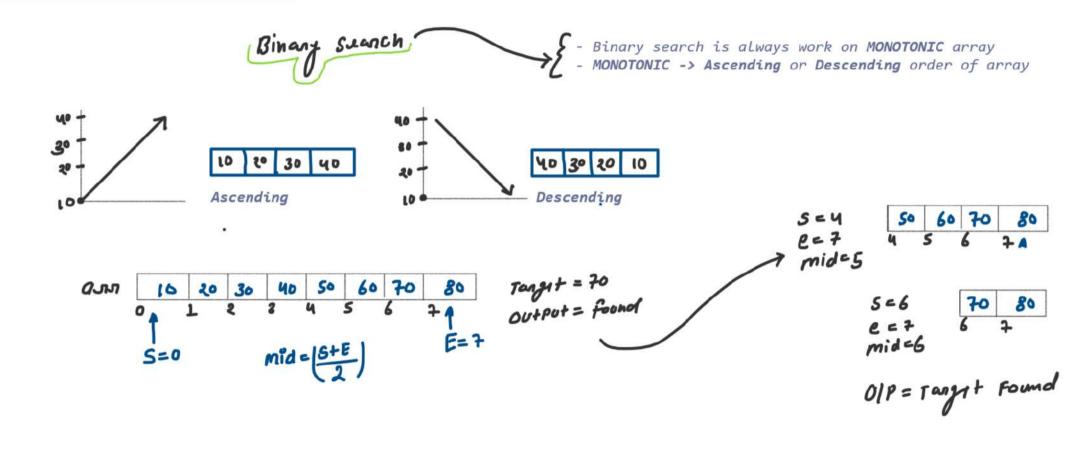
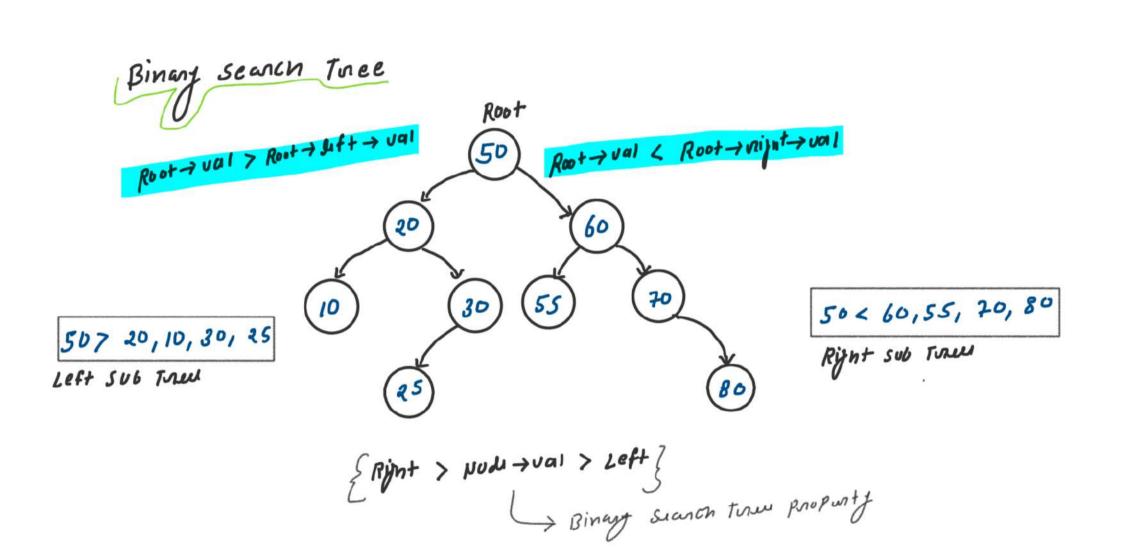
06/12/2023

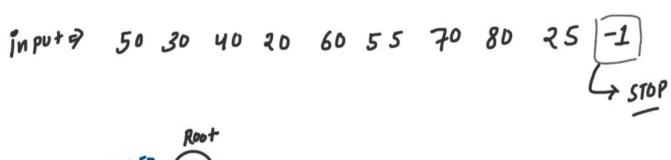
BINARY SEARCH TREE CLASS - 1

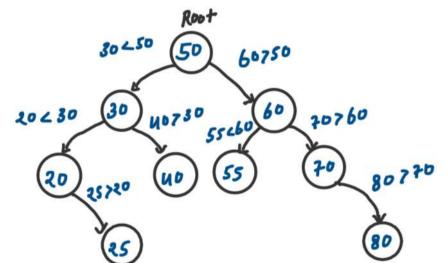
1. What is Binary Search Tree?

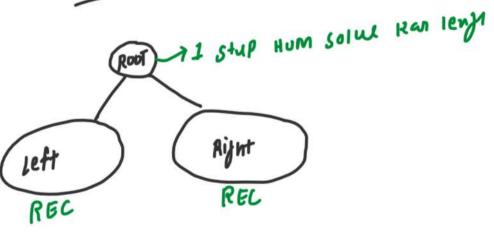




2. Create Binary Search Tree







```
// PROBLEM 01: Create binary search tree

// CREATE BINARY SEARCH TREE
Node* inserIntoBST(Node* root, int data){...}

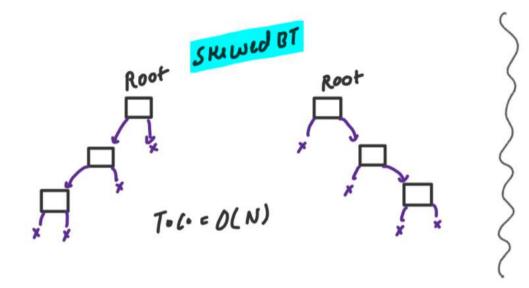
void createBST(Node* &root){
    cout<< "Enter Data: " << endl;
    int data;
    cin >> data;

    while (data != -1) {
        root = inserIntoBST(root, data);
        cin >> data;
    }

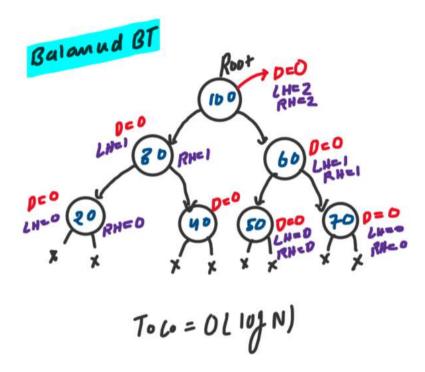
}

/*
Binary Tree Input: 50 30 40 20 60 55 70 80 25 -1

OUTPUT:
Level Wise Order:
50
30 60
20 40 55 70
25 90
```



in the Binary Tree



Overall time complexity:

Worst Case: O(N)

Average Case: O(N Log N)

Where N is number of nodes in binary search tree

Space Complexity: O(H), Where H is height of BST

3. Binary Search Tree Traversals

PRE-DRDER [NLR)

50 30 20 25 40 60 55 70 80

IN-DRDER [LNR)

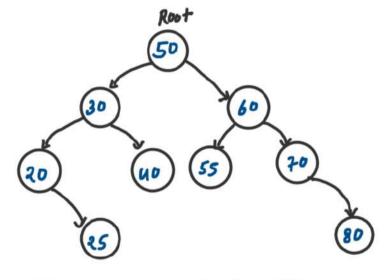
Always Print

IN-DRDER [LNR)

Nucles in

POST-DRDER [LRN)

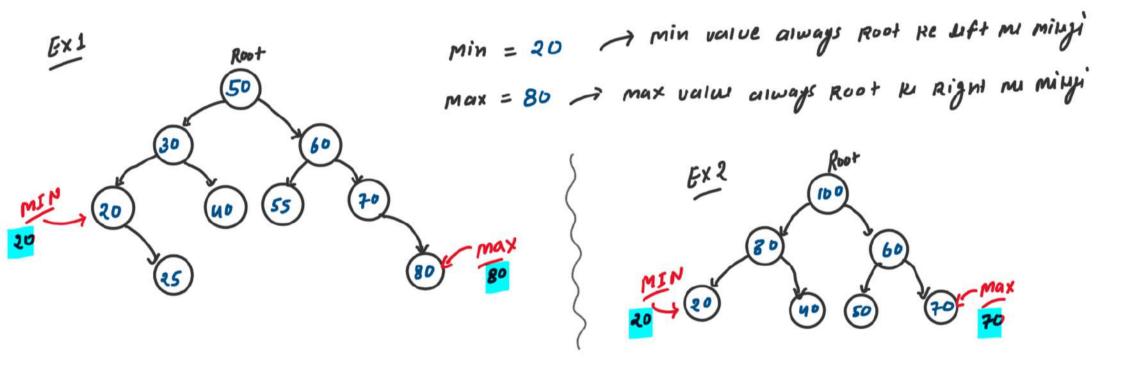
25 20 40 30 55 80 70 60 50



Time and space complexity: O(N), where N is total number of nodes in BST



4. Min and Max Value in BST



```
. .
Node* minValue(Node* root){
    if(root == NULL){
        return NULL;
    Node* temp = root:
    while(temp->left != NULL){
        temp = temp->left:
    if(root == NULL){
    Node* temp = root;
    while(temp->right != NULL){
        temp = temp->right;
```

Time complexity:

Skewed BST: O(N) or O(H) in the worst case Balanced BST: $O(\log N)$ in the average case

Space Complexity: O(H)

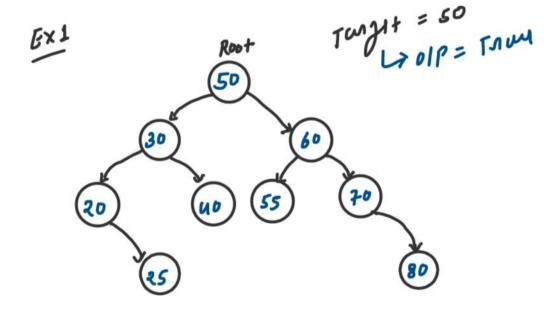
In the worst case: H is equal to O(N)

In the average case: H is equal to O(log N)

Where H is height of BST and N is number of nodes in BST



5. Target Value is Present or not in BST



```
. .
bool searchInBST(Node* root, int target){
    if(root == NULL){
    if(target == root->data){
    if(target > root->data){
       right = searchInBST(root->right, target);
    if(target < root->data){
       left = searchInBST(root->left, target);
   return right || left;
```

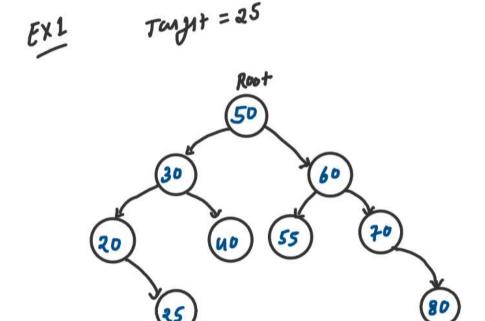
DRY RUN Tonget = 80 PRUPERTY + 80 7 60 60 780770 Time complexity: Return RILL Skewed BST: O(N) or O(H) in the worst case Balanced BST: O(log N) in the average case Right => TRUE Space Complexity: O(H) In the worst case: H is equal to O(N)

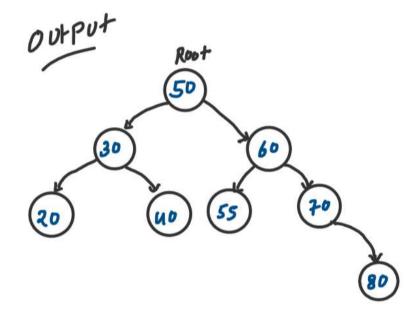
Where H is height of BST and N is number of nodes in BST

In the average case: H is equal to O(log N)



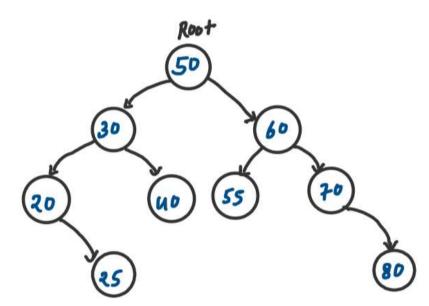
6. Delete Node from BST

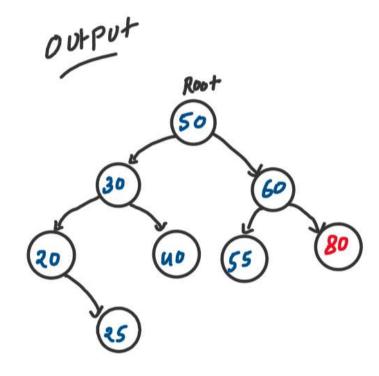




EX2

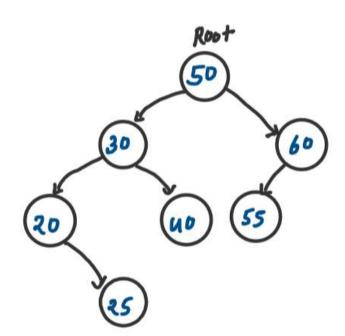
Taryst = 70

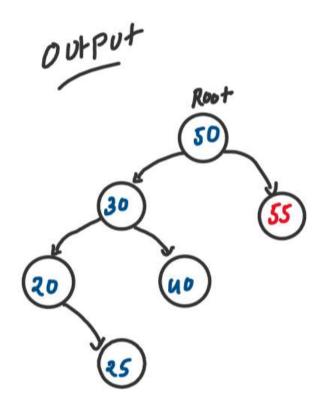




EX3

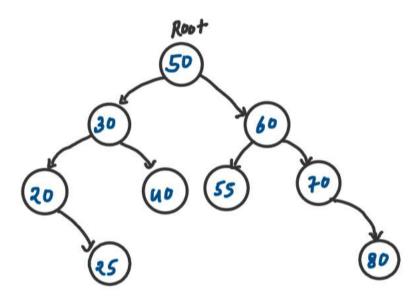
Tanget = 60

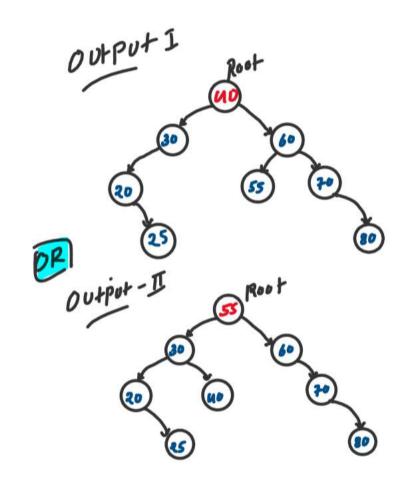




EXU

Tanget =50





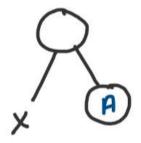
[ASE] Leaf Nody

× / ×

netum Null



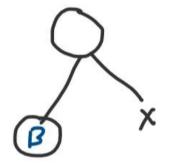
CASE 2



netum A

BX2

CASE3



Wetum B

EX3

CASE 4 LST => max Node Find Kanlo > Replace with target Nools From LST min pud Find Kano Ly Replace with ranget Mode Cy deliver min word from RST

```
// Delete node from BST
Node* deleteFromBST(Node* root, int target){
    // Base case
    if(root == NULL){
        return NULL;
    }

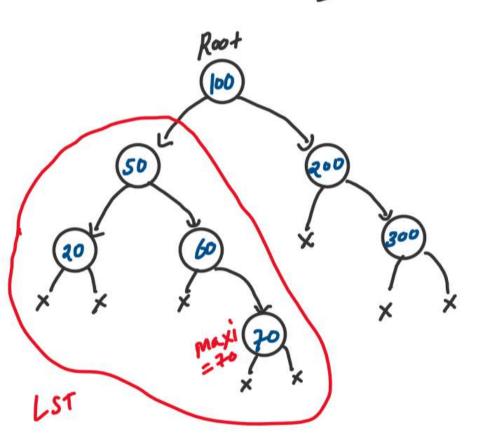
    // Step 01: target ko find krlo
    // Target root v ho skta hat
    if(target == root->data){
        ...
    }

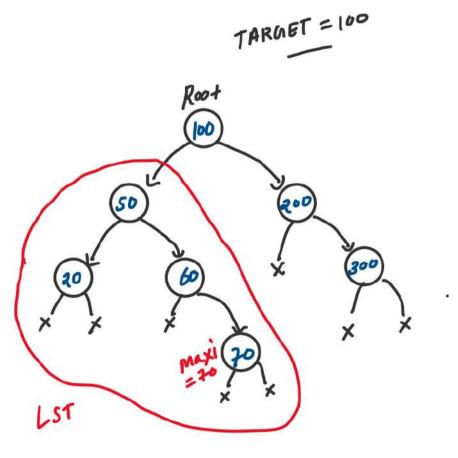
    // Target root ke left me v ho skta hat
    else if(target < root->data){
        root->left = deleteFromBST(root->left, target);
    }
    // Target root ke right me v ho skta hat
    else{
        root->right = deleteFromBST(root->right, target);
    }
    return root;
}
```

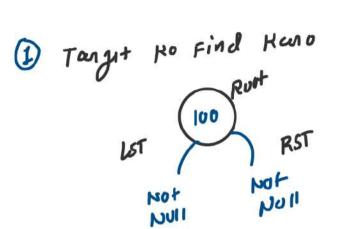
```
. .
   if(target == root->data){
       if(root->left == NULL && root->right == NULL){
       else if(root->left == NULL && root->right != NULL){
           Node* childSubTreeA = root->right;
       else if(root->left != NULL && root->right == NULL){
           return childSubTreeB;
       else if(root->left != NULL && root->right != NULL){
           Node* maxi = maxValue(root->left);
           root->left = deleteFromBST(root->left, maxi->data);
```

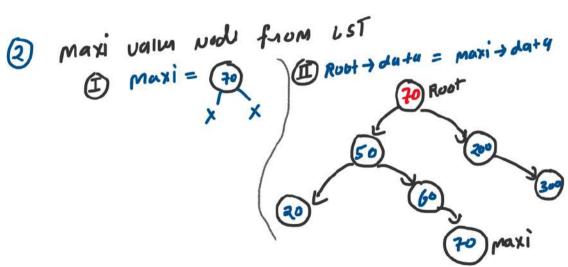
DRY RUN

TARGET = 100

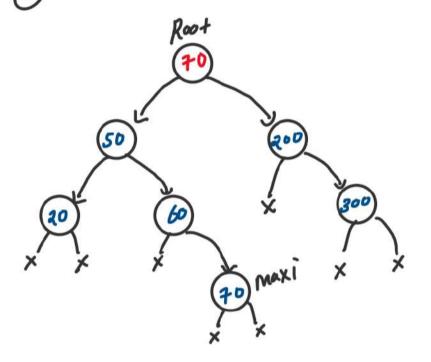






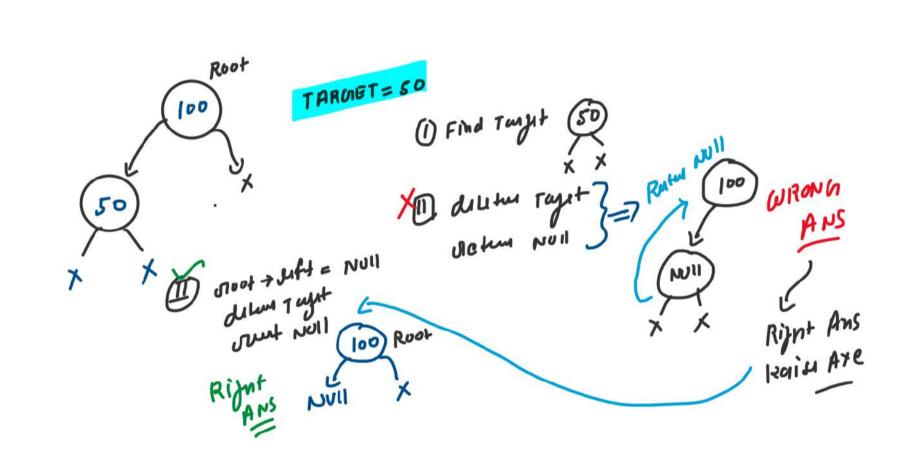


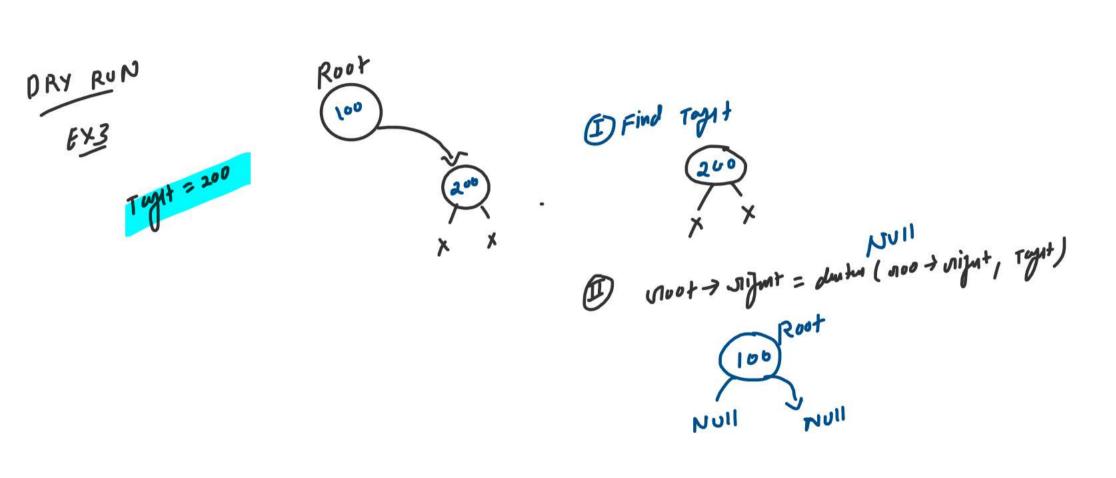
(I) Rout > data = maxi > data



(I) Root > suft = deliter (nout > left, maxi > dut) poot > Juft Final output

DRY RUN





Time complexity: CASES 1, 2, and 3: O(1) CASE 4: O(N)

Overall Time Complexity:

Skewed BST: O(N) or O(H) in the worst case Balanced BST: O(log N) in the average case

Space Complexity: O(H)

In the worst case: H is equal to O(N)

In the average case: H is equal to O(log N)

Where H is height of BST and N is number of nodes in BST