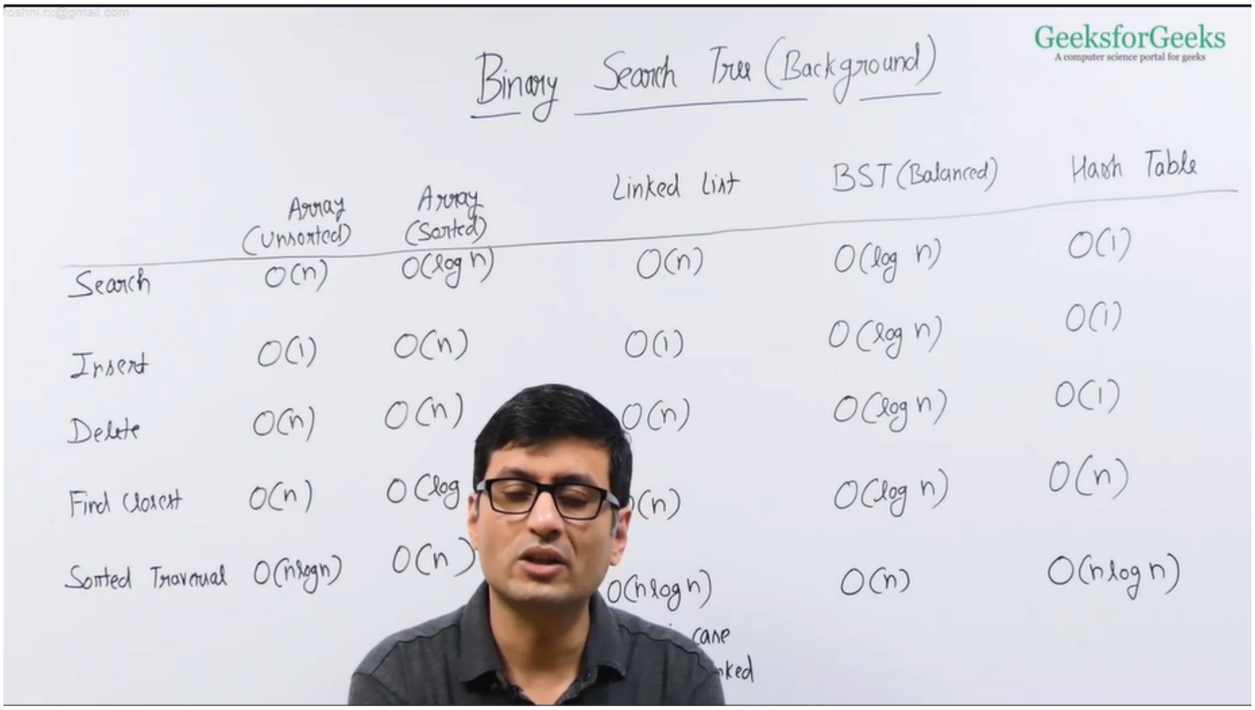
BINARY SEARCH TREES



1. **SEARCH**

Time:O(h) Aux. Space:O(h)

```
bool search(Node *root,int x)
{
    if(root==NULL)
    {
        return false;
    }
    if(root->key==x)
    {
        return true;
    }
    else if(root->key<x)
    {
```

```
        return search(root->right,x);
    }
    else
    {
        return search(root->left,x);
    }
}
```

2. **INSERT**

Time:O(h) Aux.Space:O(h)

```
Node* insert(Node *root,int x)
{
    if(root==NULL)
    {
        return new Node(x);
    }
    else if(root->key>x)
    {
        root->left=insert(root->left,x);
    }
    else
    {
        root->right=insert(root->right,x);
    }
    return root;
}
```

3. **DELETION**

Time:O(h) Aux. Space:O(h)

```
Node *getSuccessor(Node *curr)
{
```

```
    curr=curr->right;
    while(curr!=NULL && curr->next!=NULL)
    {
        curr=curr->left;
    }
    return curr;
}
Node *DeleteNode(Node *root,int x)
{
    if(root==NULL)
    {
        return NULL;
    }
    if(root->key<x)
    {
        root->right=DeleteNode(root->right,x);
    }
    else if(root->key>x)
    {
        root->left=DeleteNode(root->left,x);
    }
    else
    {
        if(root->left==NULL)
        {
            Node *temp=root->right;
            delete root;
            return temp;
        }
        else if(root->right==NULL)
        {
            Node *temp=root->left;
            delete root;
            return temp;
        }
```

```
        else
        {
            Node *temp=getSuccessor(root);
            root->key=temp->key;
            root->right=DeleteNode(root->right,temp->key);
        }
        return root;
    }
}
```

4. FLOOR IN BST

Time:O(h) Aux. Space:O(1)

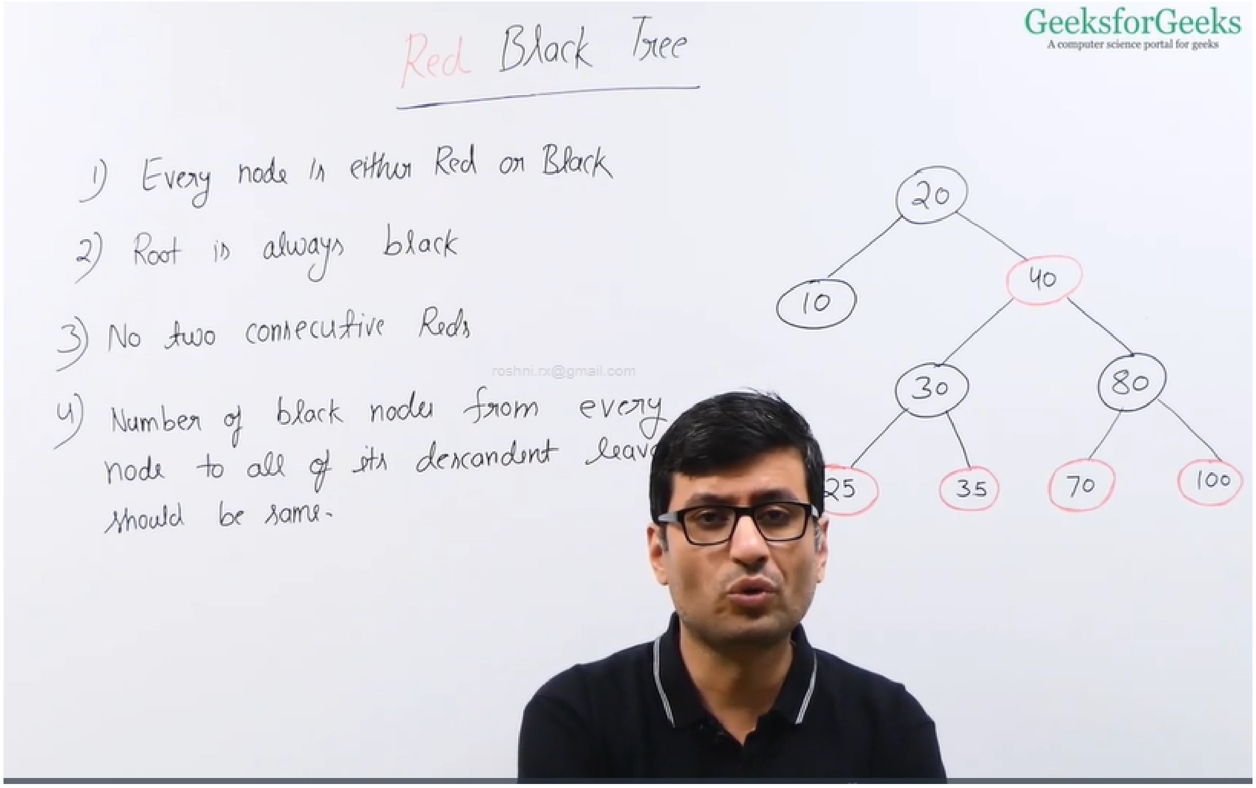
```
Node* fbst(Node *root,int x)
{
    Node* res=NULL;
    while(root)
    {
        if(root->key==x)
        {
          return root;
        }
        else if(root->key<x)
        {
          res=root;
          root=root->right;
        }
        else
        {
            root=root->left;
        }
    }
    return res;
}
```

5. CEIL IN BST

Time:O(h) Aux. Space:O(1)

```
Node *cbst(Node *root,int x)
{
    Node *res=NULL;
    while(root)
    {
        if(root->key==x)
        {
            return root;
        }
        else if(root->key<x)
        {
            root=root->right;
        }
        else
        {
            res=root;
            root=root->left;
        }
    }
    return res;
}
```

6. RED BLACK TREE



7. CEILING ON LEFT SIDE IN AN ARRAY

Time:O(nlogn)

```
void ceiling(int arr[],int n)
{
    set<int>s;
    cout<<-1<<" ";
    s.insert(arr[0]);
    for(int i=1;i<n;i++)
    {
        auto it = s.upper_bound(arr[i]);
        if(it!=s.end())
        {
            cout<<(*it)<<" ";
        }
        else
```

```
        {
            cout<<-1<<" ";
        }
        s.insert(arr[i]);
    }
}
```

8. FIND KTH SMALLEST IN BST

Time:O(h)

```
struct Node
{
  int key;
  struct Node *left;
  struct Node *right;
  int lCount;
  Node(int k){
      key=k;
      left=right=NULL;
      lCount=0;
  }
};

Node *kth(Node *root,int k)
{
    if(root==NULL)
    {
        return root;
    }
    int count = root->lCount + 1;
    if(count==k)
    {
        return root;
    }
```

```
    if(count>k)
    {
        return kth(root->left,k);
    }
    else
    {
        return kth(root->right,k-count);
    }
}
```

9. CHECK BST

Time:O(n) Aux. Space:O(h)

```
int prevv = INT_MIN;
bool isBST(Node *root)
{
    if(root==NULL)
    {
        return true;
    }
    if(isBST(root->left)==false)
    {
        return false;
    }
    if(root->key<=prevv)
    {
        return false;
    }
    prevv=root->key;
    return isBST(root->right);
}
```

10. FIX BST WITH 2 NODES SWAPPED

```
Node *prevv=NULL,*first=NULL,*second=NULL;
void fixBST(Node* root)
{
    if (root == NULL)
        return;
    fixBST(root->left);
    if(prevv!=NULL && root->key<prevv->key){
        if(first==NULL)
            first=prevv;
        second=root;
    }
    prevv=root;

    fixBST(root->right);
}
```

11. PAIR SUM WITH GIVEN BST

T:O(n) S:O(n)

```
bool isPairSum(Node *root, int sum, unordered_set<int> &s)
    {
        if(root==NULL)return false;

        if(isPairSum(root->left,sum,s)==true)
            return true;

        if(s.find(sum-root->key)!=s.end())
            return true;
        else
            s.insert(root->key);
        return isPairSum(root->right,sum,s);
    }
```

12. VERTICAL SUM IN A BST

T:O(nlog hd)
hd=> total no. of possible horizontal distances

```
void vSumR(Node *root,int hd,map<int,int> &mp){
    if(root==NULL)return;
    vSumR(root->left,hd-1,mp);
    mp[hd]+=root->key;
    vSumR(root->right,hd+1,mp);
}
```

13. VERTICAL TRAVERSAL OF BST

```
void vTraversal(Node *root){
    map<int,vector<int>> mp;
    queue<pair<Node*,int>> q;
    q.push({root,0});
    while(q.empty()==false){
        auto p=q.front();
        Node *curr=p.first;
        int hd=p.second;
        mp[hd].push_back(curr->key);
        q.pop();
        if(curr->left!=NULL)
            q.push({curr->left,hd-1});
        if(curr->right!=NULL)
            q.push({curr->right,hd+1});
    }
    for(auto x:mp){
        for(int y:x.second)
            cout<<y<<" ";
        cout<<endl;
    }
}
```

14. TOP VIEW OF BST

```
void topView(Node *root){
    map<int,int> mp;
    queue<pair<Node*,int>> q;
    q.push({root,0});
    while(q.empty()==false){
        auto p=q.front();
        Node *curr=p.first;
        int hd=p.second;
        if(mp.find(hd)==mp.end())
            mp[hd]=(curr->key);
        q.pop();
        if(curr->left!=NULL)
            q.push({curr->left,hd-1});
        if(curr->right!=NULL)
            q.push({curr->right,hd+1});
    }
    for(auto x:mp){
        cout<<x.second<<" ";
    }
}
```

15. BOTTOM VIEW OF BST

```
void bottomView(Node *root){
    map<int,int> mp;
    queue<pair<Node*,int>> q;
    q.push({root,0});
    while(q.empty()==false){
        auto p=q.front();
        Node *curr=p.first;
        int hd=p.second;
        mp[hd]=(curr->key);
        q.pop();
```

```
        if(curr->left!=NULL)
            q.push({curr->left,hd-1});
        if(curr->right!=NULL)
            q.push({curr->right,hd+1});
    }
    for(auto x:mp){
        cout<<x.second<<" ";
    }
}
```

16. FIND CLOSEST ELEMENT IN BST

T:O(h) S:O(h)

```
void in(Node *root,int &x,int K)
{
    if(root)
    {
        in(root->left,x,K);
        x = min(x,abs(root->data-K));
        in(root->right,x,K);
    }
}
    int minDiff(Node *root, int K)
    {
        int min_diff = INT_MAX;
        in(root,min_diff,K);
        return min_diff;
    }
```