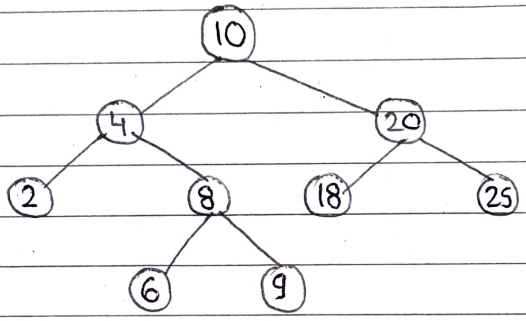
Date _____ Page _____

Day - 165

Check BST - 2

* Check BST:



⇒ If the inorder of the tree is in ascending order then the tree is BST.

Code

```
void inorder(Node * root, vector<int> &ans){
    if(!root) return;

    inorder(root->left, ans);
    ans.push_back(root->ans);
    inorder(root->right, ans);
}
```

Date _____ Page _____

```
bool isBST (Node * root){
    inorder(root, ans);
    for(int i=1; i<ans.size(); i++){
        if(ans[i] <= ans[i-1])
            return 0;
    }
    return 1;
}
```

⇒ So, can we solve it without using vector.

⇒ By using vector, we are checking curr element with prev element.

⇒ So, we can do this without using vector.

⇒ we can take prev & curr pointer & check the same thing.

⇒ So, we will do checking in the inorder order.

⇒ So, our prev is INT_MIN.

⇒ we will store prev & check.

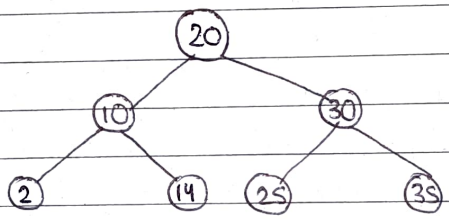
Code

```
bool isBST(Node * root, int &prev){
    if(!root)
        return 1;
    bool l = BST(root->left, prev);
    if(l == 0)
        return 0;
```

Date _____ Page _____

```
if(root->data <= prev)
    return 0;
prev = root->data;
return isBST(root->right, prev);
}
```

* **Min. Distance b/w BST Nodes:**



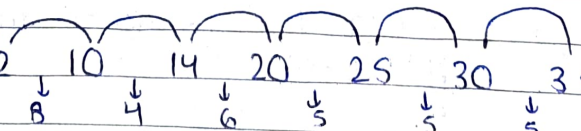
⇒ We will do subtraction b/w the data of two nodes i.e. abs(A-B).

⇒ And after doing that, the min of all the distance is our answer.

⇒ So, we will find the inorder then do the subs. b/w the adjacent two nodes.

⇒ Simply, in the end, we will return the min.

Ex:



⇒ We will take prev & do inorder traversal & store the min distance in a variable.

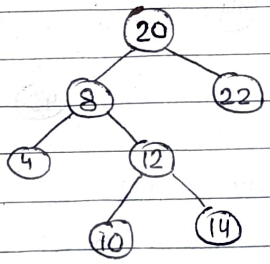
Date _____ Page _____

Code

```
void minDist(Node * root, int &prev,
                int & ans){
    if(!root) return;
    minDist(root->left, prev, ans);
    if(prev != INT_MIN)
    ans = min(ans, root->data - prev);
    prev = root->data;
    minDist(root->right, prev, ans);
}

int main(){
    int prev = INT_MIN;
    int ans = INT_MAX;
    minDist(root, prev, ans);
    return ans;
}
```

* Sum of k smallest element in BST:



⇒ We have to return the k smallest nodes sum.

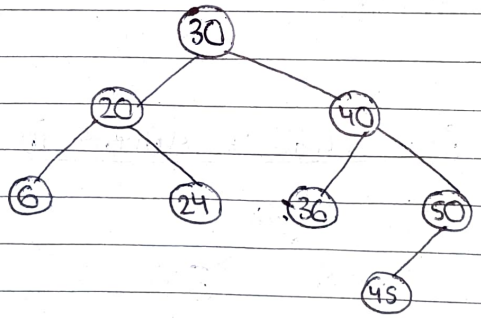
⇒ So, 4 + 8 + 10 = 22

Date _____ Page _____

Code

```
void kSum( Node * root, int & sum,
                        int &k){
    if(!root) return;
    kSum(root → left, sum, k);
    k--;
    if(k >= 0)
        sum += root → data;
    if(k <= 0)
        return;
    kSum(root → right, sum, k);
}
```

* k^{th} Largest Element in BST:



⇒ We know that inorder gives answer in ~~descending order~~ ascending order.

⇒ So, if we reverse L N R to R N L, it will answer in descending order.

Date _____ Page _____

Code

```
void kLargest( Node * root, int &ans,
               int &k){
    if(!root) return;
    kLargest( root->right, ans, k);
    k--;
    if(k==0)
        ans = root->data;
    if( k<=0)
        return;
    kLargest(root->left, ans, k);
}
```

* Array to BST:

nums: | 1 | 2 | 3 | 4 | 5 | 6 |

⇒ We have to create a Balance BST.

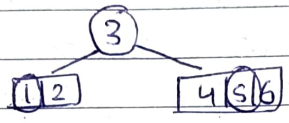
⇒ That means, height of left tree - height of right subtree → (-1, 0, 1)

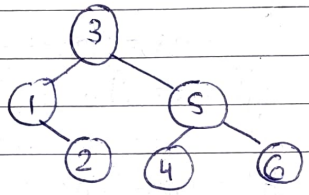
⇒ We have to create a tree then return the preorder that is lexicographically smaller.

⇒ First we select the mid then create node using that node.

Date _____ Page _____

⇒ And remaining left side nodes goes to left side & right side nodes to right side.

⇒ 

⇒ 

⇒ So, we can find preorder without creating the tree.

⇒ Simply add the node data to the ans. & don't create the node.

Code

```
void ArrayToBST( vector<int> arr,
  int start, int end, vector<int> &ans){
      if( start >= end )
          return;
    int mid = start + (end - start)/2;
    ans.push_back( arr[mid] );
    ArrayToBST( arr, start, mid-1, ans);
    ArrayToBST( arr, mid+1, end, ans);
}
```