Inorder predecessor in BST -

BF → store inorder in array & print previous element

optimised ↳

```
Node *getPredecessor (Node * curr)
{
    if (root == NULL)
        return NULL;

    Node *temp = curr->left;
    if (temp == NULL)
        return NULL;
    while (temp->right != NULL)
    {
        temp = temp->right;
    }
    return temp;
}
```

left node doesn't exist

PAGE No.
DATE

Inorder Successor in BST

BF → store inorder in an array & print next element.

Morris Traversal –

```
void Morris Traversal (Node *root)
  {
Node *curr = root;
  while (curr != NULL)
  { if (curr->left == NULL)
    {cout << curr->data;
    curr = curr->right;
    }

  else
    {
      pred = getPred(curr);
      if (pred->right == NULL)
      {
        pred->right = curr;
        curr = curr->left;
      }                                  } creating link

      else if (pred->right != NULL && pred->right
                                        != curr)
      {
        pred->right = NULL;
        cout << curr->data;              } removing link
        curr = curr->right;
      }
```

```
else
{
    Node *ptr = curr->left;
    while( pred->right != NULL && pred != curr)
    {
        pred = pred->right;
    }
    if (pred->right == NULL)
    {
        pred->right = curr;
        curr = curr->left;
    }
    else
    {
        pred->right = NULL;
        cout<<curr->data;
        curr = curr->right;
    }
}
```

find pred

link creation

link remove

Flatten a Binary Tree to Linked List -

```
N      flatten (root)
{
  Node* curr = root;
while(curr != root)
  {
   if (curr->left != NULL)
    {
Node* pred = curr->left;
        while(pred->right != NULL)
                pred = pred->right;
    pred->right = curr->right;
    curr->right = curr->left;
    curr->left = NULL;
    }
  curr = curr->right;
  }
```

find pred

Creating BST -

```
void createBST(Node* &root)
{
 cout << "Enter value for Root Node";
  int data;
  cin >> data;

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

Node* BuildBST(Node* root, int data)
{
 if (root == NULL)
  {
   root Node* temp = new Node(data);
    return temp;
  }
if(data > root → data)
  { root → right = buildBST(root → right,
                            data); }
else
{ root → left = buildBST(root → left,
                         data);
} } return root;
}
```

PAGE No.
DATE

Search an element in a BST -

```
bool search(Node* root, int target)
{
    if(root == NULL)
        return false;

    if(root->data == target)
        return true;

    if(target < root->data)
    {   bool leftAns = search(root->left, target);
        if(leftAns == true)
            return true;
    }

    if(target > root->data)
    {   bool rightAns = search(root->right, target);
        if(rightAns == true)
            return true;
    }
    return false;
}
```

PAGE No.
DATE

Delete element in BST

```
Node* deleteNode(Node*root, int target)
{
if(root == NULL)
    return NULL;

if(root->data == target)
{ // 0 child
if(root->left == NULL && root->right == NULL)
  { delete root;
    return NULL;
  }
  // 1 child
if(root->left != NVLL && root->right == NULL)
  {
    Node* temp = root->left;
    delete root;
    return temp;
  }

if(root->left == NULL && root->right != NULL)
{
  Node* temp = root->right;
  delete root;
  return temp;
}
```

// 2 child

```
if (root->left != NULL && root->right != NUL
{
    int pred = maxValue(root->left)->data;
    root->data = pred;
    root->left = deleteNode(root->left, pred);
    return root;
}
}
else if(target > root->data)
    return(root->right, target);
else
    return (root->left, target);
}
```

PAGE No.
DATE

Check tree is BST/not -

BF - Property → Inorder is sorted.
Take out inorder & check.
O(N) & O(N)

optimised - on basis of range.

BF Code →

```
bool isBSTUtil(root, &prev)
{
  if(root)
  {
    if(!isBST(root->left, prev))
    return false;

    if(prev != NULL && root->data <=
                       prev->data)
    return false;

    prev = root;

    return (isBST(root->right, prev));
  }
  return true;
}
bool isBST(root) {
  *prev = NULL;
  return isBSTUtil(root, prev);
}
```

PAGE No.
DATE

optimised – bool isBST (root, l=NULL, r=NULL)

```
{
    if(!root)
        return true;

    if (l!=NULL && root->data<=l->data
        || r!=NULL && root->data>=r->data)
    return false;

    return isBST(root->left, l, root) &&
           isBST(root->right, root, r);
}
```

PAGE No.
DATE

VVImp

LCA in BST -

```
Node* lca(Node* root, Node *p, Node *q)
{
  //base
  if(root == =NULL)
    return NULL;

if(root -> data < p -> data && 
   root -> data < q->data)
{   //right
  return lca(root -> right, p, q);
  }

if(root -> data > p -> data && 
   root -> data > q->data
{
  return lca(root -> left, p, q);
  }
else
  {
   return root;
    }
  }
```

PAGE No.
DATE

Largest BST -

for this, first we will create a class.

```
class Info
 {
 public:
  int mini;
  int maxi;
  int size;
  bool isBST;

 Info( ) { }
          {

 Info( int a, int b, int c, bool d)
    {
       mini = a;
       maxi = b;
       size = c
       isBST = d;
    }
 };
 Node* build
```

```
Node* solve ( Node* root, int & ans)
   { if (root == NULL)
      Info Node* temp = n.
        return (INTMIN, INTMAX, 0, true);
```

```
Info leftAns = solve(root->left, ans);
Info RightAns = solve(" " ->right, ans);
Info curr;

curr.size = leftAns.size + rightAns.size + 1;
curr.mini = min(leftAns.mini, root->data);
curr.maxi = max(rightAns.maxi, root->data);

if(leftAns.isBST && rightAns.BST &&
   root->data > left.maxi && root->data
   < right.mini)
{
    curr.isBST = true;
}
else
    curr.isBST = false;

if (curr.isBST)
{
    ans = max(ans, curr.size);
}
return curr;
```