Day:

Date:

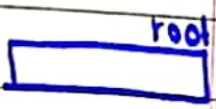
①

Name: Sadia Javed
Roll No: Sp22-BCS-113
Section: BCS-B
Submitted to: Mam Yasmeen Jana
Assignment NO: 4

. Preorder traversal:

(go to main function)

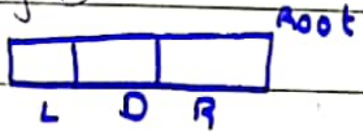
struct node *root = new node(1) →



∴ go to the newNode() function:

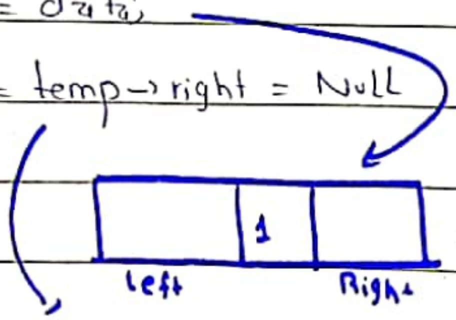
Node *temp = new Node;

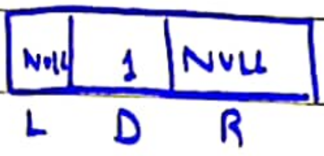
∴ go to struct Node{ and int data;
int data;
struct Node *left, *right;
}



∴ go back to new Node (int data)

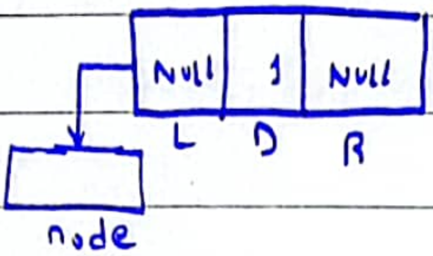
temp->data = data;
temp->left = temp->right = NULL
return temp;





∴ go back to the main function

root→left = new Node(2)



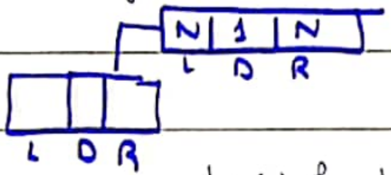
∴ Call new node() function

Node *temp = new Node;

∴ Call struct node {

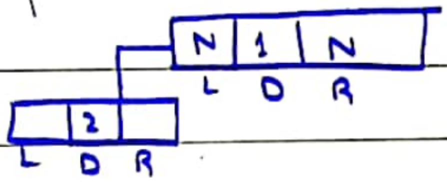
int data;

struct Node *left, *right;

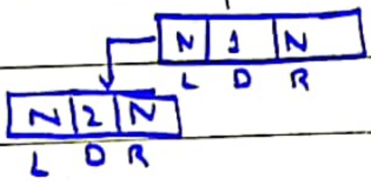


∴ go back to the new Node() function

temp→data = data;



temp→left = temp→right = NULL;

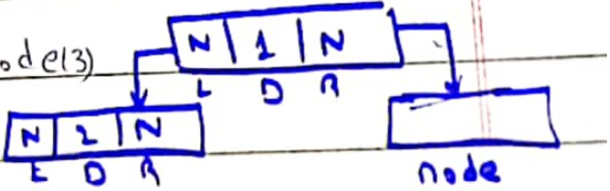


return temp;

}

∴ go again to main function;

temp→right = new Node(3)

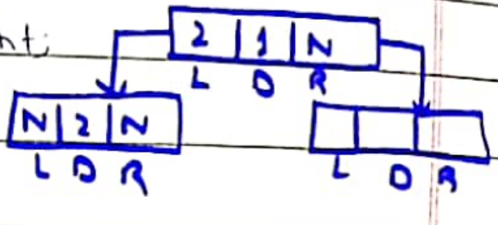


∴ call newnode function

```
Node *temp = new Node;
```

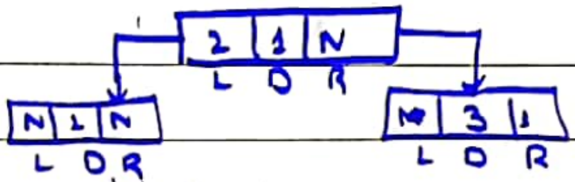
∴ call struct node function

```
Struct node{
  int data;
  struct node *left, *right;
  }
```

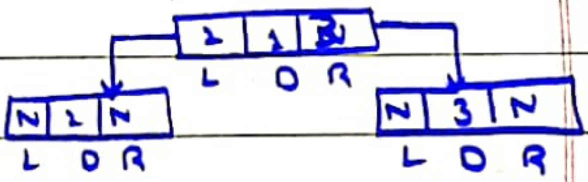


∴ go back to newnode() function

```
temp->data = data
```

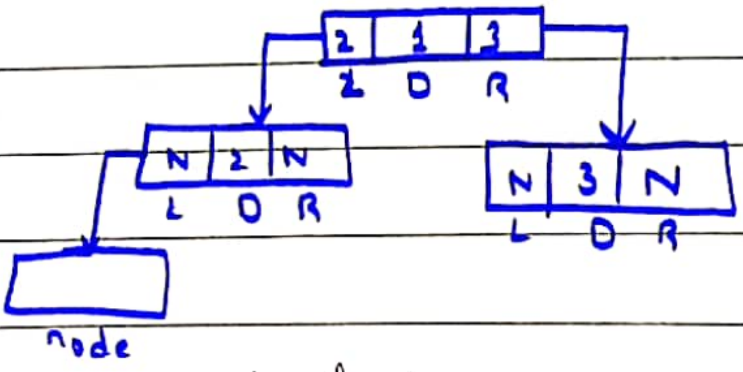


```
temp->left = temp->right = Null;
  return temp;
```



∴ go back to main function;

```
root->left->left = new Node(4)
```



∴ call newNode function

```
Node * temp = new Node;
```

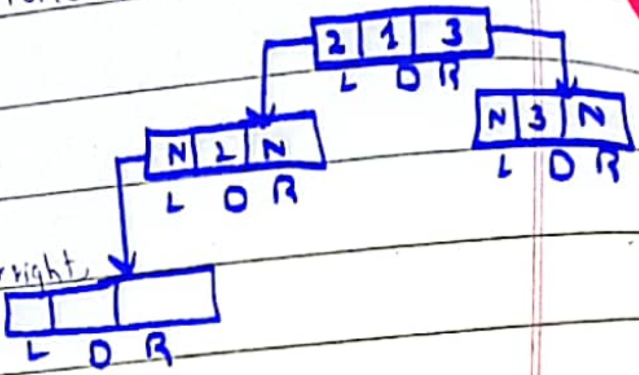
∴ call struct Node function

struct node{

int data;

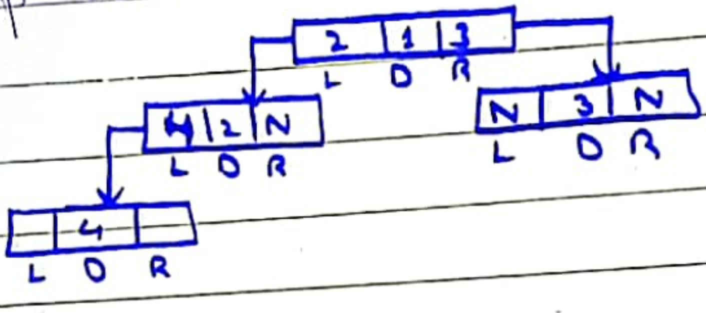
struct node *left,*right;

}



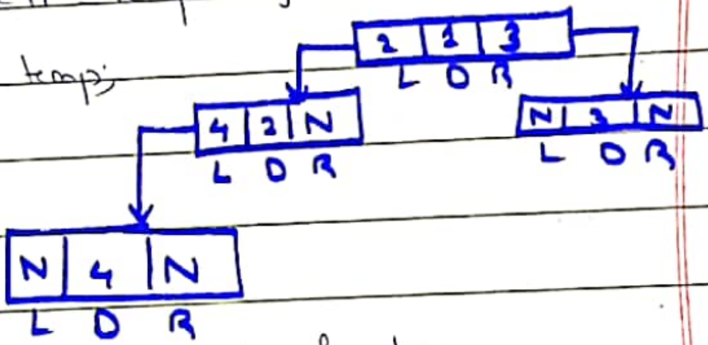
∴ go back to new Node function

temp->data = data



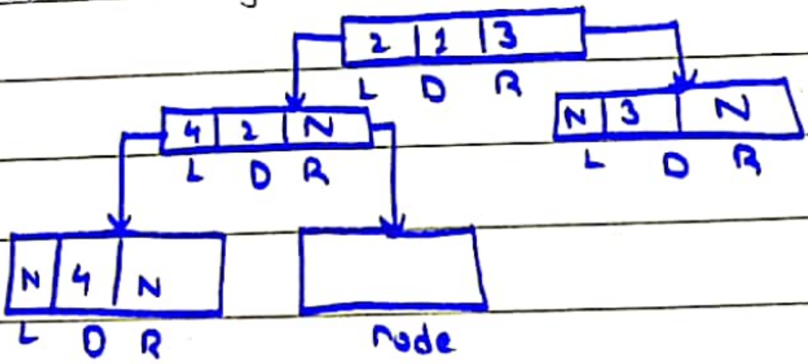
temp->left = temp->right = NULL;

return temp;



∴ go back to main function

root->left->right = new Node(5)



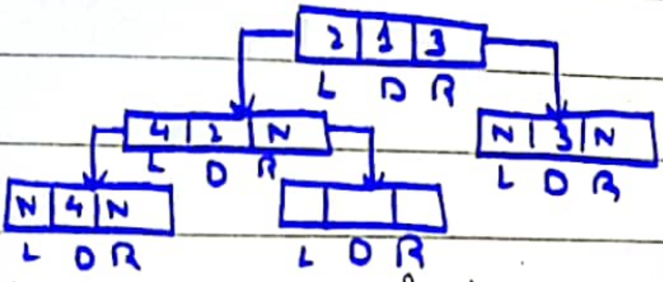
∴ call new Node function

Node *temp= new Node;

∴ go to struct node function{

int data;

struct node *left, *right;

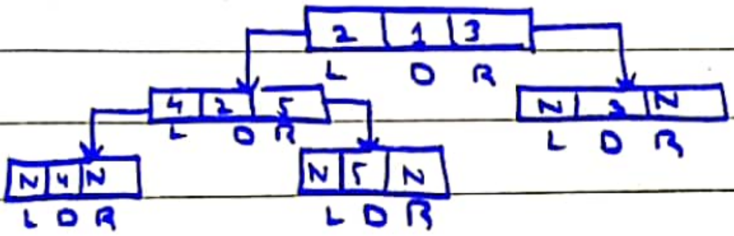


∴ go back to the new Node function

temp-> data= data;

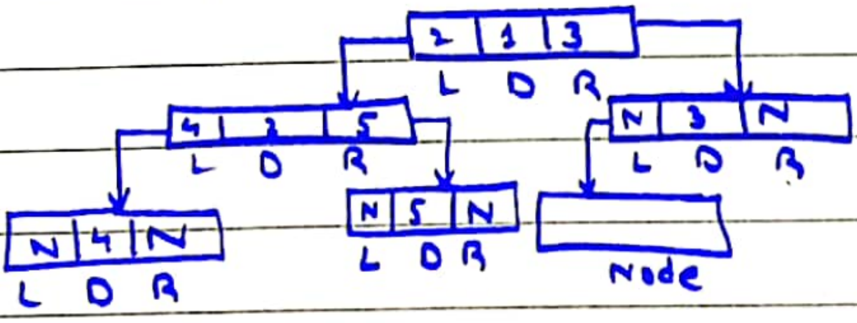
temp->left = temp->right = NULL;

return temp;



∴ go back to the main function

root->right->left = newNode(6)



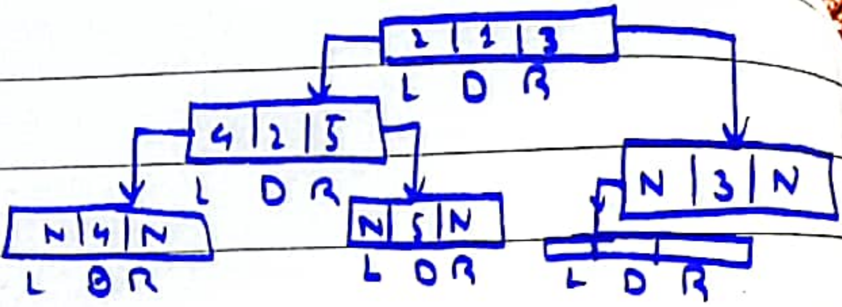
∴ call new Node function

Node* temp= new Node;

∴ go to struct node {

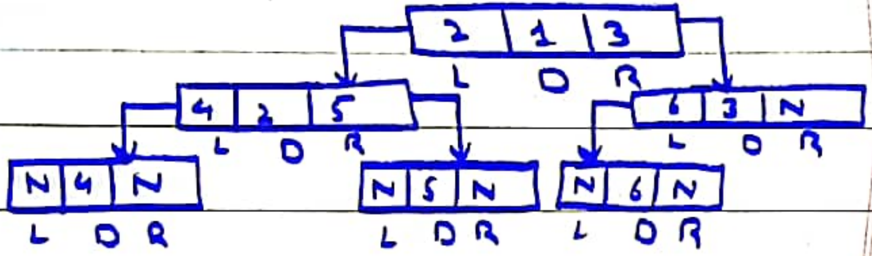
int data;

```
Struct node* left, *right;
```



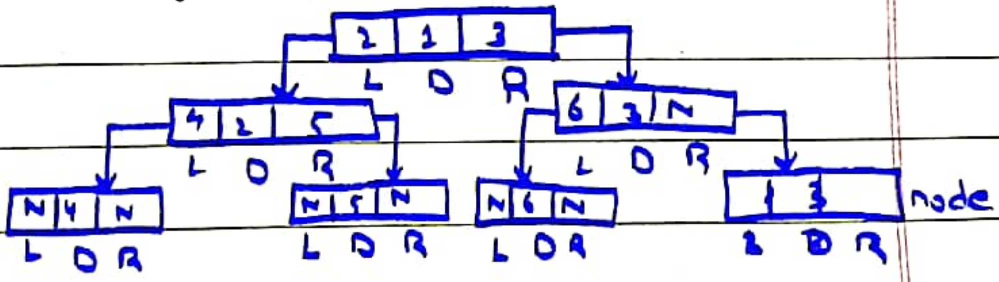
∴ Go back to the new Node function.

```
temp-> data = data;
temp->left = temp->right = NULL;
return temp;
```



∴ go back to the main function

```
root->right->right = new Node(7)
```

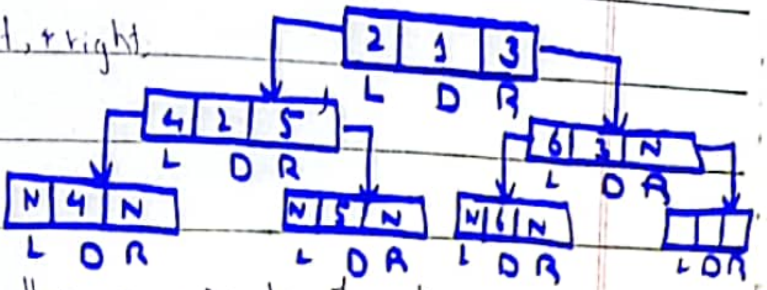


∴ Go back to the $\overset{x}{\text{the}}$ new Node function

```
temp->data = data
temp->left = temp->right = NULL,
return temp;
```

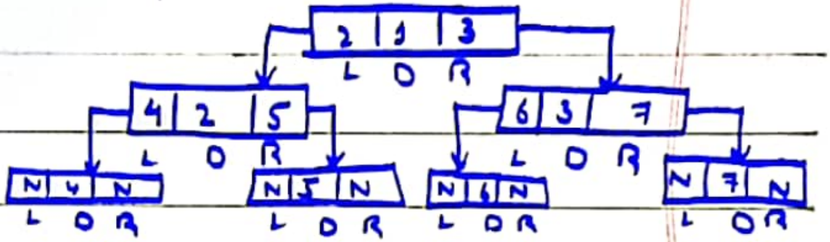
Date:

```
go to struct node { function
    int data;
    struct node *left, *right;
```



∴ go back to the new Node function

```
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}
```



∴ Now go to the main function

```
print preorder(root)
```

∴ go to the print pre-order function

```
    if (root == NULL) { -> false
                   1
        return;
    }
             cout<<node->data;
    print preorder(root->left)
    print preorder(root->right)
             cout<<node->data;
    print 1

print pre-order(root->left) 2
```

∴ go to if condition

if($\overset{2}{\text{root}}$==Null) → false

{ return;

}

Cout<< node→data;　　　　1 2

print 2

print preorder (root→left) → 4

cout<<node→data

print 4　　　　1 2 4

∴ go back to the root 2 &

go to its right

print preorder (root→right) → 5

cout<<node→data

print 5　　　　1 2 4 5

∴ Now go back to the root 1 and

then go to its right

print preorder (root→right) → 3

cout<<node→data

print 3　　　　1 2 45 3

∴ go back to the left side of root 3

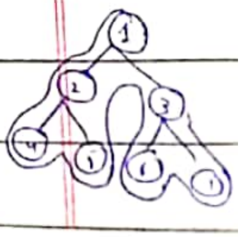
print preorder (root→left) → 6

print 6　　　　1 2 4 5 3 6

∴ Go to the right side of root 3

print preorder(root→right) → 7

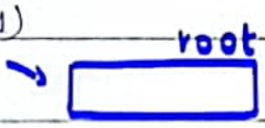
print 7　　　　1 2 4 5 3 6 7

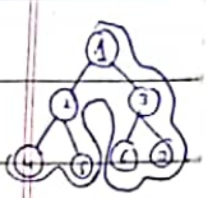


∴ Post traversal :

Go to main function.

struct node *root = new node(1)





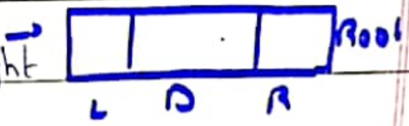
∴ go to the newNode() function

Node *temp = new Node;

∴ Go to struct Node{

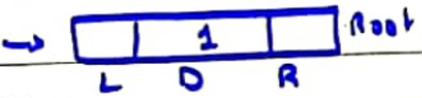
int data;

struct Node *left, *right



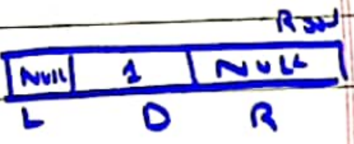
∴ Go back to newNode (int data)

temp->data = data;



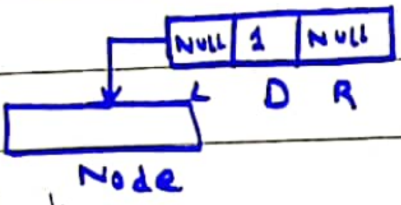
temp->left = temp->right = NULL;

return temp;



∴ Go back to the main function;

root->left = new Node (2)



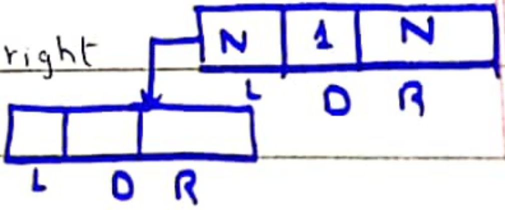
∴ Call newnode() function

Node * temp = new Node;

∴ Call struct node {

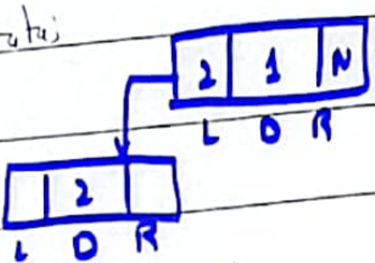
int data;

struct Node *left, *right



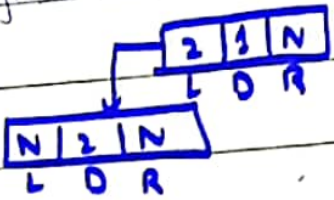
∴ go back to the new Node() function;

temp→data = data;



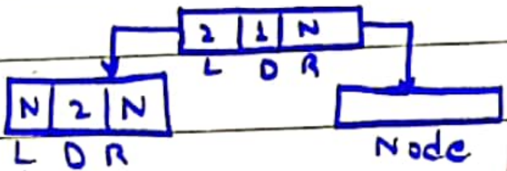
temp→left = temp→right = Null;

return temp;



∴ go again to main function

temp→right = new Node(3)

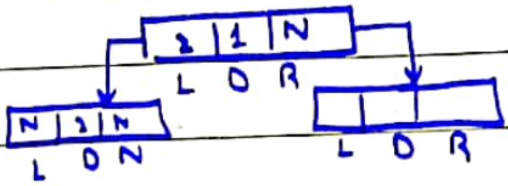


∴ Call new node function

Node *temp = new Node,

∴ Call struct node function

```
struct node{
int data,
struct node *left, *right;
}
```



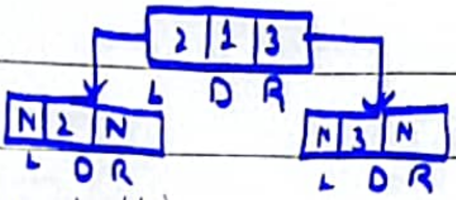
∴ Go back to new Node() function

temp→data = data;

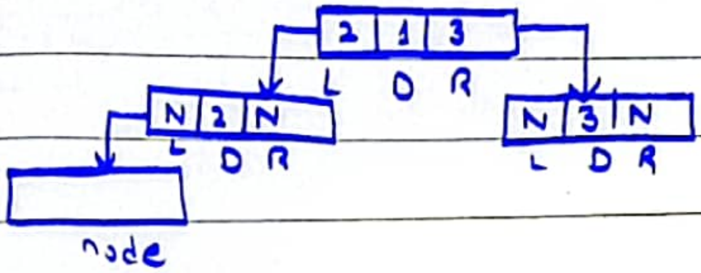
temp→left = temp→right = Null

return temp;

∴ go back to main function;



root→left→left = new Node (4)



∴ call new Node function

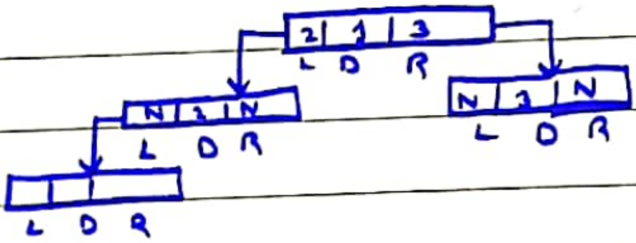
Node* temp = new Node;

∴ call struct Node function

struct node{

int data;

struct node * left, * right

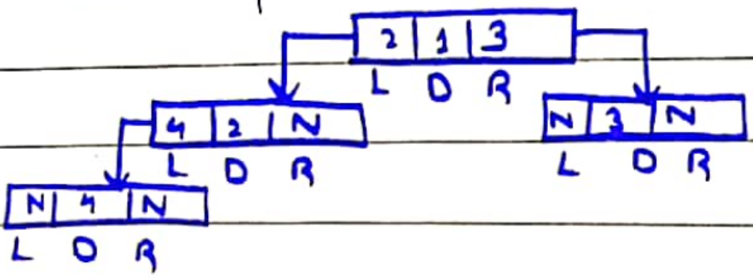


∴ Go back to new Node function

temp→data = data

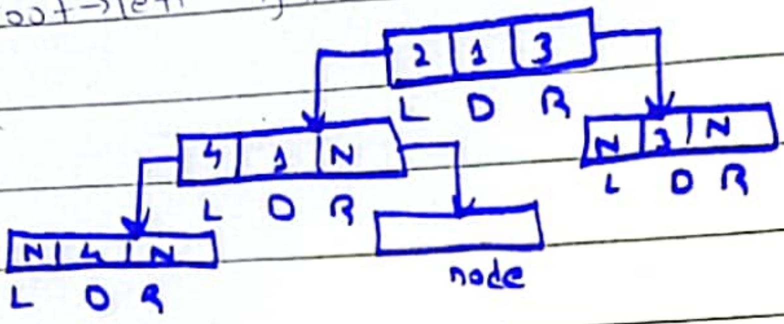
temp→left = temp→right = Null;

return temp;



Now go back to main function

root->left->right = new Node(5)



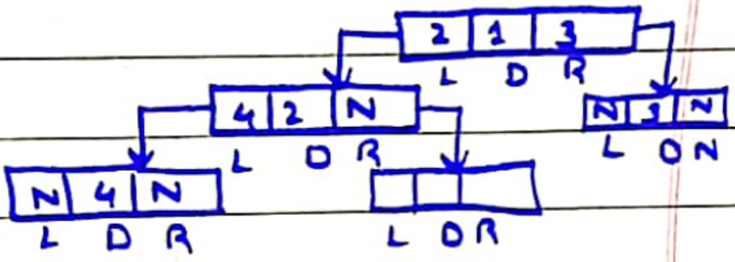
∴ Call new Node function

Node *temp = new Node;

∴ go to struct node function

int data

struct node *left, *right;

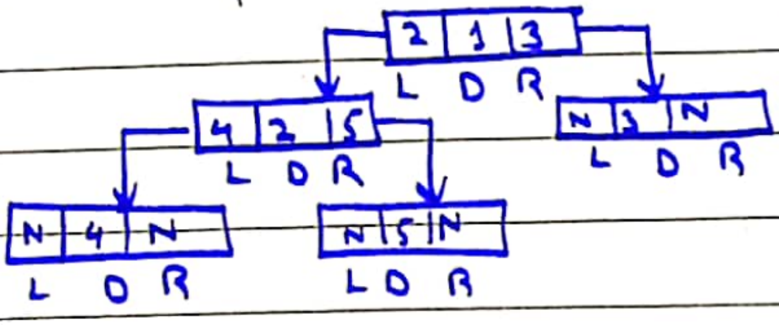


∴ go back to the new Node function

temp->data = data;

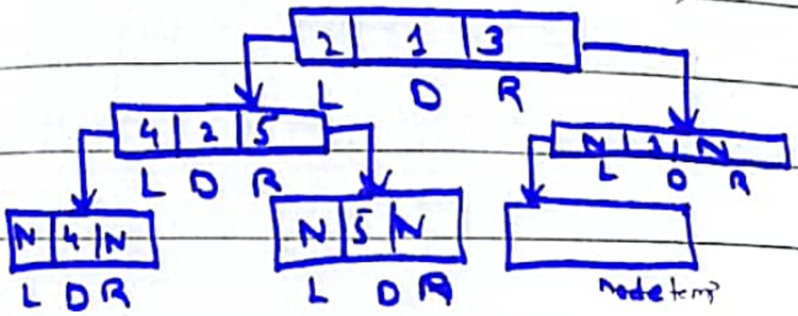
temp->left = temp->right = Null;

return temp;



∴ Again go back to the main function.

root→right→left = new Node(1)



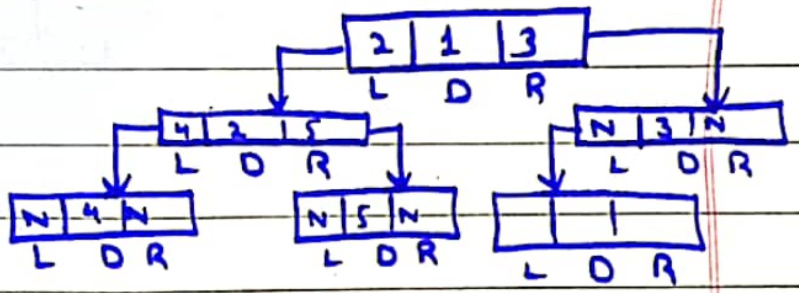
∴ Call new Node function

Node *temp = new Node;

go to struct node function

int data;

struct node *left, *right;

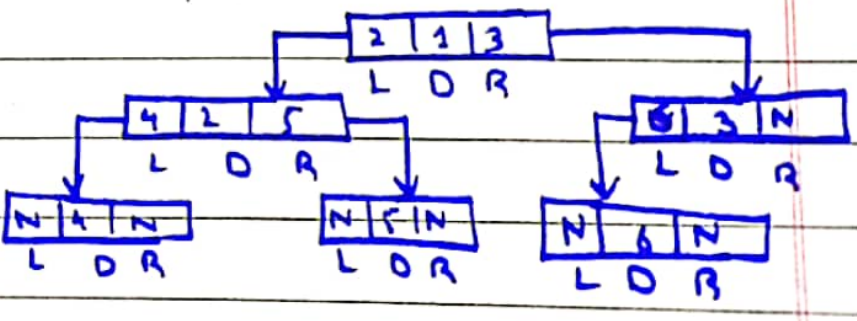


∴ go back to the new node function

temp → data = data;

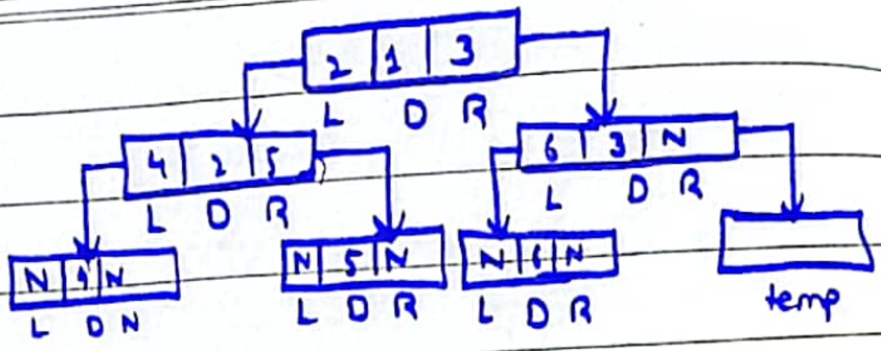
temp → left = temp → right = Null;

return temp;



∴ go back to the main function

root→right→right = new Node(7)



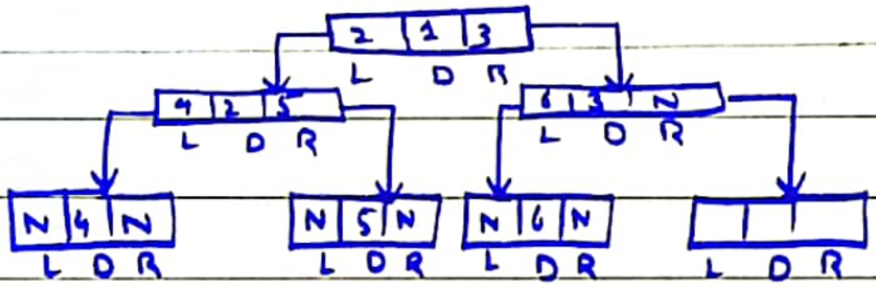
∴ call new Node function

Node * temp = new Node;

∴ go to struct Node function

int data;

Struct node * left, *right;



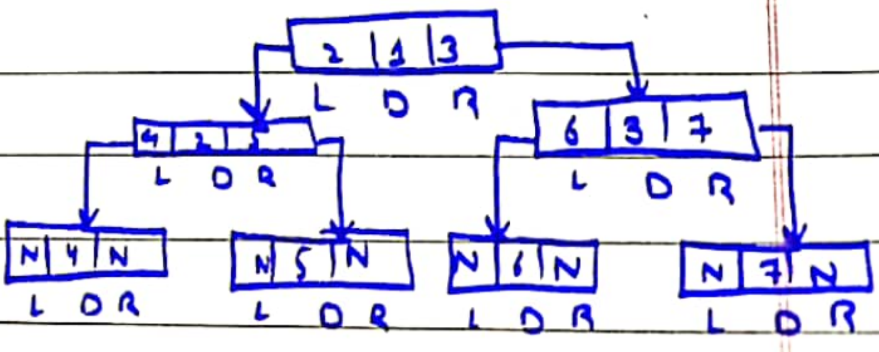
∴ Now go to the new node function

temp→data = data;

temp→left = temp→right = NULL;

return temp;

}



∴ Go back to the main function

print ~~pre~~postorder(root);

. go to the print ~~pre~~Post order function

if ($\overset{1}{\text{root}}$ == NULL) → false

{ return root;

}

print postorder (root → left) → 2

∴ Again go to if condition

if ($\overset{2}{\text{root}}$ == NULL) → false

{

return root;

}

print postorder (root → ~~right~~ left) → 4

print 4

∴ Back to the previous call

print postorder (root → right) → 5

So there is no child of 5

So print 5

∴ Now go back to the root 2

print the data of the current node

print 2

∴ Back to the main root → 1

∴ Now call Print postorder

print postorder (root → right) → 3

∴ go to the print postorder

print postorder (root → left) → 6

So there is no child of 6

print 6.

∴ Back to the previous call.

print postorder(root→right) → **7**

So there is no child of 7

print 7

Now go back to the root 3.

print 3.

Now go back to the original root.

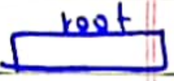
print 1.

The final output is: **4 5 2 6 7 3 1**

Inorder Traversal:

∴ Go to main function:

Struct node *root = newnode(1) →



∴ go to s new node function

struct node *new Node (int $\overset{1}{\text{data}}$){

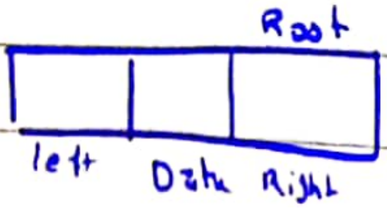
struct node *temp = new struct node;

∴ go to struct part

struct node {

int data;

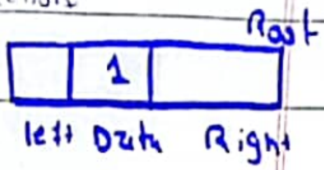
struct node *left, *right;



}

Now go to the new node function

~~Node~~ ~~temp~~ node→data = $\overset{1}{\text{data}}$;

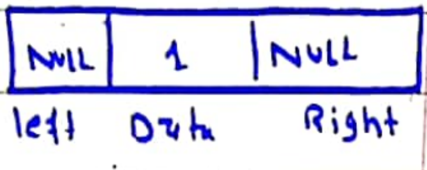


~~temp~~ node →left = Null;

~~temp~~ node →right = Null;

return ~~temp~~ node;

}



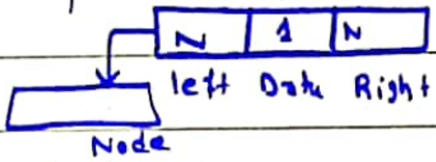
∴ Again go to main function

root→left = new Node(2);

∴ go to new Node function

struct node * new Node (int $\overset{2}{\text{data}}$){

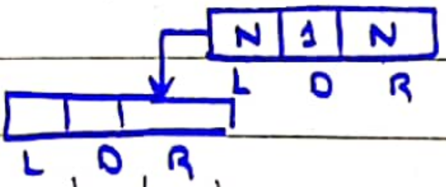
struct temp *node *temp = new struct node;



∴ go to struct part

int data;

Struct node *left, *right;



∴ go to the new node function;

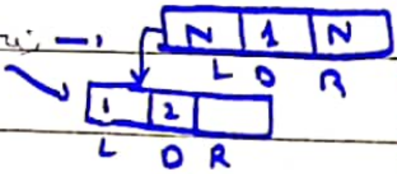
Node→data = $\overset{2}{\text{data}}$; →

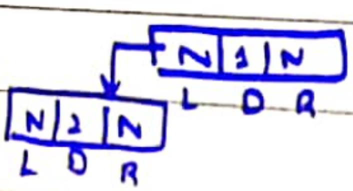
Node→left = NULL;

Node→right = Null;

return node; →

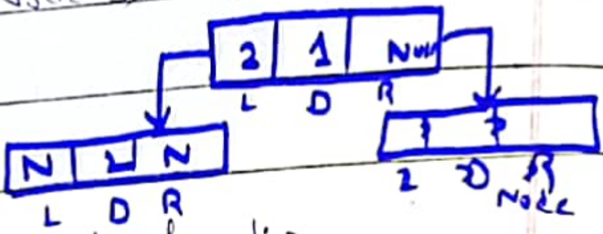
}





∴ Again go back to the main function

root→right = new Node(3)



∴ Go back to new Node function.

struct node *new node(int $\overset{3}{data}$){

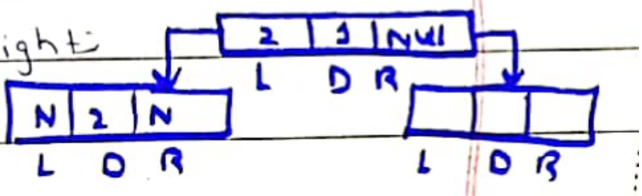
struct node * ~~temp~~ node = new struct node

∴ go to struct part

int data;

struct Node *left;

struct Node * right;



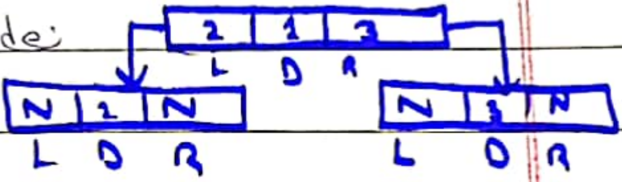
Go back to new Node() function

Node→data = data;

Node→left = Null;

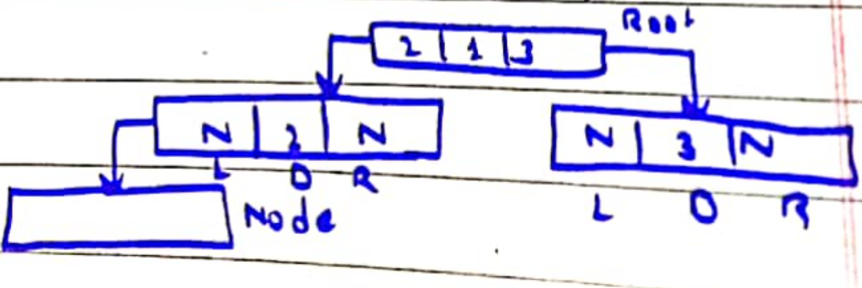
Node→right = Null;

return Node;



∴ Again go back to main function

root→left→left = new Node(4)



move to new node function.

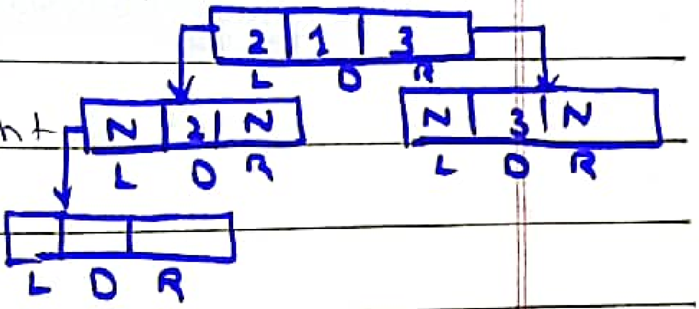
struct node * new node (int $\overset{4}{\text{data}}$){

Struct Node * Node = new struct node;

∴ move to the struct part.

int data;

Struct node * left, right



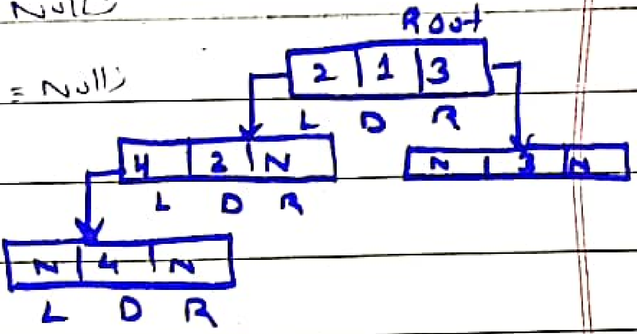
∴ Now go back to newNode

Node→data = Null;

Node→right = Null;

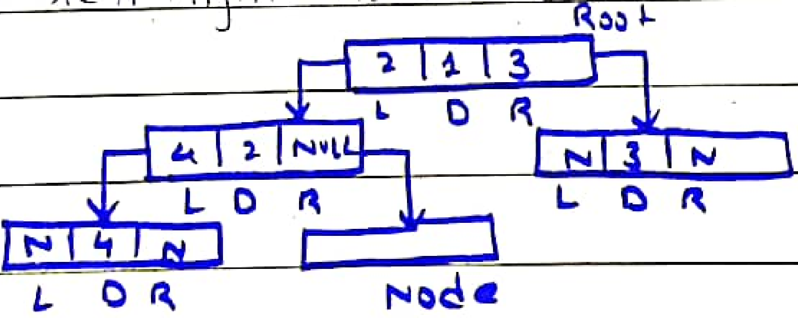
return Node;

}



Again move back to main function

root→left→right = new Node(5);



∴ Go to new Node function

Struct node * new Node (int $\overset{5}{\text{val}}$)

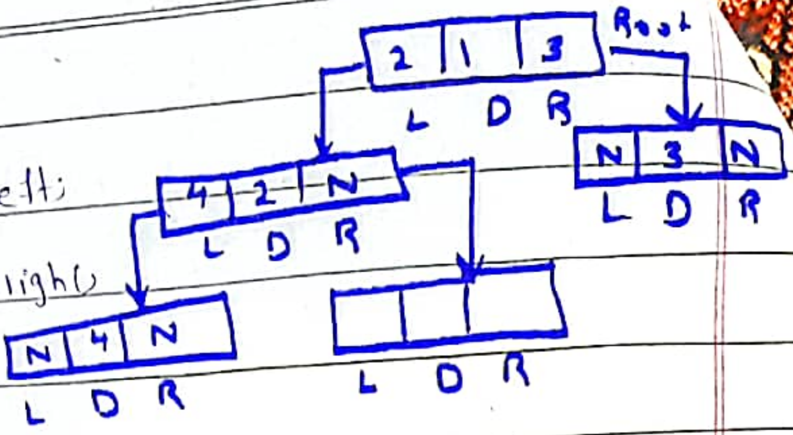
Struct node *node = new struct node;

move to struct part

int data;

struct node *left;

struct node *right;



Now go to newNode()

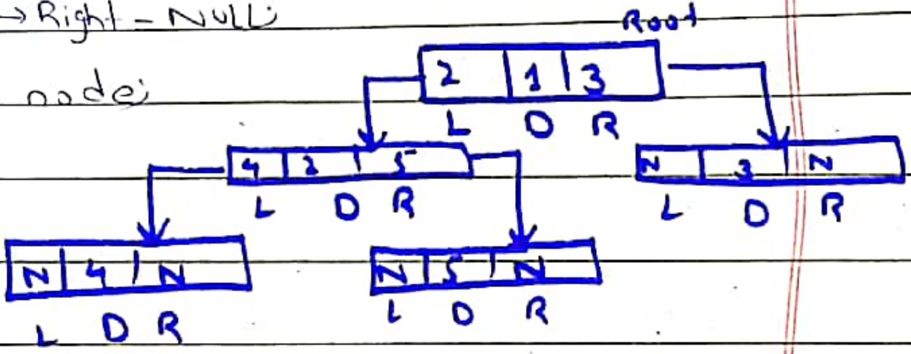
Node→data = data;

Node→left = NULL;

Node→Right = NULL;

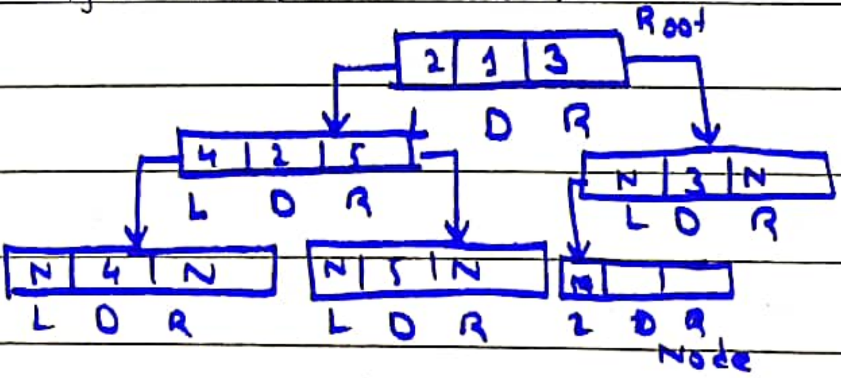
return node;

}



Again go back to main function

root→right→left = newNode(6)



Go to newNode function

struct node *newNode(int data)

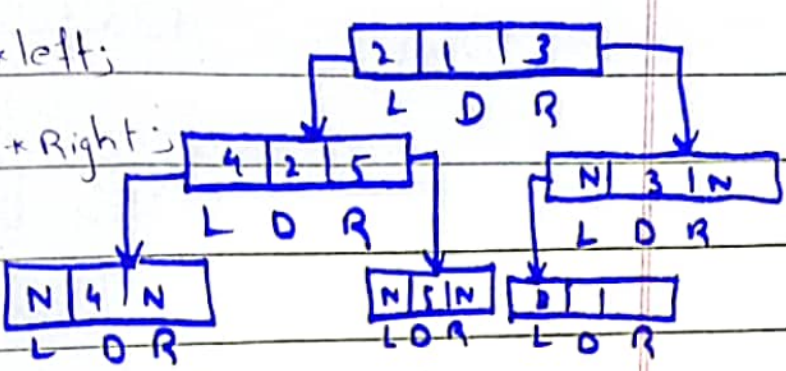
struct node *node = new struct node()

∴ move to struct part

int data;

struct node *left;

struct node *Right;



Go to new Node function;

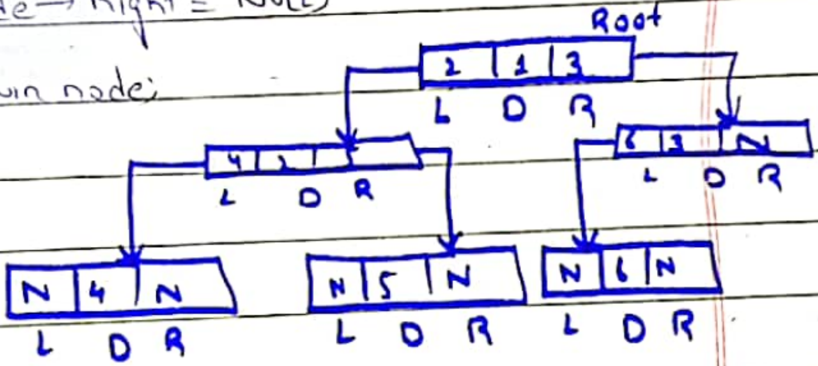
Node → left = data;

Node → left = Null;

Node → Right = Null;

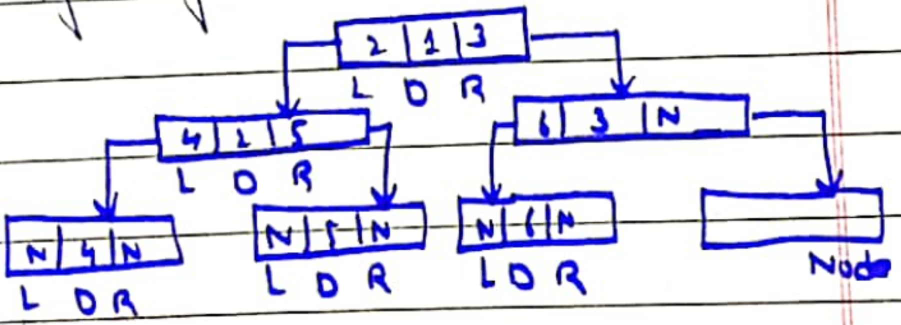
return node;

}



Now go back to main function

root → right → right = new Node (7)



Now go back to new Node function

struct node *new Node (int $\overset{7}{\text{data}}$){

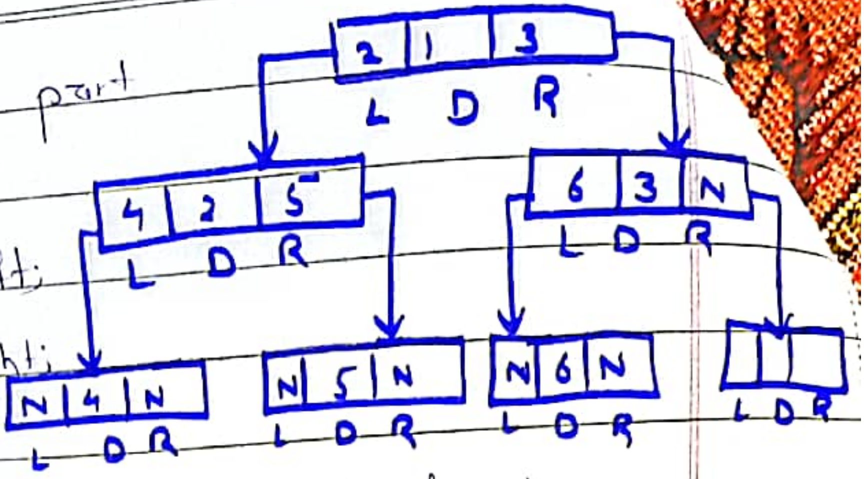
struct node * Node = new struct node ();

∴ move to struct part

int data;

struct node *left;

Struct node *right;



Now go back to new Node function

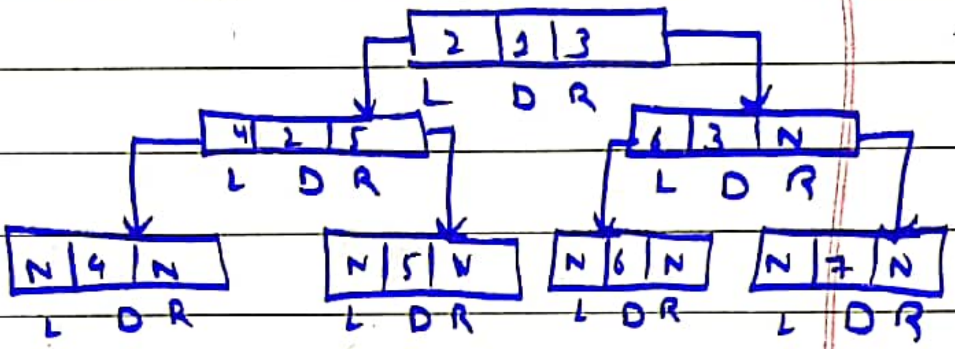
node→data = data;

Node→left = NULL;

Node→right = NULL;

return Node;

}



Traverse Indorder(root)

∴ go to the traverse Indororder function

if (root == NULL) → false

{ return root;

}

Call traverse Inorder(root→left) → 2

∴ Go to the left of the root 2.

Traverse Inorder (root → left) → 4

print 4

Date:

4 2

∴ Go back to the previous node & print **2**.
2 and go to its right side.
traverse Inorder (root→right) → **5**

Cout << root → data;

print 5   **4 2 5**

∴ Back to the root 1

Cout << root → data → **1**

print 1   **4 2 5 1**

∴ Go to the right child & then move to left child.
Its data is **6**.

print 6   **4 2 5 1 6**

∴ Go to the right child whose element or data is 7 but first we print the root 3.

Cout << root → data → **3**

print 3   **4 2 5 1 6 3**

∴ Go to its right child & print 7.

print 7   **4 2 5 1 6 3 7**

output is:

4
2
5
1
6
3
7