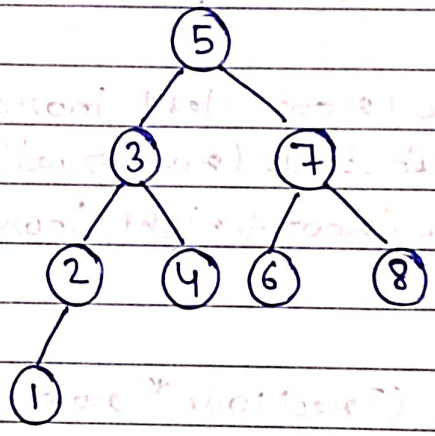
13/12/2023
Wednesday

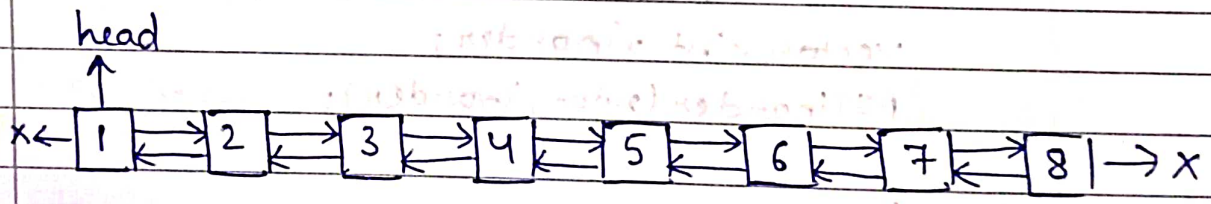
* **Convert Binary Search Tree To Doubly linked list -**

Given a BST, convert it into DLL in-place. The left and right pointers in nodes are to be used as previous & next pointers respectively. The order of nodes in DLL must be same as inorder of given BST. The first node of inorder traversal must be the head node of the DLL.

I/P -



O/P -



Approach -

Reverse inorder traversal krenge i.e. RNL. By doing this, at the last head leftmost node of BST pr hi point kr rha hoga or koi extra processing bhi nhi lgegi. If inorder traversal krenge i.e. LNR, then usme kuch extra steps lgenge so that head leftmost node pr point kre.

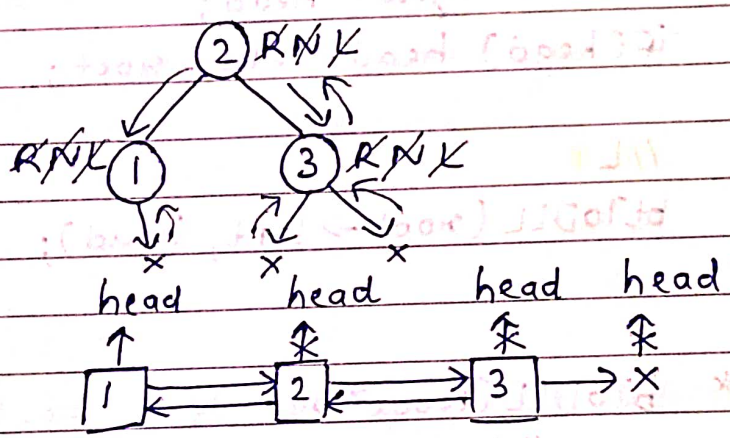
Step 1 - Recursive call for right subtree.

Step 2 - Process current node.

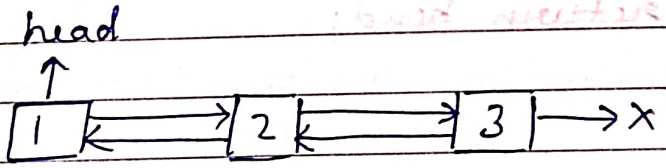
- root -> right = head ;
- if (head) head -> left = root ;
- head = root ;

Step 3 - Recursive call for left subtree.

for instance,



So, the DLL formed is -



Code-

```
class Solution{
public:

void  btToDLL(Node* root, Node* &head){

        if(!root) return;

        //R
        btToDLL(root->right, head);

        //N
        root->right = head;
        if(head) head->left = root;

        //L
        btToDLL(root->left, head);
    }

Node*  bToDLL(Node* root){
        Node* head = NULL;
        btToDLL(root, head);
        return head;
    }
};
```

→ Agr inorder traversal se kre i.e. LNR

Code-

```
class Solution{
public:
void btToDLL(Node* root, Node* &head, Node* &temp){

    if(!root) return;

    //L
    btToDLL(root->left, head, temp);

    //N
    if(!head) head= root;
    if(temp) temp->right = root;
    root->left = temp;
    temp = root;

    //R
    btToDLL(root->right, head, temp);
  }

Node* bToDLL(Node* root){
      Node* head = NULL;
      Node* temp = NULL;
      btToDLL(root, head, temp);
      return head;
    }
};
```

Note- In dono code mei hum btToDLL() function mei root ko by reference bhi pass kr skte hai but fir hume ek helper use krna pdega like temporary variable so that original tree effect na ho or segfault na aaye.

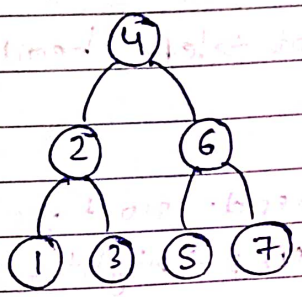
__/__/__

* **Sorted linked list to BST –**

Given a singly linked list which has data members sorted in ascending order. Construct a balanced BST which has same data members as the given linked list.

I/P – Linked list = 

O/P –



Approach –

Base case – if head is null or length of linked list <=0, return NULL.

// left subtree

Make recursive call for left subtree. As LL is in sorted order, that's why mid node of LL is the root node of BST i.e. at every call in left subtree, the length of LL gets reduced to 1/2 everytime.

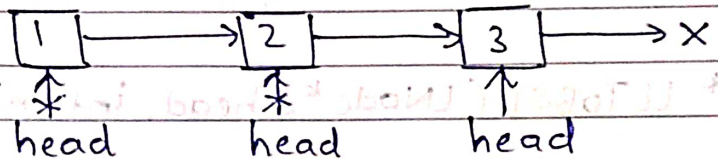
// current node

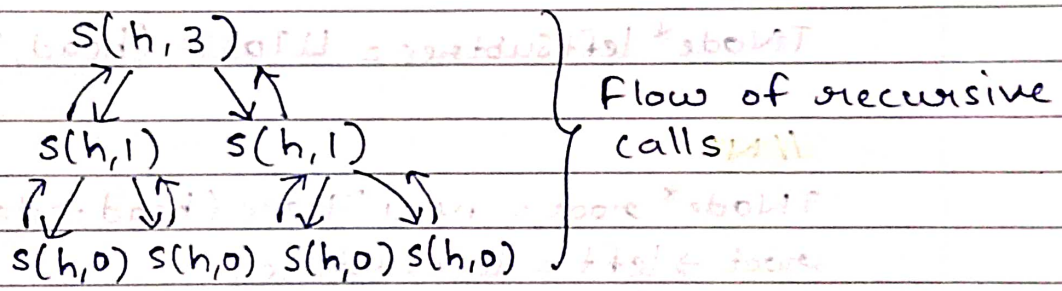
- Create root which consists head->data.
- root -> left = leftSubtree.
- head = head->next.

//right subtree

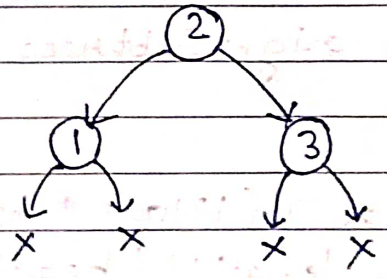
Make recursive call for right subtree. As left subtree ($n/2$) part and root node i.e. $\left(\frac{n}{2}+1\right)$ part of n is processed. Remaining is $n-\left(\frac{n}{2}+1\right)$.

For instance,





BST —



→ This same code will work for doubly linked list also.

Code-

```
class Solution{
public:
int lengthLL (LNode * &head ){
        LNode * temp= head;
        int len= 0;
        while (temp){
                len++;
                temp= temp ->next;
              }
        return len;
      }

TNode* LLToBST. (LNode*&head, int n) {

      if (head== NULL || n<=0 ) return NULL;
      // L
      TNode* leftSubtree = LLTOBST (head, n/2);

      // N
      TNode* root = new TNode (head ->data);
      root -> left = leftSubtree;
      head= head -> next;

      // R
      TNode * rightSubtree = LLTOBST (head, n-n/2-1);
      root -> right = rightSubtree;
      return root;
  }
TNode * sortedlistToBST ( LNode * head){
        int length= lengthLL ( head);
        return LLTOBST (head, length);
      }
};
```