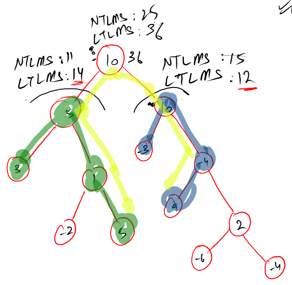
```java
public static class Pair{
    int NTLMS,LTLMS;
    Pair(int a,int b){
        NTLMS = a;
        LTLMS = b;
    }
}
public static int maxPathSum(TreeNode root) {
  return maxPathSumHelper(root).LTLMS;
}
public static Pair maxPathSumHelper(TreeNode node) {
  if(node == null){
      return new Pair(0,0);
  }

  Pair lpair = maxPathSumHelper(node.left);
  Pair rpair = maxPathSumHelper(node.right);

  int NTLMS = Math.max(lpair.NTLMS,rpair.NTLMS)+node.val;
  int LTLMS = Math.max(lpair.LTLMS,rpair.LTLMS);
  if(node.left != null && node.right != null){
      LTLMS = Math.max( LTLMS , lpair.NTLMS+node.val+rpair.NTLMS );
  }

  return new Pair(NTLMS,LTLMS);
}
```



Pair

NTLMaxSum

LTL MaxSum

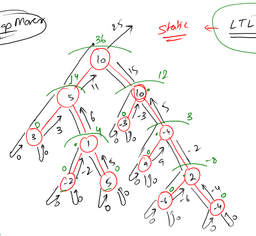
NTLMS = Max(L.NTLMS, R.NTLMS) + node.val

LTLMS = Max of:
- R.LTLMS ✓
- L.LTLMS ✓
- L.NTLMS + node.val + R.NTLMS

Heap Mover

static ← LTL Max Sum = ~~-∞~~ ~~14~~ 36

faith: NTL Max Sum



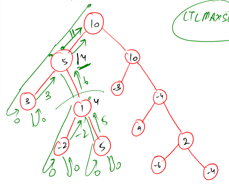
Node
↳ LTLMaxSum ⇒ left.NTLMaxSum + Right.NTLMaxSum + node.val

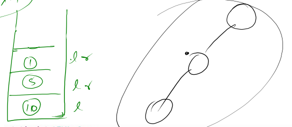
→ 2child

→ 1child

→ leaf → 0

LTLMAXSUM = ~~-∞~~ ~~4~~ 14





```java
static int LTLMaxSum;
public static int maxPathSum(TreeNode root) {
  LTLMaxSum = Integer.MIN_VALUE;
  maxPathSumHelper(root);
  return LTLMaxSum;
}
public static int maxPathSumHelper(TreeNode node){
    if(node == null){
        return 0;
    }
    int leftNTLMaxSum = maxPathSumHelper(node.left);
    int rightNTLMaxSum = maxPathSumHelper(node.right);
    if(node.left != null && node.right != null){
      LTLMaxSum = Math.max(LTLMaxSum, leftNTLMaxSum+node.val+rightNTLMaxSum );
    }
    return Math.max(leftNTLMaxSum,rightNTLMaxSum)+node.val;
}
```

Diameter → Max Distance Between Any two Nodes

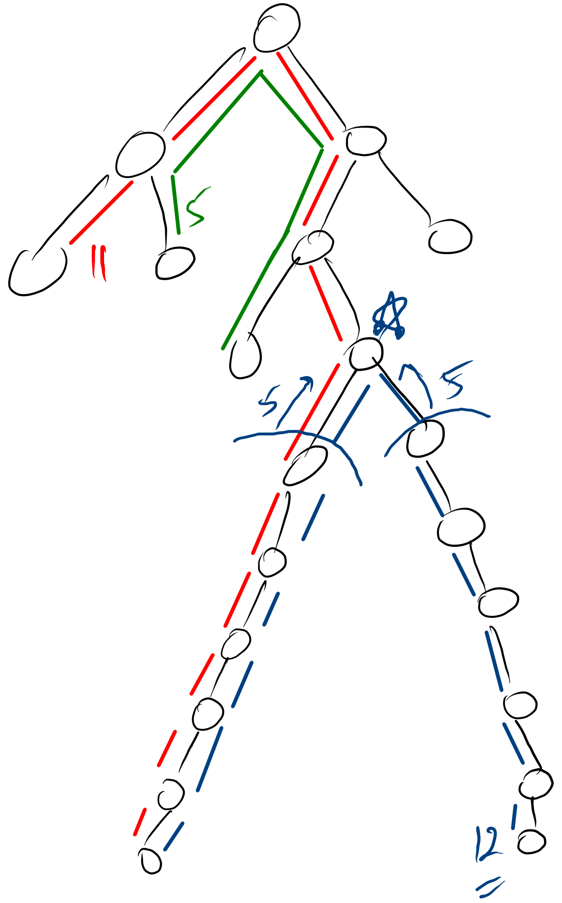
↳ Edge

H.W.

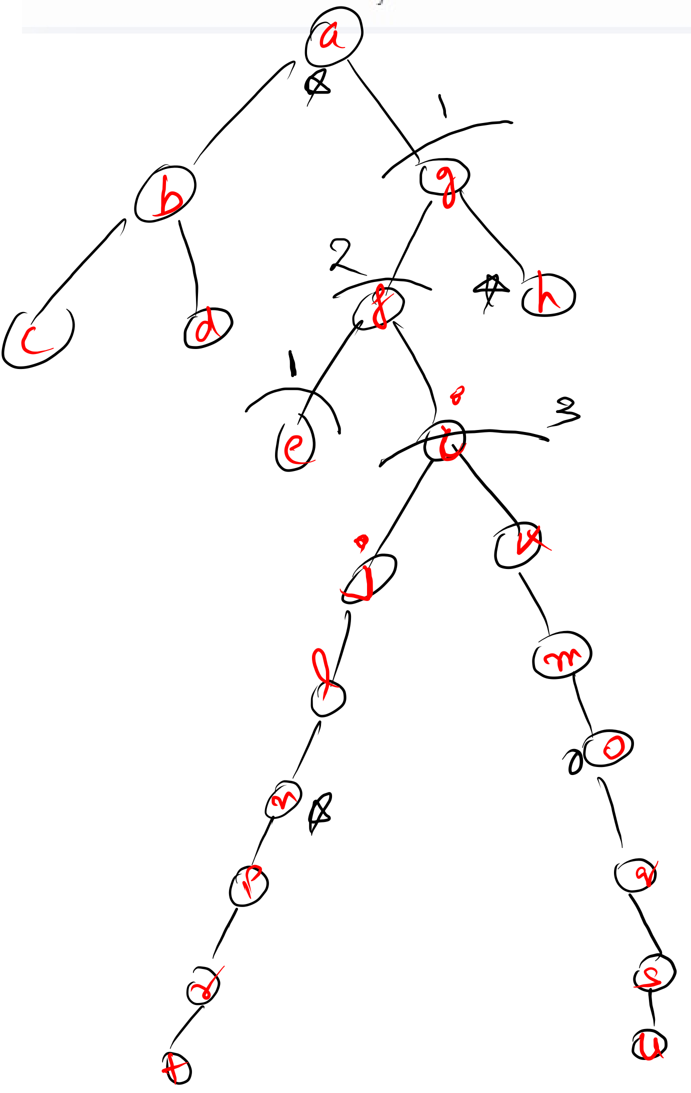
① HeapMover (static)

② Pair → HT
→ dia of Tree

Dia of Node → LHt + RHt + 2
→ 12



1. Given the root of a binary tree.
2. The value of a target node target, and an integer k.
3. You have return an array of the values of all nodes that have a distance k from the target node.



Distance

H.W.

Distance K way

Target = i

Distance → 3

① Node To root →

0	1	2	3
i	f	g	a

+

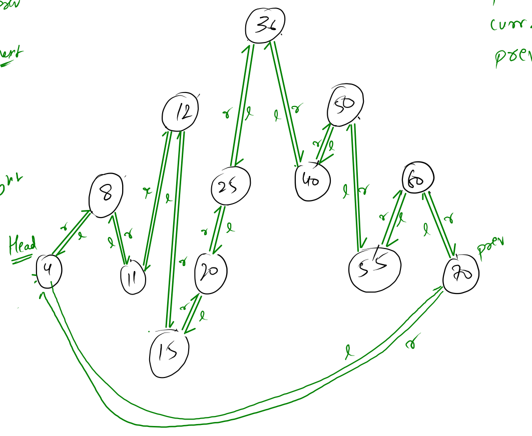
Print K levels down

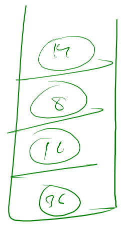
✓ n o h a

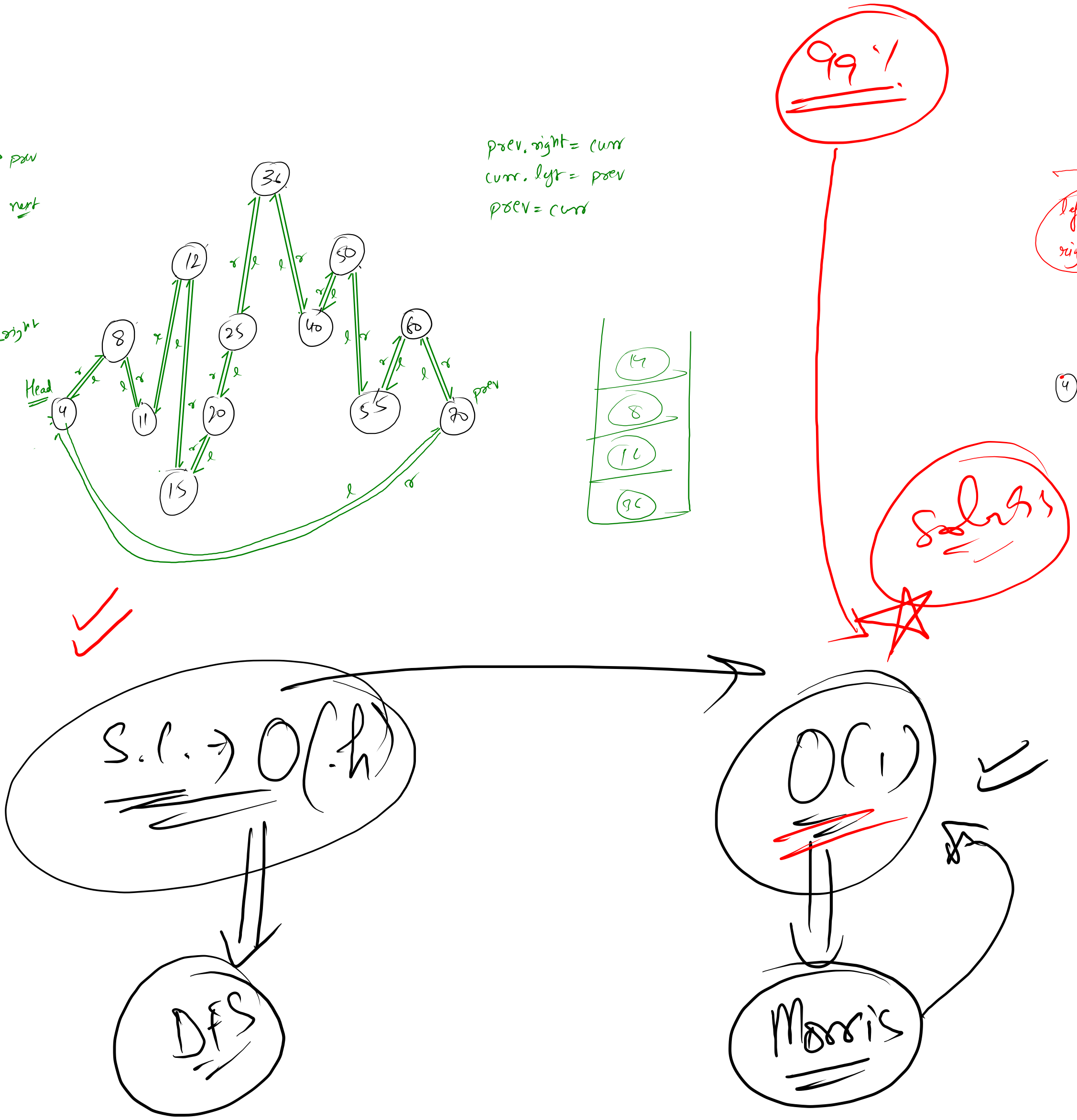
Left → prev
right → next

Head → Dummy.right

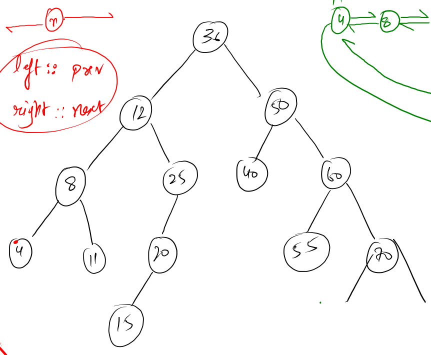
prev.right = curr
curr.left = prev
prev = curr

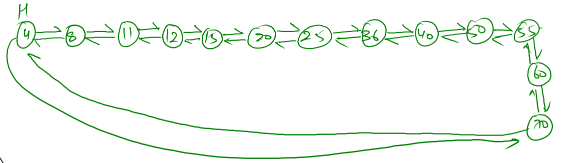


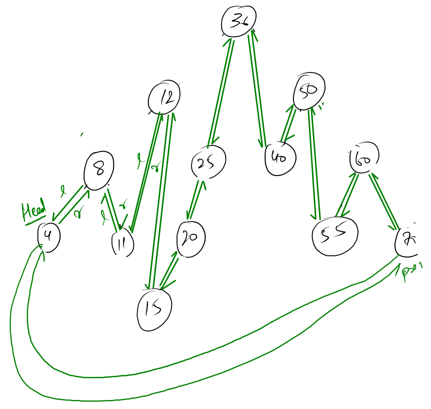




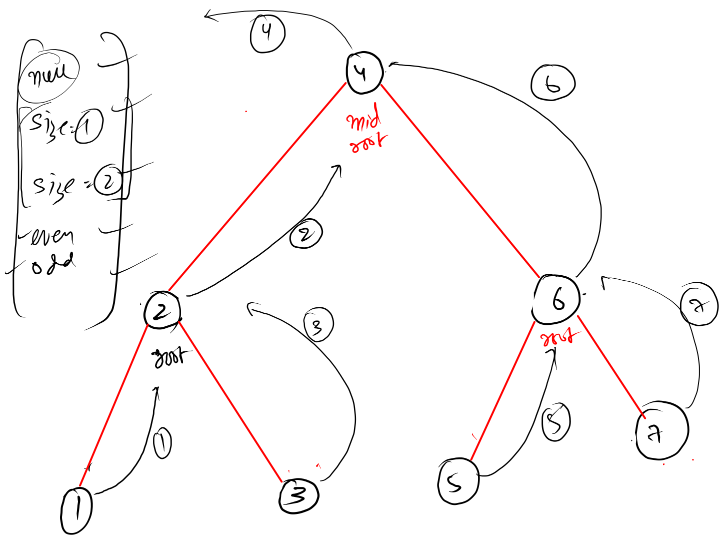
left :: prev
right :: next

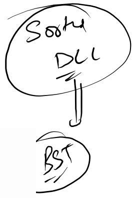


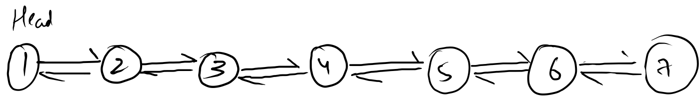


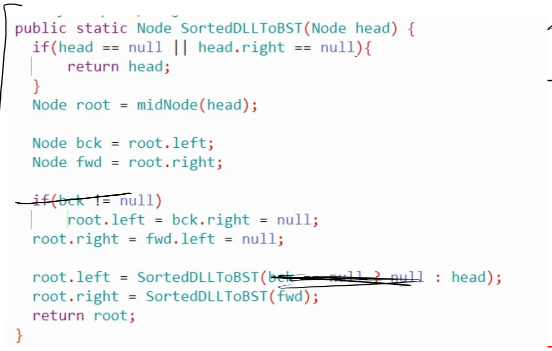


```java
public static Node bToDLL(Node root) {
  Stack<Node> st = new Stack<>();
  getAllLeftNodes(root,st);

  Node dummy = new Node(-1);
  Node prev = dummy;

  while(st.size() > 0){
      Node curr = st.pop();
      curr.left = prev;
      prev.right = curr;
      prev = curr;

      getAllLeftNodes(curr.right,st);
  }

  Node head = dummy.right;
  head.left = null;
  dummy.right = null;

  // circular
  head.left = prev;
  prev.right = head;

  return head;
}
public static void getAllLeftNodes(Node curr,Stack<Node> st){
    while(curr != null){
        st.push(curr);
        curr = curr.left;
    }
}
```



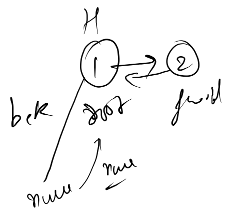


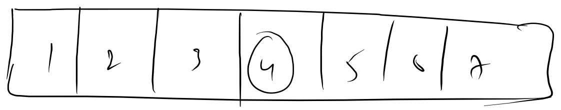


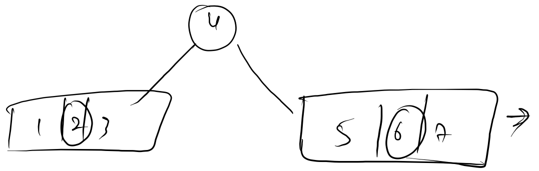

```
public static Node SortedDLLToBST(Node head) {
  if(head == null || head.right == null){
      return head;
  }
  Node root = midNode(head);

  Node bck = root.left;
  Node fwd = root.right;

  ~~if(bck~~ != null)
      root.left = bck.right = null;
  root.right = fwd.left = null;

  root.left = SortedDLLToBST(~~bck == null ? null~~ : head);
  root.right = SortedDLLToBST(fwd);
  return root;
}
```

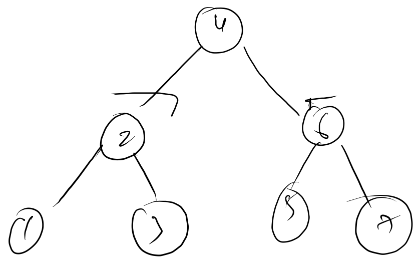
left :: prev
right :: next

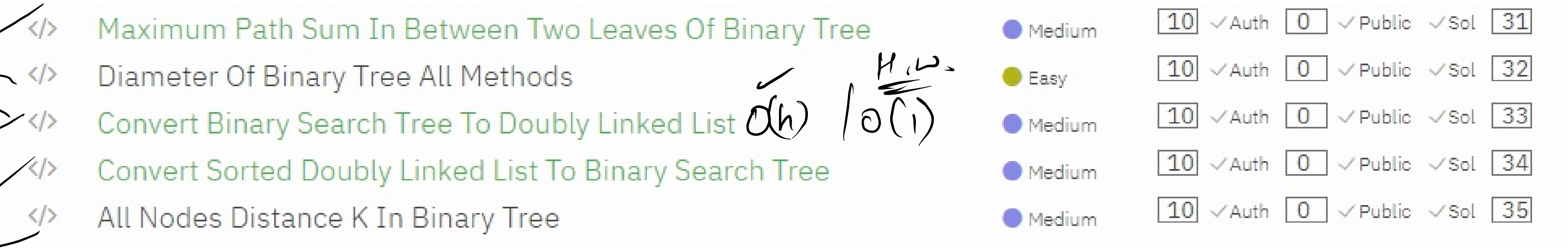
mid = slow











</>	Maximum Path Sum In Between Two Leaves Of Binary Tree	● Medium	10	✓Auth	0	✓Public	✓Sol	31
</>	Diameter Of Binary Tree All Methods	● Easy	10	✓Auth	0	✓Public	✓Sol	32
</>	Convert Binary Search Tree To Doubly Linked List	● Medium	10	✓Auth	0	✓Public	✓Sol	33
</>	Convert Sorted Doubly Linked List To Binary Search Tree	● Medium	10	✓Auth	0	✓Public	✓Sol	34
</>	All Nodes Distance K In Binary Tree	● Medium	10	✓Auth	0	✓Public	✓Sol	35

H.W.

$O(h)$ | $\overset{\text{H.W.}}{O(1)}$

~~Thurs~~ $\xrightarrow{\text{shifted}}$ Sunday

S

Tue → Sunday

7:00 - 09:00

Tue → K.V.=1