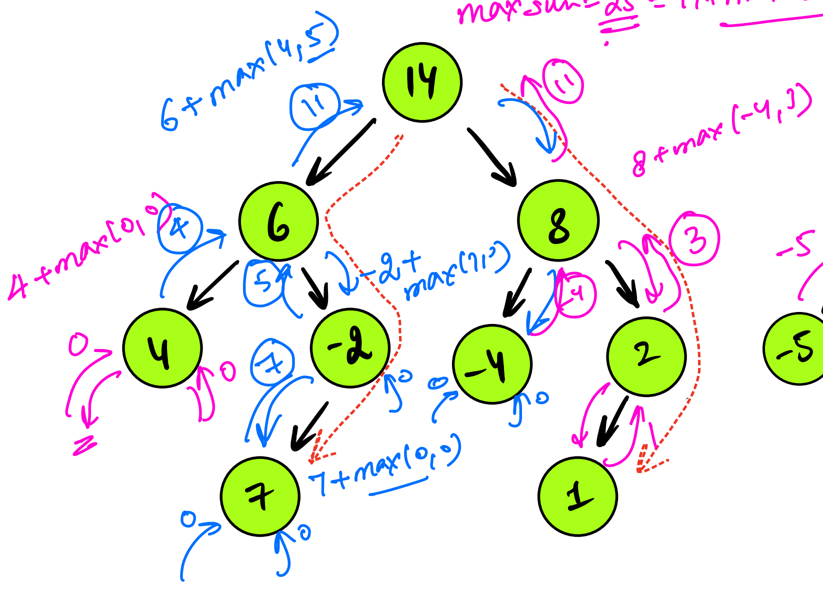
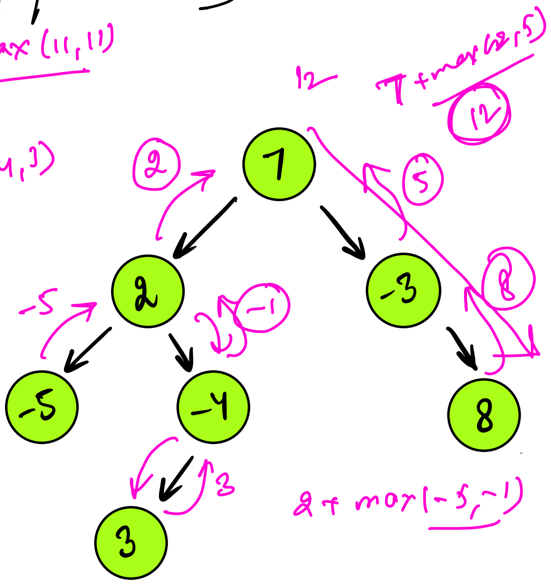
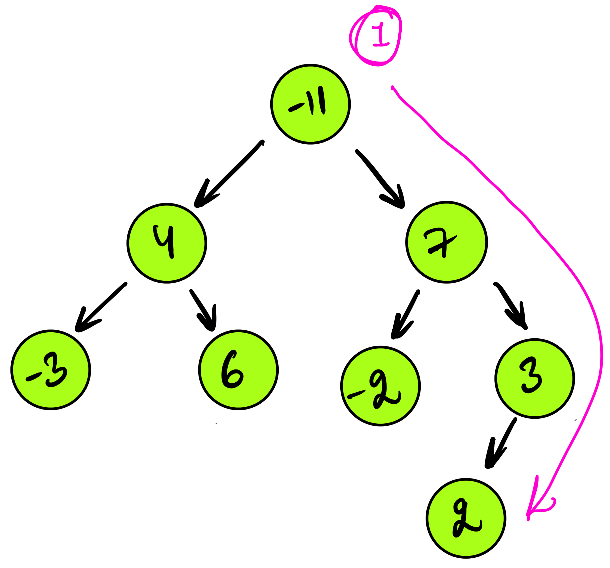
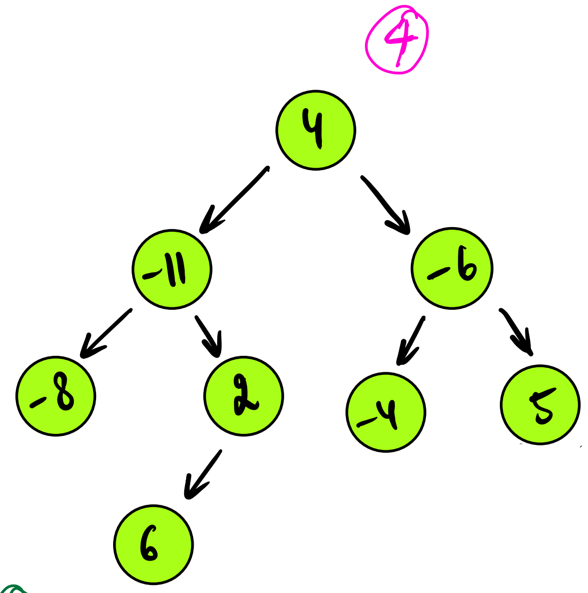
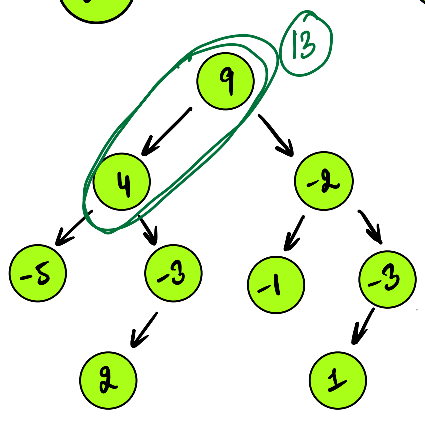
Q Find max path seen starting from root → max (sum of nodes)







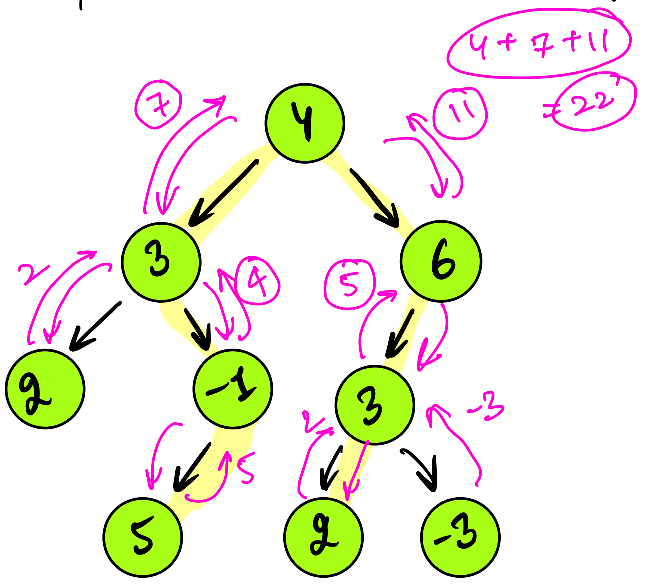


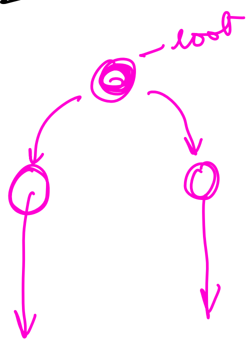


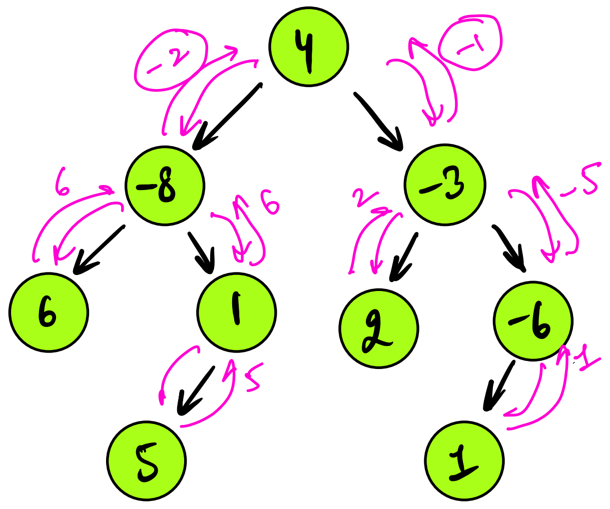
root.data + max(LST' value , RST' value , 0);

```
int   maxpathsum( Node root)
 {
        if( root = = null) return 0;

        int l  = maxpathsum( root.left);
        int r =  maxpathsum( root.right);

        return    root.data  +  max( l, r, 0);
 }
```

• max path sum across/contain the root







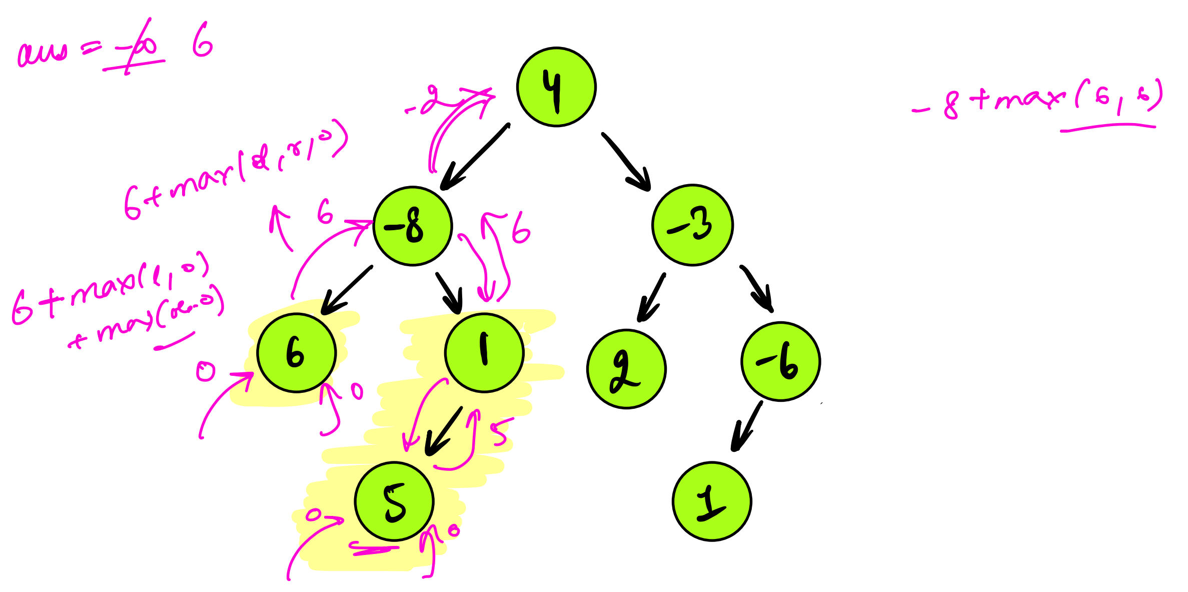
4 + max(l, 0) + max(r, 0)

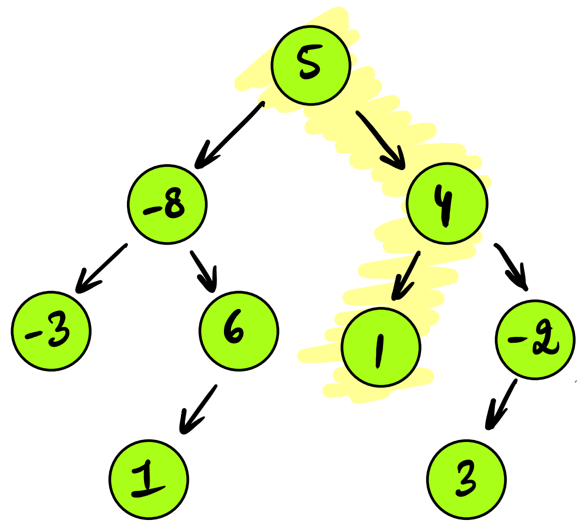
max(l, r, 0)

$\downarrow$

max(max(l, r), 0)

• Maximum path sum going through any node

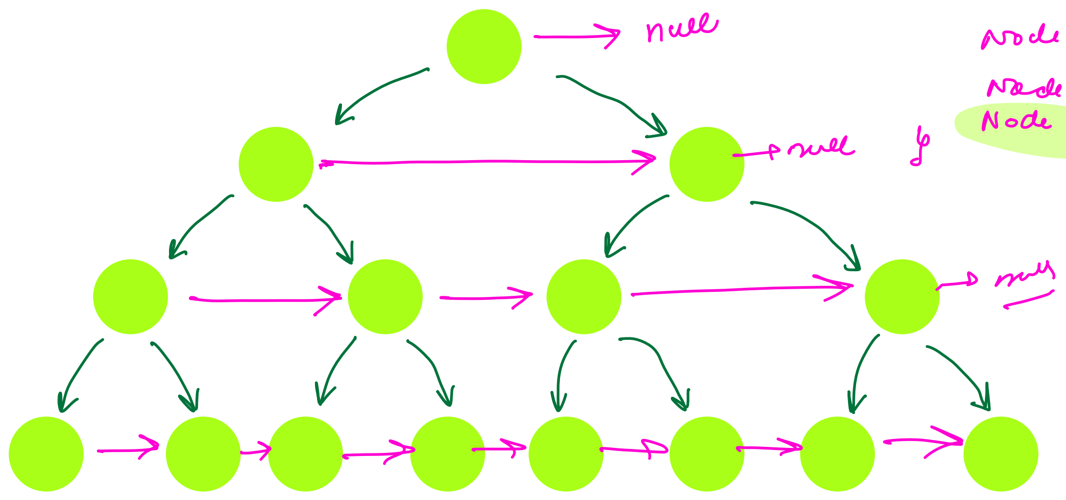




```
ans = -∞
int    maxpathsum( Node root)
 {
        if( root = = null) return 0;

        int l  = maxpathsum( root.left);
        int r =  maxpathsum( root.right);
        ans = max( ans, root.data + max(l,0) + max(r,0));
        return    root.data  +  max( l, r, 0);
 }
```

Q. Population Right Next pointers. perfect

class Node{
int data;
Node left;
Node right;
Node next;
} null

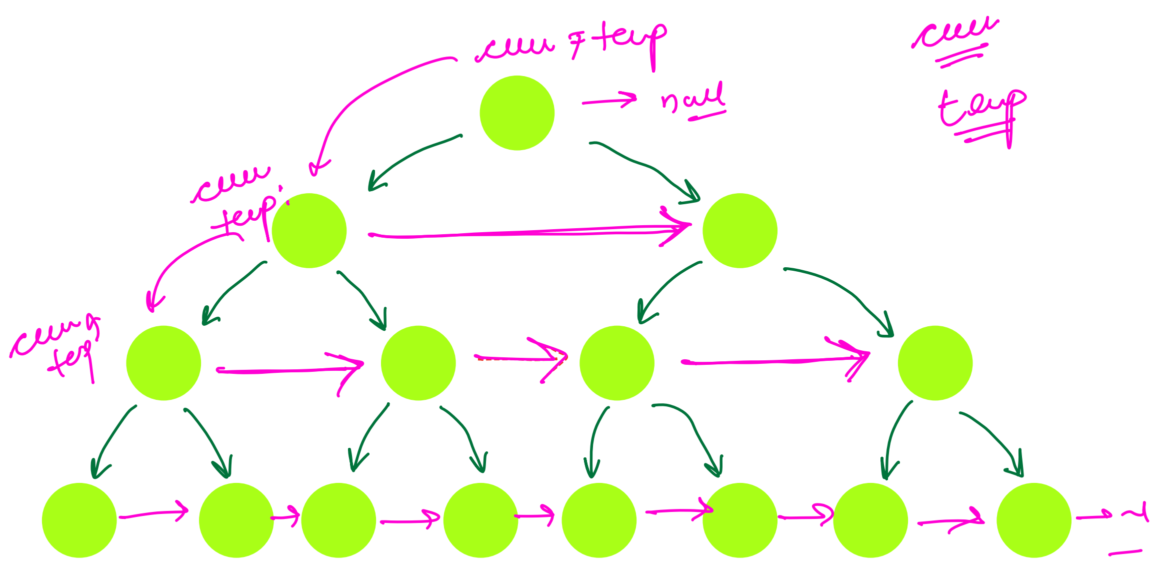


level order traversal (by adding null)

temp.next = q.front()

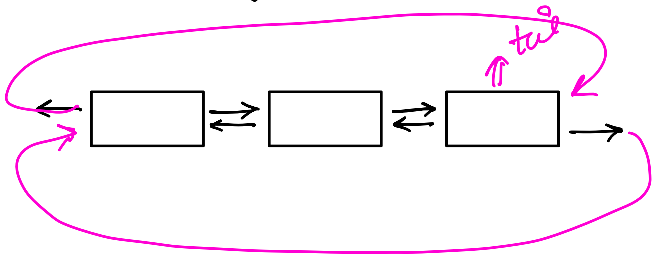
S.C = O(1)

S.C = O(max node in a level)



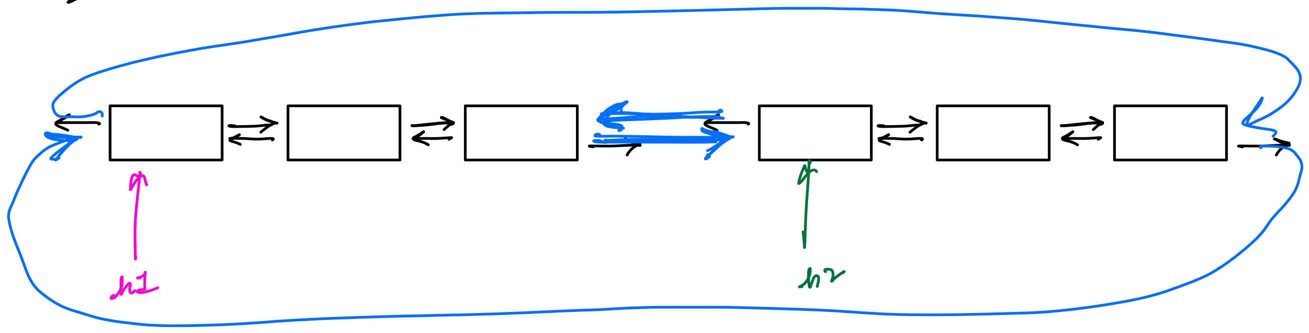
```
while( curr != null && curr->left != null)
{           temp = curr;
        while( temp != null)
        {   temp.left.next = temp.right;
            temp.right.next = temp.next.left;   → if(temp.next != null)
                temp = temp.next;
        }
        curr = curr.left
}
```

• circular doubly linked list



tail.next = head;
head.prev = tail;

• 2 CDLL , merge these



```
Node   Merge2CDLL( Node h1 , Node h2)
{
        if( h1 == null)  return h2;
        if (h2 == null)  ret  h1;
    Node t1 = h1.prev;
    Node t2 = h2.prev;

        t1.next = h2;
        t2.next = h1;
        h1.prev = t2;
        h2.prev = t1;
    return h1;
}
```

T.C: O(1)

• merge 3 CDLL

$$\begin{tikzcd}
A \arrow[r, shift left] & B \arrow[l, shift left] \arrow[r, shift left] & C \arrow[l, shift left] \arrow[ll, bend left, <->]
\end{tikzcd}$$

```
Node merge3CDLL( h1, h2, h3)
 {
 return merge2CDLL(merge2CDLL( h1, h2), h3)
 }
```

T.C = $\underline{O(1)}$

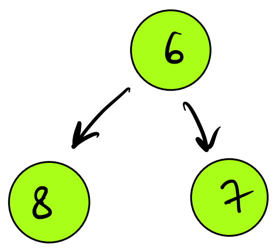
Q Given a BT, convert it into CDLL in-place

All nodes on LST will come on leftside of a node and all nodes on RST will come on right side.

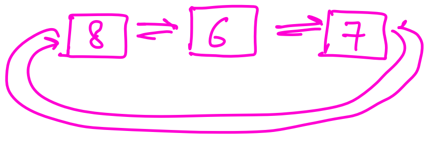
⇓

inorder

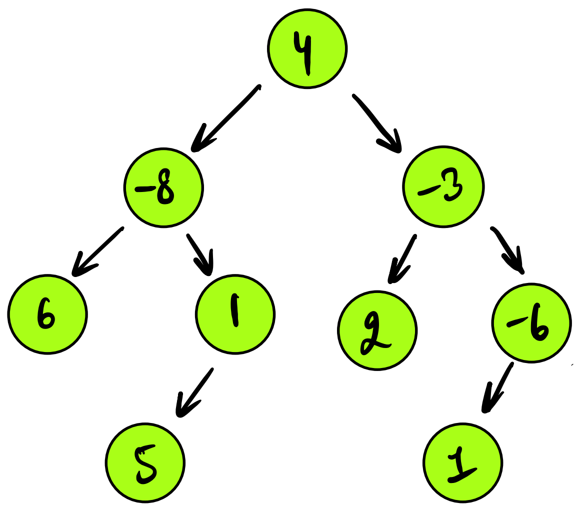
left ≡ prev
right ≡ next



⇒

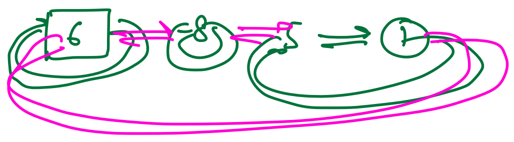


convert tree → CDLL

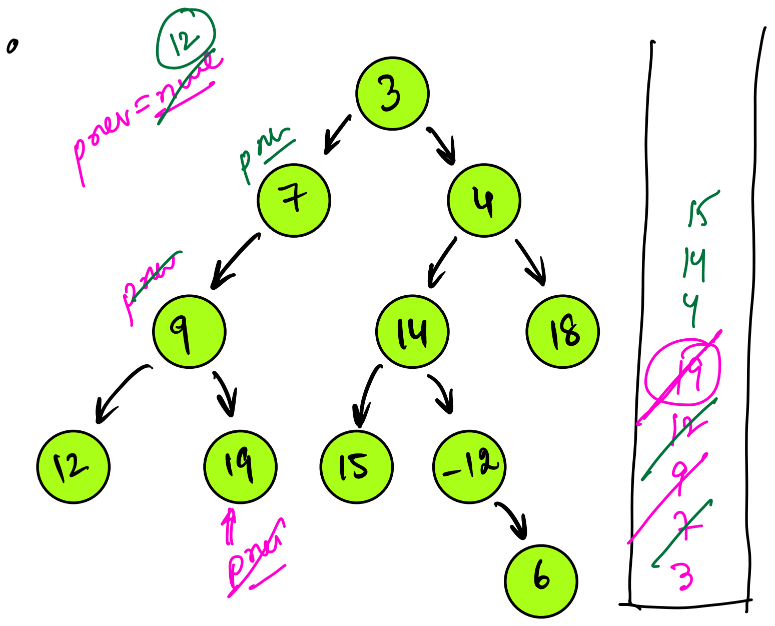


temp.next = Temp
temp.prev = Temp

T.C : O(n)



```
Node  BT2CDLL( Node root)
 {
      if(root == null) return NULL;
   Node l = BT2CDLL(root.left);
   Nod r = BT2CDLL(root.right);
     root.left = root;
      root.right = root;
   return merge3CDLL(l, root, r)
}
```



```
            curr = root                prev = NULL;
           stack  s;
while ( ! s.empty () || curr != null)
{          while ( curr != null)
           {    s.push( curr);
                curr = curr.left;
           }
       curr = s.top();
           if (curr.right == null || curr.right == prev)
           {          print( curr.data);
                        s.pop();
                      prev = curr;   curr = null;
           }
          else
          { curr = curr.right ; }
}
```

12   19   9   7

preorder

```
            curr = root
           stack  s;
while ( ! s.empty () || curr != null)
{          while ( curr != null)
           {         print( curr.data);
                      s.push( curr.right);
                     curr = curr.left;
           }
        curr = s.top();

              s.pop();
        ~~curr = curr.right;~~

}
```

postorder

```
            curr = root          prev = NULL;
          stack s;
while ( !s.empty() || curr != null)
{         while( curr != null)
          {    s.push( curr);
               curr = curr.left;
          }
        curr = s.top();
            if (curr.right == null || curr.right == prev)
            {          print( curr.data);
                         s.pop();
                       prev = curr;   curr = NULL;
            }
            else
           { curr = curr.right; }
}
```