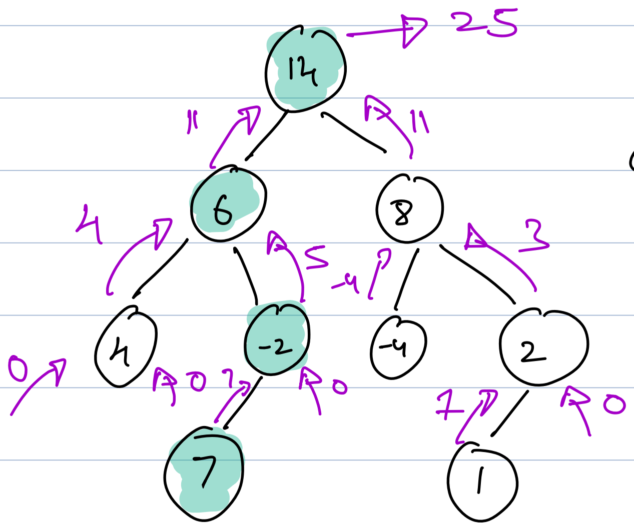
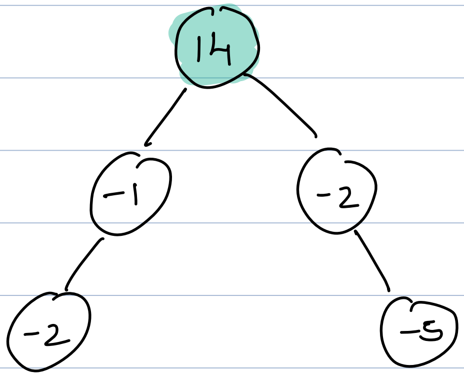
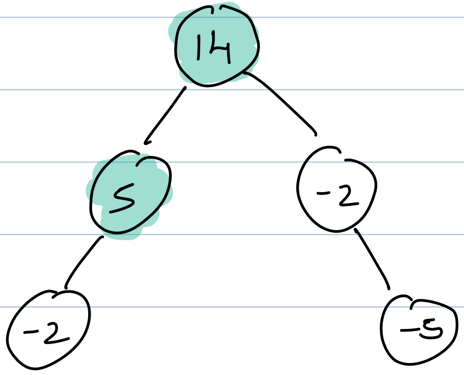
Max Path Sum → Starting from the root.



ans ⇒ 25



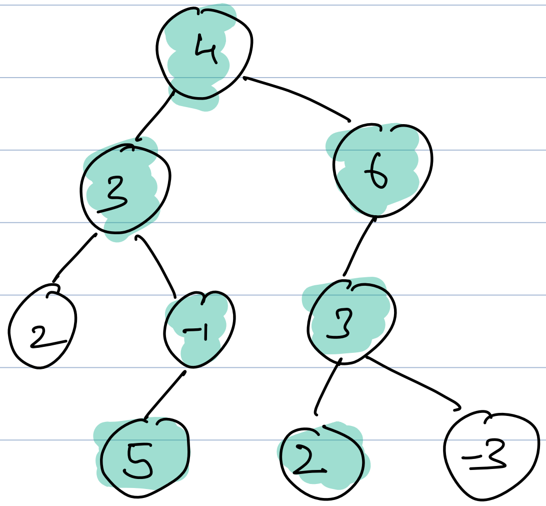
ans = 14



ans ⇒ 19

```
node.data + max (LST, RST, 0);

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

    int x => max_Sum (root.left);
    int y => max_Sum (root.right);

    return root.val + max (x, y, 0);
}
```

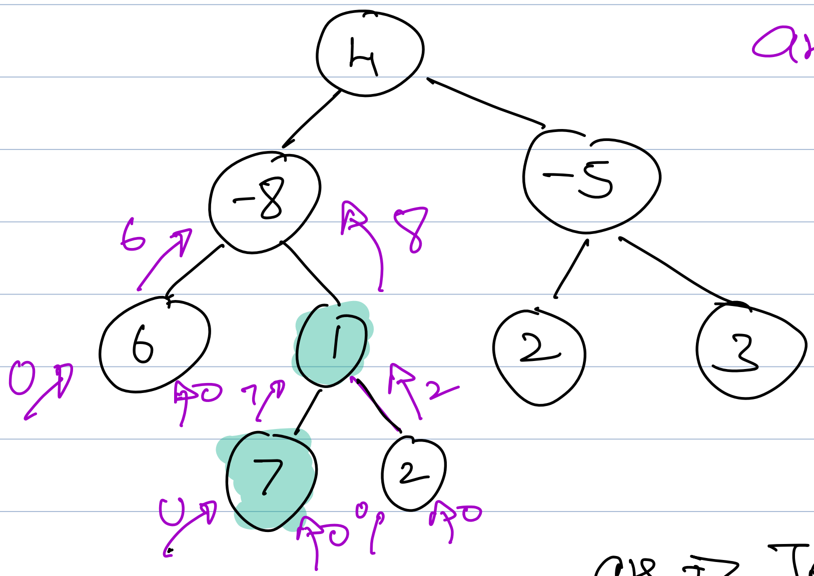
Max Sum passing through root.



⇒ 22

⇒ root.val + max(x, 0) + max(y, 0)

Max path sum in the tree.



ans = 10

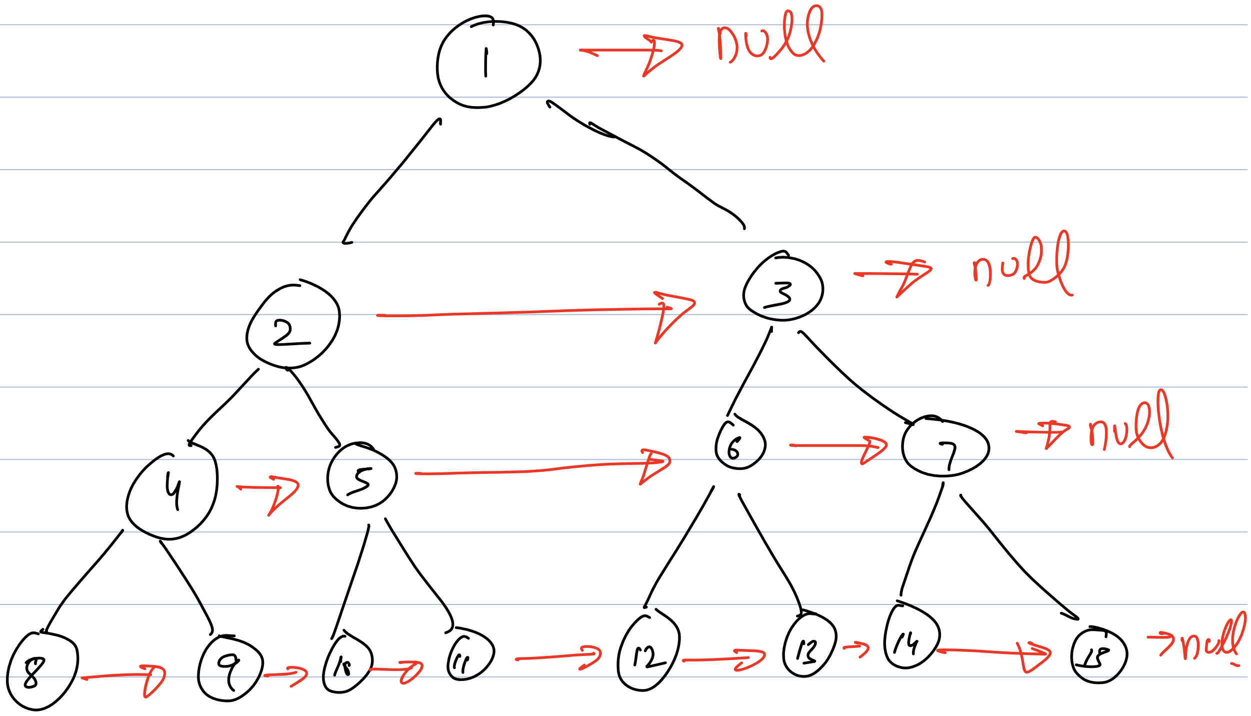
ans ⇒ Integer.Min;

```
⇒   int   maxSum (Node root) {

          if (root == NULL)
              return 0;

          int x ⇒ maxSum (root.left);
          int y ⇒ maxSum (rool.right);

  int curr = root.val + max (x, 0) + max(y,0);

        ans = max (curr, ans)

        return   rool.val + max (x,y,0);
}
```

Q Given a perfect binary tree.

→ every parent has 2 children

→ every level is completely filled.



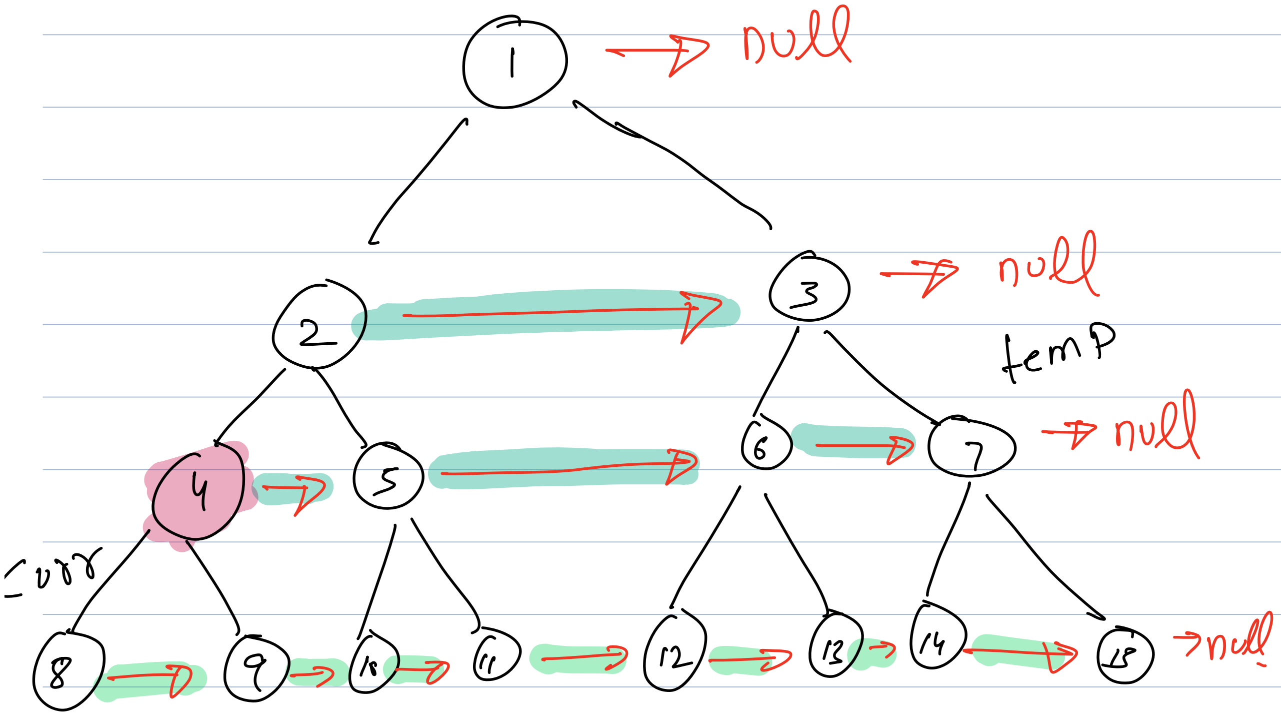
Class Node {
int val;
Node left;
Node right;
Node next;
}

Tc: O(n)

Sc: O(n)

reduce

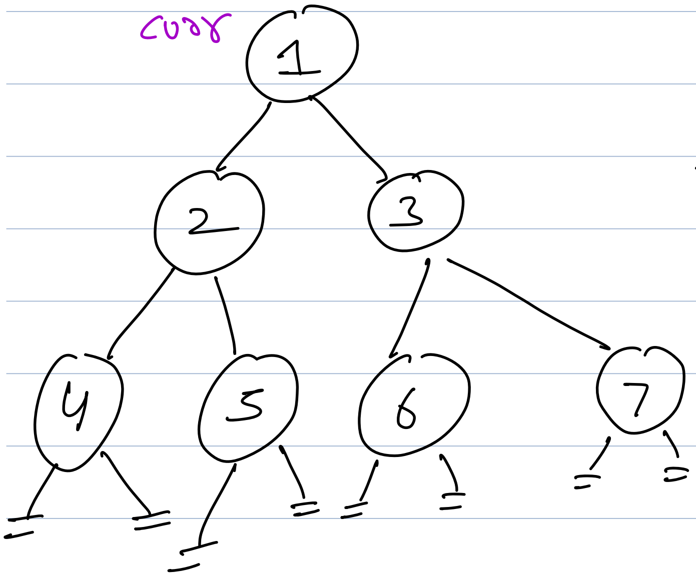
SC : $O(1)$



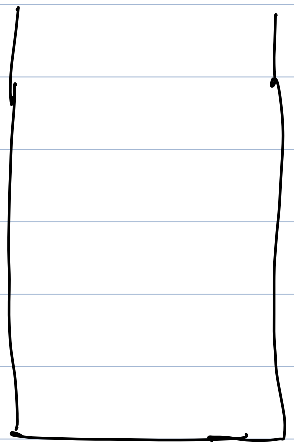
```
temp.left.next  =  temp.right

temp.right.next  =  temp.next.left.
```

```
void    makeNextPointers  (Node  root)  {

    Node   curr   ⇒  root;
    Node   temp ;

    while ( Curr != NULL  &&  curr. left != null) {

        temp  ⇒  curr.

        while  ( temp != null)  {

            temp.left.next   =  temp.right
          if ( temp.next  != null )  {
           temp.right.next  =  temp.next.left.
          }

          temp  ⇒  temp.next;

      }

      curr ⇒  curr.left;

} }
```

Inorder



Left Root Right



4, 2, 5, 1, 6, 3, 7

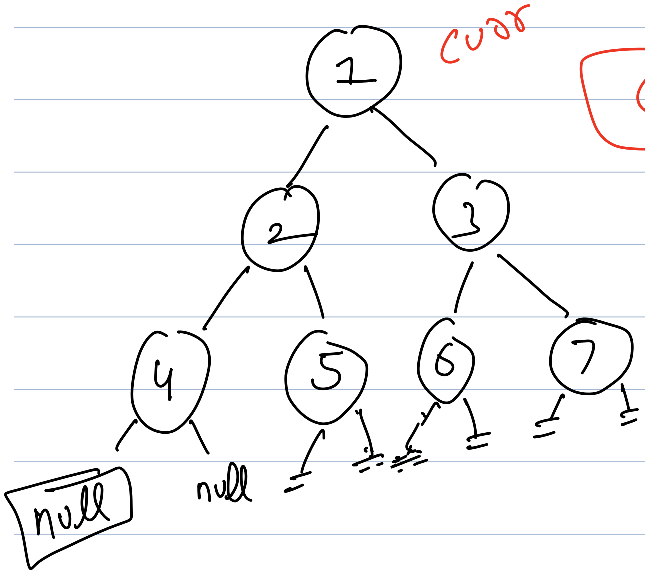
```
Stack <Node> st;
Node curr = root;

while (!st.empty() || curr != NULL) {

    while (curr != NULL) {
        st.push(curr);
        curr = curr.left;
    }

    curr = st.top();
    st.pop();
    print (curr.val);

    curr = curr.right;

  }
}
```

TC : O(n)
SC : O(h)

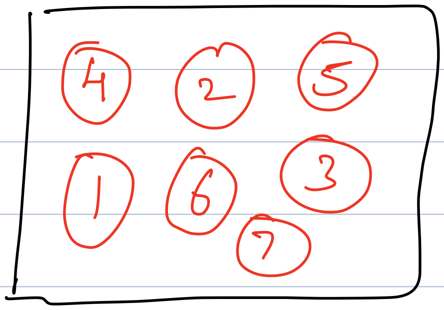
Preorder : Root, Left, Right

```
Stack <Node> st;
Node curr = root;

while (!st.empty() || curr != NULL) {

    while ( curr != NULL) {
        print (curr.val);
        st.push (curr);
        curr = curr.left;
    }
    curr = st.top();
    st.pop();

    curr = curr.right;

  }
}
```

Postorder !   ⇒ Left Right Root



curr = null



4 5, 2, 6, 7, 3, 1

Prev ⇒

```
Stack <Node> st;        Set <Node> S;
Node curr = root;

while (!st.empty() || curr != NULL) {

    while (curr != NULL) {
        st.push(curr);
        curr = curr.left;
    }

    curr = st.top();
    if (S.contains(curr)) {
        print(curr.val);
        st.pop();           prev = curr
        curr = NULL
    } else {                            TC: O(n)
        S.insert(curr);
        curr = curr.right;              SC: O(H)
    }

}
}
```

```
Stack <Node> st;          Node prev = null
 Node curr = root;

 while ( !st.empty() || curr != NULL ) {

      while ( curr != NULL ) {
           st.push (curr);
           curr = curr. left;
      }

    curr = St. top();
    if. ( curr. right == null || curr. right = prev) {
         print (curr.val);
          st. pop(),          → prev = curr
         curr = NULL
    } else {
         curr = curr. right;
    }

}
```

TC: O(n)

SC: O(H)