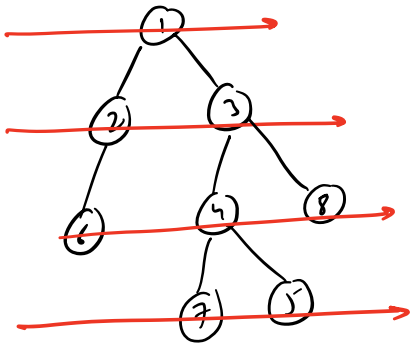
# Level Order Traversal [LOT]



Print

1

2 3

6 4 8

7 5

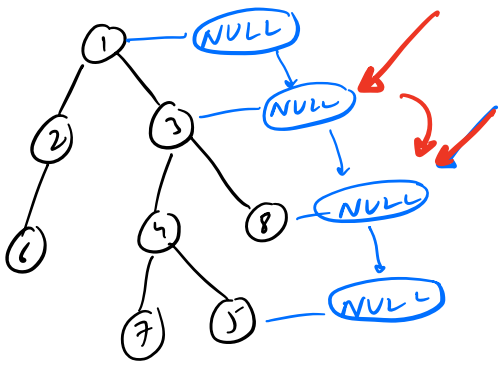
Print

1

2 , 3

6 , 4 , 8

→ 7, 5,

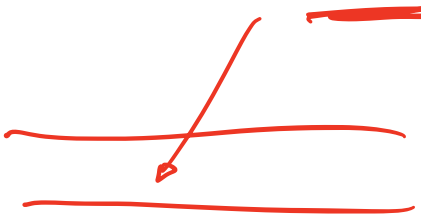



1, NULL, 2, 3, NULL, 6, 4, 8, NULL, 7, 5, NULL

→ if (root == NULL) return;

```
queue<Node> q;
q.enque(root), q.enque(NULL);
while(q.size() > 1) {
    Node f = q.front();
    q.dequeue();
    if(f == NULL) {
        print(newLine);
        q.enque(NULL);
    }
```

N + H



```
else {
    print(f.data);
    if (f.left != NULL) {
        q.enqueue(f.left);
    }
    if (f.right != NULL) {
        q.enqueue(f.right);
    }
}
}
```

TC = O(N)

SC = O(W)

O(N)

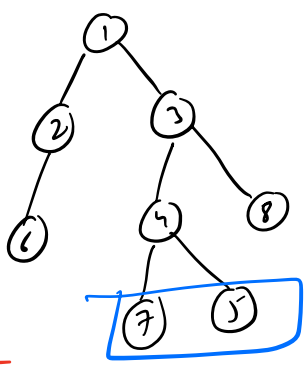
II)

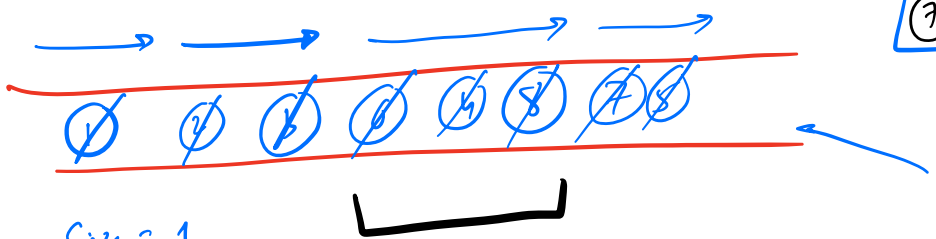
1

2 3

6 4 8

7 5





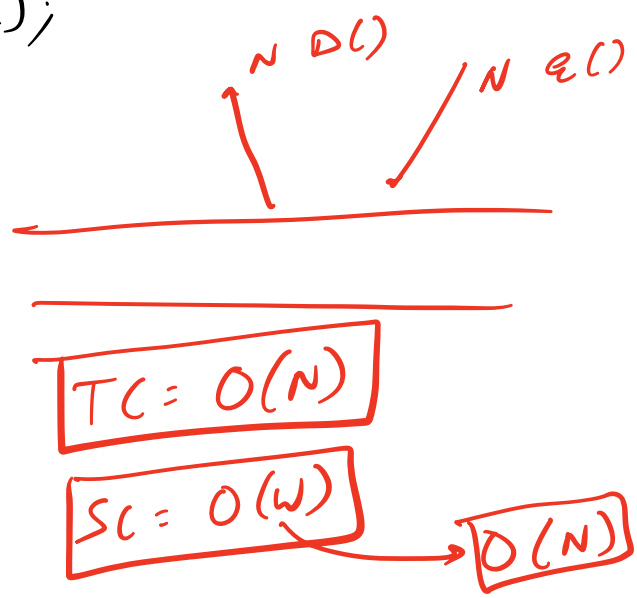
Size = 1

= 2

= 3

= 2

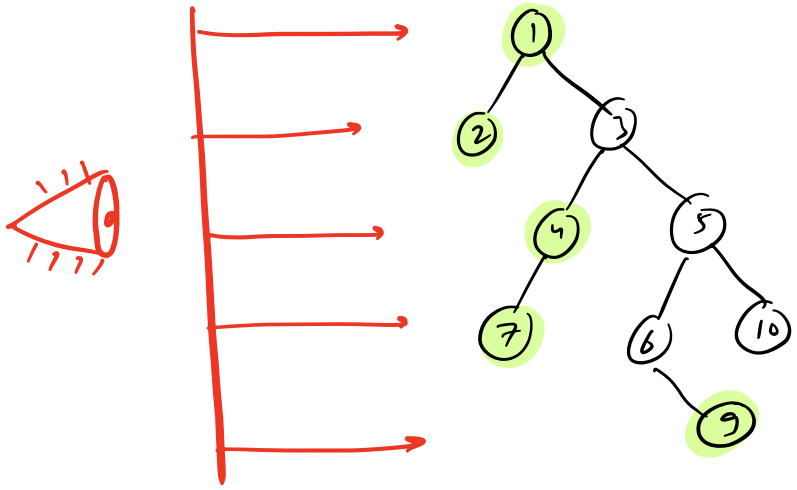
```
queue<Node> q;
q.enqueue(root);
while(!q.isEmpty()) {
    s2 = q.size();
    for(i=0; i<s2; i++) {
        Node f = q.front();
        q.dequeue();
        print(f.data);
        if(f.left != NULL) {
            q.enqueue(f.left);
        }
        if(f.right != NULL) {
            q.enqueue(f.right);
        }
    }
    print(newline);
}
```



Q Print the Left view of the B.T. !

1, 2, 4, 7, 9

Obs: 1st element of every level



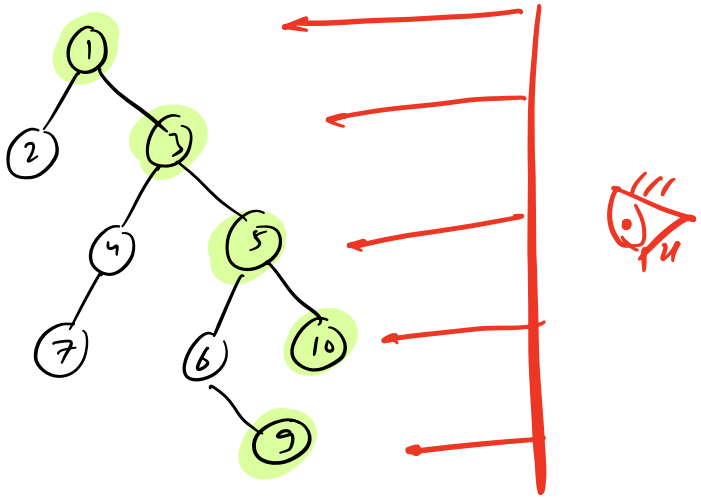

```
Queue<Node> Q;
Q.enqueue(root);
while(!Q.isEmpty()) {
    s2 = Q.size();
    for(i=0; i<s2; i++) {
        Node f = Q.front();
        Q.dequeue();
        print(f.data);   // if(i==0)
        if(f.left != NULL) {
            Q.enqueue(f.left);
        }
        if(f.right != NULL) {
            Q.enqueue(f.right);
        }
    }
}
```

~~print(newline);~~

}

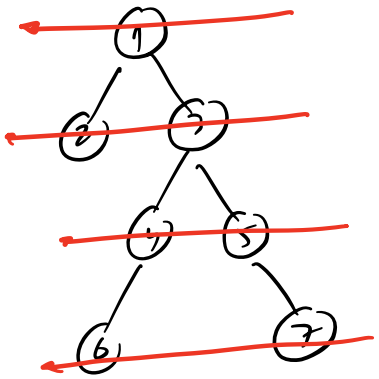
| TC : O(N) |
| --- |
| SC : O(W) → O(N) |

## Print Right View




```
Queue<Node> q;
q.enqueue(root);                    l = 0
while(!q.isEmpty()) {
    sz = q.size();
    for(i=0; i<sz; i++) {
        Node f = q.front();
        q.dequeue();
        print(f.data);              // if(i==sz-1)
        if(f.left != NULL) {
            q.enqueue(f.left);
        }
        if(f.right != NULL) {
            q.enqueue(f.right);
        }
    }
    print(newline);                 → l++;
}
```

Q Given a B.T. Do LOT (L←R)

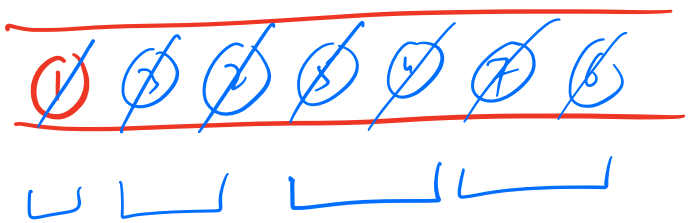


1

3 2

5 4

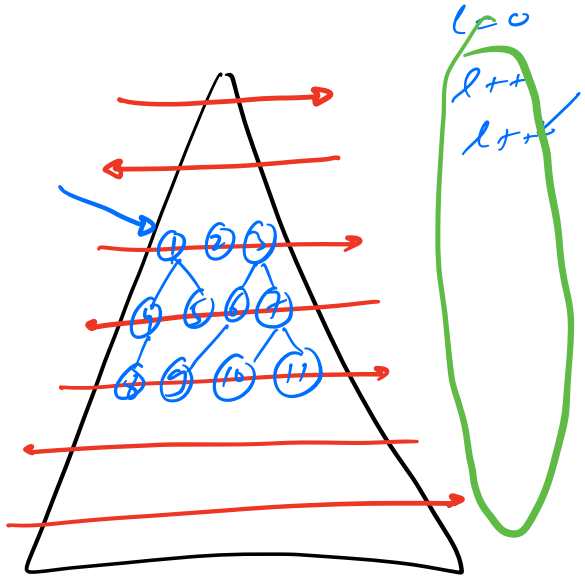
7 6



Idea: push right child before left child!

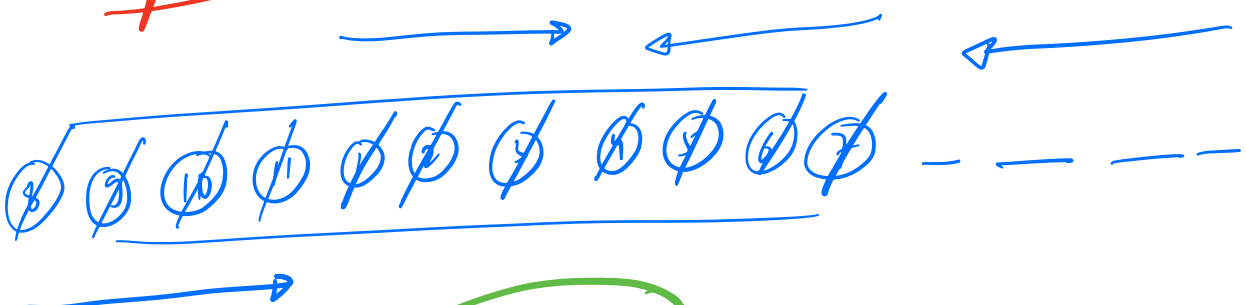
Q Given a B.T.

Zig-Zag LOT ?




1 2 3

deque !

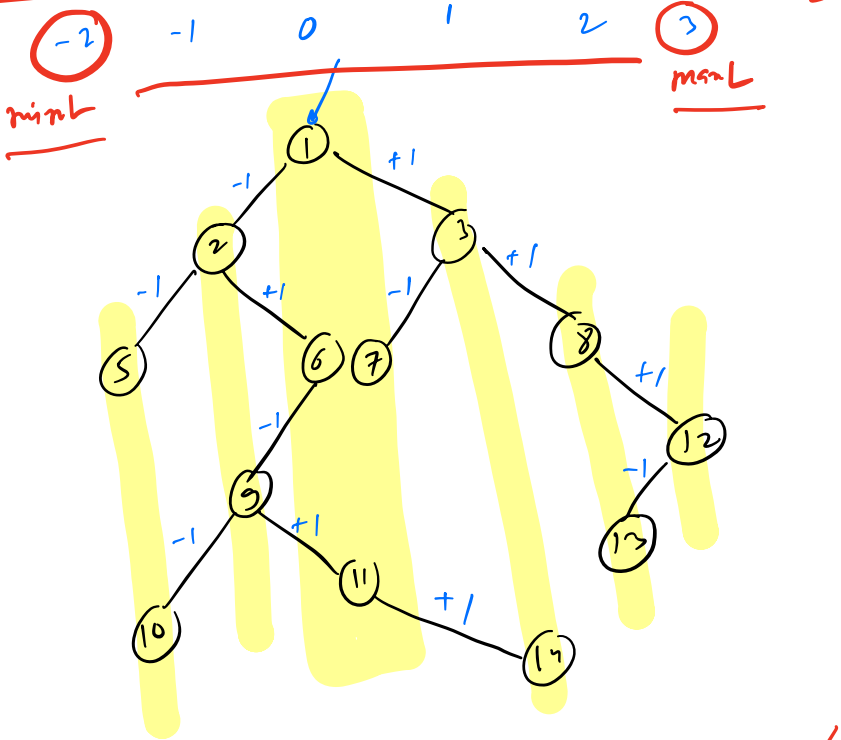


HW → Queue + Stack

stack : 2

deque

Q Given a B.T. Do Vertical Order Traversal [ VOT ]




< 5, 10 >

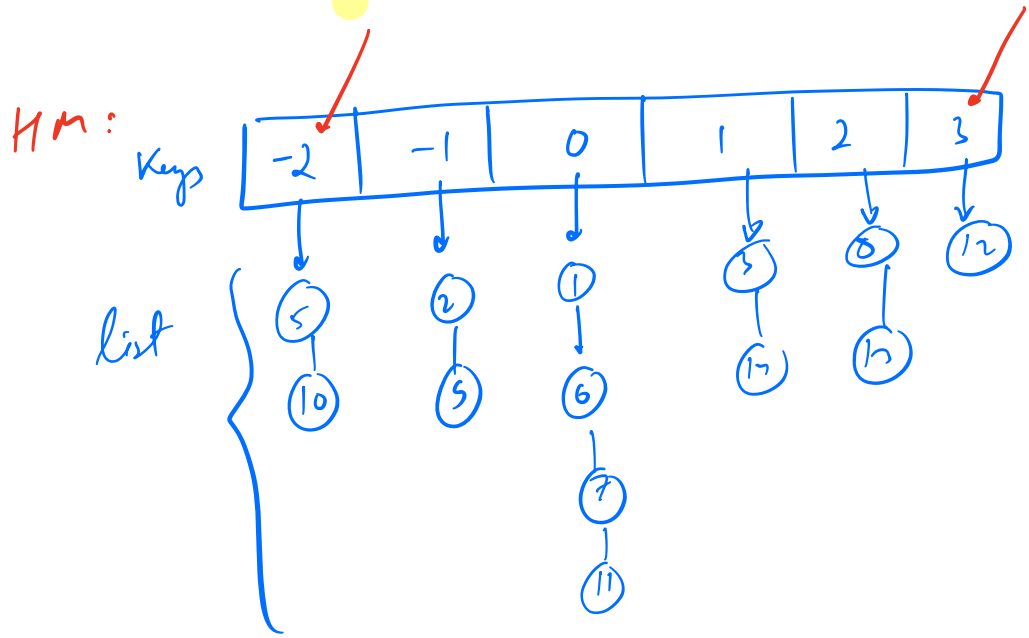
< 2, 9 >

< 1, 6, 7, 11 >

< 3, 14 >

< 8, 13 >

< 12 >




HashMap < int, List < Node > > hm;

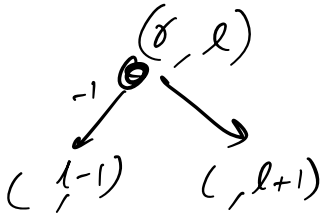
1)

HashMap < int, List < Node >> hm;

maxL = -∞, minL = +∞

(root, 0)

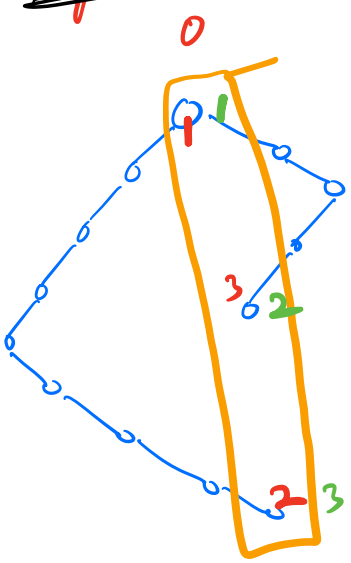
```
void preorder ( Node root, int l) {
    if ( root == NULL) ret;
    maxL = max(maxL, l);
    minL = min ( minL, l);
    hm[l].add(root);
    preorder ( root.left, l-1);
    preorder ( root.right, l+1);
}
```



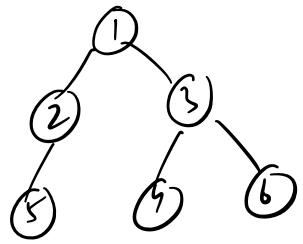

```
for ( i = minL; i <= maxL; i++) {
    print (hm[i]);
    print ( new Line);
}
```

→ print the list

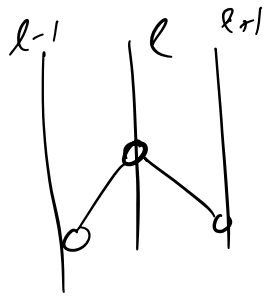
CORNER CASE :



II LOT




```
class Info {
    Node node;
    int l;
}
Queue<Info> Q;
Q.enqueue(Info(root, 0));
maxL = 0, minL = 0;
while(!Q.isEmpty()) {
    Info f = Q.front();
    Q.dequeue();
    hm[f.l].add(f.node);
    maxL = max(maxL, f.l), minL = min(minL, f.l);
    if(f.node.left != NULL) {
        Q.enqueue(Info(f.node.left, f.l-1));
    }
    if(f.node.right != NULL) {
        Q.enqueue(Info(f.node.right, f.l+1));
    }
}
```



```
for( i = minL ; i <= maxL ; i++) {
    print (hm[i]);
    print( newLine);
}
```

→ N

→ print the list

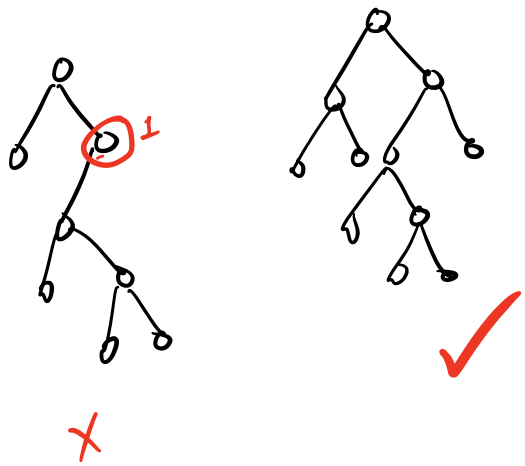
TC = O(N)

SC = O(W + N)

SC = O(N)

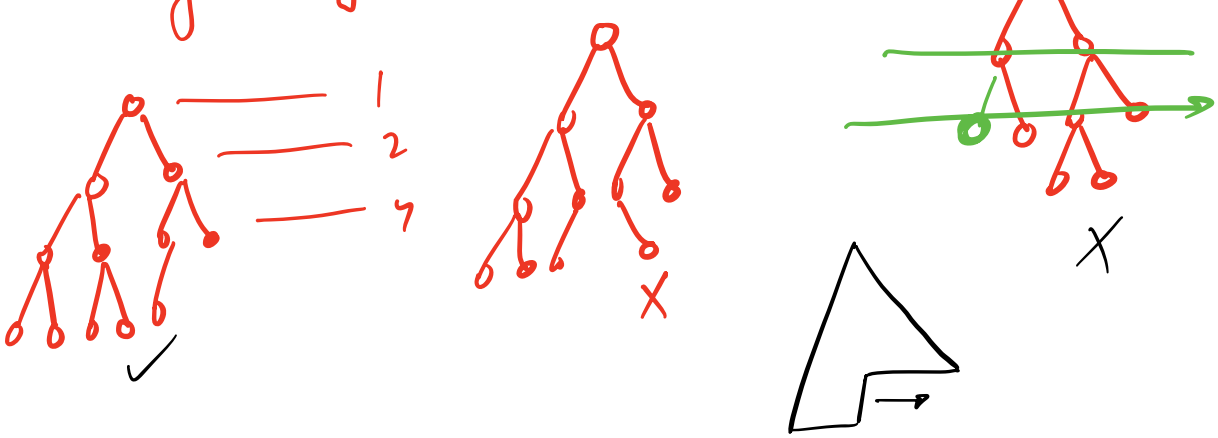
(a) Full B.T / Proper BT

→ A B.T wherein every node has either 2 child or 0 child !



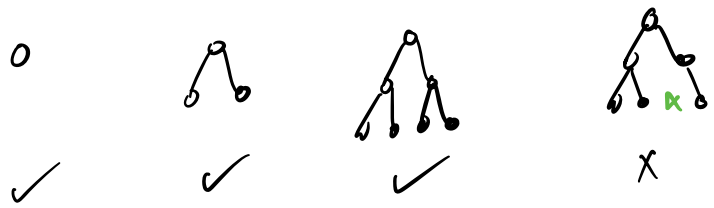

(b) Complete B.T

A B.T. wherein every level is fully filled except possibly the last level, which is filled from L → R



④ Perfect BT

A B.T with every level fully filled!

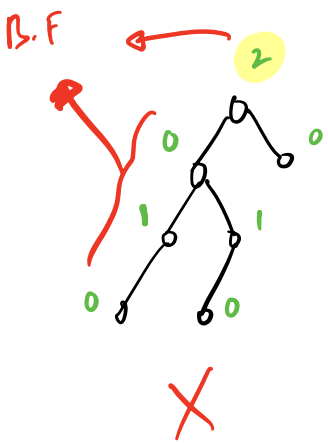


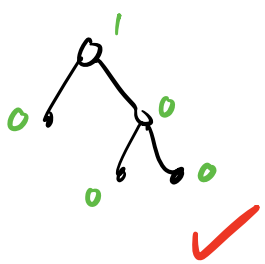

---

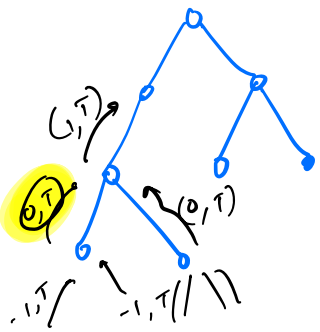
Q Given a B.T. Check if it is height balanced!

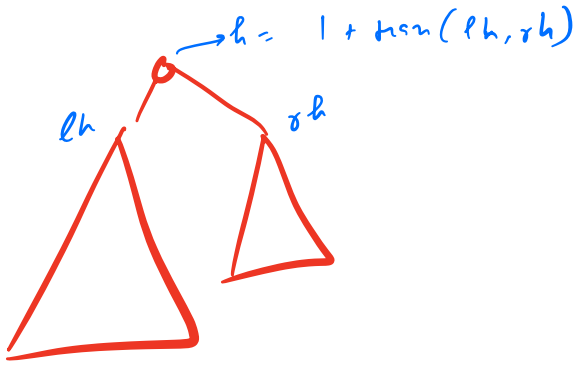
→ Absolute diff b/w the height of left S.T & right S.T <= 1 for all nodes!

balance factor (node)
= Absolute diff b/w the height of left S.T & right S.T






```
ret check(root).ok;

Info {
  int h;
  bool ok;
}

Info check (Node root) {
  if (root == NULL) {
    ret Info (-1, true);
  }
  Info LInfo = check (root.left);
  if (LInfo.ok == false) {
    ret Info (-1, false);
  }
  Info RInfo = check (root.right);
  if (RInfo.ok == false || (abs(LInfo.h - RInfo.h) > 1)) {
    ret Info (-1, false);
  }
  ret Info (1 + max(LInfo.h, RInfo.h), true);
}
```

TC = O(N)

SC = O(N)