

Striver

Dsa

Sheet

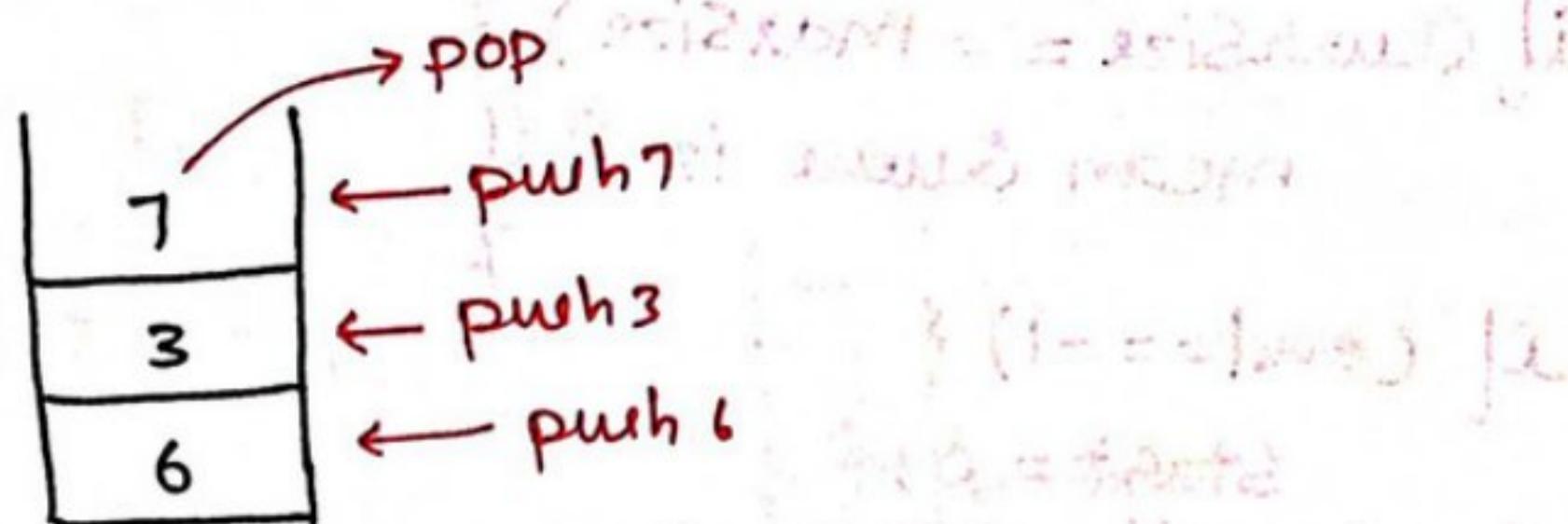
"180"

Part - 2

## Implement Stack Using Arrays : Problem - 74

push(6)  
push(3)  
push(7)  
top() → 7  
pop → ~~pop~~ 7

size - 2



size = 1000;  
int arr[ ] = new int [size];  
int top = -1; // initially.  
void push (int x){  
 top++;  
 arr [top] = x;  
}

int pop () {  
 int x = arr [top];  
 top--;  
 return x;  
}

Time complexity - O(N)

Space complexity - O(N)

## Implement Queue using Arrays : Problem. 75

```
private int arr[];  
private int start, end, currsize, maxsize ;  
  
public Queue () {  
    arr = new int [16];  
    start = -1;  
    end = -1;  
    currsize = 0;  
}  
  
public Queue (int maxSize) {  
    this .maxSize = maxSize;  
    arr = new int [maxSize];  
    start = -1;  
    end = -1;  
    currsize = 0; }
```

```

public void push(int newElement) {
    if (currSize == maxSize) {
        mean Queue is full;
    }
    if (end == -1) {
        start = 0;
        end = 0;
    } else {
        end = (end + 1) % maxSize;
        arr[end] = newElement;
        currSize++;
    }
}

public int pop() {
    if (start == -1) {
        mean Queue Empty;
    }
    int popeed = arr[start];
    if (currSize == 1)
        start = -1;
    end = -1;
    else
        start = (start + 1) % maxSize;
    currSize--;
    return popeed;
}

public int top() {
    if (start == -1)
        mean queue Empty;
    return arr[start];
}

public int size() {
    return currSize;
}

```

(Q) 1009  
(E) 1009  
(R) 1009  
T = O(1)  
C = O(1)  
S = O(1)

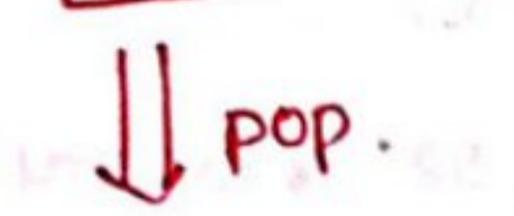
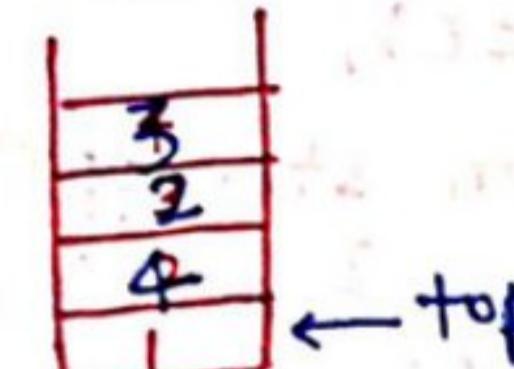
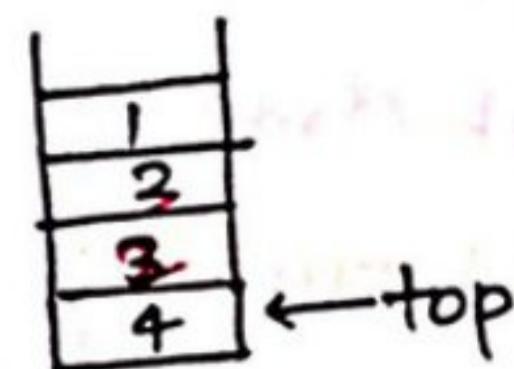
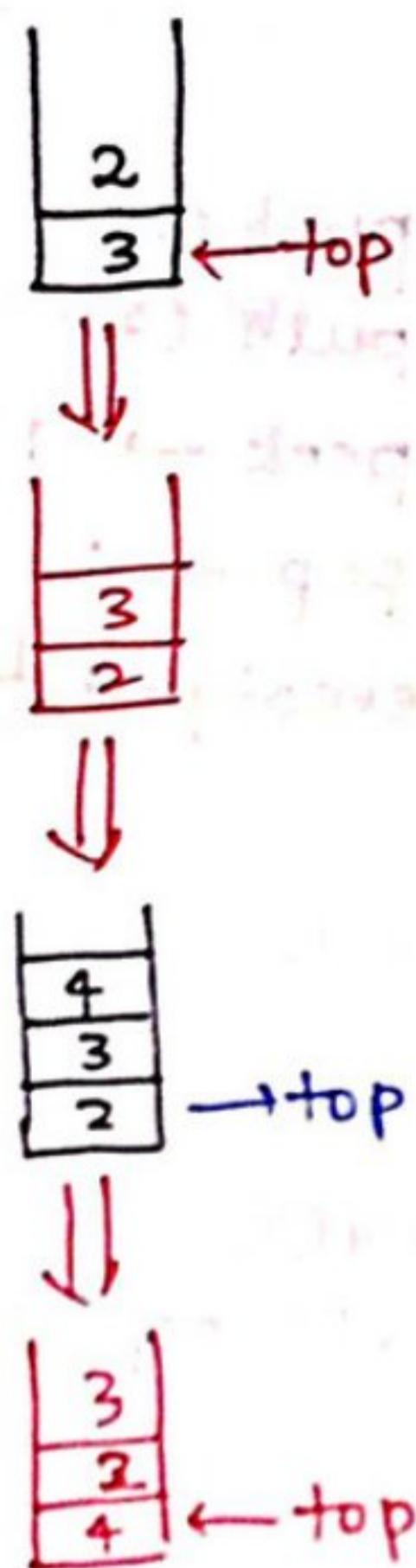
(Q) 1010  
(E) 1010  
(R) 1010  
T = O(1)  
C = O(1)  
S = O(1)

(Q) 1011  
(E) 1011  
(R) 1011  
T = O(1)  
C = O(1)  
S = O(1)

(Q) 1012  
(E) 1012  
(R) 1012  
T = O(1)  
C = O(1)  
S = O(1)

## Implement stack using Queue : Problem - 76

push (3)  
push (2)  
push (4)  
push (1)  
top - ①  
pop - (1)  
top () → ④



after pushing a new element remove all the element from the queue and again put into the queue.

```
Queue<Integer> q;  
void push(int x){  
    q.add(x);  
    for(i=0;i<q.size();i++) {  
        q.add(q.remove());  
    }  
}  
int pop(){  
    return q.remove();  
}  
int top(){  
    return q.peek();  
}  
int size(){  
    return q.size();  
}
```

Tc- O(N)  
Sc- O(N)

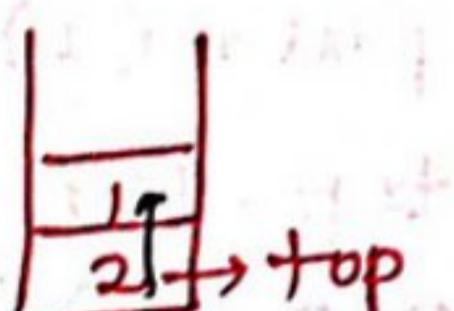
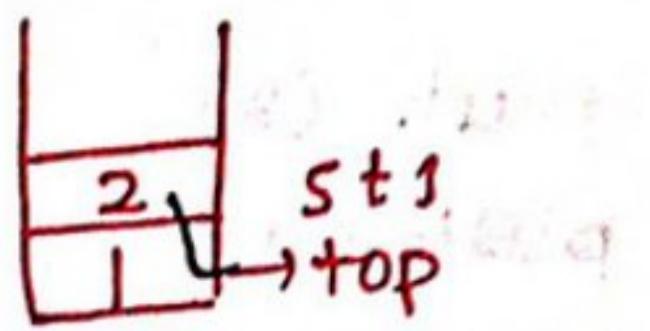
## Implement Queue Using Stacks :: Problem: 77

```

Stack<int> s1();
Stack<int> s2();
public void push(int x) {
    s1.push();
}
public int pop() {
    peek();
    return s2.pop();
}
public int peek() {
    if (s2.isEmpty())
        while (!s1.isEmpty())
            s2.push(s1.pop());
    return s2.peek();
}
public boolean empty() {
    return s2.isEmpty() && s1.isEmpty();
}

```

push(1)  
 push(2)  
 peek → 1  
 pop → 1  
 empty → false



SC -  $O(2N)$   
 TC -  $O(N)$

## Check for Balance paranthesis :: Problem: 78

first put all the open brackets into stack and if any closing brackets arrives match with the top of stack and remove. in the last check the size of stack == 0 so it is balance otherwise no.

SC -  $O(N)$   
 TC -  $O(N)$

## Next Greater Element : Problem-79

$N=4$ ,  $\text{arr}[] = [1, 3, 2, 4]$

output  $[3, 4, 4, -1]$

traverse the array from last to  $0^{\text{th}}$  index.

```

Stack<int> st();
long nge[] = new int[n];
for(i = arr.length-1; i >= 0; i--) {
    if(!st.isEmpty()) {
        while(!st.isEmpty() && st.peek() <= arr[i])
            st.pop();
    }
    if(st.isEmpty())
        nge[i] = -1;
    else
        nge[i] = st.peek();
    st.push(arr[i]);
}

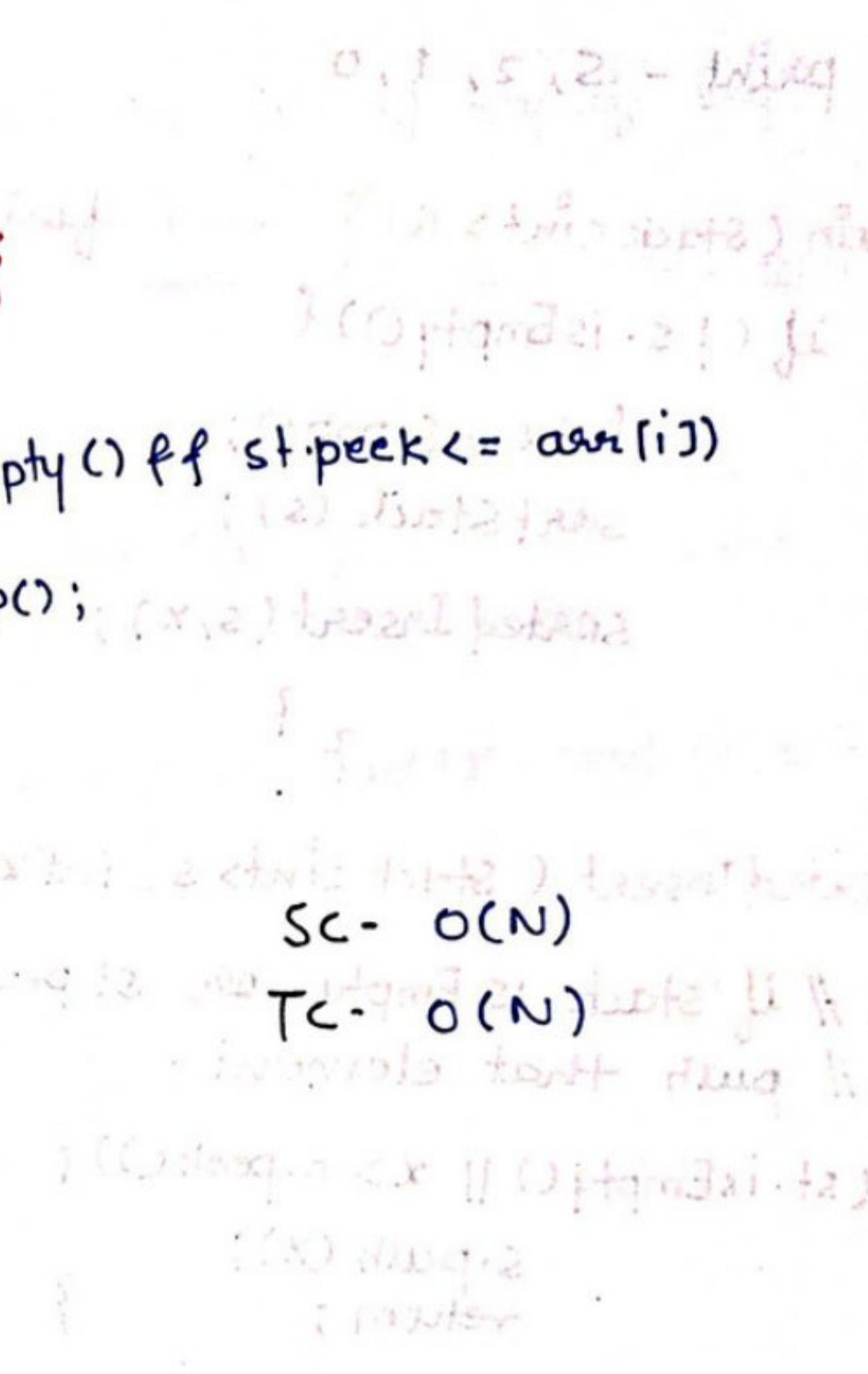
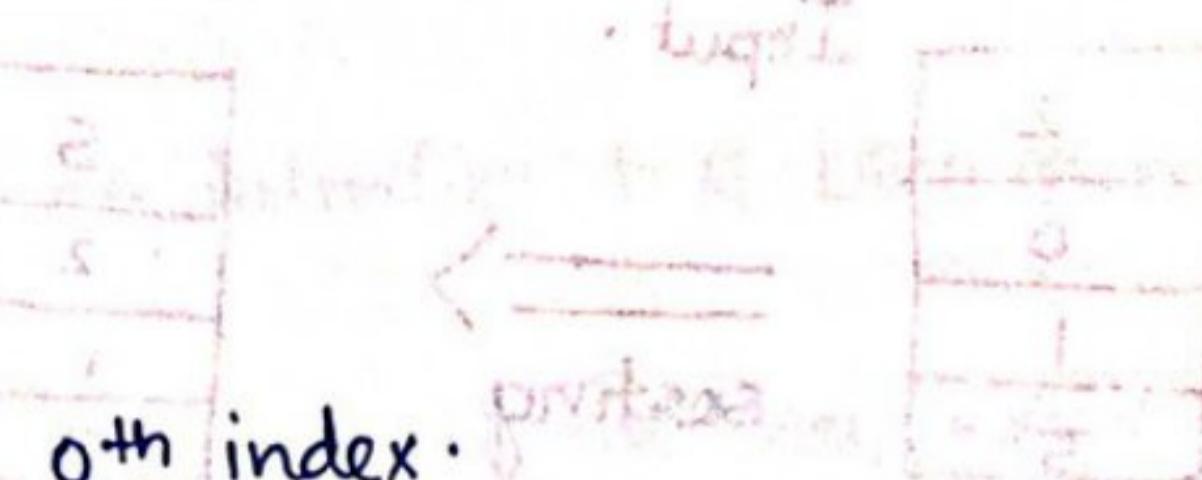
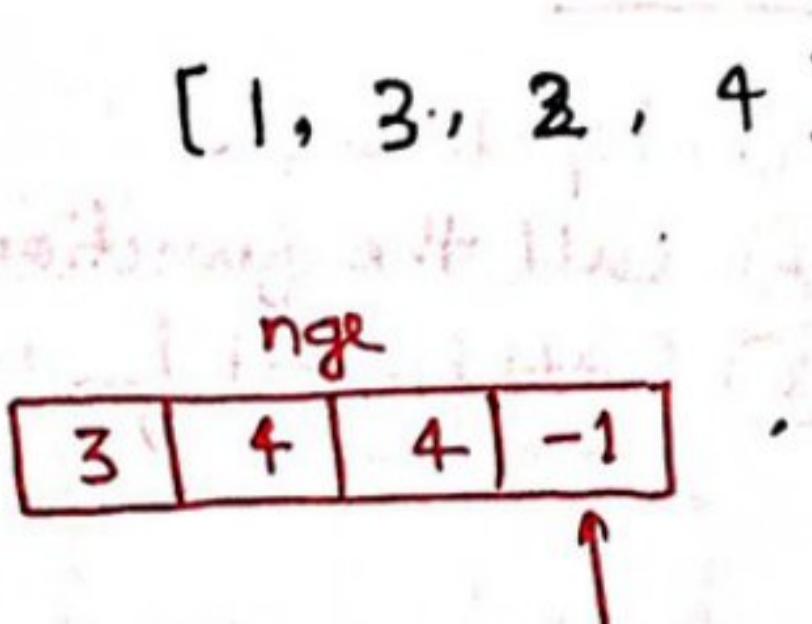
```

SC -  $O(N)$   
TC -  $O(N)$

this is the optimal Approach.

In Brute just select a number and then find the greater number than the selected number if you get then push in the array. otherwise push -1.

Time complexity -  $O(N^2)$ .



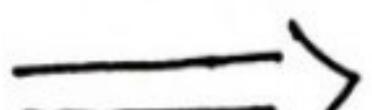
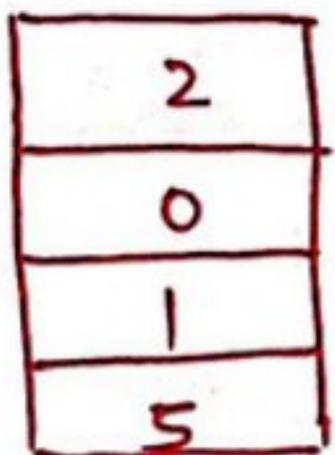
3
2
4

initially stack size = 0

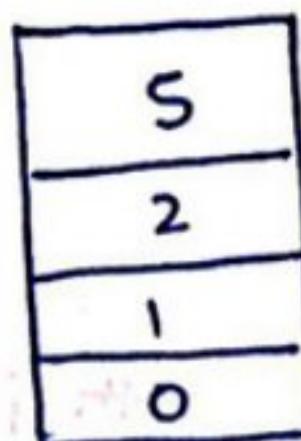
$3 > 2$  (false) remove 2

## Sort a Stack - Problem - 80

Input.



sorting



print - 5, 2, 1, 0

main (stack<int> s) { → fun name - sortStack

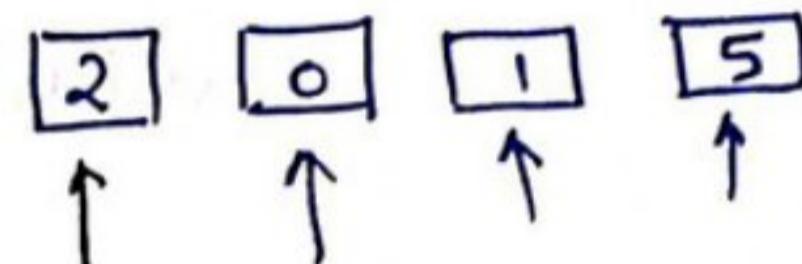
if (!s.isEmpty()) {

    int x = s.pop();

    sortStack(s);

    sortedInsert(s, x);

}



sortedInsert (stack<int> s, int x) {

// if stack is Empty or st.peek is less than x)

// push that element.

if (st.isEmpty() || x > s.peek()) {

    s.push(x);

    return;

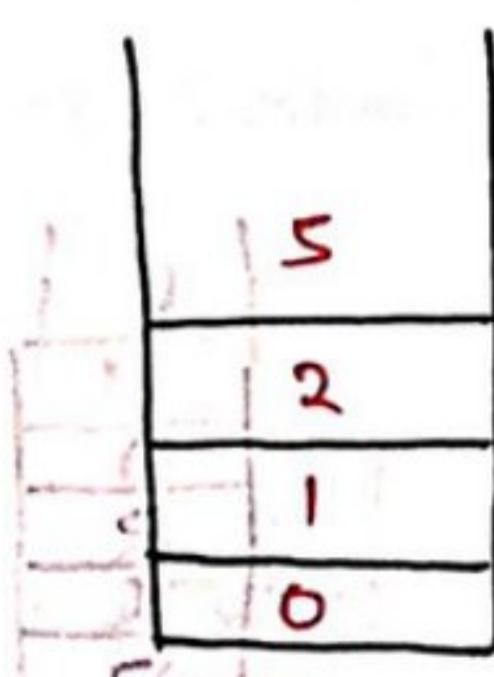
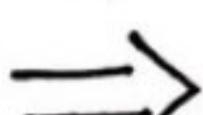
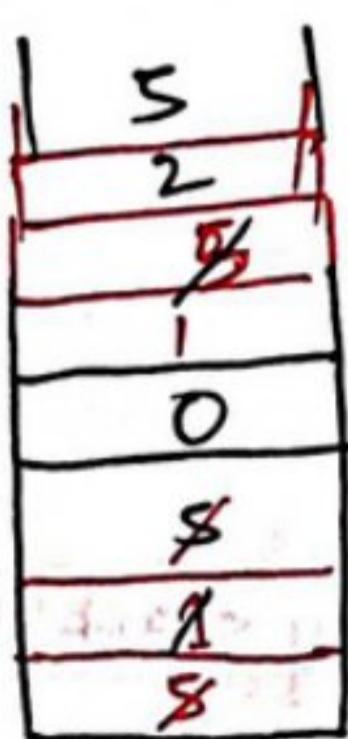
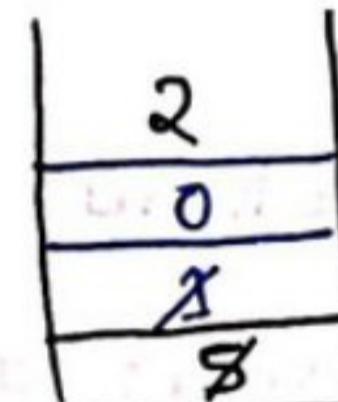
}

int temp = s.pop();

sortedInsert (s, x);

s.push(temp);

}



temp = 8

temp = 8

temp = 5

temp = 1

temp = 5  
temp = 1

Hypothesis

- ① Pop element
- ② call the function (himself)
- ③ sorted insert fun (s, x)

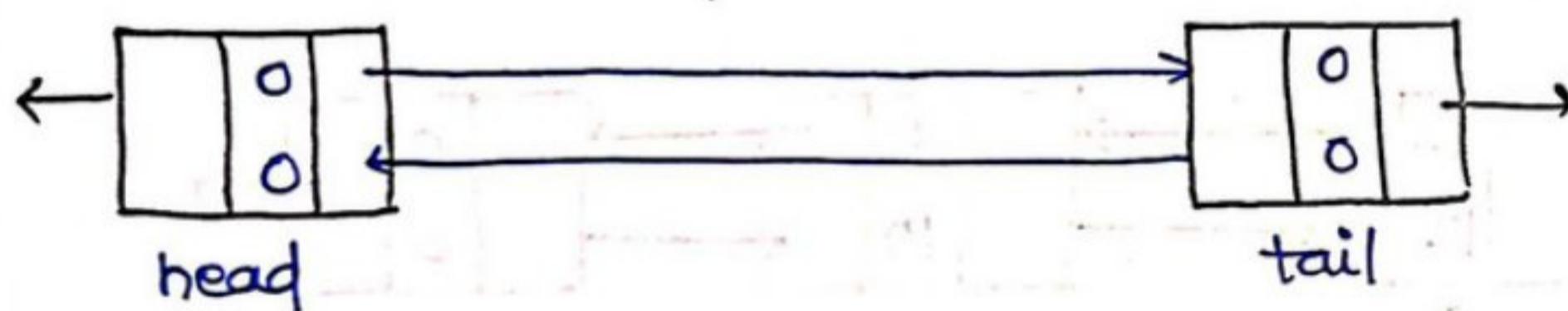
## LRU cache - Problem - 82

- LRU Cache (int capacity) - we need to initialize the LRU cache with positive size capacity.
- int get (int key) - return the val of key if the key exists otherwise return -1;
- void put (int key, int value) - Update the val. of key if key exists . otherwise, add the key-val pair to cache. if no of keys exceeds the capacity from this operation, evict the least recent.

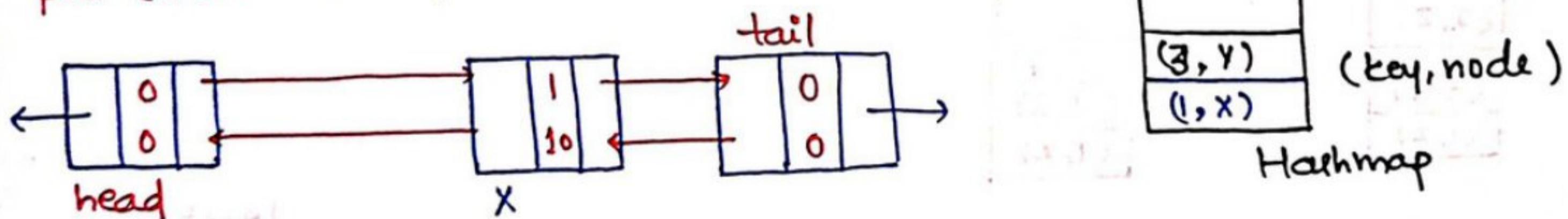
Intuition - While inserting {key, val} pair into DLL make sure that we are inserting it from the back tail to head.

Size = 3      put(1, 10)    put(3, 15)    put(2, 12)    get(3)    put(4, 25)

Create a DLL and map.



put (1, 10) Not present so put



While inserting consider:

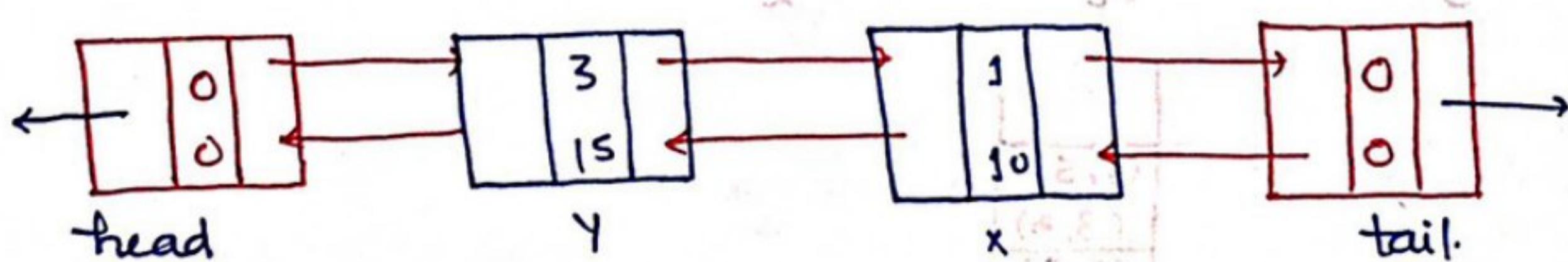
① Check if {key, val} is already present

② Check for capacity if capacity == size  
so remove the pair which is just right after head node and add

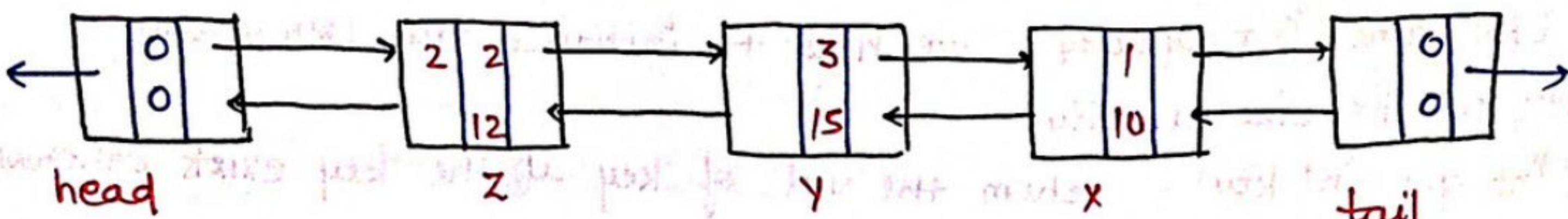
right after head node.

③ if key is not present return -1

put (3, 15) Not present so put



put (2, 12) Not present so put.

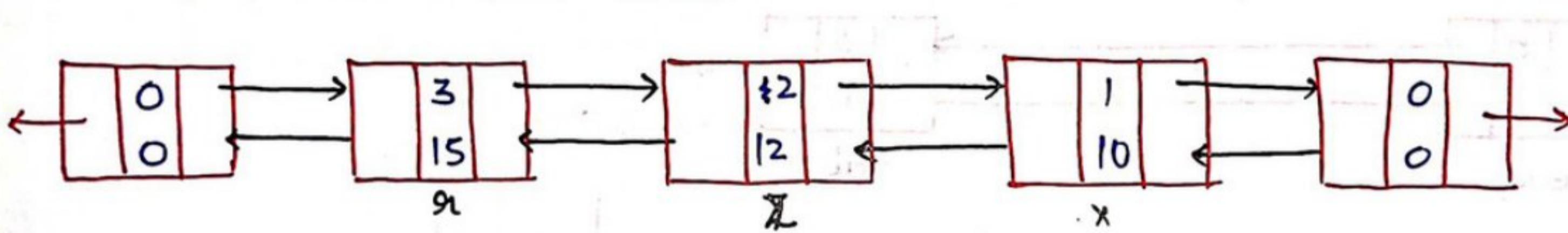
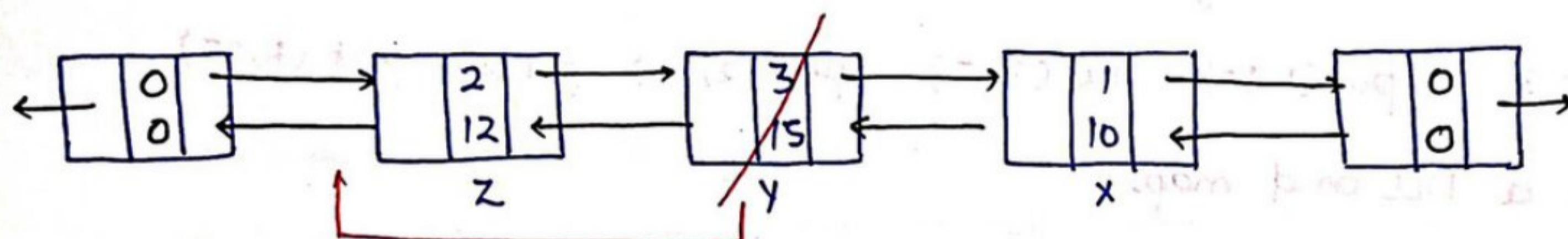


(2, z)
(3, y)
(1, x)

get(3), yes, so it is present.

Now most recent element should be change.

Delete the Node and add it right after head node.



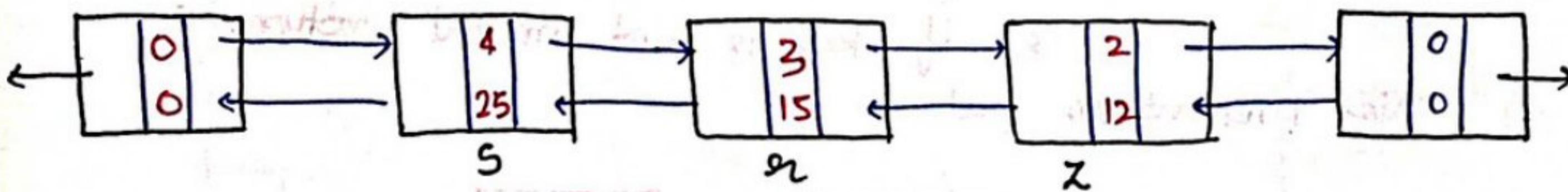
(2, z)
(3, y)
(1, x)

(3, y)
(2, z)
(1, x)

put (4, 25) because capacity full. so we have to remove least recent used element.

1
10

will remove and add (4, 25)



(4, s)
(3, y)
(2, z)

```

class LRUCache {
    Node head = new Node (0,0);
    Node tail = new Node (0,0);
    Map <int, Node> map = new HashMap ();
    int capacity;
    public LRUCache (int capacity) {
        capacity = -capacity;
        head.next = tail;
        tail.prev = head;
    }
}

class Node {
    Node prev, next;
    int key, val;
    Node (int key, int value) {
        this.key = key;
        this.value = value;
    }
}

public int get (int key) {
    if (map.containsKey (key)) {
        Node node = map.get (key);
        remove (node);
        insert (node);
        return node.value;
    }
    else return -1;
}

public void put (int key, int val) {
    if (map.containsKey (key)) {
        remove (map.get (key));
    }
    if (map.size == capacity) {
        remove (tail.prev);
    }
    insert (new Node (key, val));
}

```

```

private void remove (Node node) {
    map.remove (node.key);
    node.prev.next = node.next;
    node.next.prev = node.prev;
}

private void insert (Node node) {
    map.put (node.key, value) node;
    node.next = head.next;
    node.next.prev = node;
    head.next = node;
    node.prev = head;
}

```

Time complexity -  $O(N)$

Space complexity -  $O(1)$

### LFU Cache (Least frequent used) : Problem 83

get(key) → get the value of key if exists, else -1

put(key, val) → update the value of key if it is present, or insert the key if not present.

when the cache is full, it removes the LFU one if there are multiple LFU so we choose with the help of LRU.

Size = 2

put (1, 10)

put (2, 20)

get (1) → 10

put (3, 30)

get (2) → -1

get (3) → 30

put (4, 4)

get (1) → -1

get (3) → 30

get (4) → 4

freq.

1 → ~~(1, 10)~~ (2, 20)

2 → ~~(1, 10)~~

Now cache is full and remove

1 → ~~(3, 30)~~

2 → ~~(1, 10)~~ (3, 30)

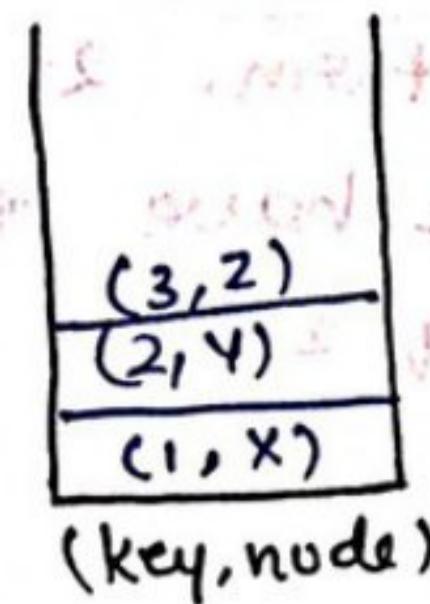
because here is a tie so we remove one of them by taking help of LRU.

1 → (4, 4)

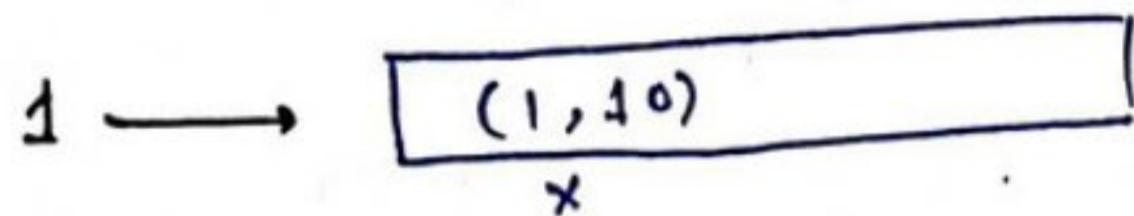
2 → (4, 4) (3, 30) (4, 4)

3 → (3, 30)

$\rightarrow$  DLL  
 Map<freq, list> freqList;  
 Map<key, node> keynode;  
 capacity = 0  
 freq = 0

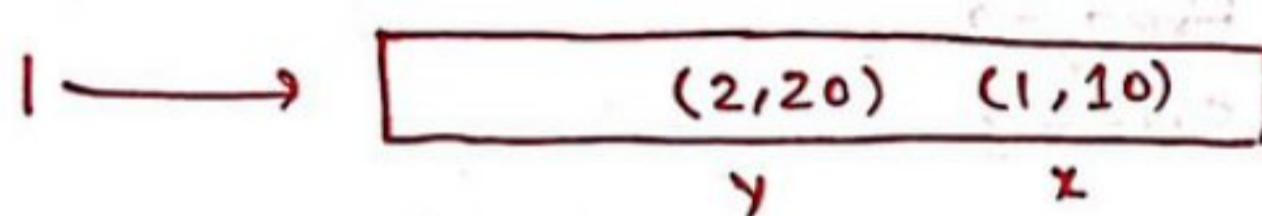


put(1, 10), check that (1, 10) is present in the keynode map or not.  
 if it is not so check for the capacity.



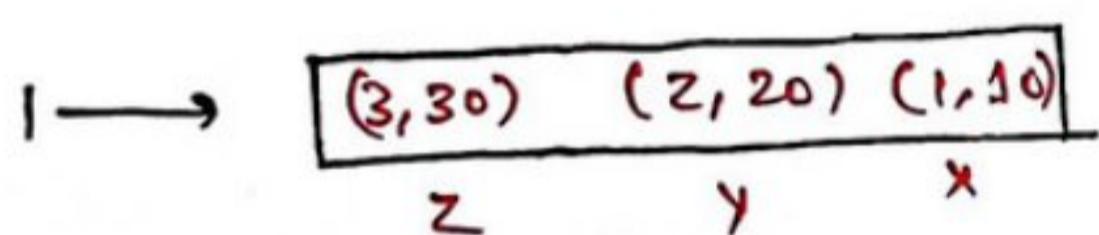
freq = 1  
 capacity = 1

Now put (2, 20) → verify it is present in the map or not also check the capacity.



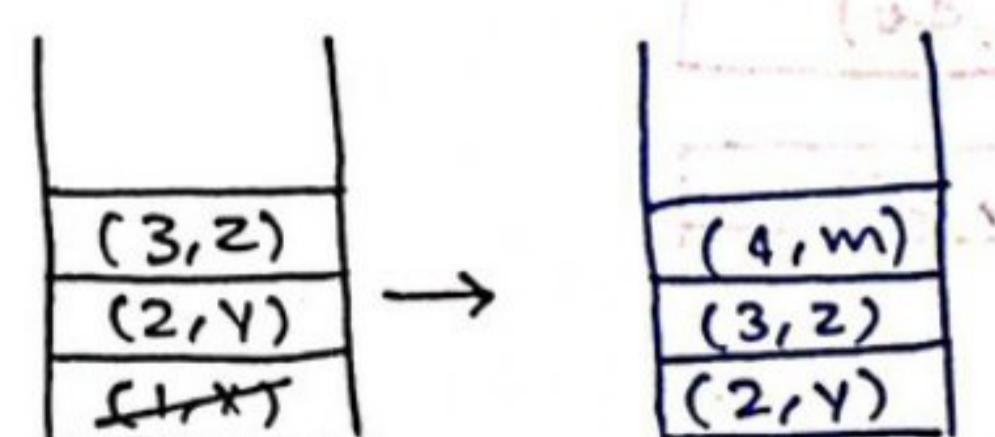
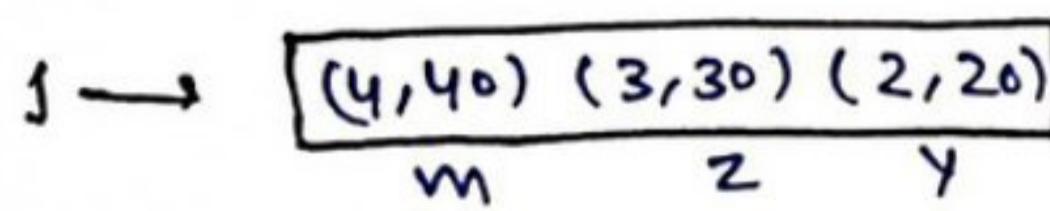
freq = 1  
 capacity = 2

put (3, 30). → verify is present in the map or not.



freq = 1  
 capacity = 3.

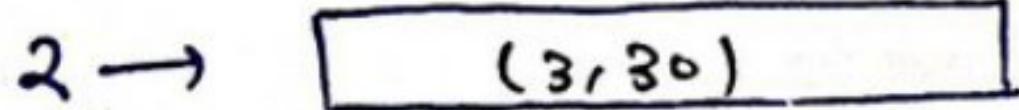
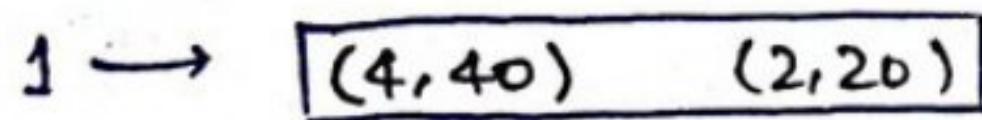
put (4, 40) → it is not present in the map but capacity is full so we have to remove, so we remove LFU element. but because it has three elements so choose according to LRU.



get(3) → yes it is in the map.

and check for the value of 3. it is 30.

and now we have to delete the z node. frequency will be 2.



get(2) → yes it is exist.  
the value return. 20

and after that we have to remove the value (2, 20)  
and put it into freq 2.

1 → (4, 40)

2 → (2, 20) (3, 30)

(2, 20)
(3, 30)

get(4) → it is exist. val = 40

1 →

2 → (4, 40) (2, 20) (3, 30)

freq[1] has a empty list  
so we have to increase the  
freq.

freq = 2

size = 3

put(5, 50) → No we can insert  
it, but capacity is full.

1 → (5, 50)

2 → (4, 40)(2, 20)

(4, W)
(5, X)
2, Y

freq = 1

put(2, 25) → 2 is exist, so we update it and change  
freq.

1 → (5, 50)

2 → (4, 40)

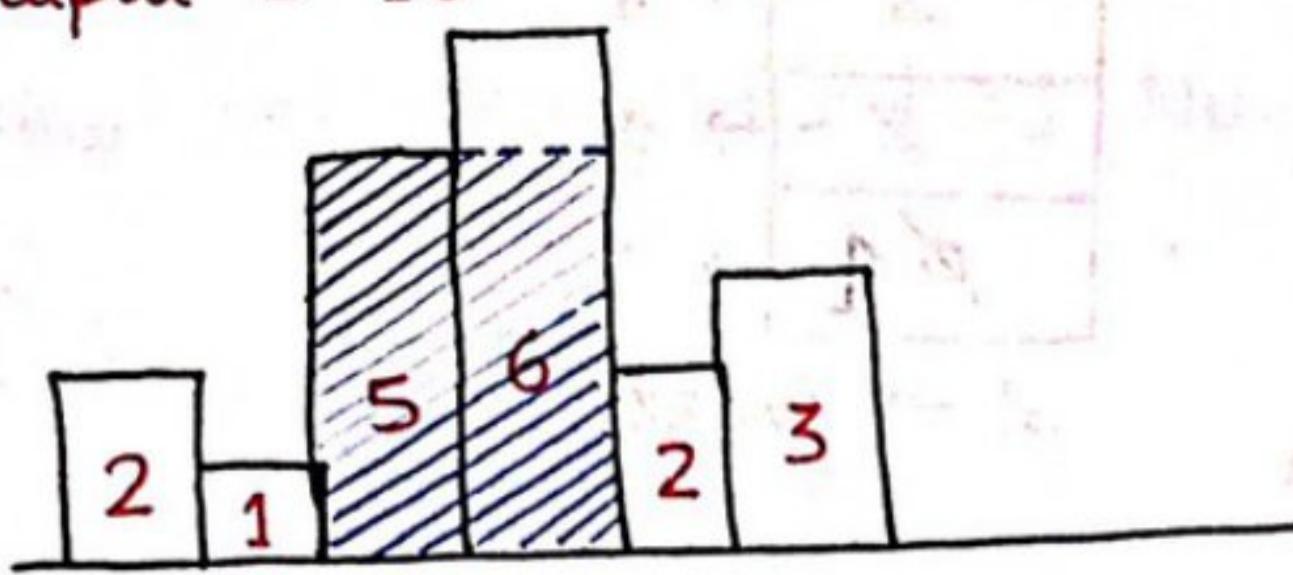
3 → (2, 25)

(5, 50)
(4, 40)
(2, 25)

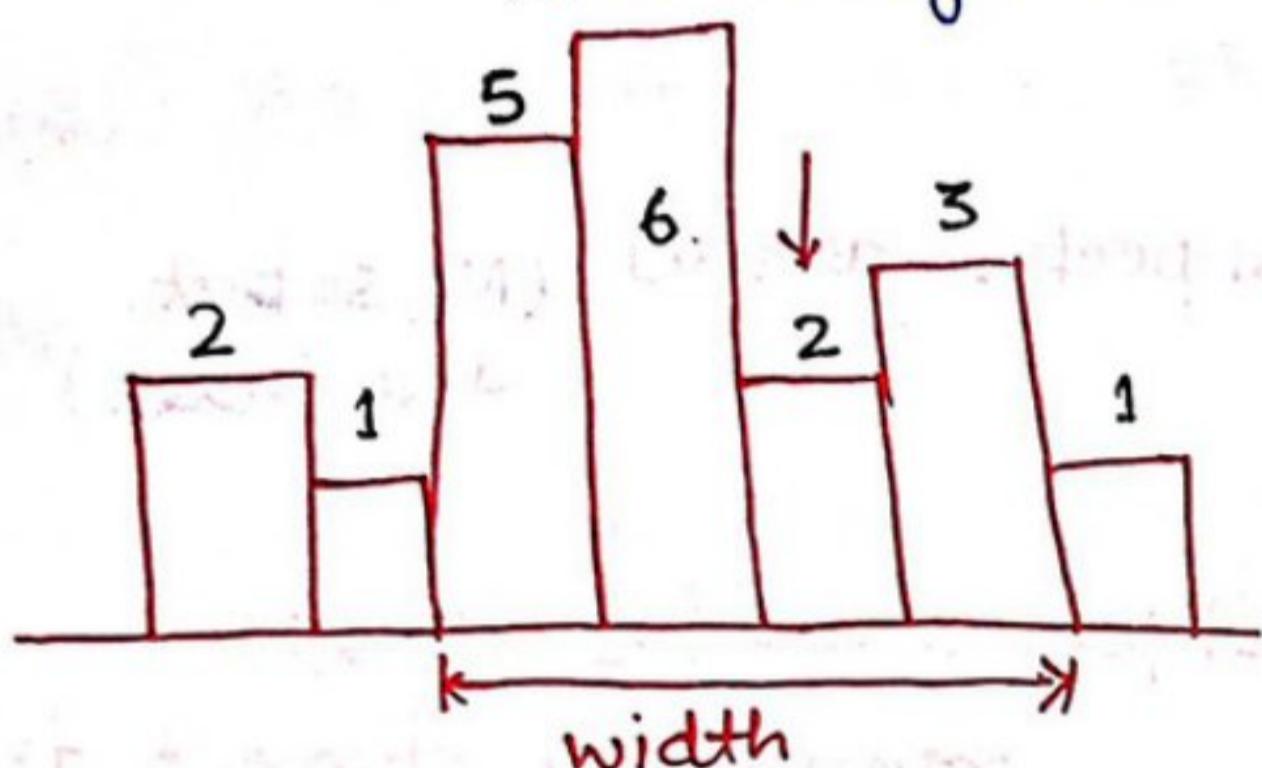
## Largest Rectangle in a histogram - Problem - 04

Input  $\Rightarrow$  heights = [2, 1, 5, 6, 2, 3]

Output = 10



Total rectangle len = 10, width of each flight = 1



### Brute force approach:

find the right smaller and left smaller element and find the largest rectangle area of the Histogram.

main (arr, n) {

    int maxArea = 0;

    for (i = 0; i < n; i++) {

        int minHeight = INT\_MAX;

        for (j = i; j < n; j++) {

            minHeight = Math.min (minHeight, arr[j]);

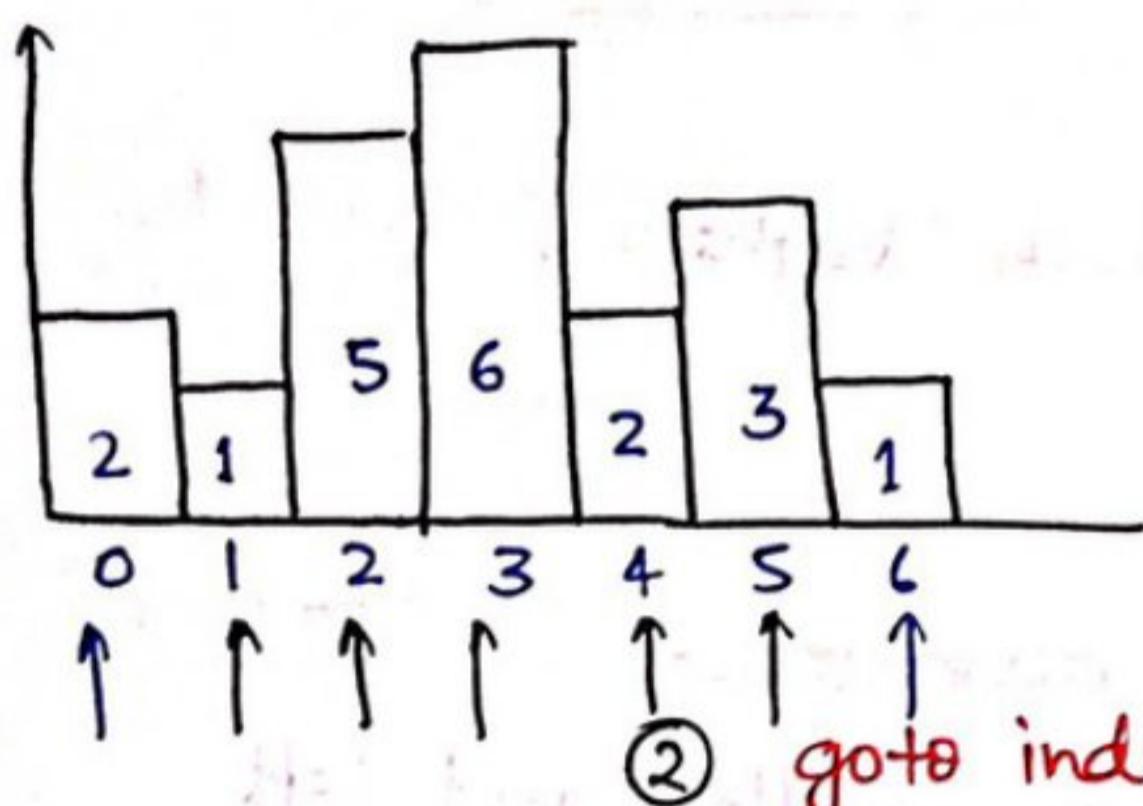
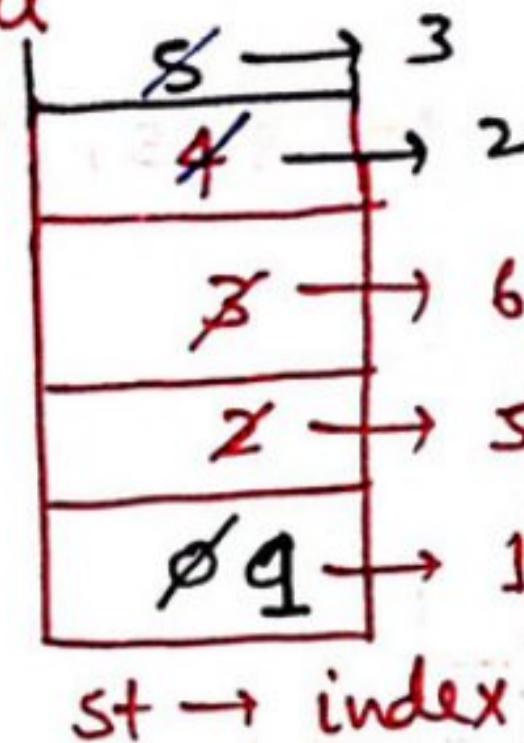
        maxArea = Math.max (maxArea, minHeight \* (j - i + 1));

Time complexity -  $O(N^2)$

Optimised Approach: The intuition behind the approach is the same as finding the smaller element on the both side but in an optimised way using the concept of next greater element for next smaller element.

0	0	2	3	2	5	0
0	1	2	3	4	5	6

left smaller



- ① check  $st.\text{peek} > arr[0]$  (No, so put the index)

- ③ goto  $\text{idx}-1$  and

Now  $st.\text{peek} > arr[1]$  (No)  
and put the idx into stack.

$st.\text{peek} > arr[1]$  (No, so remove the element in  
and put the current  $\text{idx}$ .)

- ④ goto  $\text{idx} \rightarrow 4$

$st.\text{peek} > arr[4]$  (Yes) remove 3, 2 from stack  
because they both are greater

right smaller -

0	6	3	3	5	5	6
0	1	2	3	4	5	6

- ① On  $\text{idx} = 6$   
 $st$  is empty put the curr  $\text{idx}$  in the array. and put 6 inst.

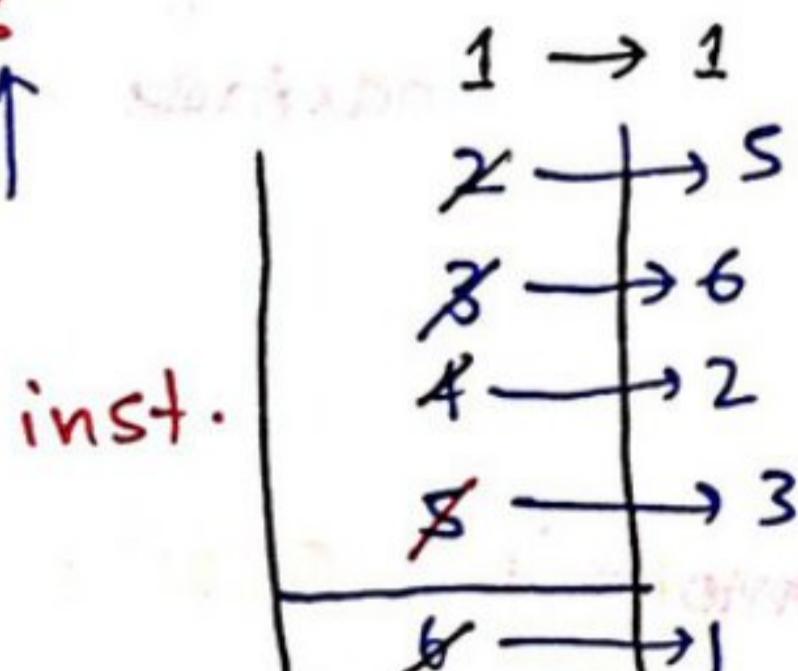
- ② On  $\text{idx} = 5$

$st.\text{peek} > arr[5]$ , No

put the  $\text{idx}$  into stack and store the  $\text{idx}$  in the array.

- ③ No when we go to  $\text{idx} = 4$  val 2.

$arr[st.\text{peek}] > arr[4]$  Yes, so we have to remove the value from  $st$ . and put the  $\text{idx}$  in  $arr$ .



and stack.

Now 6 is the boundary so put  $6-1=5$  in the arr.  
because 6th idx is smaller.

Now  $st \cdot peek > arr[3]$  No so put it into st and arr

Now  
go to 2

$st \cdot peek > arr[2]$ , Yes so remove the 3 from stack.

Now after this for  $idx-1$  st will empty so put the last idx.  
for 0th index -

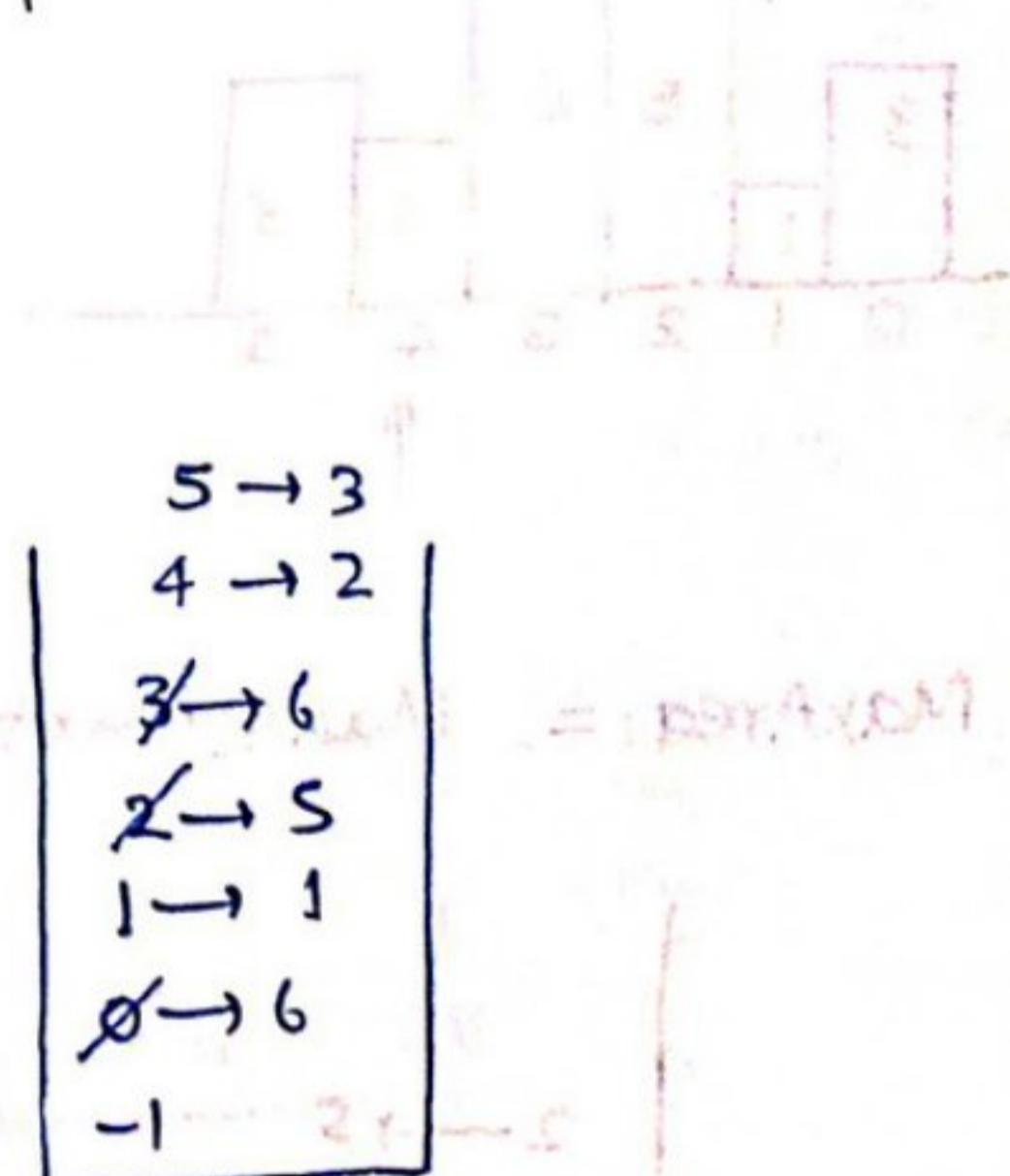
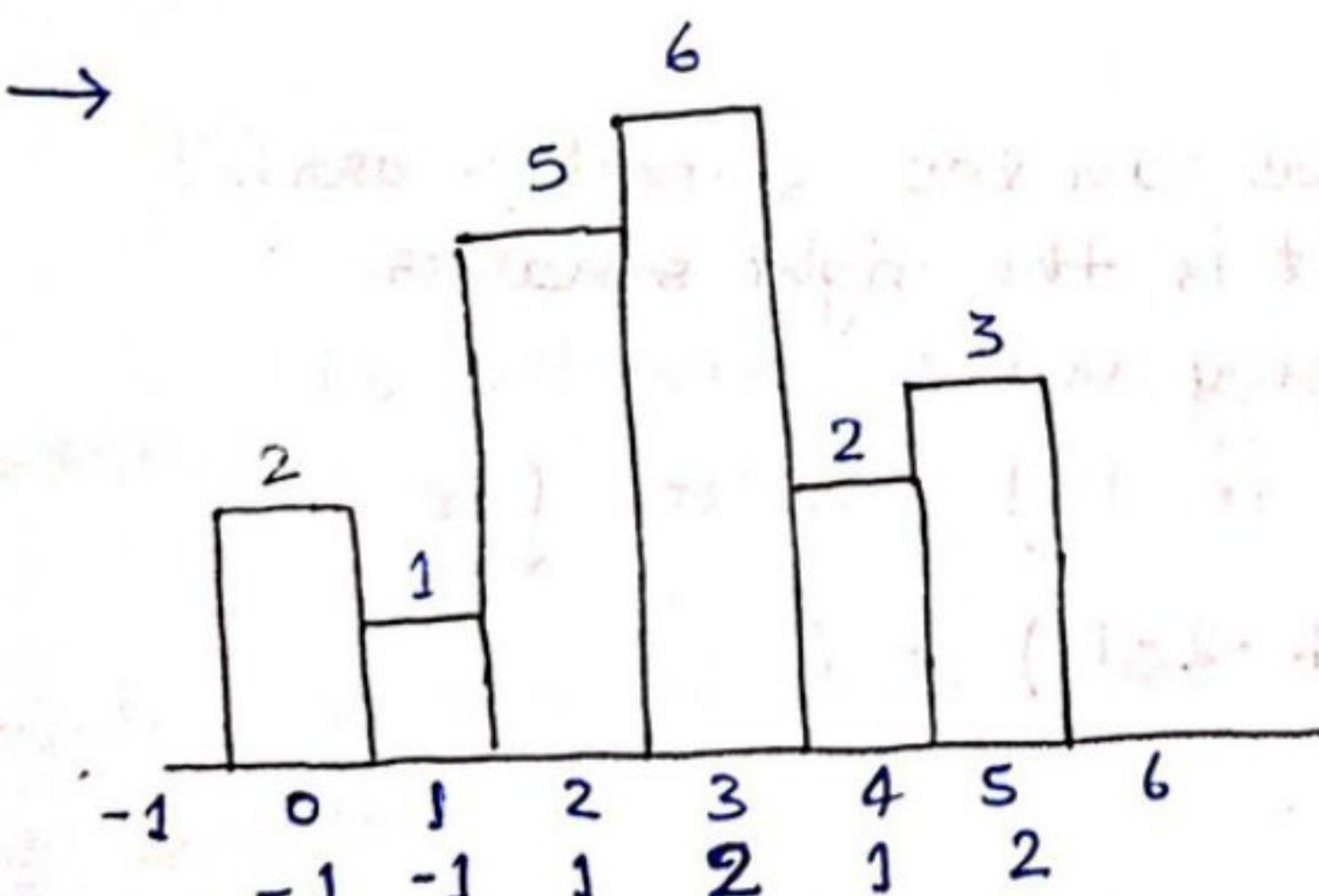
$$\frac{(\text{right smaller} - \text{left smaller}) * a[i]}{\text{idx}}$$

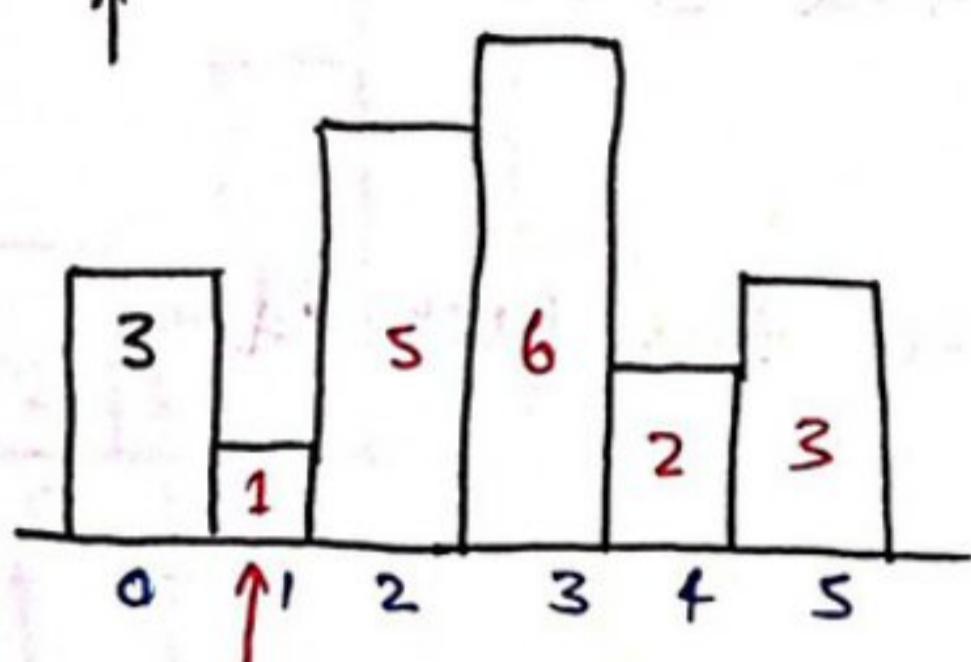
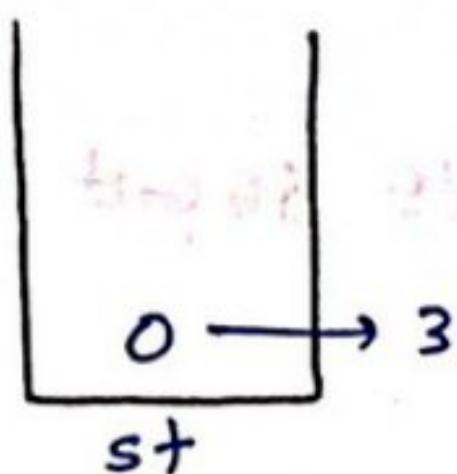
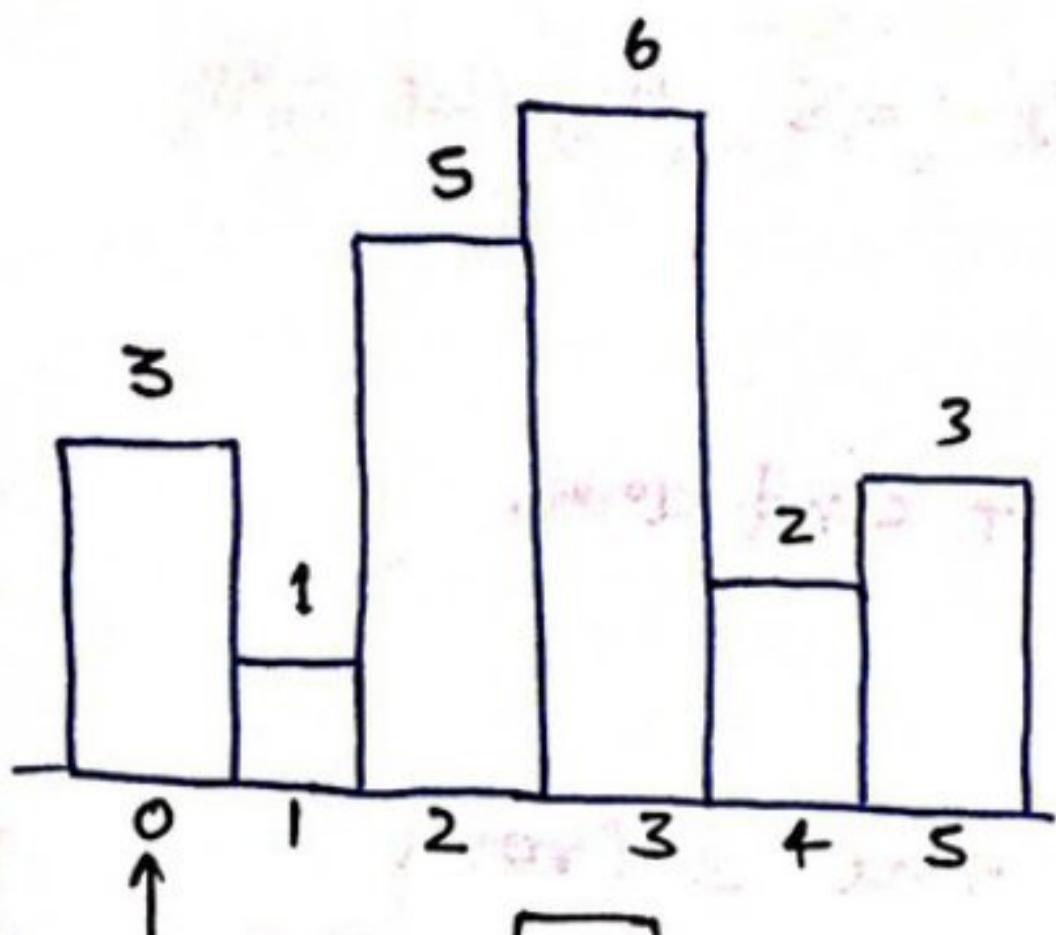
$$Tc = O(N) + O(N) \approx O(N)$$

$$Sc = O(3N)$$

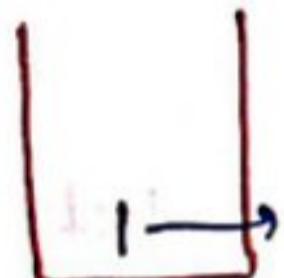
### Optimised solution - 2

- pop → bigger or same value
- ans → push self idx.

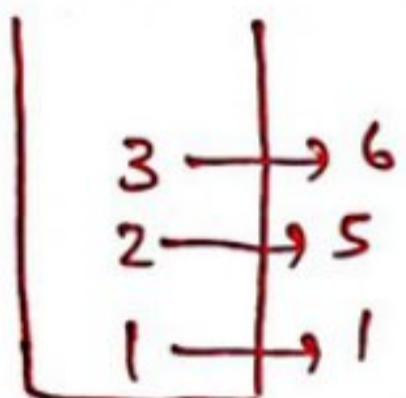




current val is greater than st.peek  
so pop it from st and put the curr idx.

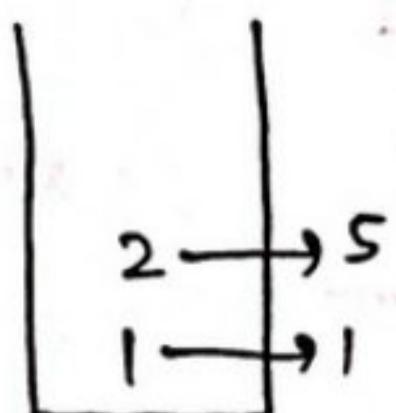


idx 2 and 3 val is greater than arr[st.peek] so simply put it into st



Now we can see  $st.peek > arr[i]$   
mean it is the right smaller.  
so popping out 6 from the st.  
and 5 is left smaller for 6.

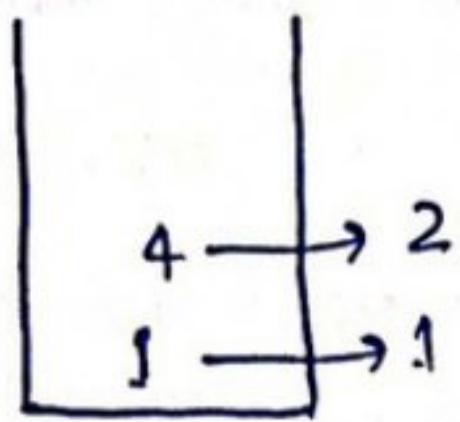
$$\text{MaxArea} = \text{Max}((\text{MaxArea}, (6 * (4 - 2 - 1))) = 6$$



again  $st.peek > arr[i]$

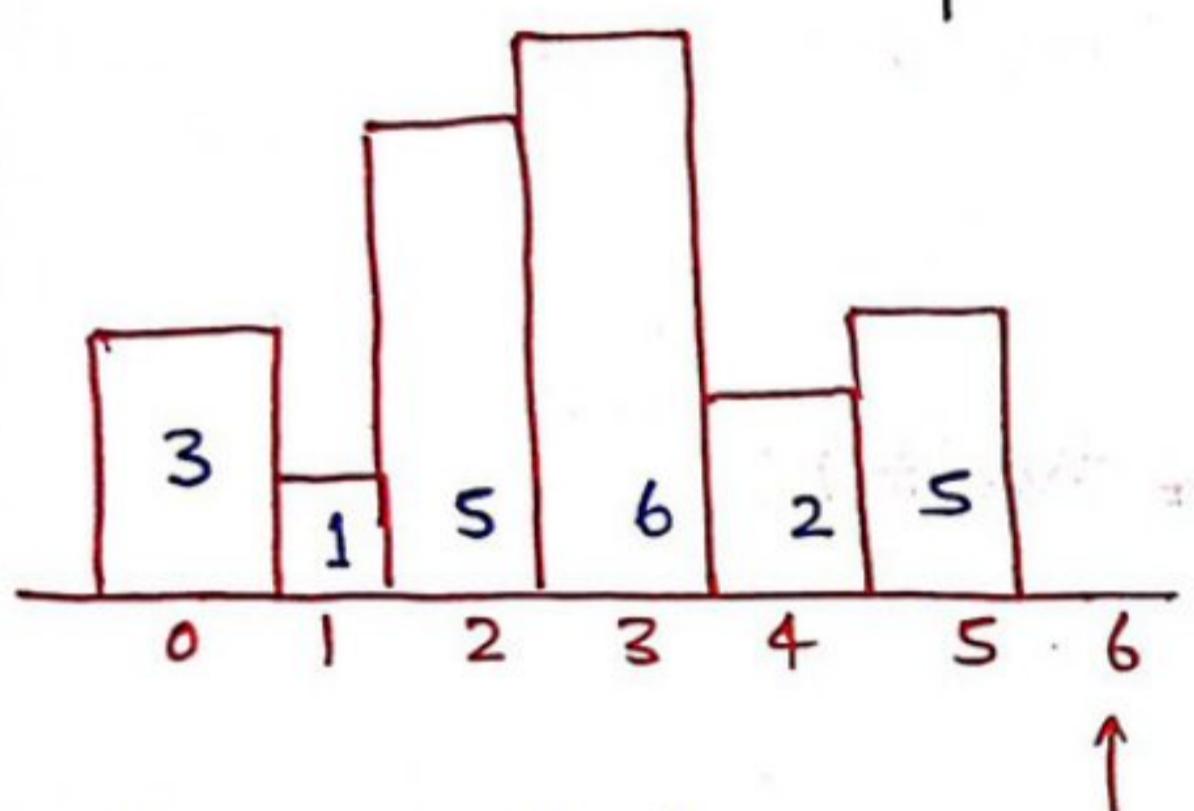
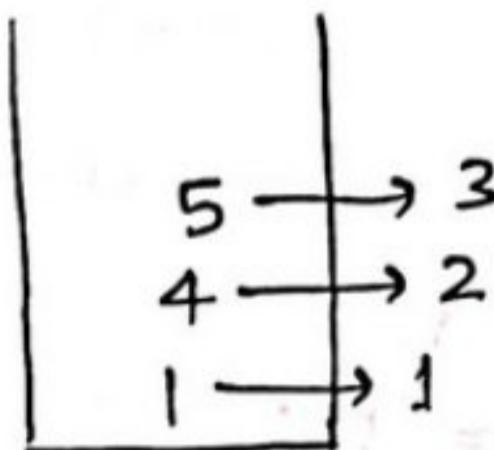
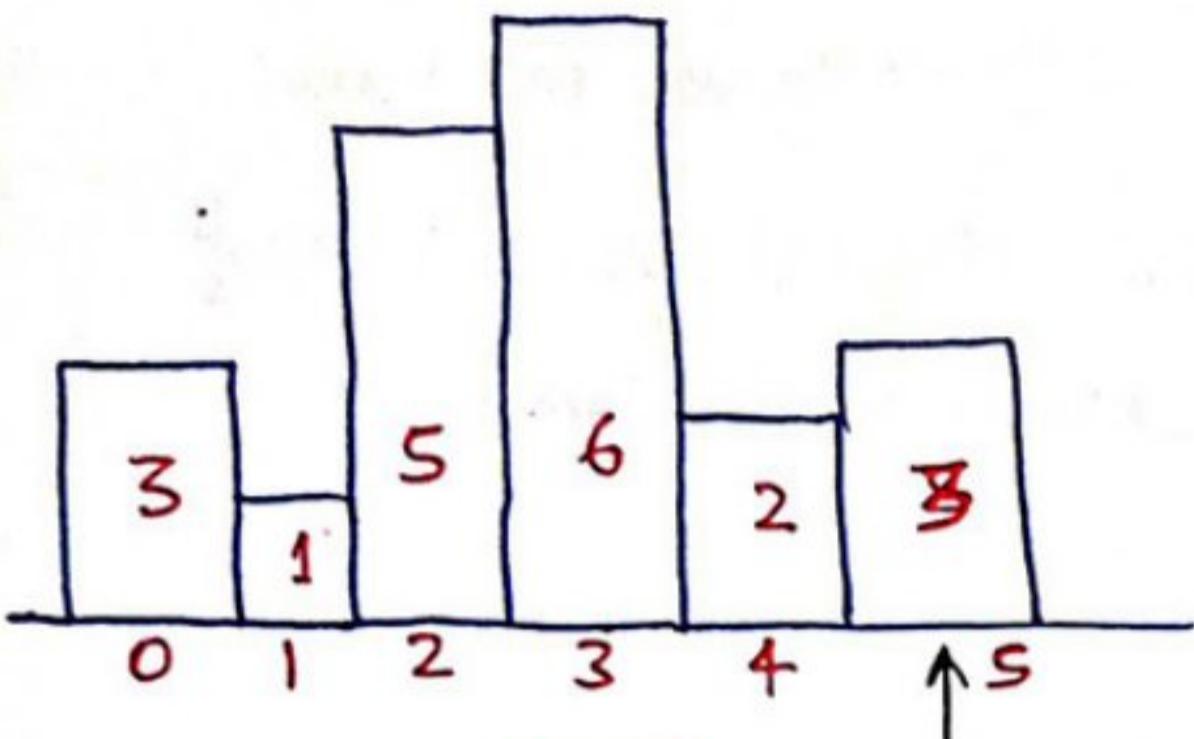
Since 2 is smaller than 5 so it means it is right smaller and popping out 5 from the stack, 1 is left smaller for 5

$$\text{MaxArea} = \max(6, (5 * (4 - 1 - 1)) = 10$$

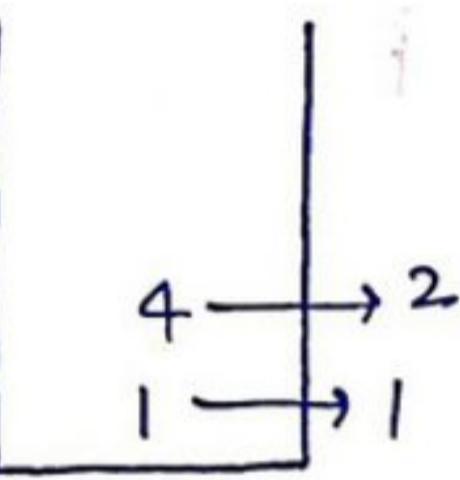


Now 1 is smaller than 2 so we directly

push 2.



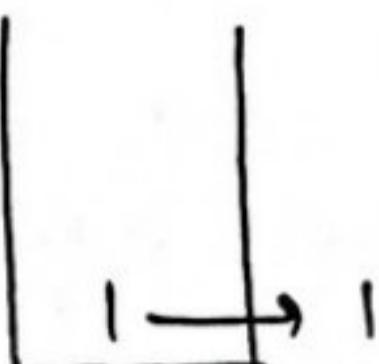
Now we will do one more iteration and consider 6 as a right smaller and so pop out 3 and consider 2 as left smaller.



$$\text{MaxArea} = (10, (3 * (6 - 4 - 1)) = 10$$

consider 6 is right smaller for 4 so pop 4 and 1 is left smaller.

$$\text{MaxArea} = 2 * (6 - 1 - 1) = 8 , \quad \underline{\text{MaxArea} = 10}$$



Now pop out 1 6 is the right smaller and 0 is the left smaller

$$\text{MaxArea} = (50, 1 * (6 - 0 - 1)) = 10$$

```
main(ARR){
```

```
Stack<int> st;
```

```
int MaxA = 0;
```

```
for(i=0; i < n; i++) {
```

```
    while(!st.isEmpty() && (i == n || arr[st.peek()] >= arr[i])) {
```

```
        int height = arr[st.peek()];
```

```
        st.pop();
```

```
        int width;
```

```
        if(st.isEmpty()) {
```

```
            width = i;
```

```
}
```

```
        else {
```

```
            width = i - st.peek() - 1;
```

```
}
```

```
        maxA = Math.max(maxA, width * height);
```

```
}
```

```
        st.push(i);
```

```
}
```

```
return MaxA;
```

```
}
```

Time complexity -  $O(N) + O(N)$

Space -  $O(N)$

## Sliding window Maximum : Problem - 05

We use the intuition of NGE here.

main(arr) {

Stack<int> st();

int[] nge = new int[arr.length];

st.push(arr.length - 1);

nge[arr.length - 1] = arr.length;

for(i = arr.length - 2; i >= 0; i--) {

while(st.size() > 0 && arr[i] >= arr[st.peek()]) {

st.pop();

if(st.isEmpty()) {

nge[i] = arr.length;

else {

nge[i] = st.peek();

}

st.push(i);

}

for(i = 0; i < arr.length - k; i++) {

int j = i;

while(nge[j] < i + k) {

j = nge[j];

}

print(arr[j]);

}

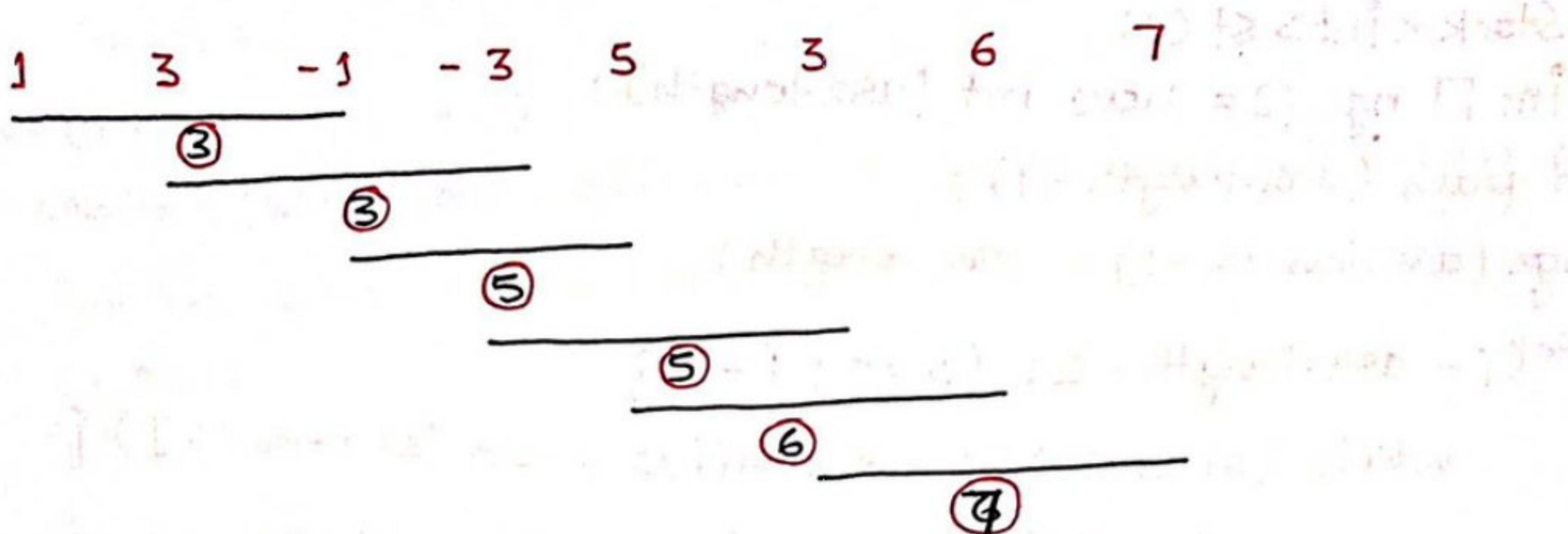
Time complexity -  $O(N) + O(N)$

Space complexity -  $O(2N)$ .

## Approach-2

nums = [1, 3, -1, -3, 5, 3, 6, 7], k=3

Output = [3, 3, 5, 5, 6, 7]



Naive solution - traverse from 0 to  $n-k$  so we can get the max in the every window.

Now firstly traverse from 0 to 2 and find the max from it and put into the array.

```

for(i = 0 → s) {
    maxi = arr[i];
    for(j = i+k-1) {
        maxi = max(arr[j], maxi)
    }
}
    
```

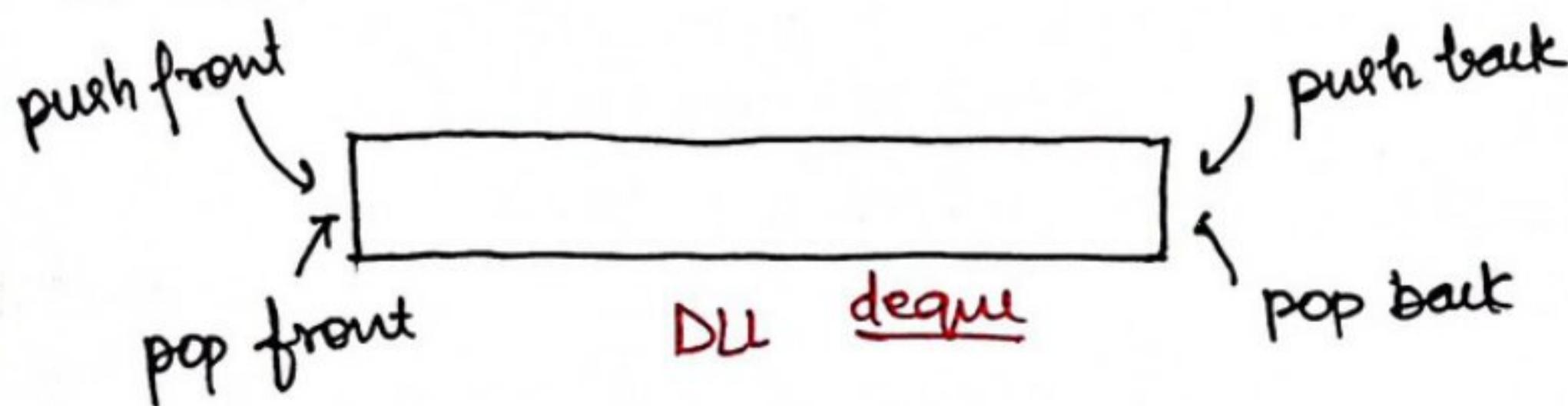
Time complexity -  $O(N^2)$

Space -  $O(1)$

## Optimised Approach

We use the concept of NGE but over here we store the elements in decreasing order. So we need a deque.

It allows



1	3	-1	-3	5	3	6	7
---	---	----	----	---	---	---	---

idx 0 1 2 3 4 5 6 7



In the 0th idx check does the deque has anything  $\rightarrow$  No. So push back the val 0

0	
---	--

then go to idx 1.

so now compare that peek element of dq, which is 1 is less than 3.

1	3	2	5	6
---	---	---	---	---

$$1 < 3$$

which can be my

answer.

take this 0 out after this dq, be empty so push 1 into dq.

∅	1	
---	---	--

Move to the 2nd idx. with val -1

Now  $dq.\text{peek} > arr[i]$  so no need to pop because we are maintaining the decreasing order

∅	1	2
---	---	---

Now we can see 2 is the last index of the subarray of size K.

So, if i look from the front the max element will be on the front.

Now pointer will move on to 3rd idx and if we check  $arr[dq.\text{peek}] > arr[i]$  so we can push into dq maintaining the decreasing order.

3	
---	--

ans

1	2	3
---	---	---

Now we have one more subarray of size 3.

3	3	
---	---	--

ans

Now move to the 4th idx.

0	1	2	3	4	5	6	7
1	3	-1	-3	5	3	6	7

Now my boundary is 2 to 4 but dq. contains 3 so we have to pop it.

3	3	
---	---	--

Now arr[dq.peek] < arr[i] so we have to pop 3 from dq.

X	2	3	4
---	---	---	---

this 3 to 5 my boundary  
so check is there any idx in the  
dq out of the boundary → No

3	3	5	5	
---	---	---	---	--

also arr[dq.peek] > arr[i] so push

4	5
---	---

Now boundary 4-6 No element in the dq is out of boundary.  
but. dq.peek < arr[i] pop both elements.

4	5	6	
---	---	---	--

4	5	6	
---	---	---	--

3	3	5	5	6	7
---	---	---	---	---	---

Move to 7 and boundary 5-7  
everything in the dq. is in the boundary.

but. dq.peek < arr[i] → pop 6.

6	7
---	---

check from front so it 7 and  
correspond to 7 so push it into the array.

6	7	8
---	---	---

Time complexity -  $O(N) + O(N) \approx O(N)$

Sc -  $O(K)$

main (arr, k) {

int si = 0;

int arr[] = new int [n-k+1];

Deque<int> dq;

for (i=0; i < a.length; i++) {

if (!dq.isEmpty() && dq.peek() == i-k) {

dq.poll();

// remove numbers  
out of range k.

// remove smaller numbers in k range as they are useless.

while (!dq.isEmpty() && arr[dq.peekLast()] < arr[i])

dq.pollLast();

dq.offer(i);

if (i >= k-1) {

arr[si++ ] = a[dq.peek()];

}

}

return arr;

// any i which is greater than k-1  
be the max in the window because  
it will be max because we are  
storing everything in decreasing  
order.

L2

L3

L4

L5

L6

L7

L8

L9

L10

L11

L12

L13

L14

L15

L16

L17

L18

L19

L20

L21

L22

L23

L24

L25

L26

L27

L28

L29

L30

L31

L32

L33

L34

L35

L36

L37

L38

L39

L40

L41

L42

L43

L44

L45

L46

L47

L48

L49

L50

L51

L52

L53

L54

L55

L56

L57

L58

L59

L60

L61

L62

L63

L64

L65

L66

L67

L68

L69

L70

L71

L72

L73

L74

L75

L76

L77

L78

L79

L80

L81

L82

L83

L84

L85

L86

L87

L88

L89

L90

L91

L92

L93

L94

L95

L96

L97

L98

L99

L100

L101

L102

L103

L104

L105

L106

L107

L108

L109

L110

L111

L112

L113

L114

L115

L116

L117

L118

L119

L120

L121

L122

L123

L124

L125

L126

L127

L128

L129

L130

L131

L132

L133

L134

L135

L136

L137

L138

L139

L140

L141

L142

L143

L144

L145

L146

L147

L148

L149

L150

L151

L152

L153

L154

L155

L156

L157

L158

L159

L160

L161

L162

L163

L164

L165

L166

L167

L168

L169

L170

L171

L172

L173

L174

L175

L176

L177

L178

L179

L180

L181

L182

L183

L184

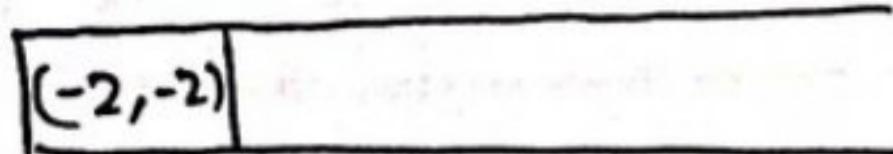
L185

L186

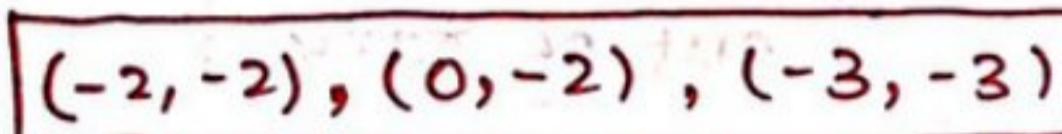
L187

## Implement Min Stack :-

push (-2)	getMin()
push (0)	pop()
push (-3)	getMin()



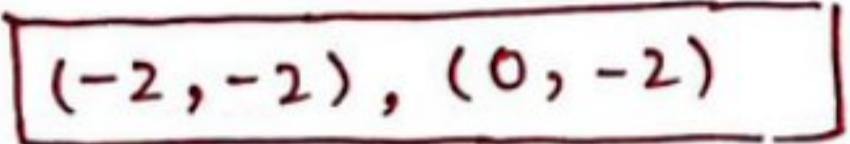
Bruteforce - The Naive approach is we have to push the pair and val, min element till now on the stack.



we push all the elements.

Now  $\text{getMin}() = -3$

if pop so st.



Tc - O(1)

Sc - O(2N)

$\text{getMin}() = -2$

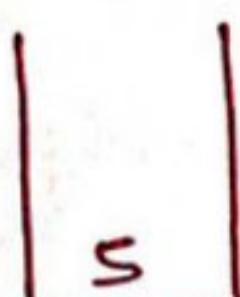
## Optimal solution -

push (5)

push (3)

initially min Element will be max set

first we push 5 into the stack because it is empty.



minElement = 5

Now we have to push 3 so there can arrive two conditions.

$3 \geq \text{minElement}$

$\downarrow$   
current element

so directly push the cur  
val into the stack

$||$      $3 < \text{minElement}$

$\downarrow$   
if yes

so push  $2 \times \text{val} - \text{minElement}$   
into stack and update the  
val.

3 &lt; 5

Use  $\min$  two longer  $\rightarrow n$ 

so -

 $2 \times 3 - 5 = 1$  so we have to push 1 into the

stack.



$$\minElement = 8 / 3$$

Now if pop operation call so we have to check  
 $st \cdot peek < \minElement$  so its means it was a flag  
 that was pushed by us and its mean the min  
 element will change after pop operation

$$st \cdot peek < \minElement$$



$$\minElement = 2 \times \minElement - st \cdot peek$$

Stack&lt;long&gt; st;

Long min;

public MinStack() {

    min = Long.MAX\_VALUE;  
}

① void push(int val) {

Long x = Long.valueOf(val);

    if ( $st \cdot size == 0$ ) st.push(x);

min = x;

    else if ( $val \geq min$ ) {

st.push(x);

} else {

        st.push( $2 * x - min$ );

min = x;

}

}

② void pop() {

    if ( $st \cdot size == 0$ ) return;

Long val = st.pop();

    if ( $val < min$ ) {        min =  $2 * min - val$ ;

}

③ int top() {

Long val = st.peek();

    if ( $val < min$ ) {

return min.intValue();

}

return val.intValue();

}

## Rotting Oranges :- Problem - 07

given  $\rightarrow \frac{m \times n}{\text{size}}$  Matrix

0  $\rightarrow$  represent empty cell

1  $\rightarrow$  fresh orange

2  $\rightarrow$  Rotten orange

Every min any fresh orange that is 4-directionally adjacent to a rotten orange becomes rotten. Return the min no of min to rot all the orange. if impossible -1.

2	1	1
1	1	0
0	0	1

min 0

initially 2 is rotten orange

2	2	1
2	1	0
0	1	1

min 1

min 2

min 3

2	2	2
2	2	0
0	1	1

2	2	2
2	2	0
0	2	1

min Time = 4 minutes.

2	2	2
2	2	0
0	2	2

min 4

let's take a example to think the intuition -

0	1	2
0	0	2
1	0	2
2	2	2

(1,1)
(2,2)
(1,2)
(0,2)
(2,1)
(0,1)
(1,0)

Q

cnt = 2

$$x = 0 + 2 = 2$$

$$y = 1 + 0 = 1$$

$$x = 2 + 0 = 2 \rightarrow x$$

$$y = 0 - 1 = -1$$

$$\begin{aligned}x &= 2 + 1 = 3 \\y &= 0 + 0 = 0\end{aligned}\quad ]_X$$

$$x = 2 + (-1) = 1$$

$$y = 0 + (0) = 0$$

because  $q[0][1][0] == 0$  no change

$\min = 1$

$$\begin{aligned}x &= 0 + 0 = 0 \\y &= 2 + 1 = 3\end{aligned}\quad ]_X$$

$$\begin{aligned}x &= 0 + 0 = 0 \\y &= 2 - 1 = 1\end{aligned}\quad ] \checkmark \text{ push}(0, 1) \text{ into } q.$$

$$\begin{aligned}x &= 0 + 1 = 1 \\y &= 2 + 0 = 2\end{aligned}\quad ] \text{ push}(1, 2) \text{ into } q.$$

$$\begin{aligned}x &= 0 - 1 = -1 \\y &= 2 + 0 = 2\end{aligned}\quad ] \quad x \text{ } \textcircled{0} \text{ } -1 < 0$$

$$\begin{aligned}cut &= cut + q.size \\&= 2 + 3 = 5\end{aligned}$$

$$\begin{aligned}x &= 2 + 0 = 2 \\y &= 1 + 1 = 2\end{aligned}\quad ] \text{ push}(2, 2) \text{ into } q.$$

$$\begin{aligned}x &= 2 + 0 = 2 \\y &= 1 - 1 = 0\end{aligned}\quad ] \quad x \text{ already vis.}$$

$$\begin{aligned}x &= 2 + (1) = 3 \\y &= 1 + (0) = 0\end{aligned}\quad ]_X$$

$$\begin{aligned}x &= 2 - 1 = 1 \\y &= 1 + 0 = 0\end{aligned}\quad ] \quad x \text{ zero}$$

$\text{pop} \rightarrow 0, 1$

$$\begin{array}{l} x = 0 + 0 = 0 \\ y = 1 + 1 = 2 \end{array} ] \quad \text{already present}$$

$$\begin{array}{l} x = 0 + 0 = 0 \\ y = 1 - 1 = 0 \end{array} ] \quad \text{Empty cell.}$$

$$\begin{array}{l} x = 0 + 1 = 1 \\ y = 0 + 0 = 0 \end{array} ] \quad \text{put it } (1, 1) \text{ into } q.$$

Similarly.  $(1, 2) \rightarrow$  change pop and all the elements will pop so it will take 2 min to rot all the oranges..

2 min to  
ans

```
main(grid[][]) {
    if (grid == null || grid.length == 0) return 0;
    int n = grid.length;
    int m = grid[0].length;
    Queue<int[]> q = new LinkedList<>();
    int cut-fresh = 0;
    for (i=0; i<n; i++) {
        for (j=0; j<m; j++) {
            if (grid[i][j] == 2) {
                q.offer(new int[] {i, j});
            }
            if (grid[i][j] == 0) {
                cut-fresh++;
            }
        }
    }
}
```

if (fresh\\_cut == 0) return 0;

BO - moldoff - may2 sheet 3 min 10

int cntMin = 0, cnt = 0;

int dx[] = {0, 0, 1, -1};

int dy[] = {1, -1, 0, 0};

while (!q.isEmpty()) {

int size = q.size();

cnt += size;

for (i = 0; i < size; i++) { int[] point = q.poll();

for (j = 0; j < 4; j++) {

int x = point[0] + dx[j];

int y = point[1] + dy[j];

if (x < 0 || y < 0 || x >= n || y >= m || grid[x][y] == 0 ||

grid[x][y] == 2) continue;

grid[x][y] = 2;

q.offer(new int[] {x, y});

return cnt - fresh == cnt ? cntMin : -1;

Time complexity -  $O(n \times n) * 4$

Space complexity -  $O(n \times n)$ .

## Online Stock Span - Problem - 08

arr: 

100	80	60	70	60	75	85
-----	----	----	----	----	----	----

consecutive smaller  
or equal

→ Before it.

op: 

1	1	1	2	1	4	6
---	---	---	---	---	---	---

like if we are on 75  
so check on the right 60, 70, 60 is less than 75  
also there are in consecutive order so including 75 it will  
be 4.

Nearest greater to left.



this problem is similar to this.

0	1	2	3	4	5	6
100	80	60	70	60	75	85

If we are in the fifth idx ↑ so nearest greater to  
its left will be 80 whose idx is 1  

$$80 \quad 5 - 1 = 4$$
 that will be our ans.

- ① Create an array
- ② Create a stack.
- ③ Use the algo of Nearest greater to left.

Time complexity -  $O(N)$

Space complexity -  $O(2N)$ .

## Celebrity Problem - Problem - 09

Celebrity

- ↳ known by everybody
- ↳ knows nobody

	0	1	2	3	4
0	0	1	1	1	1
1	1	0	0	1	0
2	1	0	0	1	0
3	OK	OK	OK	0	OK
4	0	1	0	1	0

so 3 know nobody and 3 is known by everybody so its celebrity.

there can't be two celebrities and also its is possible that no body is celebrity.

3
2
1
0

pop two elements.

2      3

	0	1	2	3
0	X	✓	✓	✓
1	✓	X	✓	X
2	X	X	X	X
3	✓	✓	✓	X

then check 2 is known by 3 , No

and check 3 is known by 2 , No

we will eliminate one of them so, because.

2 is known by 3 so 3 can't be a celebrity

so 2 push into the st.

2
1
0

Now pop 1 and 2

check 1 knows 2 (yes) so it can't be a celebrity.

push 2 into st

2
0

Now pop 0 and 2

0 knows 2 (yes)

but 2 doesn't know 0 so push 2 into the stack, now st.size == 1 so 2 is a celebrity.

```
main (arr[][]) {
```

```
    Stack<int> st;
```

```
    for (i=0; i<n; i++) {
```

```
        st.push(i);
```

```
    while (st.size() >= 2) {
```

```
        int i = st.pop();
```

```
        int j = st.pop();
```

```
        if (arr[i][j] == 1) {
```

```
            st.push(j); // if i knows j → i is not a celebrity
```

```
}
```

```
    else {
```

```
        st.push(i);
```

```
}
```

```
}
```

```
    int pot = st.pop();
```

```
    for (i=0; i<n; i++) {
```

```
        if (i == pot) {
```

```
            if (arr[i][pot] == 0 || arr[pot][i] == 1)
```

```
                return -1;
```

```
}
```

```
}
```

```
return pot;
```

Tc - O(N)

Sc - O(N)

## Reverse words in a String : Problem - 90

Input:  $s = \text{"The sky is blue"}$

Output: "blue is sky The"

We can follow the Brute-force initially. use a stack.  
and if there is a character add it into the string and if  
there is any space put it into the stack and initialize  $\text{str} = ""$   
again.

$\text{st} = \text{" "}$  → Because we want to add the last  
string as well.

```
Stack<String> st();
```

```
String str = "";
```

```
for (i=0; i < s.length; i++) {
```

```
    if (s.charAt(i) == ' ') {
```

```
        st.push(str);
```

```
        str = " ";
```

```
}
```

```
else {
```

```
    str = s.charAt(i);
```

```
}
```

```
}
```

```
String ans = "";
```

```
while (st.size() != 1) { // otherwise it adds a empty all. in the
```

```
ans = ans + st.peek() + " "; // last.
```

```
st.pop();
```

```
}
```

```
ans += ans + st.peek(); → // finally add the last element in  
return ans; the string.
```

Time complexity -  $O(N)$

Space complexity -  $O(N)$ .

## Optimal Approach

- Start traversing the String from the end until we hit a space. It means we have gonna part a word.
- Check if ans variable is empty or not.
- If it's empty, it indicates that it is the last words we need to print.
- If it is not empty we add it to result with a space after it.

main fun(s){

```
int left = 0, right = s.length() - 1;  
String temp = ""; → for storing substring.  
String ans = ""; → for storing final string.
```

```
while (left <= right) {
```

```
    char ch = s.charAt(left);
```

```
    if (ch != ' ') {
```

```
        temp += ch;  
    }
```

```
    else if (ch == ' ') {
```

```
        if (!ans.equals(" ")) {
```

```
            ans = temp + " " + ans;
```

```
}
```

```
else {
```

```
    ans = temp;
```

```
}
```

```
}temp = "";
```

```
}
```

```
left++;
```

```
}
```

Tc - O(N)

Sc - O(1)

// for adding last word

```
if (!temp.equals(" ")) {
```

```
    if (!ans.equals(" "))
```

```
        ans = temp + " " + ans;
```

```
    else ans = temp;
```

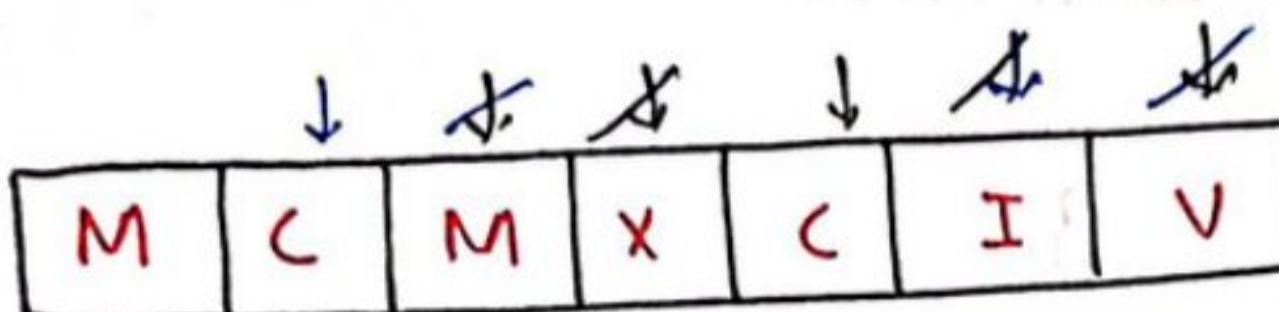
```
}
```

```
return ans;
```

## Roman to Integer : Problem-91

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

- I can be placed for V(5) and X(10) to make 4 and 9.
- X can be placed before L(50) and C(100) to make 40 and 90.
- C can be placed before D(500) and M(1000) to make 400 and 900.



- Start traverse from end initially

$$\text{res} = V, 80 \quad \boxed{\text{res} = 5}$$

then go to I

Now currently we are in I which is 1

and res is 5 mean value of (I) < res.

so simply subtract.

$$\text{res} = \text{res} - \text{val of (I)}$$

$$\boxed{\text{res} = 5 - 1 = 4}$$

the goto C -

on val of (C) which is 100 > res. so we have to add it to the result.

$$\begin{aligned} \text{res} &= \text{res} + \text{val of (C)} \\ &= 100 + 4 \\ &= 104 \end{aligned}$$

the goto X whose val is 10 > res we need to subtract val of (X) < res.

$$\begin{aligned} \text{res} &= \text{res} - \text{val of (X)} \\ &= 104 - 10 \\ &= 94 \end{aligned}$$

then goto M so res = 1094  
then goto C so res = 994

on M res = 1994

```

Map<Character, int> map;
put I, V, X, L, C, D, M with its value manually.

int res = map.get(s.charAt(s.length() - 1));
for (i = s.length() - 2; i >= 0; i++) {
    if (map.get(s.charAt(i)) < map.get(s.charAt(i + 1))) {
        result -= map.get(s.charAt(i));
    } else {
        result += map.get(s.charAt(i));
    }
}
return result;

```

Time complexity -  $O(N)$

Space complexity -  $O(N)$

## String to Integer: Problem 92

atoi algorithm → • read in and ignore leading white space.

- check for +, - if it is met in the last.
- Ignore the non digit character and return the one.
- convert "123" → 123  
"0032" → 32.
- 32 bit signed integer range  $[-2^{31}, 2^{31} - 1]$ . then clamp the integer so it remains in the range.

```

main fun (String s) {
    if (s == null) return 0;
    s = s.trim();
    if (s.length() == 0) return 0;

```

```

int sign = +1;
long ans = 0;
if (s.charAt(0) == '-' ) sign = -1;
int MAX = Integer.MAX-VAL;
int MIN = Integer.MIN;
int i = (s.charAt(0) == '+' || s.charAt(0) == '-') ? 1 : 0;
while (i < s.length()) {
    if (s.charAt(i) == ' ' || !Character.isDigit(s.charAt(i))) break;
    ans = ans * 10 + s.charAt(i) - '0';
    // Check
    if (sign == -1 && -1 * ans < MIN) return MIN;
    if (sign == +1 && ans > MAX) return MAX;
    i++;
}
return (int)(sign * ans);

```

Time complexity -  $O(N)$ .

Space complexity -  $O(1)$ .

## Longest common prefix - Problem - 93

Str = ["Flower", "flow", "flight"]

O/P = "Fl"

Sort the string and compare the first and last string, so we able to find the common prefix.

Why sorting - ? The reason why we sort the input array and compare first and last strings is that the longest common prefix of all the string must be a prefix of first and last.

This is because strings are based on their alphabetical order.

like if we have -

[flower, flow, apple, flight, food]

Sort - [apple, flower, flow, flight, good]

we can clearly see the max diff between the character is between 1st and last string.

```
main fun( String [] str) {  
    Arrays.sort(str);  
    String s1 = str[0];  
    String s2 = str [str.length - 1];  
    int idx = 0;  
    while (idx < s1.length() && idx < s2.length()) {  
        if (s1.charAt(i) == s2.charAt(i)) {  
            idx++;  
        }  
        else  
            break;  
    }  
    return s1.substring(0, idx);  
}
```

Time complexity -  $O(N \log N) + O(M)$

Space complexity -  $O(1)$

## Pattern searching | Rolling hash | Robin Karp - Problem - 94

$a = "abcd"$ ,  $b = "cdabcdab"$

$O(p) = 3$ .

given two strings  $a$  and  $b$ , return the min number of times you should repeat string  $a$  so that  $b$  is substring of  $a$ . if impossible  $\infty - 1$ .

firstly look at Robin Karp:

so we have a given text and pattern we have to check the pattern is a substring of text or not.

text  $\rightarrow a \rightarrow b \rightarrow x \rightarrow a \rightarrow b \rightarrow c$

pattern  $\rightarrow a \rightarrow b \rightarrow c$

Brute Force-

Compare the text and pattern characters one

by one.

① first both matched the character  $a$  and  $a$ .

② Now move to pointer to next idx.

text  $\rightarrow a \rightarrow b \rightarrow x \rightarrow a \rightarrow b \rightarrow c$

pattern  $\rightarrow a \rightarrow b \rightarrow c$

again  $b$  matched with  $b$ , move to the next idx.

text  $\rightarrow a \rightarrow b \rightarrow x \rightarrow a \rightarrow b \rightarrow c$

pattern  $\rightarrow a \rightarrow b \rightarrow c$

↑  
Not matched so rest the pointer  $i$  to  $0$

and set the pointer  $i$  to next of previous starting  $idx = 1$

text  $\rightarrow a \rightarrow b \rightarrow x \rightarrow a \rightarrow b \rightarrow c$

pattern  $\rightarrow a \rightarrow b \rightarrow c$

↑  
index element not match so move to  $i+1$  and  $i=0$

Test =  $a \rightarrow b \rightarrow x \rightarrow a \rightarrow b \rightarrow c$

pattern =  $a \rightarrow b \rightarrow c$

again Not match. move the idx.  $j$

test =  $a \rightarrow b \rightarrow x \rightarrow a \rightarrow b \rightarrow c$

pattern =  $a \rightarrow b \rightarrow c$

①  $j$  and  $i$  value match so move to next idx

②  $j$  and  $i$  \_\_\_\_\_

③  $j$  and  $i$  \_\_\_\_\_

at end  $i > \text{pattern.length}$ , so it is a substring of test.

Time complexity -  $O(m \cdot n)$

Space complexity -  $O(1)$ .

### Optimisation -

test =  $a \ b \ x \ a \ b \ c \ a \ b \ c \ a \ b \ y$

pattern =  $a \ b \ c \ a \ b \ y$

We want to find the pattern in the given text. It is present or not.

① If at any index the value matches so update the pointer to the next idx.  $\rightarrow$  rule

test =  $a \ b \ x \ a \ b \ c \ a \ b \ c \ a \ b \ y$

pattern =  $a \ b \ c \ a \ b \ y$

① matched so move to next idx.

② again the values matched. move to next idx.

test = a b x a b c a b c a b y  
pattern = a b c a b y

Not matching the values, so now i keep moving to the left and check that there is any x or not because there is no x so i will get to 0 and f will be increase.

increase.

text = a b x a b c a b c a b y

pattern = a b c a b y

- ① Now values matched so move the pointers to next idx.
  - ② again matched. move to next idx.
  - ③ match ④ match ⑤ match.

Now i is on c and i is on y.  $\rightarrow$  No matched  
so we checked to mean i keep moving to the left  
and check there is any c or not.

test = a b x a b c a b c a b y  
 pattern = a b c a b y

Here we find there is a  $\text{idx}$  whose value match with  $j$ . so put the  $i = \text{idx}$  and  $j = j + 1$

j. so put the  $i = i+1$  and  $t = t+1$

text = a b x a b c a b c a b y

pattern = a b c a b y

- ① matched i and j so move to next idx
  - ② matched \_\_\_\_\_
  - ③ matched \_\_\_\_\_

Now  $j > \text{pattern.length}$  mean its true.

## Implementation -

text = a b x a b c a b c a b y  
 pattern = a b c a b y      i    j    i    j    i  
 $\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow$   
 a b c a b y  

0	0	0	1	2	0
---	---	---	---	---	---

 till 0 initially.

We create a array for setting the longest prefix substring.

- ① i and j not match move j to next idx
- ② i and j met match move i to next idx.
- ③ Value of i and j match  
so put the val of  $a^{idx+1}$   
previous  $a = 0 + 1 = 1$
- ④ value of i and j  
match.  $\rightarrow$  so get the idx  $\rightarrow$  which  
i & j and add 1 it to it and put

i    j    i    x    i    i  
~~i~~    ~~j~~    ~~i~~    ~~x~~    ~~i~~    ~~i~~  
~~i~~    ~~j~~    ~~i~~    ~~i~~    ~~i~~    ~~i~~  
 a b c a b y  

0	0	0	1	2	0
---	---	---	---	---	---

  
 0    1    2    3    4    5

- ⑤ Now y not match to any one put 0.

i    j    i    x    i    i  
~~i~~    ~~j~~    ~~i~~    ~~x~~    ~~i~~    ~~i~~  
~~i~~    ~~j~~    ~~i~~    ~~i~~    ~~i~~    ~~i~~  
 text a b x a b c a b c a b y

Here i and j match - a match

Here \_\_\_\_\_ - b match

Now x and c are not matching, so

in the the array move to the previous location and  
look at the val. which having val = 0

so move to  $idx = 0$

Now a compare with x. not equal so update the  
 $j = j + 1$  because i is on the  $\downarrow$   
 $\downarrow$   
 first  $\downarrow$   
 0th.

Now i and j value match so the idx increment

Now i and j ——————

Now y is not matched with c

so we look at the val stored at previous idx. which is 2.

so look at idx 2.

it matches with the c so i and j

will be equal /match.

so the whole string matched.

Problem: 95

main fun(s1, s2) {

int n = s2.length;

int lps[n] = new int[n];

i=0, j = 1;

while (j < n) {

    while (i > 0 && s1.charAt(j) != s2.charAt(i))

        i = lps[i - 1];

    if (s2.charAt(i) == s2.charAt(i))

        i++;

    lps[j] = i;

    j++;

}

i=0, j=0;

while (i < s1.length()) {

    while (j > 0 && j < n && s2.charAt(j) != s1.charAt(i))

        j = lps[j - 1];

    if (j < n && s2.charAt(j) == s1.charAt(i))

        j++;

    i++;

    if (i == n) return 1; // main matched ..

7

return -1;

Move on to question:

main (s1, s2) {

StringBuilder sb();

int rep = B.length() / A.length();

int = 1;

for (i=0 ; i< rep+2 ; i++) {

if (sb.toString().contains(B2))

return cut;

else

res.append(A);

int ++;

}

return -1;

$$Tc = O(N) + O(M)$$

] $\rightarrow$  Robin Karp.

$$Sc = O(N).$$

## Count and Say : Problem - 98

- CountAndSay(1) = "1"

- Break the digit into minimal numbers of substrings such that each substring contains exactly one unique digit.

"3322251"

two 3's, three 2's, one 5, one 1

$$= 23 + 32 + 15 + 11$$

= "23321511"

n = 1      "1"      one ①

n = 2      "11"      two 1x

n = 3      "21"      two 1

n = 4      "1211"

n = 5      "111221"

Start with "1" build n-1 times.

main fun (int n){

String ans = "11"; "1"

if (n == 1) return "1";

if (n == 2) return "11";

for (i = 3; i < n+1; i++) {

ans = build (ans);

}

return ans;

}

// function for ans. built build (ans){

int freq = 1;

char num = charAt(0);

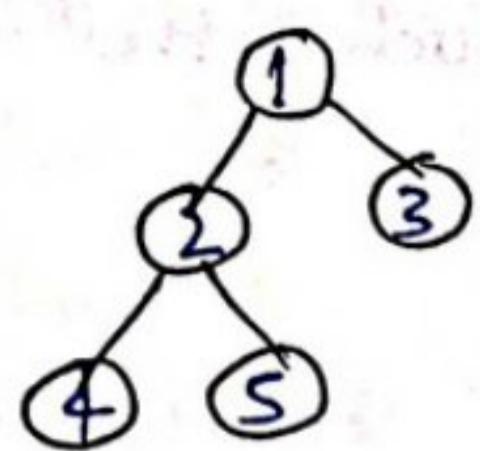
String res = "";

```
for (i = 1; i < ans.length(); i++) {
    if (curNum == ans.charAt(i)) {
        freq++;
    } else {
        char freqChar = (char) ('0' + freq);
        res = res + freqChar;
        res = res + num;
        curNum = ans[i];
        freq = 1;
    }
    char freqChar = (char) ('0' + freq);
    res = res + freqChar;
    res = res + num;
}
```

## Morris Inorder Traversal - Problem 104

Inorder - Left, root, right

I/P:



op:

4	2	5	1	3
---	---	---	---	---

ArrayList<int> lt;

Stack<Node> st;

Node temp = root.left;

st.push(root);

while (temp != null || !st.isEmpty()) {

if (temp != null) {

st.push(temp);

temp = temp.left;

}

else {

temp = st.pop();

lt.add(temp.data);

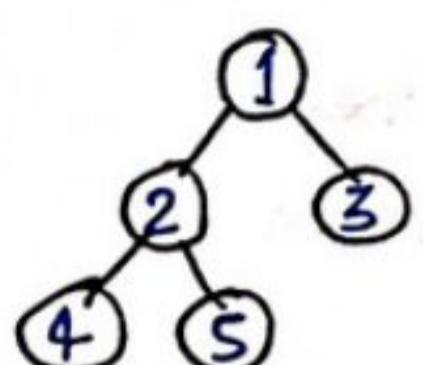
temp = temp.right;

}

return lt;

## Pre Order Morris Traversal - Problem - 105

Preorder - Root, left, right



1	2	4	5	3
---	---	---	---	---

ArrayList<int> lt;

Stack<Node> st;

st.push(root);

while (!st.isEmpty()) {

Node cur = st.pop();

lt.add(cur.data);

temp = cur.right;

temp = temp.left;

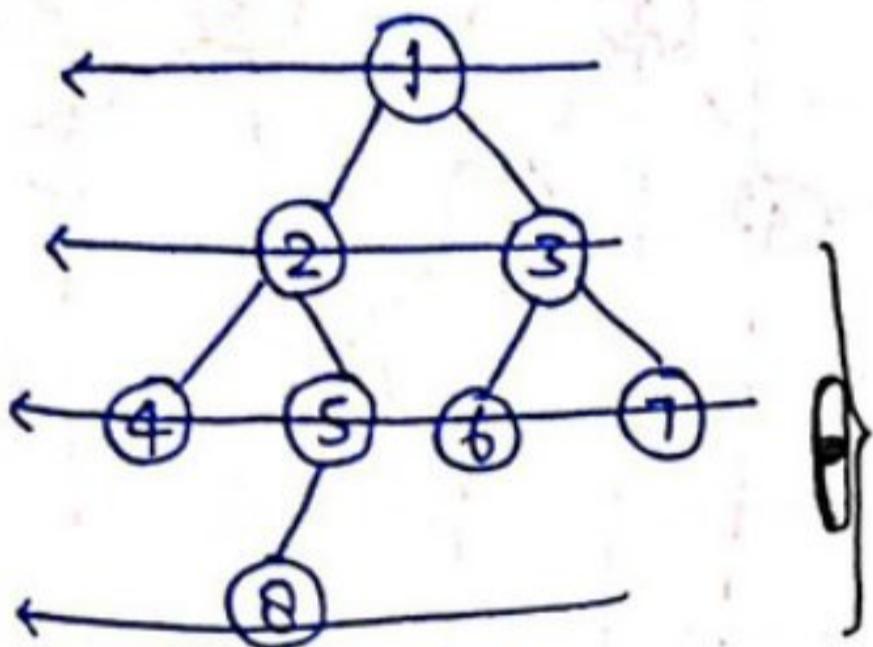
```

        right
if (cur.left != null)
    st.add(cur.left); right
if (cur.right != null)
    left st.add(cur.right);
    left
return lt;

```



## Right/Left View of Binary Tree - Problem - 106



Time complexity -  $O(N)$   
Space complexity -  $O(H)$

Intuition: We have to do a recursive level order traversal.

Root Right Left  $\rightarrow$  for Right View

Root Left Right  $\rightarrow$  for Left view

public class Solution {

    public List<int> rightSideView (TreeNode root) {

        List<Integer> res = new ArrayList<>();

        helprightSideView (root, result, 0);

        return res;

    fun helpRightView { TreeNode cur, List<int> res, int dep) {

        if (cur == null) return;

        if (depth == res.size())

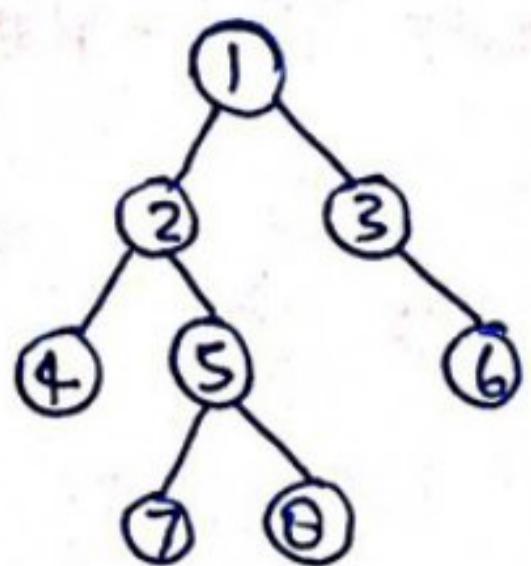
            res.add (cur.val);

        helpRightView (cur.right, res, dep+1);

        helpRightView (cur.left, res, dep+1);

} swap for left view.

## Bottom View of Binary Tree



BottomView - [ 4 | 7 | 5 | 8 | 6 ]

BottomView(root) {

```

        ArrayList<int> res;
        if (root == null) return res;
        Map<int, int> map;
        Queue<Node> q;
        root.hd = 0;
        q.add(root);
        while (!q.isEmpty()) {
            Node temp = q.poll();
            int hd = temp.hd;
            map.put(hd, temp.data); // for Top view check
            if (temp.left != null)
                temp.left.hd = hd - 1;
            q.add(temp.left);
            if (temp.right != null)
                temp.right.hd = hd + 1;
            q.add(temp.right);
        }
    }
  
```

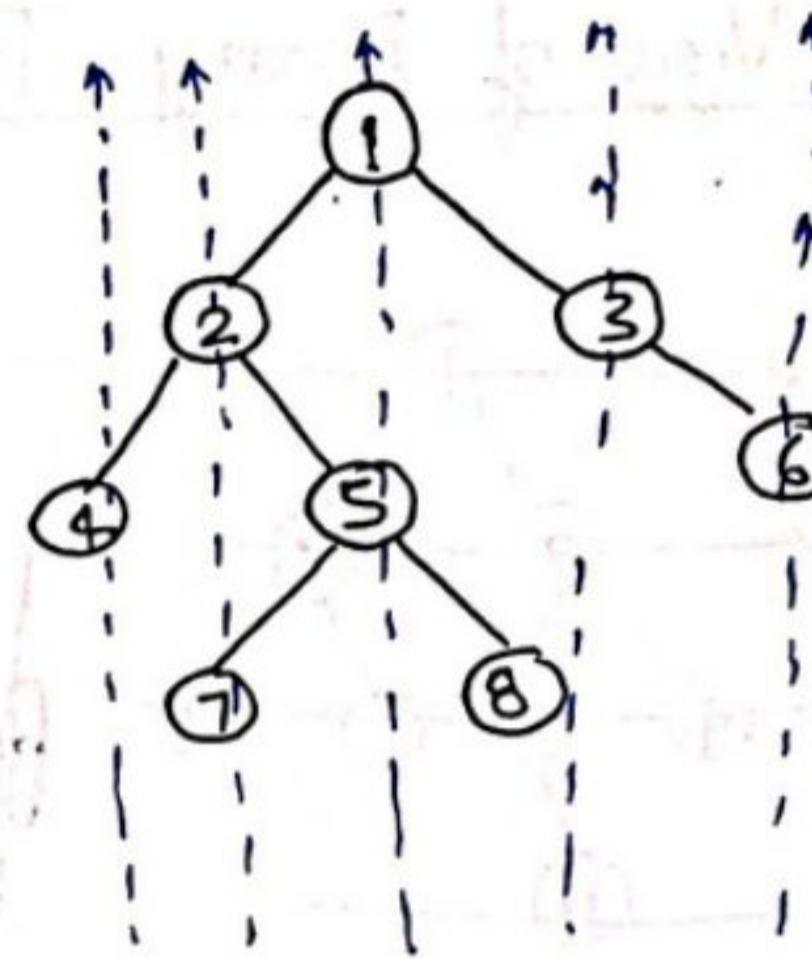
for (Map.Entry<int, int> entry : map.entrySet()) {

    res.add(entry.getValue());

return res;

Time complexity -  $O(H)$

Space " " -  $O(N)$



if (map.contains(hd) == false)

    map.put(hd, entry.getValue());

    }
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

}
 

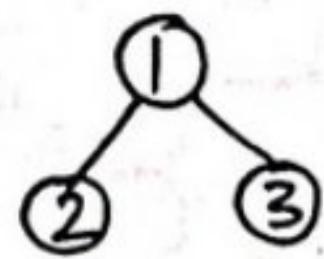
}
 

}
 

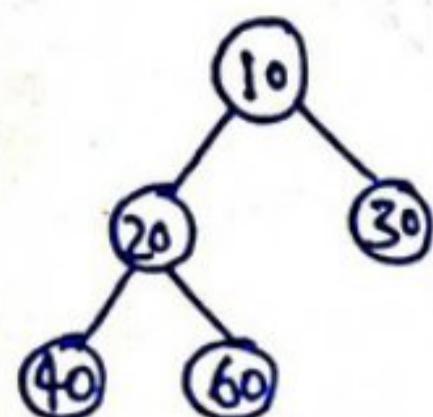
}
 

}
 
<div data-bbox

## Root to Leaf Paths



O/P: 1 2 # 1 3



I/P: O/P: 10 20 40 # 10 20 60 # 10 30

ArrayList<ArrayList<int>> ans;

ArrayList<int> lt;

main(Node root)

    help(root);

    return ans;

help(Node root) {

    if (root == null) return;

    lt.add(root.data);

    if (root.left == null &amp; root.right == null)

        ans.add(new ArrayList<>(lt));

    help(root.left);

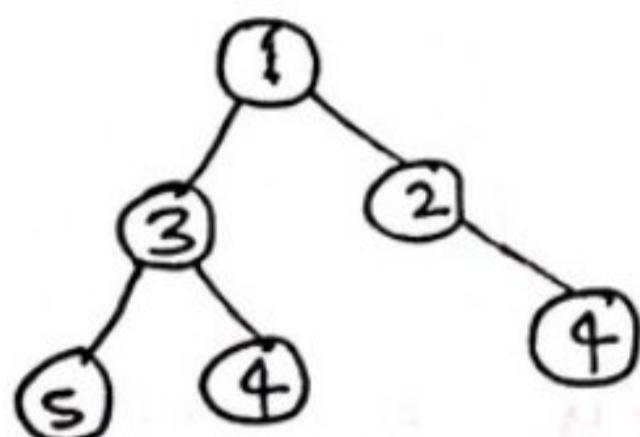
    help(root.right);

}

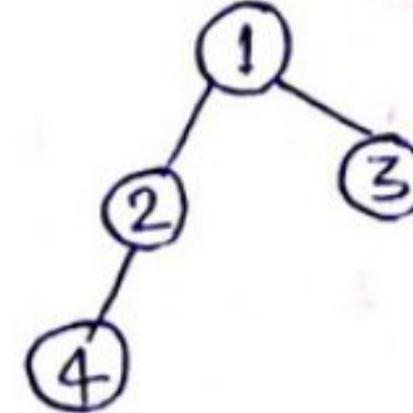
Time complexity -  $O(N)$

Space complexity -  $O(N)$

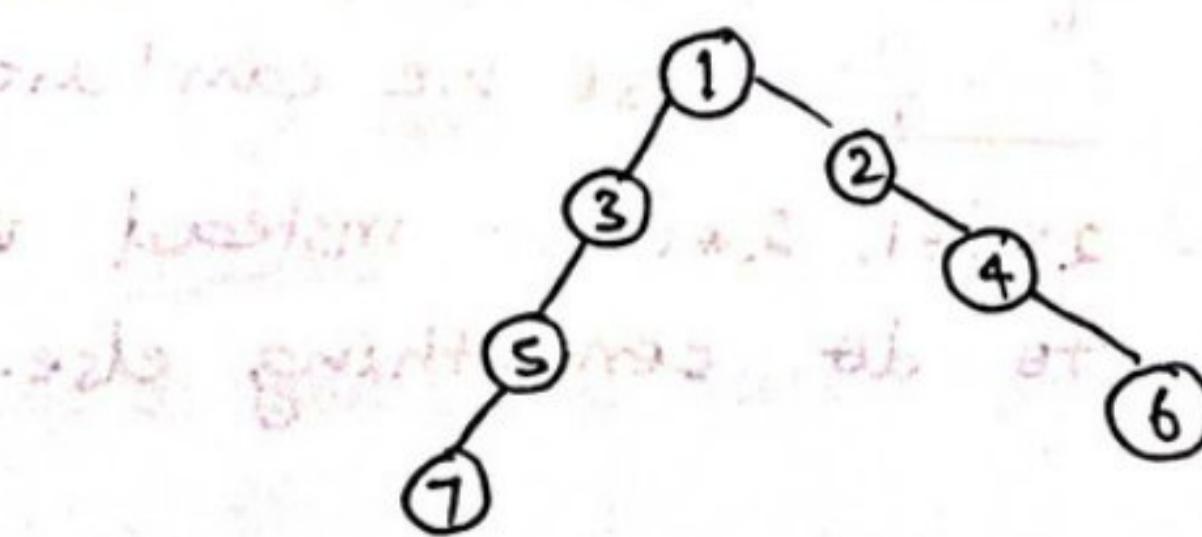
## Maximum width of Binary Tree



Max width = 4



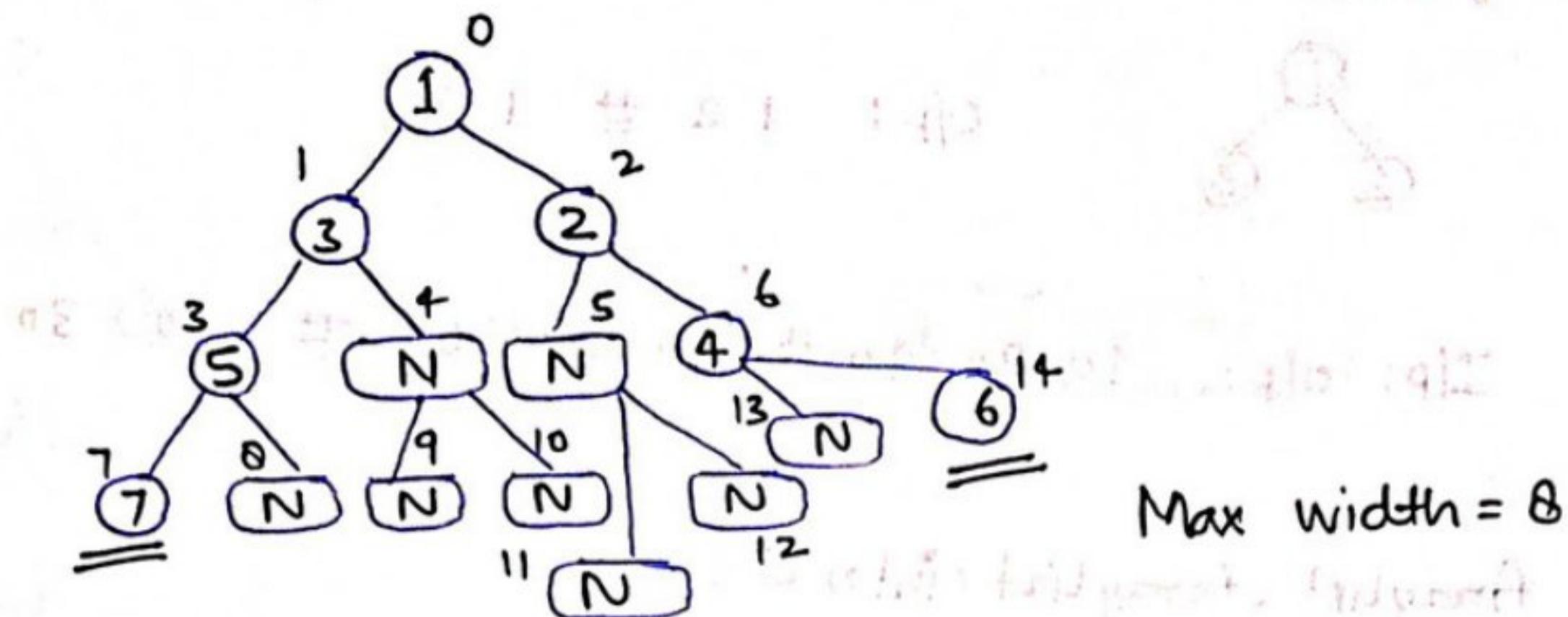
max width = 2



Max width = 8

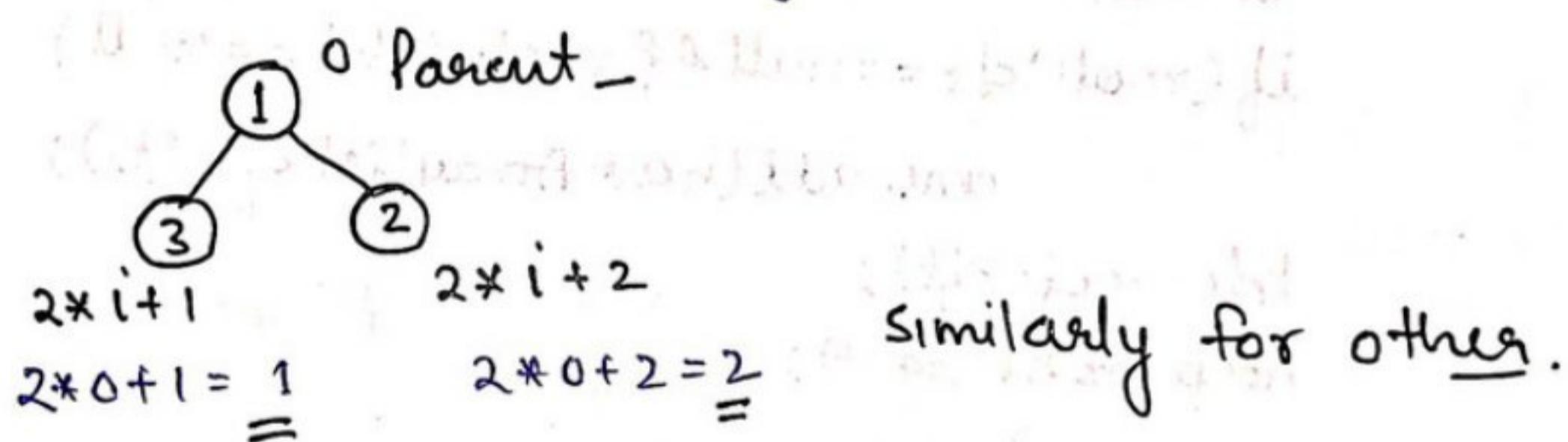
width for a level is defined as the maximum number of nodes between the left most and right most node of level.

## Intuition:

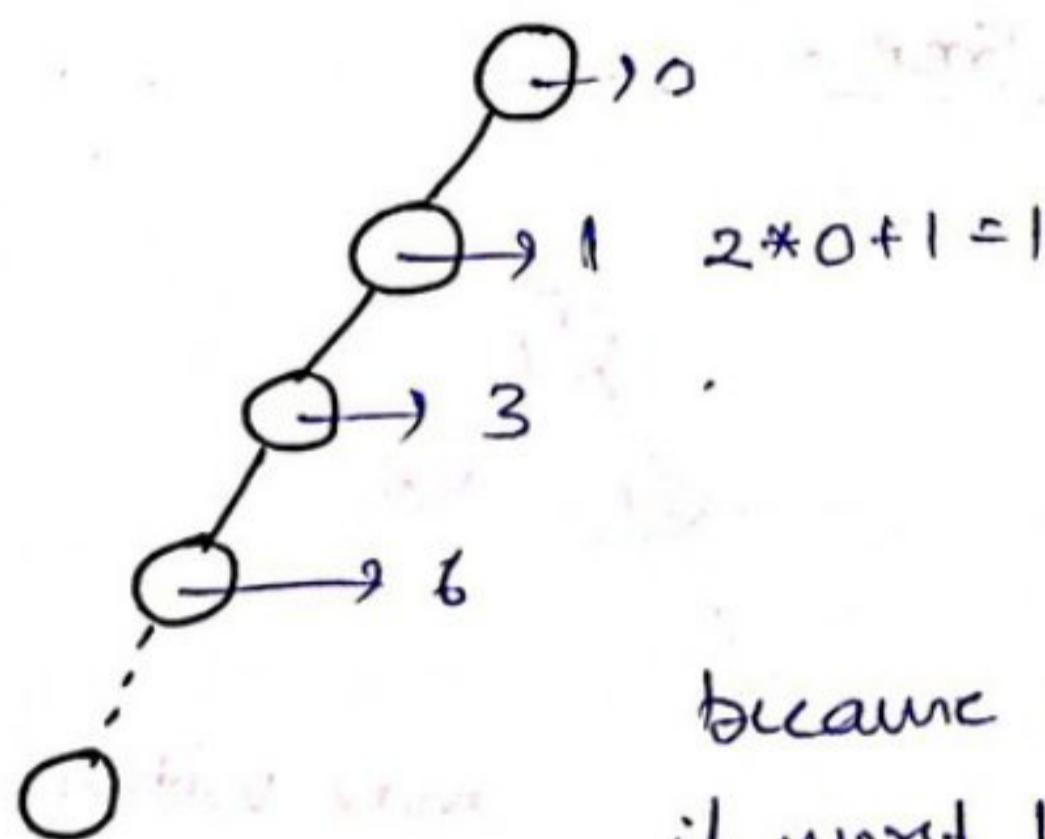


See the left most and right most node at every level. To achieve this we need unique id for every node. Once we know the left most and right most nodes width can be decided as ( $\text{rightMost} - \text{leftmost} + 1$ ).

A parent given  $2*i+1$  to left child and  $2*i+2$  to the right child.

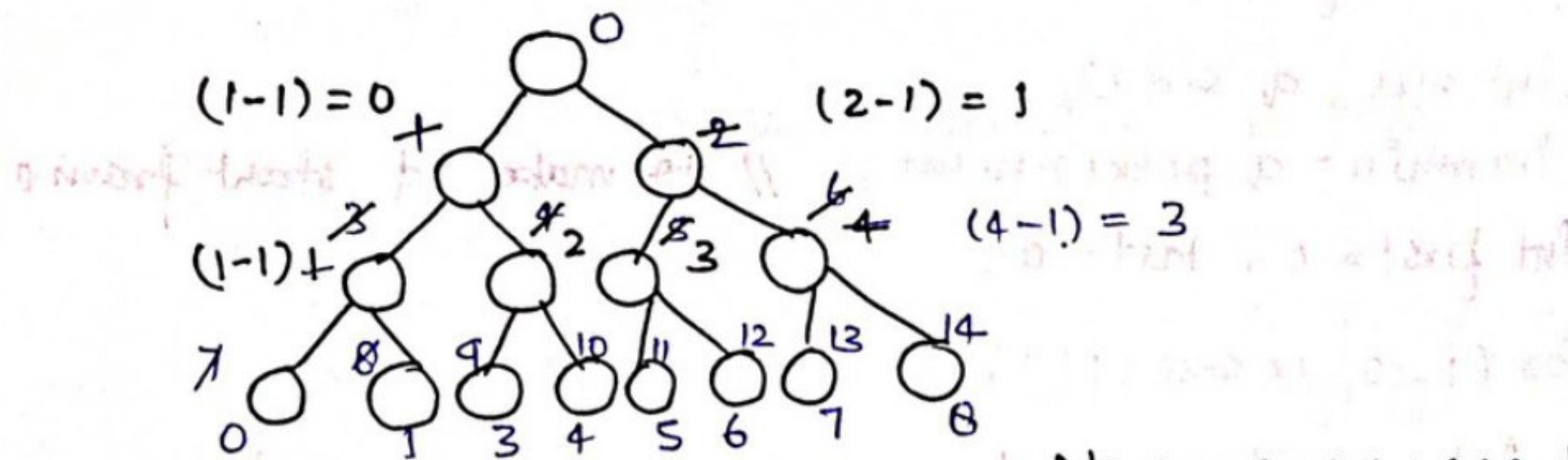


But this fails on tc.

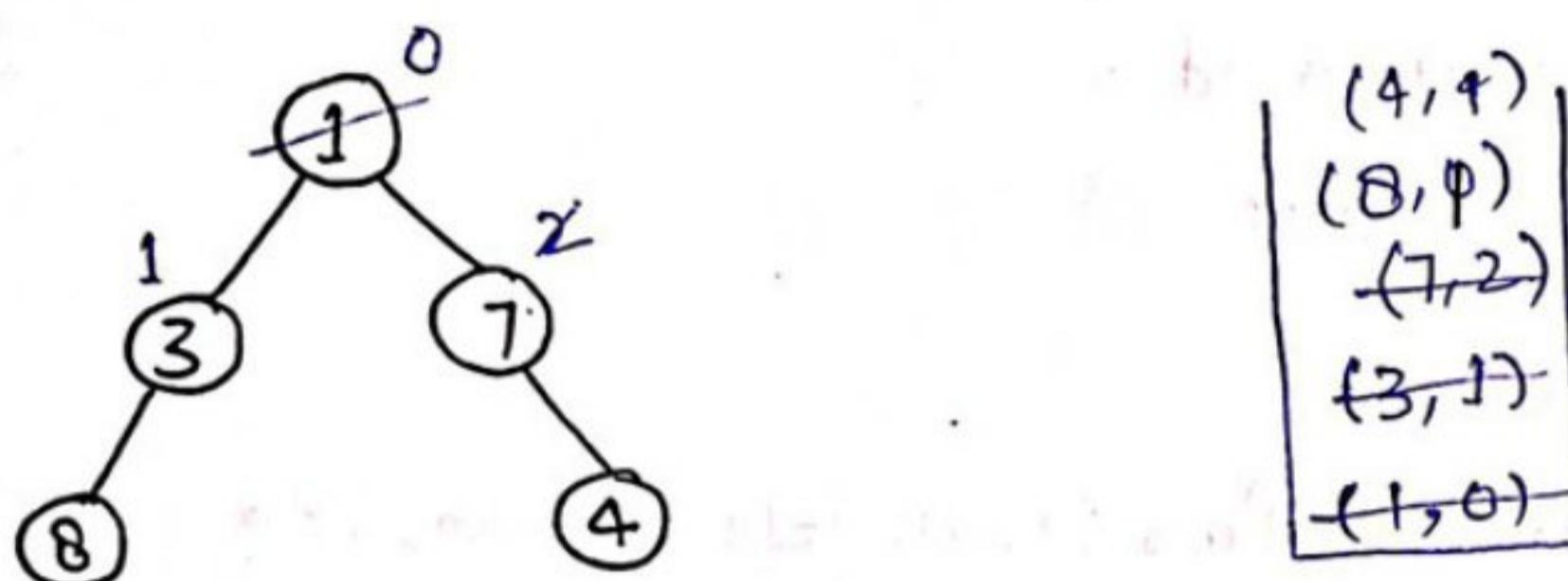


because increasing in twice if worst length is 8 so it will overflow. So we can't use exactly  $2*i+1$ ,  $2*i+2$ . instead we need to do something else.

## How to prevent overflow



Now we can see that we handle the overflow condition.



$$(3, 1) \rightarrow (1-1) = 0+1 = 1$$

$$(7, 2) \rightarrow (2-1) = 1$$

$$(\text{last} - \text{first} + 1)$$

$$(2-1) + 1 = 2^4$$

$$\text{width} = 2^2$$

```
class TreeNode {
    int data;
    TreeNode left, right;
    TreeNode (int data) {
        this.data = data;
        left = null;
        right = null;
    }
}
```

```
main (root) {
    if (root == null) return 0;
    int ans = 0;
    Queue <Pair> q = new LinkedList ();
    q.add (new Pair (root, 0));
    while (!q.isEmpty ()) {
        Pair p = q.remove ();
        ans = Math.max (ans, p.level);
        if (p.node.left != null)
            q.add (new Pair (p.node.left, p.level + 1));
        if (p.node.right != null)
            q.add (new Pair (p.node.right, p.level + 1));
    }
    System.out.println ("ans = " + ans);
}
```

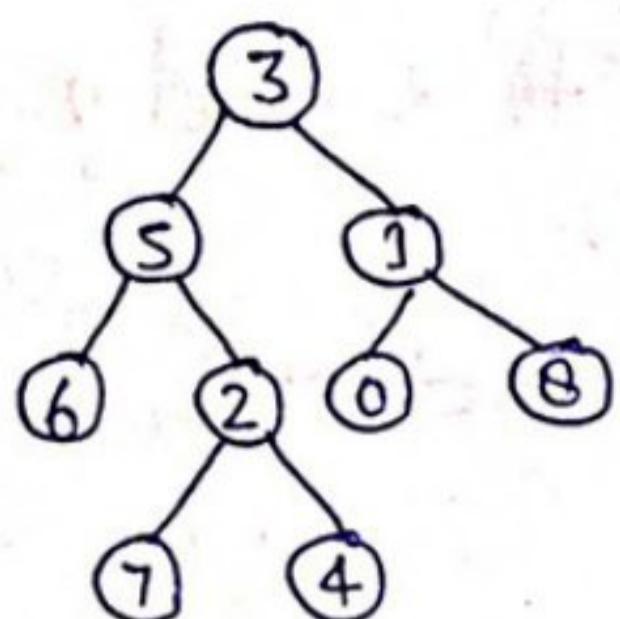
```
class Pair {
    TreeNode node;
    int num;
    Pair (TreeNode node, int num) {
        this.node = node;
        this.num = num;
    }
}
```

```
while (!q.isEmpty()) {
    int size = q.size();
    int min = q.peek().num; // to make id start from 0.
    int first = 0, last = 0;
    for (i=0; i<size; i++) {
        int cur_id = q.peek().num - min;
        TreeNode node = q.peek().node;
        q.poll();
        if (i==0) first = cur_id;
        if (i==size-1) last = cur_id;
        if (node.left!=null)
            q.add(new Pair(node.left, cur_id*2+1));
        if (node.right!=null)
            q.add(new Pair(node.right, cur_id*2+2));
    }
    ans = (ans, last-first+1);
    max = max;
}
return ans;
}
```

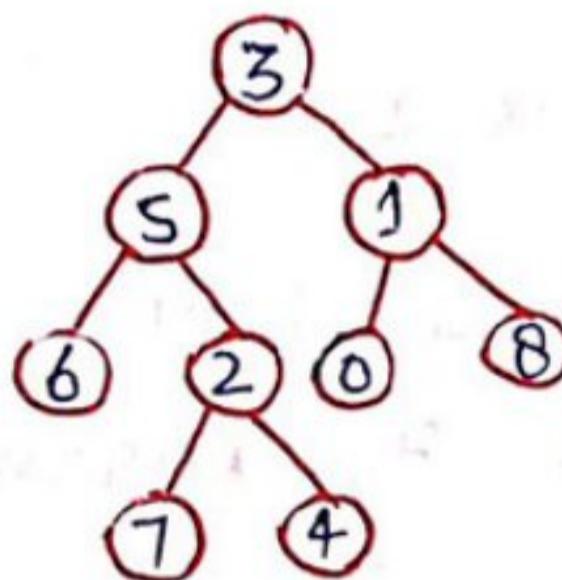
## lowest common Ancestor of a Binary Tree



the lowest parent of two Nodes.



$p=5, q=1$ , the LCA will be 3



$p=5, q=4$

lowest CA = 5

Intuition - we can observe that we can find the LCA of 2 given nodes from

- ① Left subtree or right subtree
- ② If not in both, the root will be LCA.

1. If root is null or if root is  $x$  or if root is  $y$  then return root.

2. Make a recursion call for both.

- ① Left
- ② Right

\* Make a recursive call  $\rightarrow$  left subtree



if null, mean LCA not found

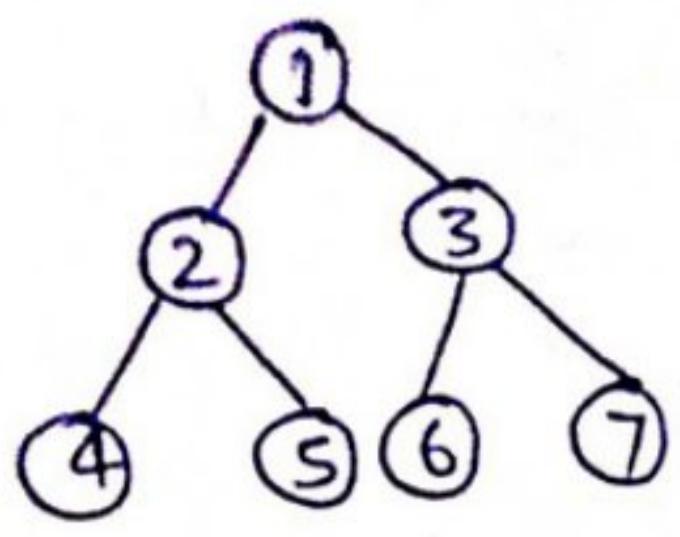
on left subtree which means we found LCA on right subtree. go return Right.

\* Make a recursive call  $\rightarrow$  right subtree



if null, mean LCA not found

on right subtree which means we found LCA on left so return left.



$$x = 4, y = 5$$

$\text{root} \neq \text{null}$  if not equal to  $x$  and  $y$   
go recursion call.

call left subtree it find 4 so this call will return  
4 to its parent.

At present the parent is 2.

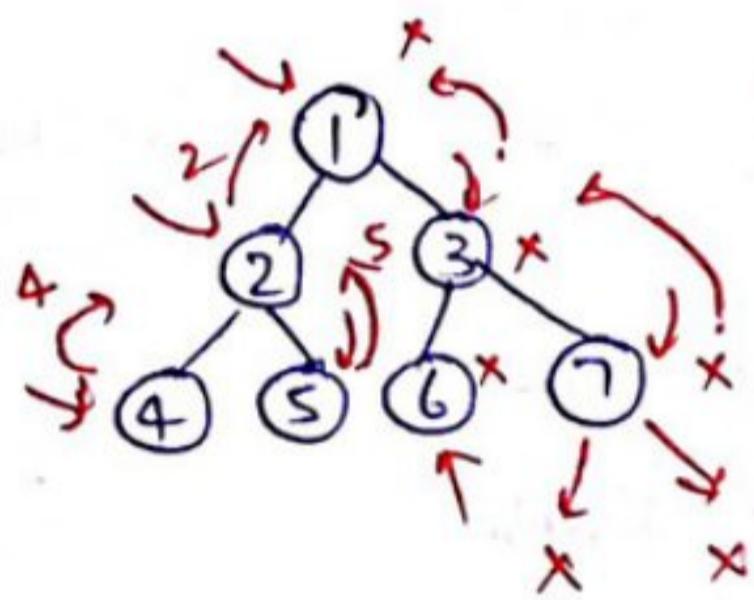
\* Call the right subtree, while calling it will find 5.  
and this call will return 5 to its parent.

Now left recursive call returns val. not null  
of

Right recursive call " " "



thus 2 will return itself. to root i.e. to 1.



$$\text{lca}(4, 5) = 2$$

Brute force - take the root to node path to  $x$  and  $y$ .

(1, 2, 4)

(1, 2, 5)

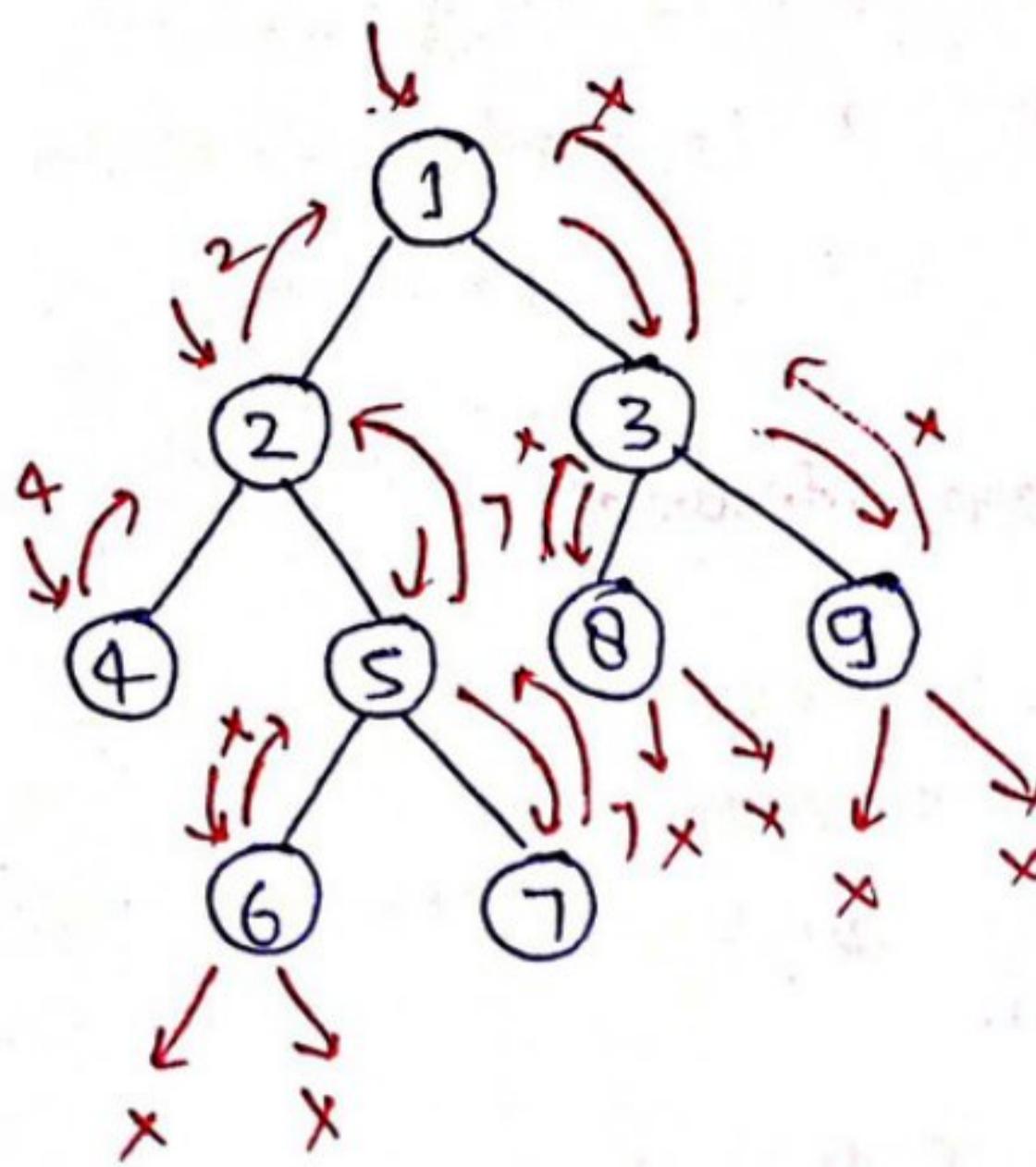
at zeroth idx match

at 1st idx match

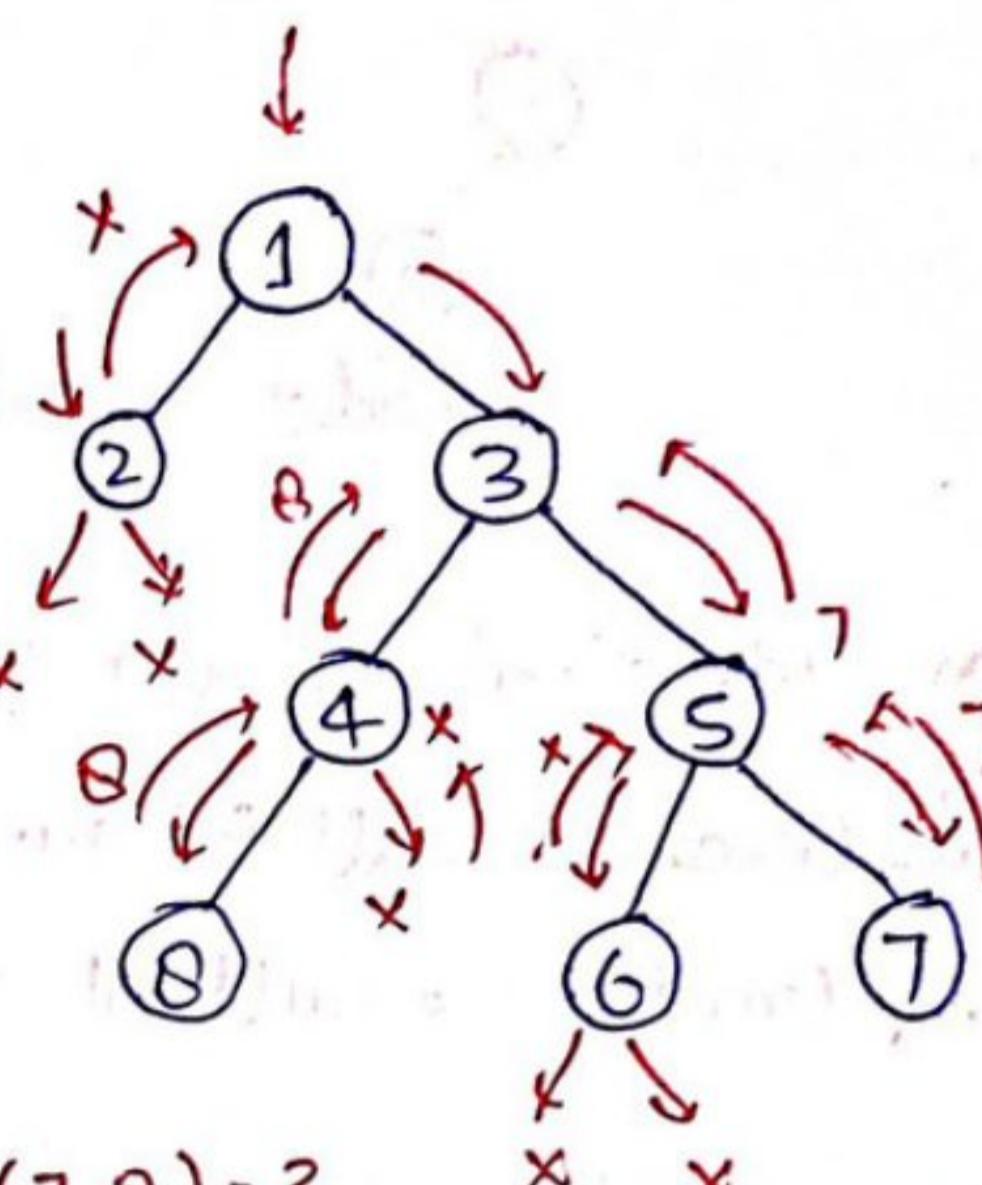
not matched on 2nd so

last match was 2 so it is LCA.

$O(N) \rightarrow$  Space  
Time



$$\text{lca}(4, 7) = 2$$



$$\text{lca}(7, 8) = 3$$

both the node we got 80 we are confirm that this the the lca we are looking for

main LCA()

```
if (root == null || root == p || root == q) return root;
```

```
TreeNode left = LCA(root.left, p, q);
```

```
TreeNode right = LCA(root.right, p, q);
```

// result

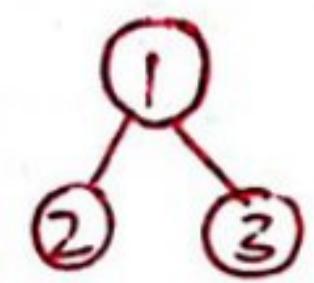
```
if (left == null) return right;
```

```
if (right == null) return left;
```

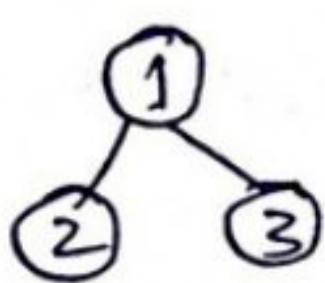
```
else return root; // if left and right both != null we found the result.
```

## Check if two trees are identical -

1



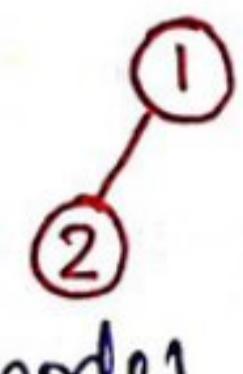
node 1



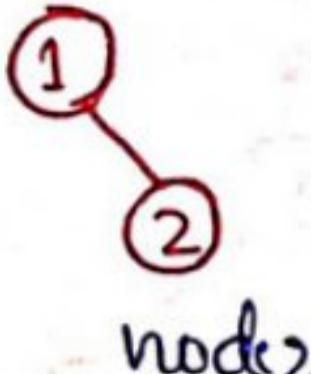
node 2

both are identical.

2



node 1



node 2

not identical.

boolean identical (node1, node2){

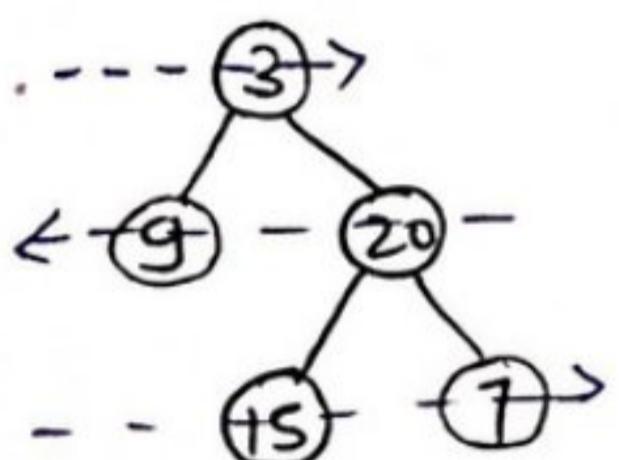
    if (node1 == null && node2 == null) return true;

    if (node1 == null || node2 == null) return false;

    return ((node1.data == node2.data) && identical (node1.left,  
                node2.left) && identical (node1.right, node2.right));

}

## Binary Tree Zig Zag level order Traversal.



O/P: [[3], [20, 9], [15, 7]]

main levelordertraversal (root) {

    List<List<int>> lt;

    if (root == null)

        return lt;

    Stack<Node> st1;

    Stack<Node> st2;

    st1.push (root);

    TreeNode curr;

boolean lter = true;

while (!st1.isEmpty() || !st2.isEmpty()) {

    List<int> level();

    if (lter) {

        while (!st1.isEmpty()) {

            curr = st1.pop();

            level <-- lter = curr.add(curr.val);

            if (curr.left != null)

                st2.push(curr.left);

            if (curr.right != null)

                st2.push(curr.right);

}

    else {

        while (!st2.isEmpty()) {

            curr = st2.pop();

            level <-- lter = curr.add(curr.val);

            if (curr.right != null) st1.push(curr.right);

            if (curr.left != null) st1.push(curr.left);

    }

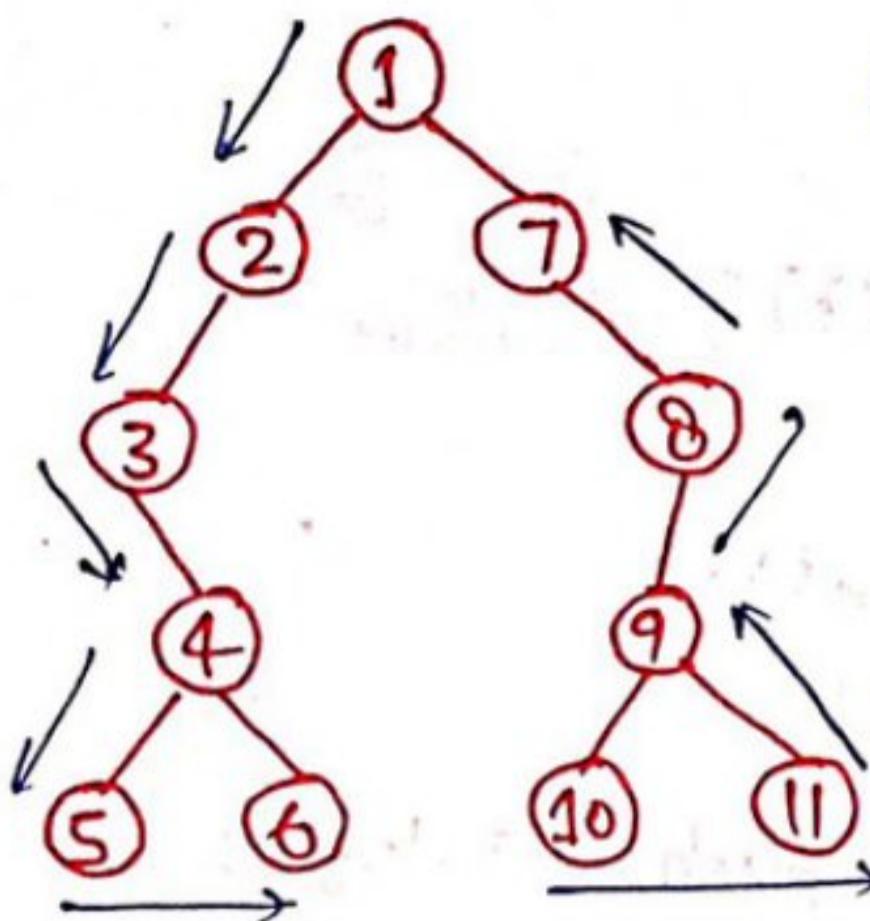
    ans.add(lter);

    lter = !lter;

return ans;

}

## Boundary Traversal of Binary Tree.



Boundary traversal:

1	2	3	4	5	6	10	11	9	8	7
---	---	---	---	---	---	----	----	---	---	---

1. Part 1: Left Boundary of tree.
2. Part 2: All the leaf nodes travelled in the left to right
3. Part 3: Right boundary of tree (exclude leaf) reverse.

```
printBoundary(node) {  
    ArrayList<int> ans();  
    if (isLeafNode) == false) ans.add(node.data);  
    addLeftBoundary(node, ans);  
    addLeaves(node, ans);  
    addRightBoundary(node, ans);  
    return ans;  
}
```

```
Boolean isLeaf(node) {  
    return (root.left == null) && (root.right == null);  
}
```

```
addLeftBoundary(node, ans) {  
    Node cur = root.left;  
    while (cur != null) {  
        if (isLeaf(cur) == false) >  
            ans.add(cur.data);  
        if (isLeaf(cur) == false) ans.add(cur.data);  
        else  
            cur = cur.right;
```

```
if(isleaf(cur) == false) ans.add(cur.data);  
if(cur.left != null) cur = cur.left;  
else  
    cur = cur.right;  
    ?
```

addRightBoundary(root, ans) {

Node cur = root.right;

ArrayList<int> tmp();

while(cur != null) {

if(isleaf(cur) == false) tmp.add(cur.data);

if(cur.right != null) cur = cur.right;

else

cur = cur.left;

}

int i;

for(i = tmp.size(); i >= 0; i--) {  
 ans.add(tmp.get(i));  
}

}

addleaves(root, ans) {

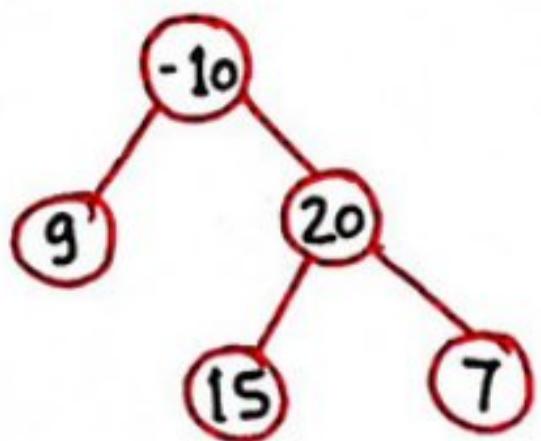
if(isleaf(root)) ans.add(root.data);  
 return;

if(root.left != null) addleaves(root.left, ans);

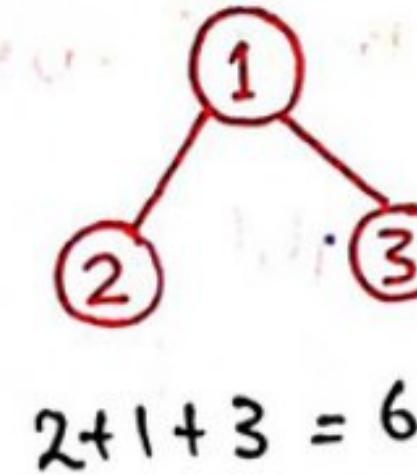
if(root.right != null) addleaves(root.right, ans);

}

## ★ Binary Tree maximum Path sum



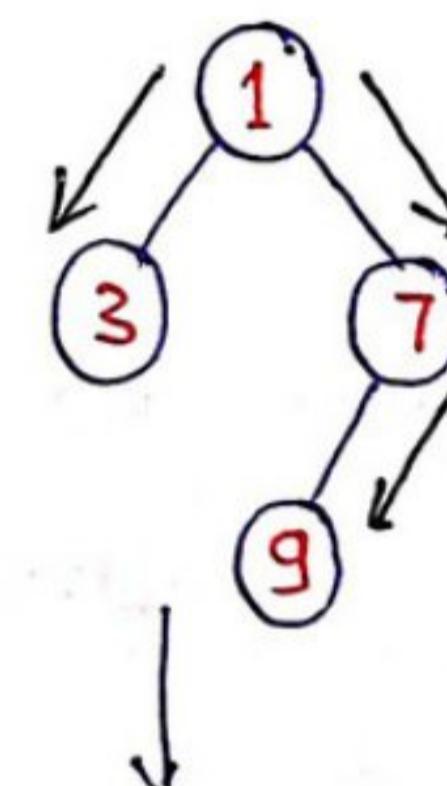
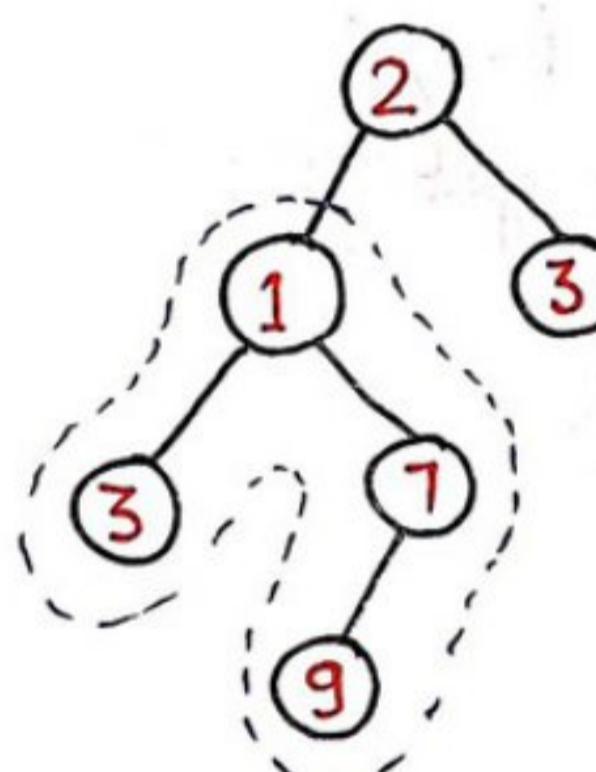
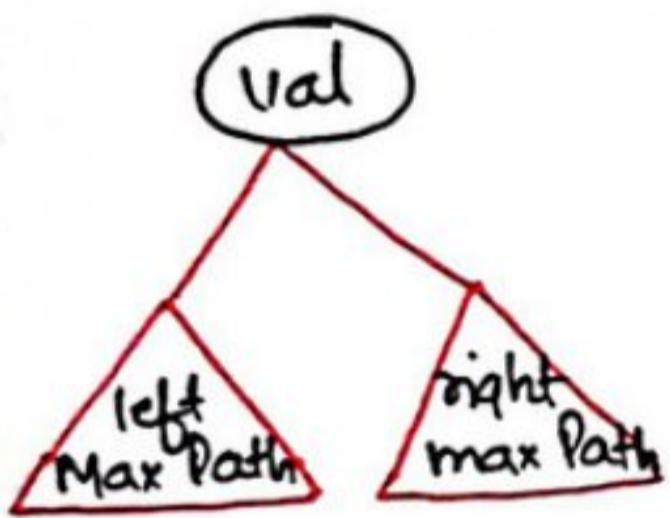
most optimal maximum path  
sum =  $15 + 20 + 7 = \underline{\underline{42}}$



A brute force approach would be to generate all paths and compare them. Generating all paths will be a time costly activity therefore we need to lookup for some thing else.

At a given node with a val, if we find the max left sumPath in the left subtree and rightSubtree,

$$\text{maxPathSum} = \text{val} + \text{leftSumPath} + \text{rightSumPath}$$



- Initialize a maximize to store final one.
- Do a simple tree traversal. At each node, find recursive its left Max Path and right Max sum path.
- cal. the  $\text{val} + \text{left max Path} + \text{right Max sum}$  and update maxi accordingly.
- Return the max Path when node is not the curving point as  $\text{val} + \max(\text{left Max Path}, \text{rightMaxPath})$ .

$$\text{left path} = 3$$

$$\text{right path} = 0 + 7 + 9 = 16$$

$$\begin{aligned} \text{max Path Sum} &= \text{val} + \text{left path} + \\ &\quad \text{right path} \end{aligned}$$

$$\begin{aligned} &= 3 + 1 + 16 \\ &= 20 \end{aligned}$$

```

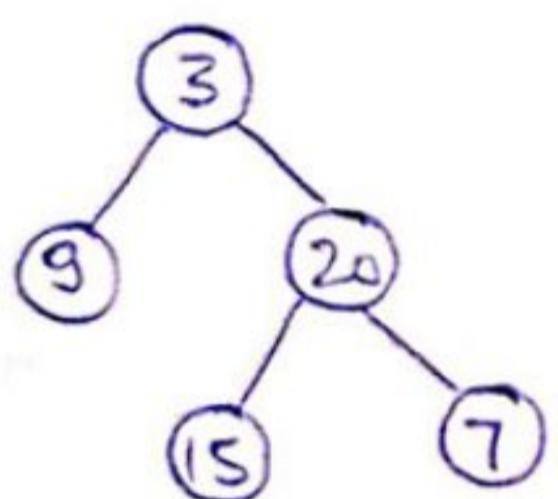
maxPathSum (root) {
    int maxVal[] = new int[1];
    maxVal[0] = Integer.MIN_VALUE;
    maxPathDown (root, maxValue);
    return maxValue[0];
}

maxPathDown (node, maxValue) {
    if (node == null) return 0;
    int left = Math.max(0, maxPathDown (node.left, maxValue));
    int right = Math.max(0, maxPathDown (node.right, maxValue));
    [ maxValue[0] = Math.max(maxValue[0], left + right + node.val); ]
    return Math.max(left, right) + node.val;
}
    }   ↳ return to its parent.

```

Construct a Binary Tree from inorder and pre order -

Problem - 120



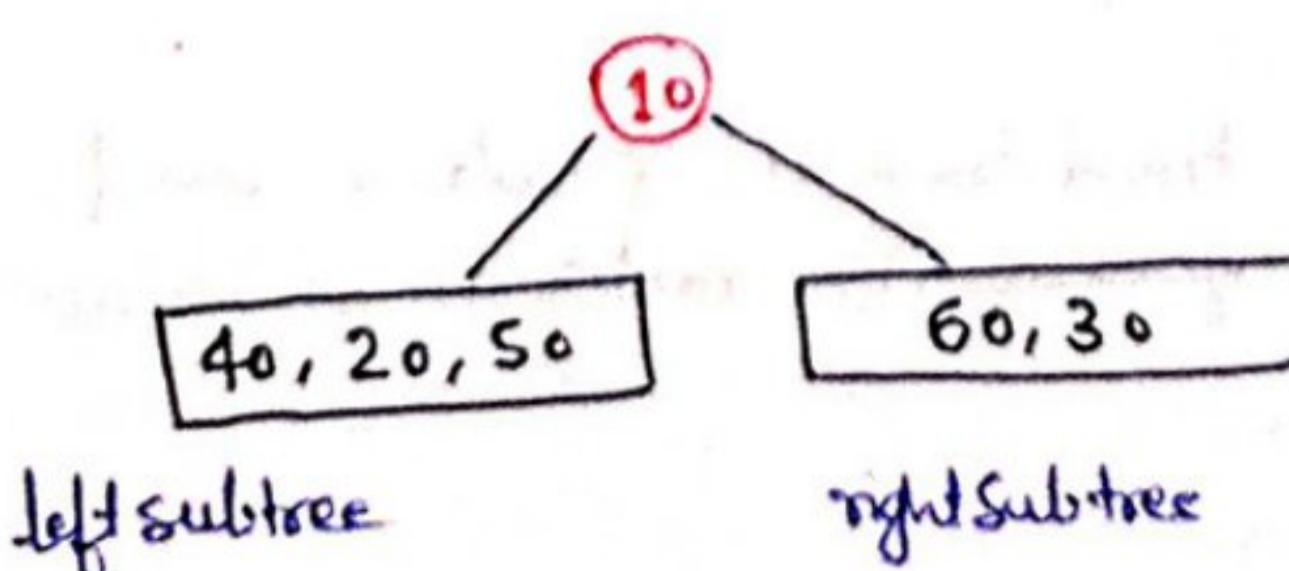
Input: [3, 9, 20, 15, 7]

Output: [9, 3, 15, 20, 7]

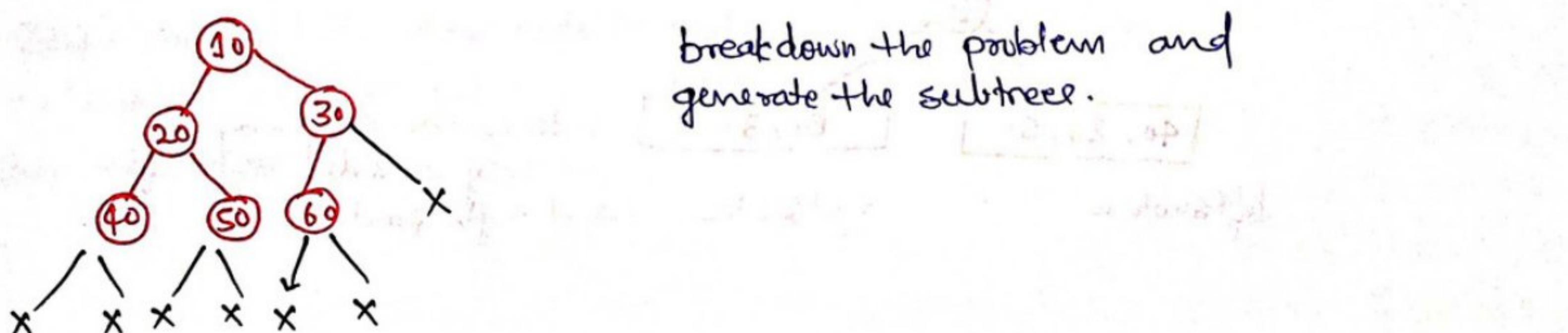
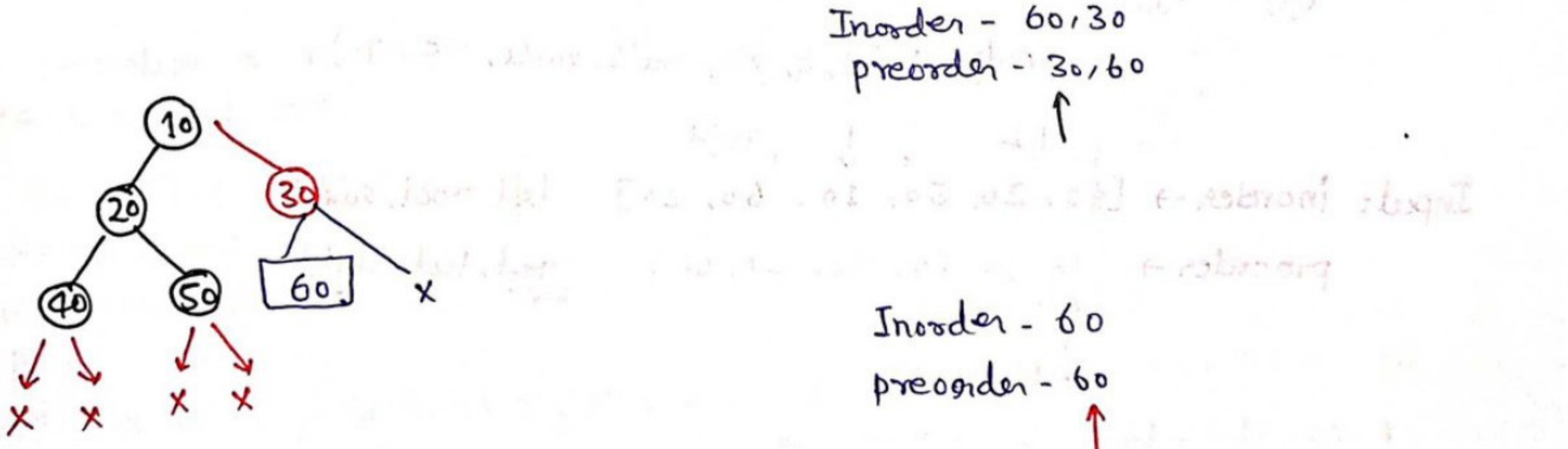
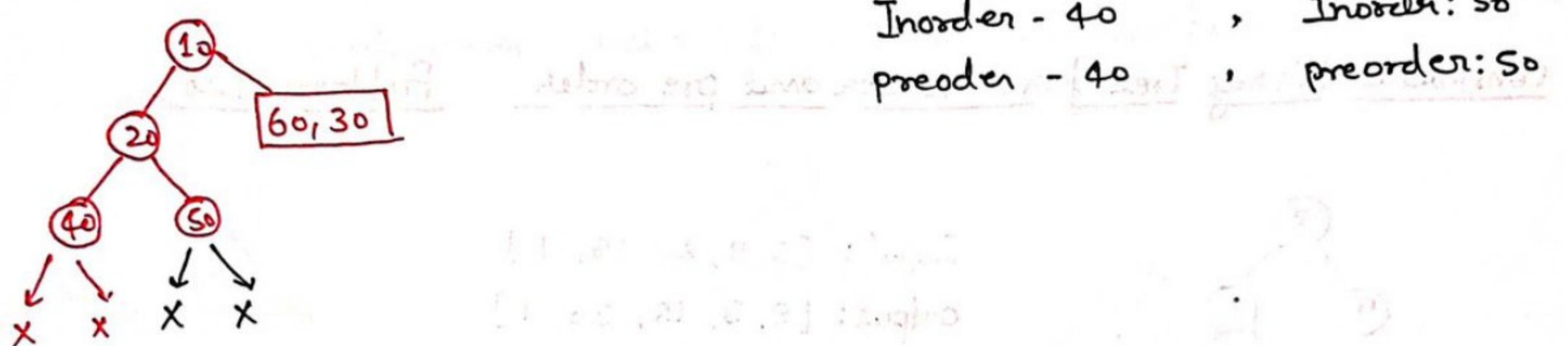
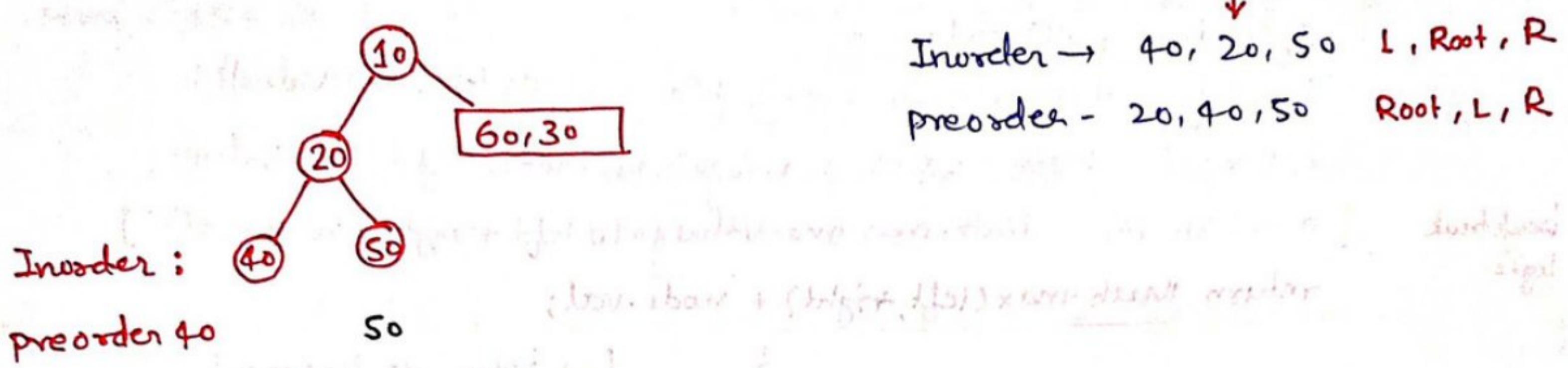
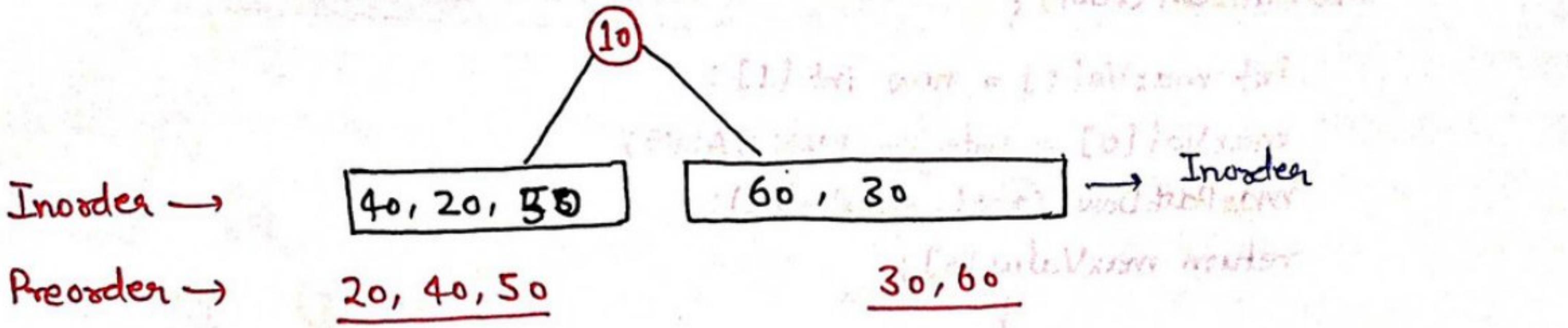
Output: [3, 9, 20, null, null, 15, 7]

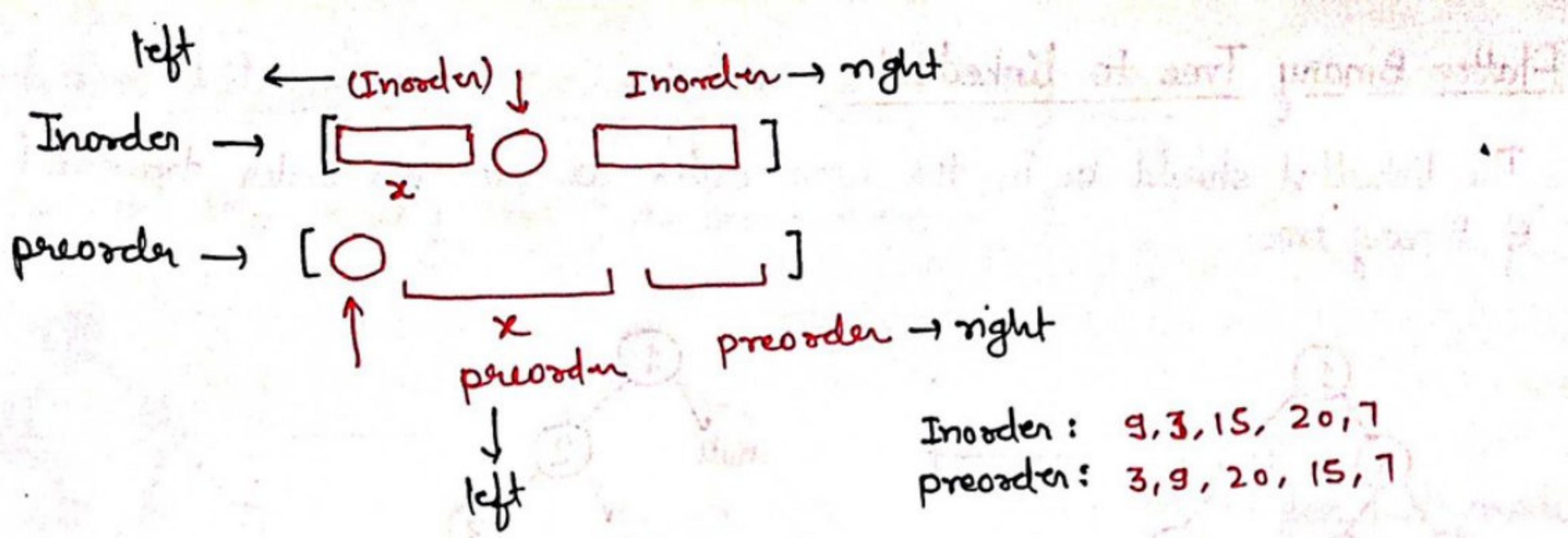
Input: inorder → [40, 20, 50, 10, 60, 30]      left      ↓      right  
root

preorder → [10, 20, 40, 50, 30, 60]      root, left, right.



order can be anything  
we only consider the left  
and right part





```
main(preorder[], inorder[]) {
    HashMap<int, int> map();
    for(i = 0; i < inorder.length; i++) {
        map.put(inorder[i], i);
    }
}
```

9 → 0
3 → 1
15 → 2
20 → 3
7 → 4

```
TreeNode root = buildTree(preorder, 0, preorder.length - 1, inorder, 0,
                           inorder.length - 1, map);
```

```
return root;
```

```
}
```

```
buildTree(preorder, int preStart, int preEnd, inorder[], int inStart,
          int inEnd, map) {
```

```
if (preStart > preEnd || inStart > inEnd) return null;
```

```
TreeNode root = new TreeNode(preorder[preStart]);
```

```
int inRoot = map.get(root.val);
```

```
int numsLeft = inRoot - inStart;
```

```
root.left = buildTree(preorder, preStart + 1, preStart + numsLeft, inorder,
                      inStart, inRoot - 1, map);
```

```
root.right = buildTree(preorder, preStart + numsLeft + 1, preEnd, inorder,
                       inRoot + 1, inEnd, map);
```

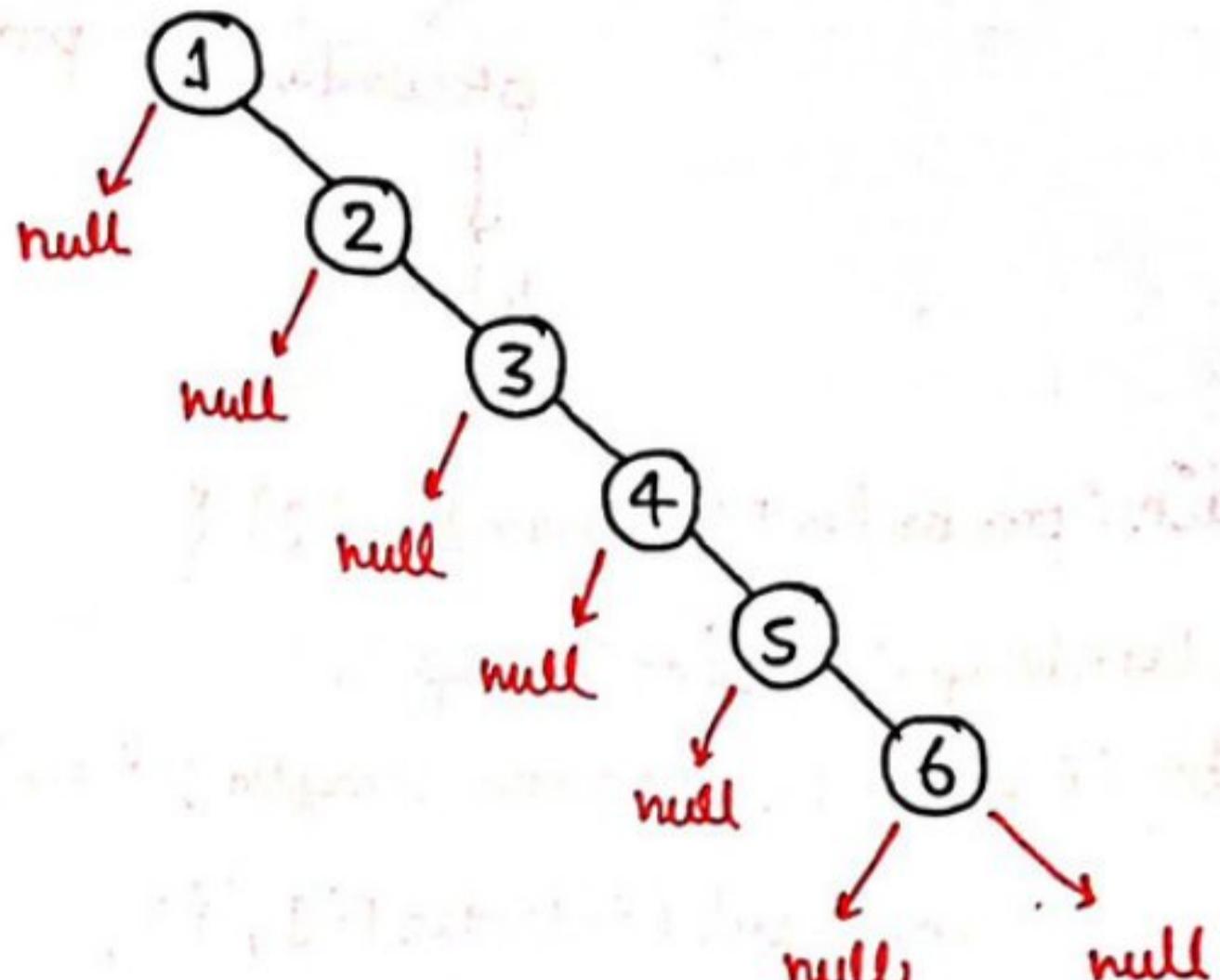
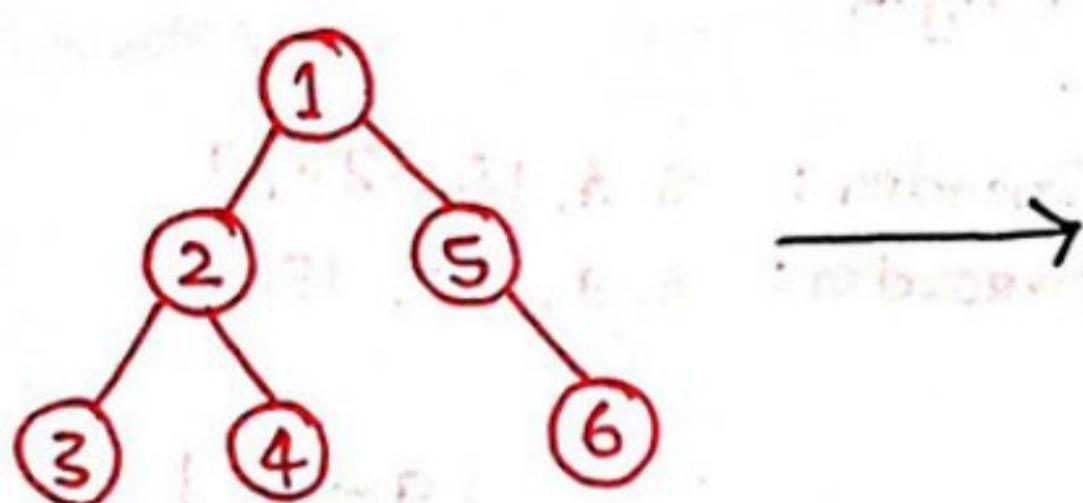
```
return root;
```

Tc - O(N)

Sc - O(N)

## Flatten Binary Tree to linkedlist

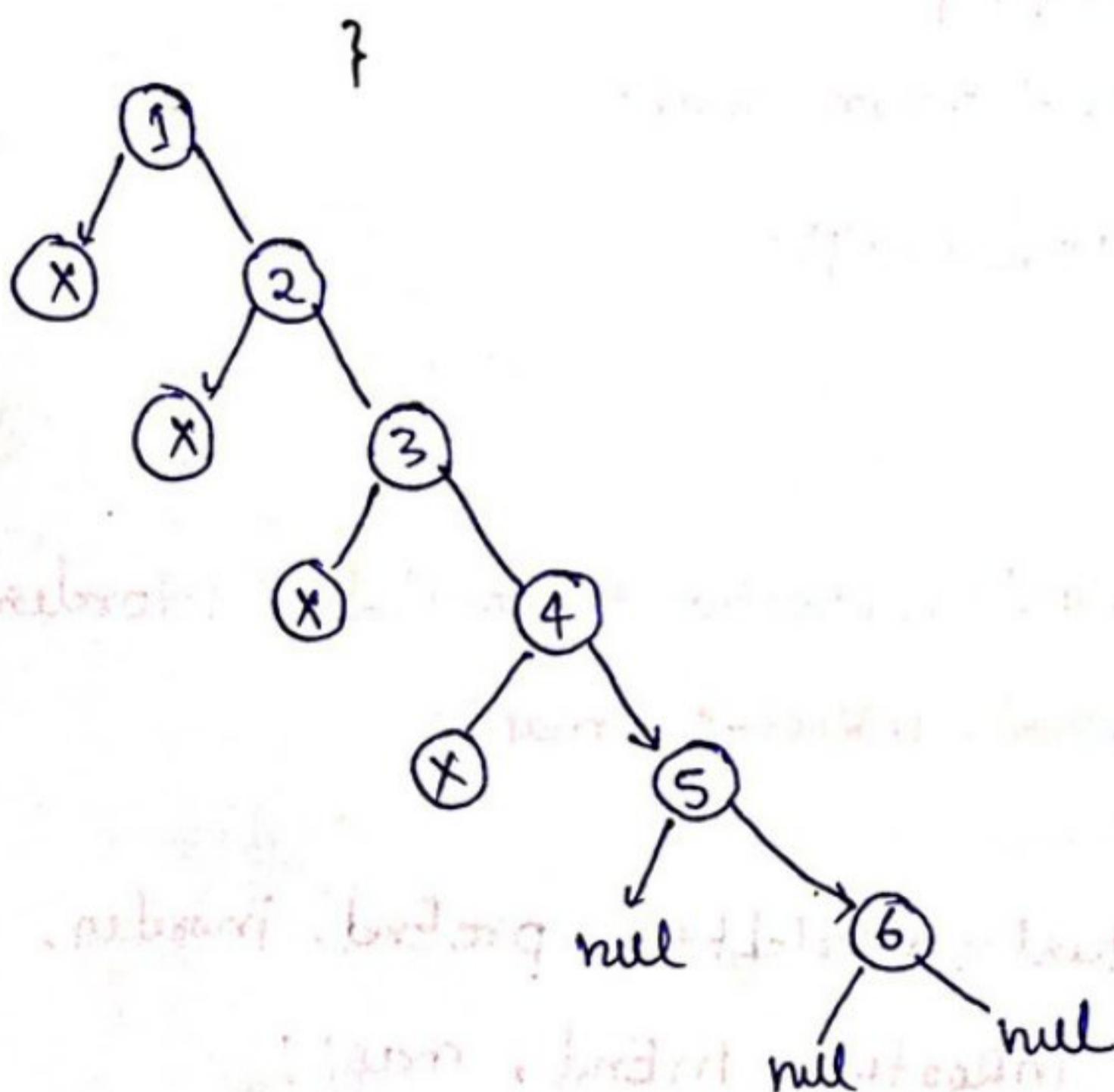
The linkedlist should be in the same order as a pre-order traversal of Binary tree.



class Solution {

```

static Node pre = null;
static void flatten(Node root) {
    if (root == null) return;
    flatten(root.right);
    flatten(root.left);
    root.right = pre;
    root.left = null;
    pre = root;
}
  
```



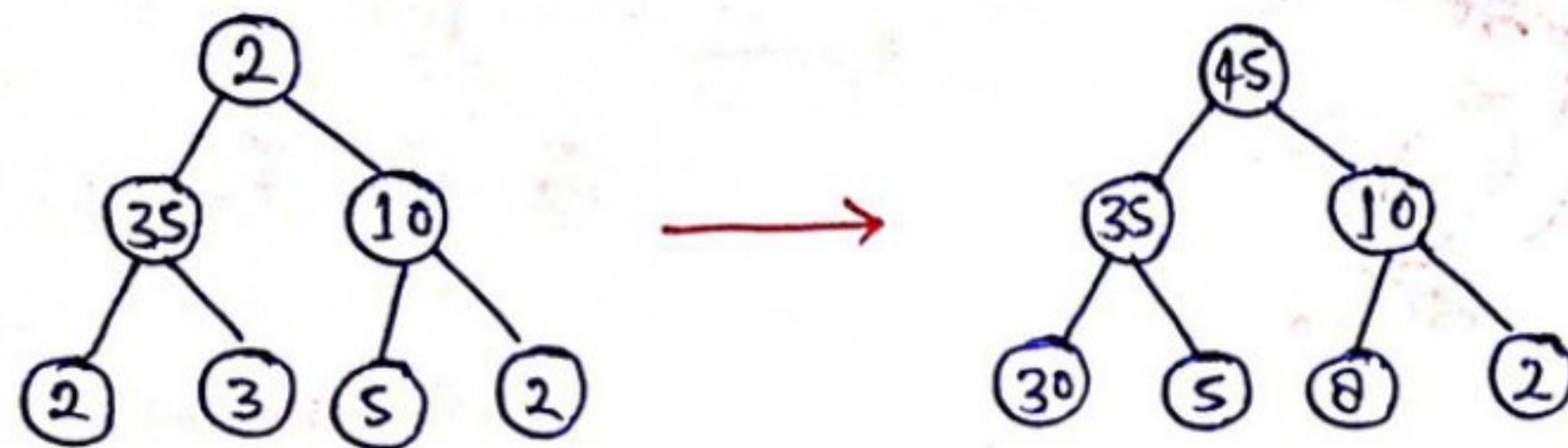
pre = null & 5
4, left, nod
4, right, null
2, left, 3
2, right, 4
2, left, 2
6, left, null
6, right, null
5, right, 6
1, right, 5

Root, left, right

Similarly

## Check for children sum property in a Binary Tree

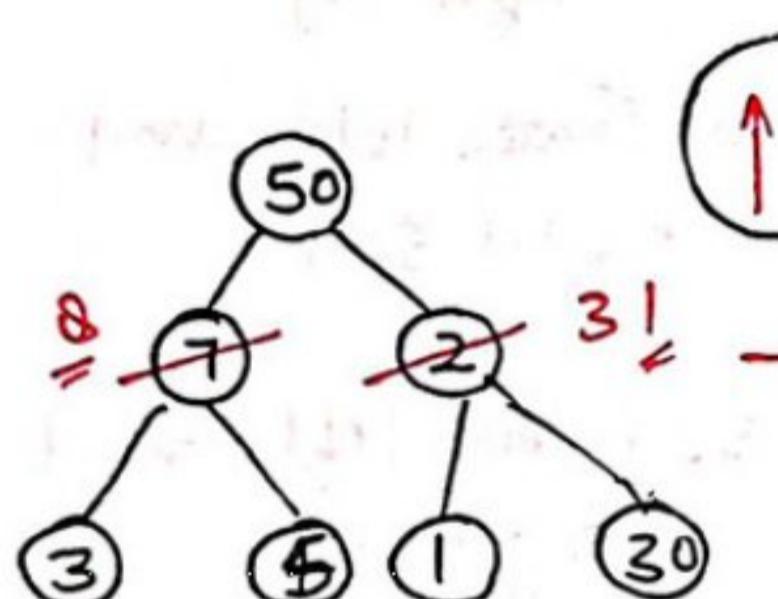
convert the binary Tree into sum tree -



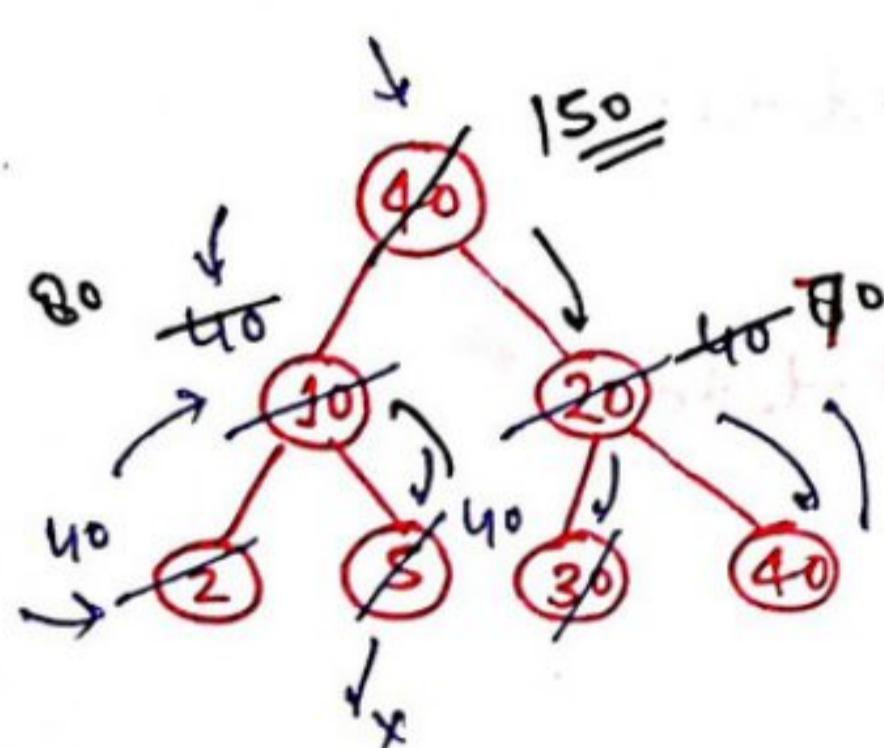
Note - A val for null node can be accessed 0

node = left + right

- Not allowed to change the structure.



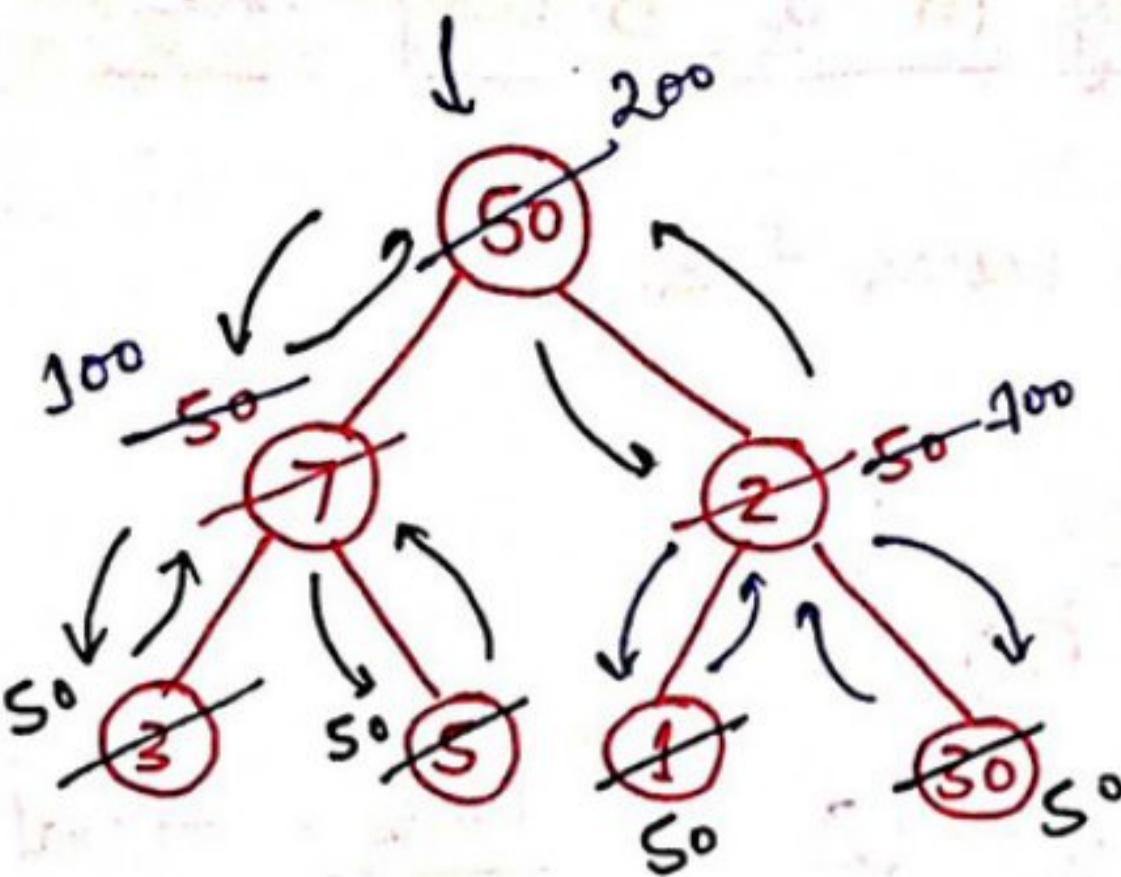
so we can see over here the sum of left and right  $8+31=91$ , and the root is 50, so we can't minimize the root val. that's the problem this is not a simple problem.



$$10+20=30 < 40$$

$$2+5=7 < 40$$

$$30+40=70 > 40 \text{ (make } \text{root} = \text{left} + \text{right})$$



main()

```

    if (root == null) return;
    int child = 0;
    if (root.left != null) {
        child += root.left.data;
    }
    if (root.right != null) {
        child += root.right.data;
    }

```

```

    if (root.data <= child)
        root.data = child;
    else {

```

```

        if (root.left != null)
            root.left.data = root.data;
        else if (root.right != null)
            root.right.data = root.data;
    }

```

Self fun recursion (root.left);

Self fun recursion (root.right);

int tot = 0;

```

    if (root.left != null)
        total += root.left.data;

```

```

    if (root.right != null)
        total += root.right.data;

```

```

    if (root.left == null || root.right == null)
        root.data = tot;

```

$7+2 = 9 < 50$  [make left and right 50]

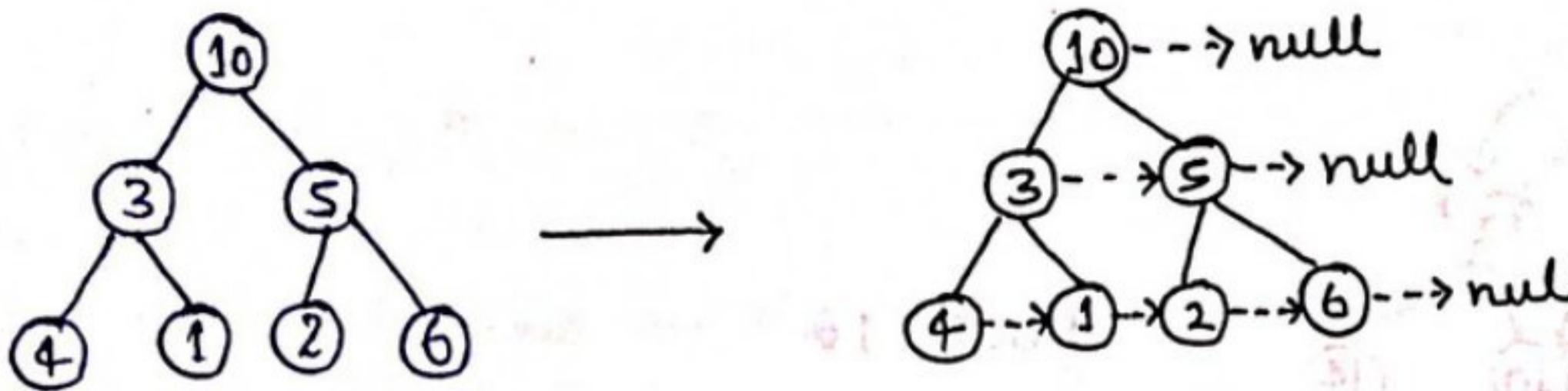
$3+5 = 8 < 50$  [make left and right 50]

$1+30 = 31 < 50$  [make left and right 50]

Tc - O(N)

Sc - O(H)

Connect nodes at the same level



connect(root){

Queue<Node> q();  
q.add(root);

Node temp = null;

while(!q.isEmpty()) {

int n = q.size();

for(i=0; i<n; i++) {

Node pre = temp;

temp = q.poll();

if(i>0) {

pre.nextRight = temp;

}

if(temp.left != null)

q.add(temp.left);

if(temp.right != null)

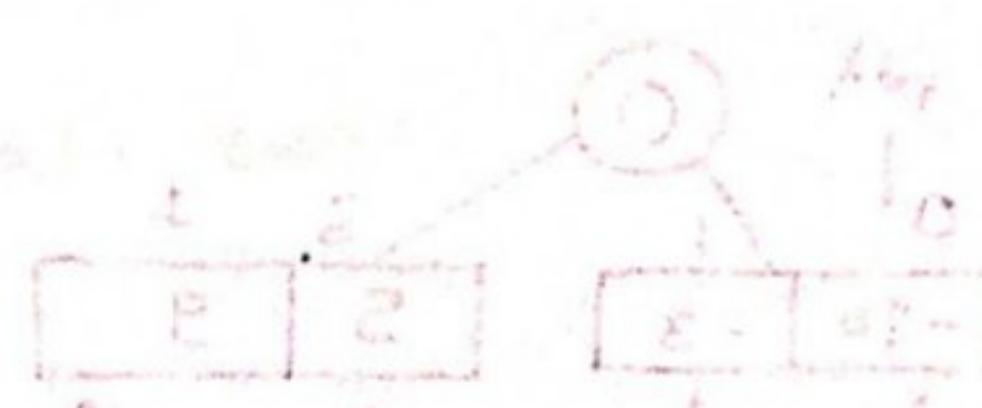
q.add(temp.right);

}

temp.nextRight = null

Time complexity - O(N)

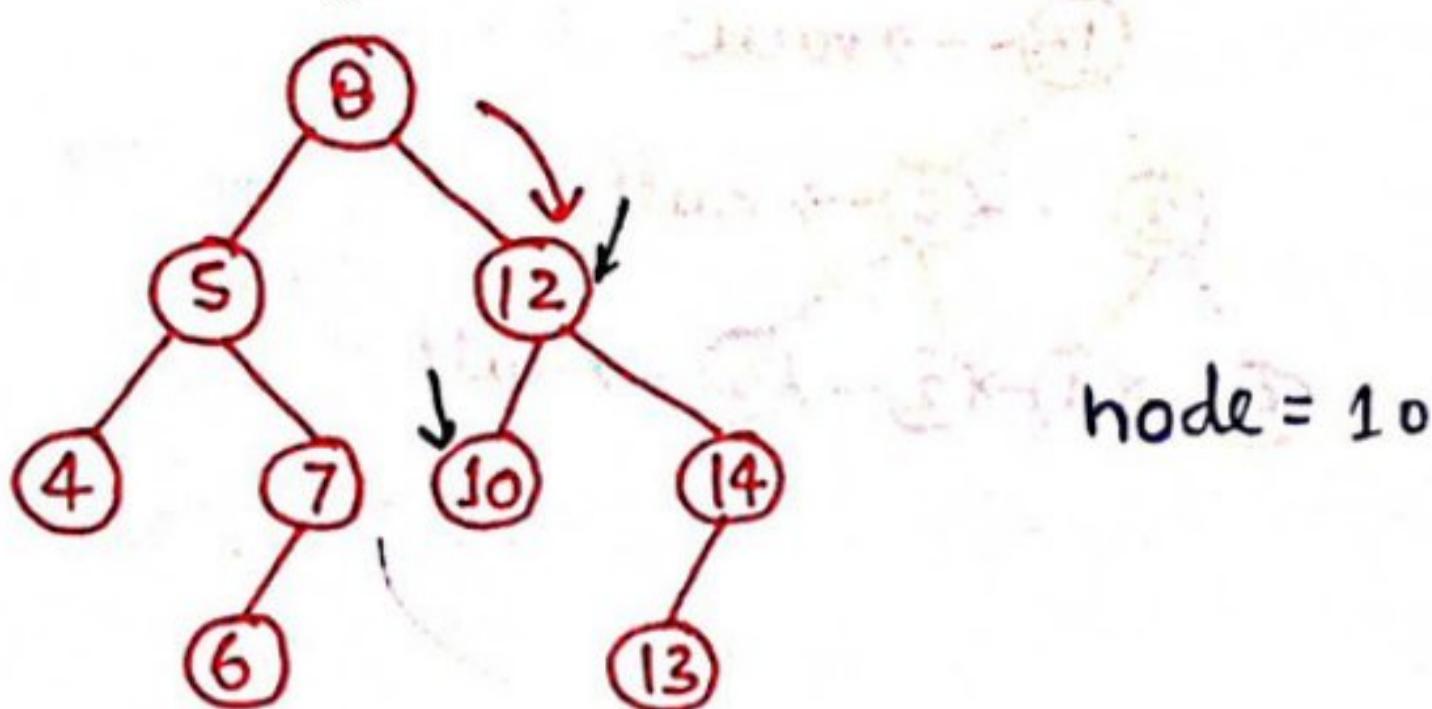
Full solution



first node <--> last node

## Search In a Binary Tree -

BST  $\rightarrow (\log_2 N)$



L < N < R

It is a BST so we can see that 8 is less than 10 so 8 is less than 10 so 10 could be lie in right.

12 is greater than 10 so goto 10

10 == val

if reach null return null:

```

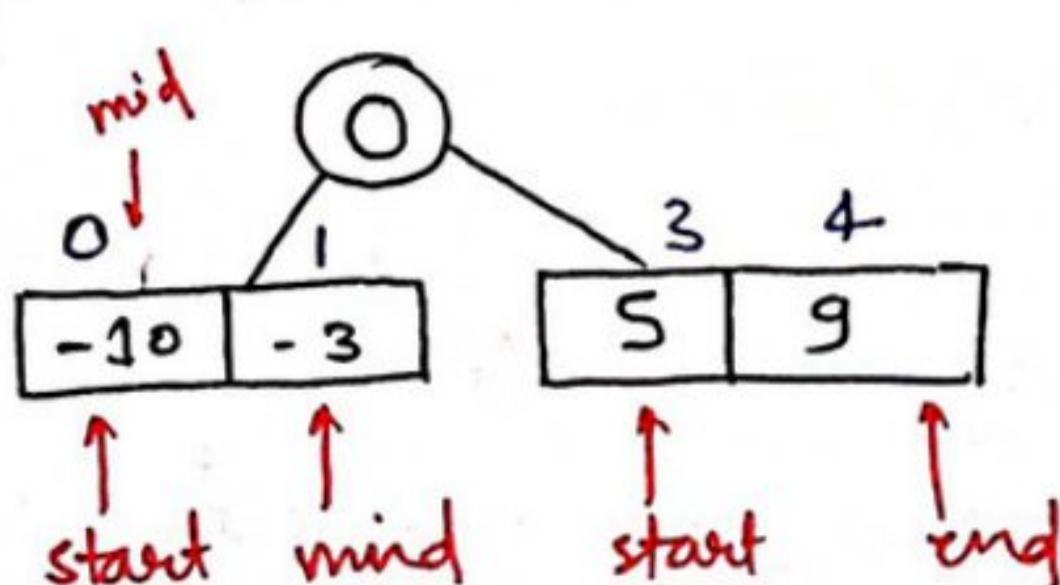
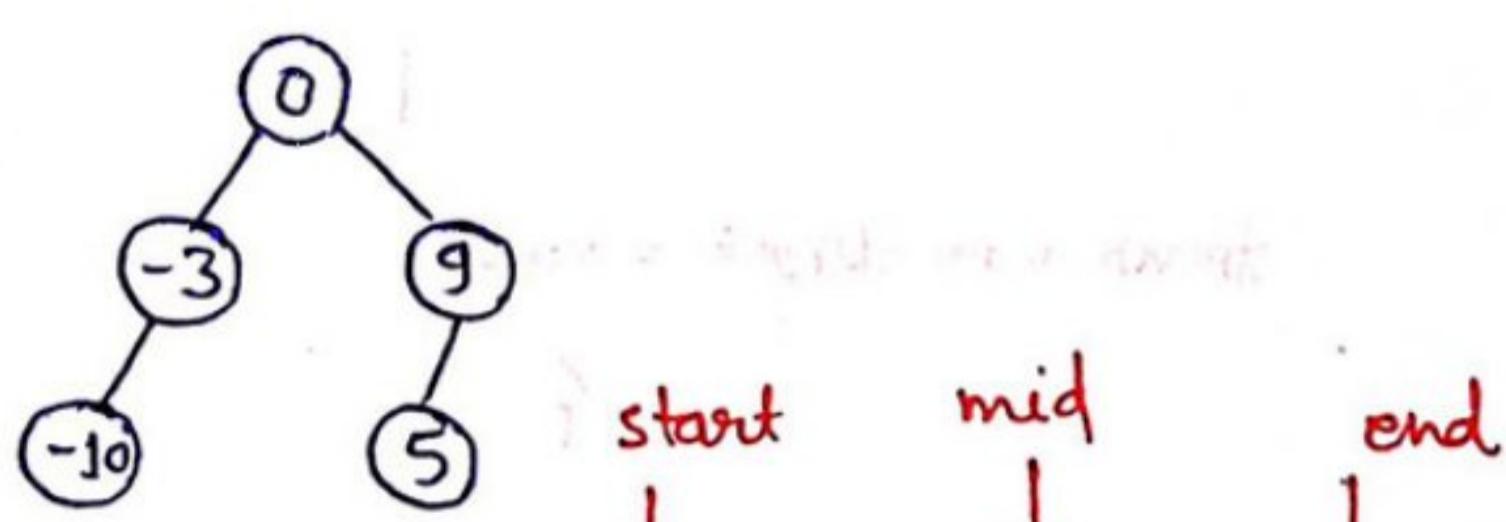
while (root != null && root.val != val) {
    root = val < root.val ? root.left : root.right;
}
return root;

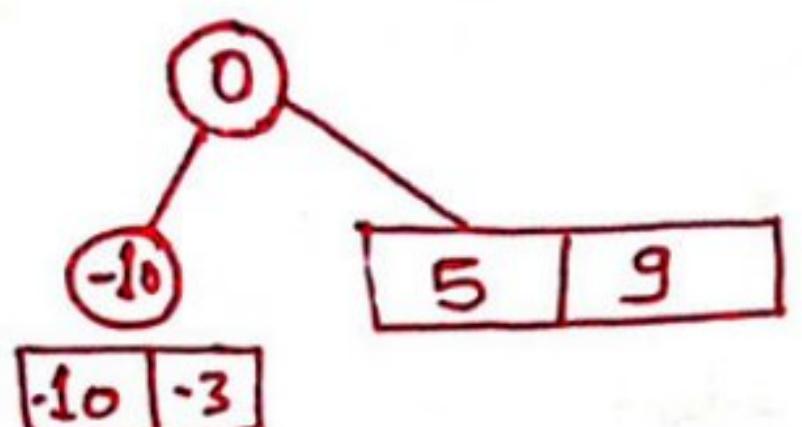
```

## Convert sorted Array to Binary Search Tree

array = [-10, -3, 0, 5, 9]

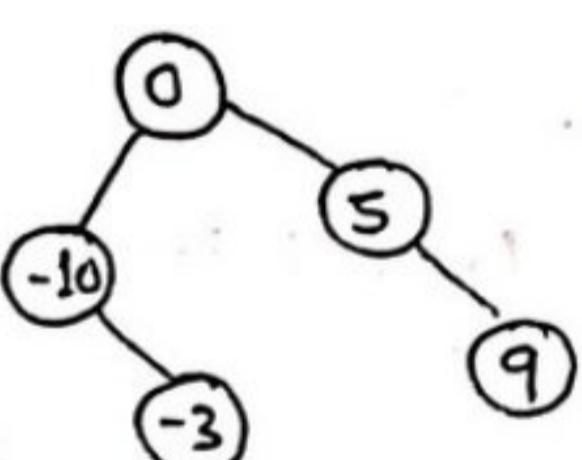
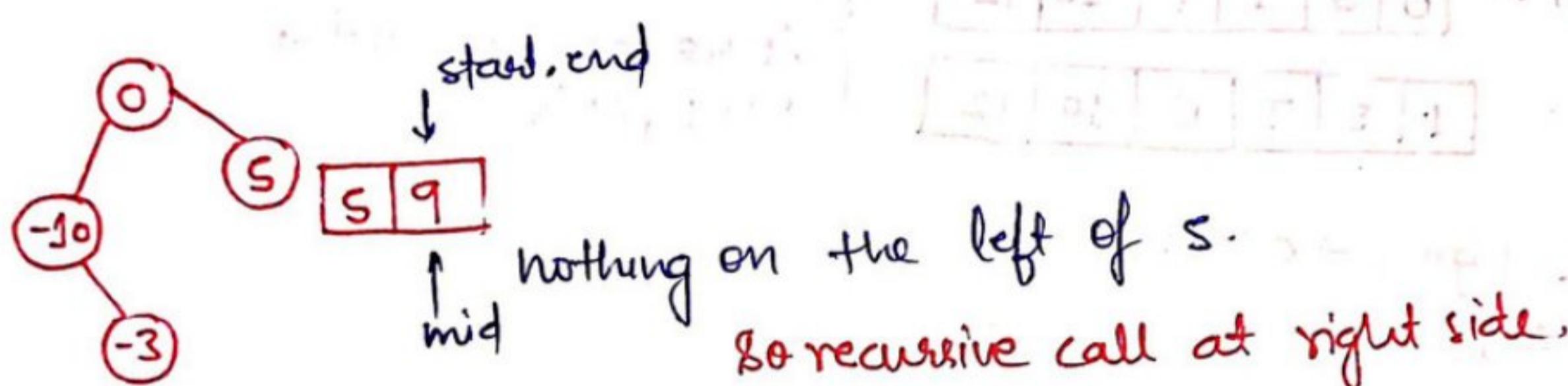
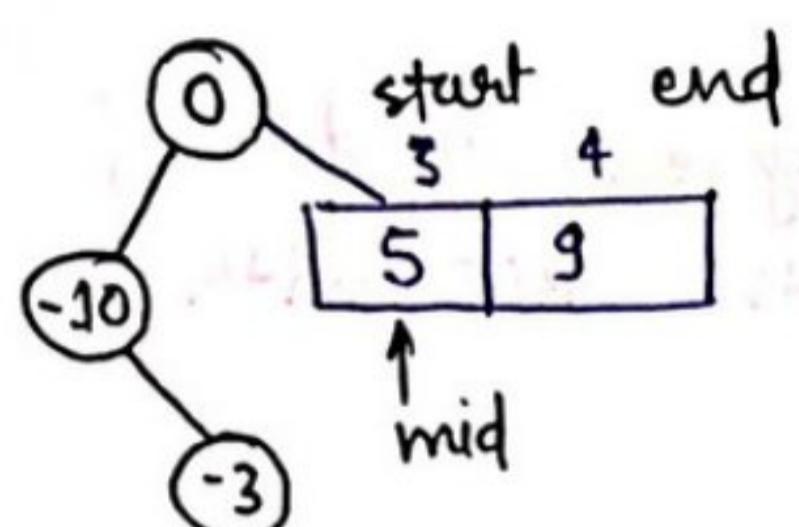
clearly show the root will be the mid element.





## DELETE

there is nothing on the left of -10 make a recursive call on right side.



```

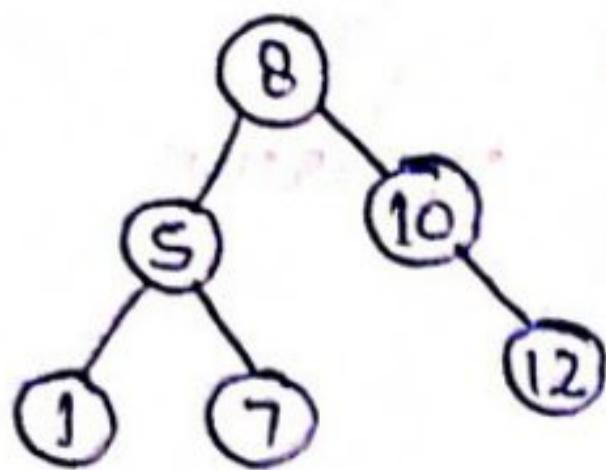
main(arr) {
    return help(arr, 0, arr.length - 1);
}

help(arr, start, end) {
    if (start > end) return null;
    int mid = (start + end) / 2;
    Node node = new Node(arr[mid]);
    node.left = help(arr, start, mid - 1);
    node.right = help(arr, mid + 1, end);
    return node;
}

```

## BST from pre-order Traversal -

preorder - [ 8 | 5 | 1 | 7 | 10 | 12 ]



Better solution - we know that preorder traversal is given and if we know the Inorder so we can construct the BST with help of both the traversal.

preorder - [ 8 | 5 | 1 | 7 | 10 | 12 ]

Inorder - [ 1 | 5 | 7 | 8 | 10 | 12 ]

} Some can create the BST by this.

$$T_c = O(N \log N) + O(N)$$

$$S_c = O(N)$$

## Optimal solution -

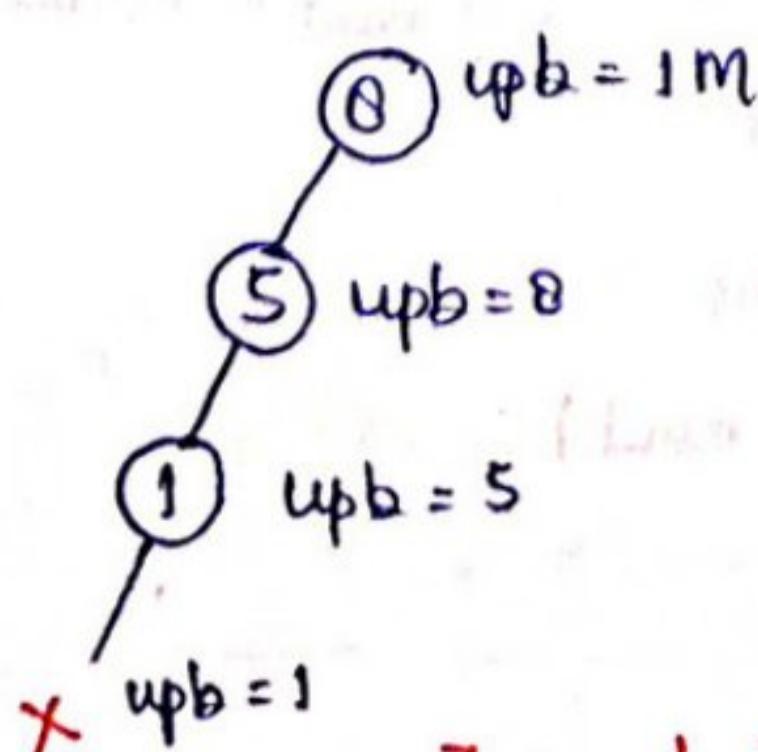
take the upperbound initially as 1M.

firstly check 8 is under the upperbound or not (yes it is).

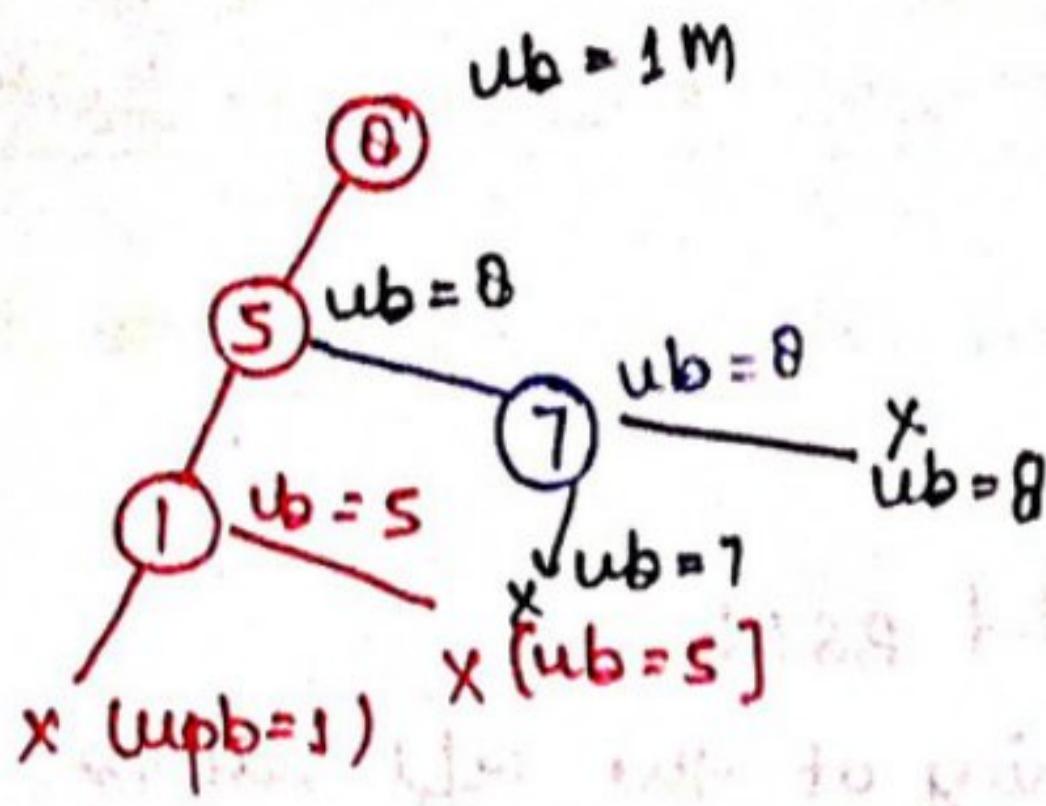
$$\textcircled{8} \quad \text{upb} = 1M$$

we dont know how many element that will lie at the left side of 8.

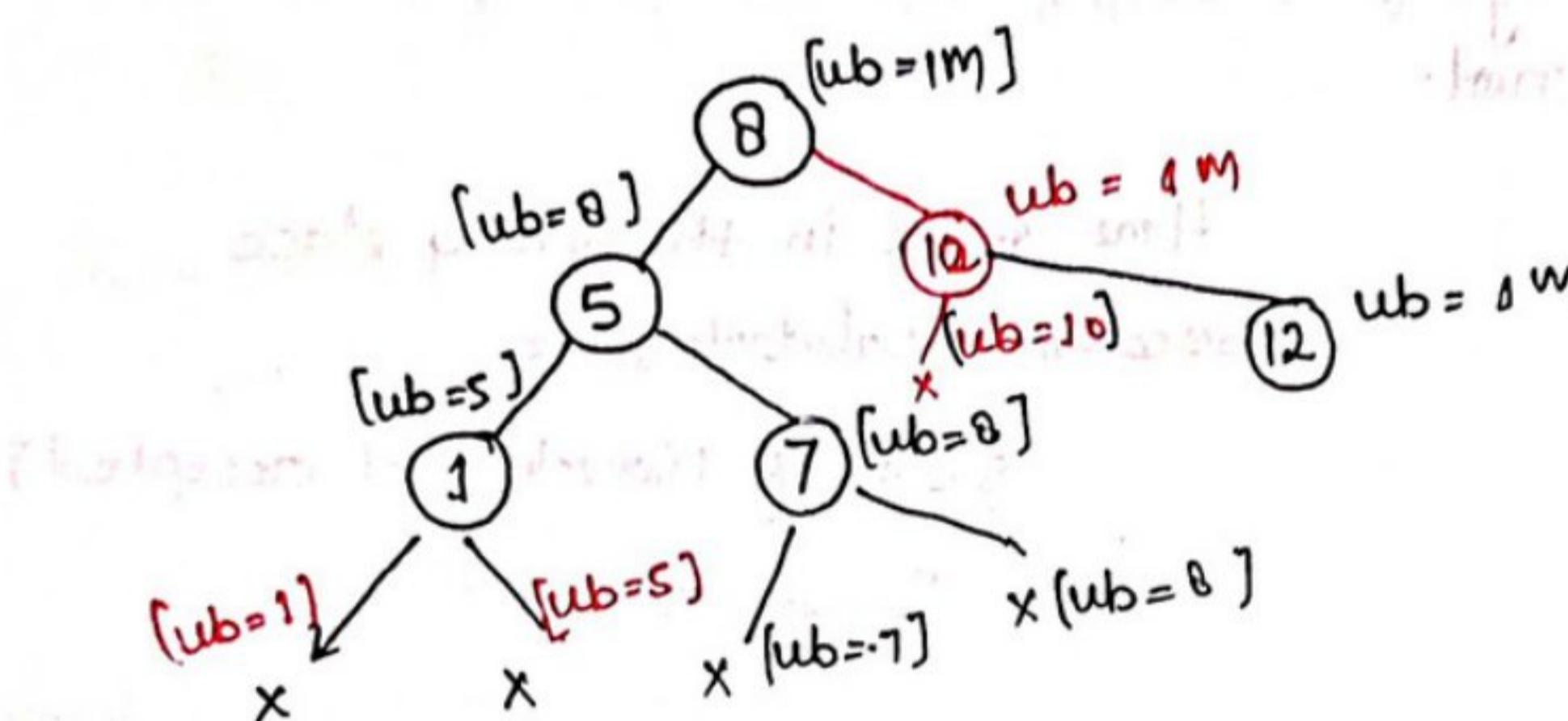
We know one thing for sure. everything at the left will always smaller than 8.



7 can't be on left. so it assign null



8	5	1	7	10	12
---	---	---	---	----	----



If we are calling left, then we have to pass that val. as a bound.  
 $\{(left, node.val)\}$

If we are calling right, then we have to the val. which is prev. Bound.

$\{(right, bound)\}$

Tc - we are traversing a singal node 3 times.

Tc -  $O(3N) \approx O(N)$

Sc -  $O(1)$ .

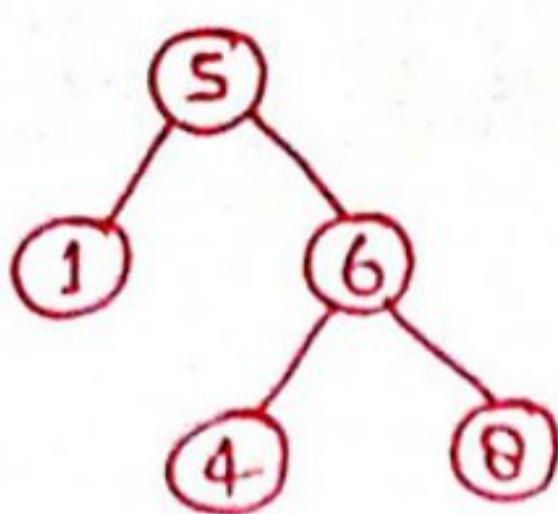
```

main (int arr) {
    return help (arr, INT_MAX, new int [] {0});
}

help (arr, bound, int [] i) {
    if (i[0] == arr.length || arr[i[0]] > bound) return null;
    TreeNode root = new TreeNode (A[i[0]+1]);
    node.left = help (arr, root, val, i);
    node.right = help (arr, bound, i);
    return root;
}

```

## Check BT is BST or not.



By this we the intuition come in our mind we will pass a range and ask every node they. you will lie at this range or not.

Range - [L, H]

initial range [-1M, 1M]

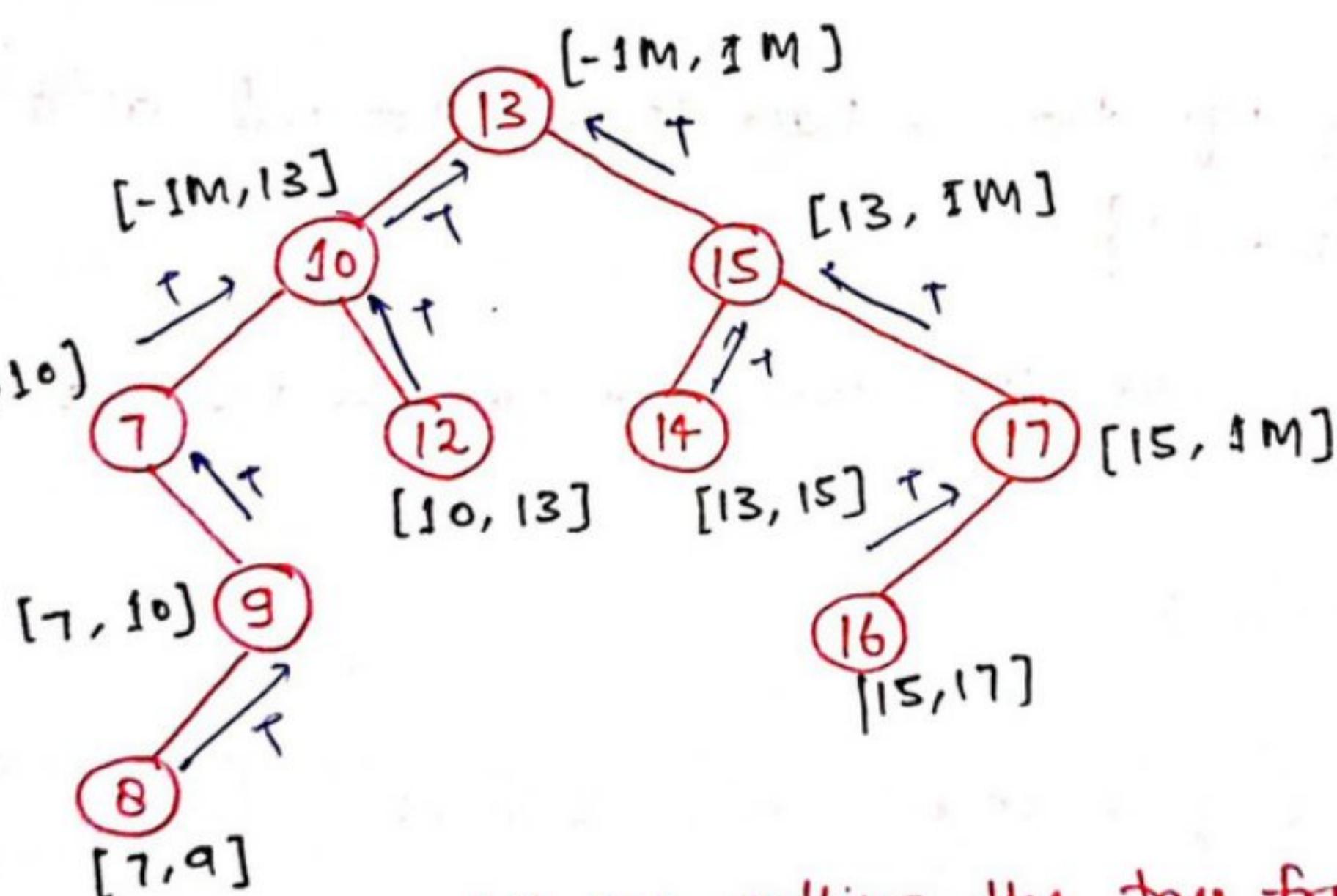
The valid BST:

- \* everything at the left has to be smaller than that node
- \* and everything at the right has to be greater.

Here 4 is in the wrong place because validation is.

$$5 < 4 < \underline{6} \text{ (which not accepted)}$$

Range [S .. E]



we are getting the tree from both

the subtree. so its a valid BST.

Tc - O(N)

Sc - O(1)

```
main(root) { return help(root, LONG_MIN, LONG_MAX); }
```

```
help(root, long minval, long maxval) {
```

if (root == null) return true;

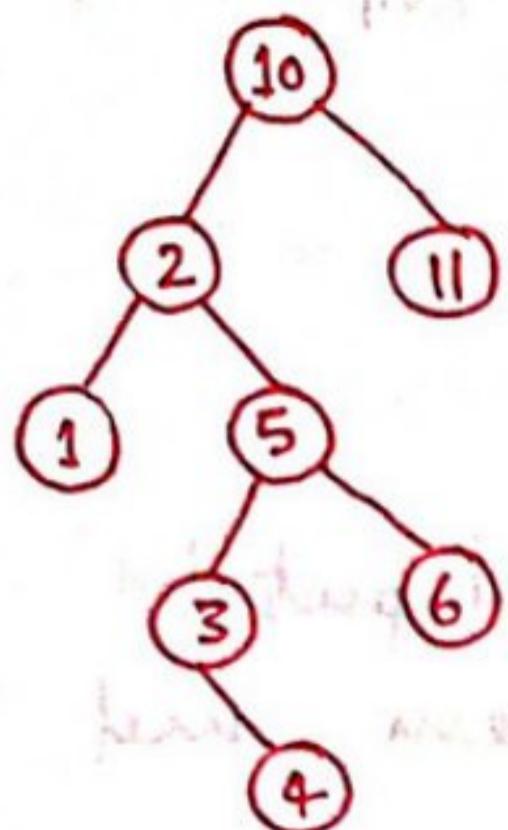
if (root.val >= maxval || root.val <= minval)  
return false;

return help(root.left, minval, root.val) && help(root.right, root.val, maxval);

## Predecessor and Successor

A key is given to us and we have to search in the tree for predecessor and successor.

If not found set it to null.



key = 8, output = [6, 10]

predecessor    successor

It is a bst so we can decide that we have to go to the left subtree/right subtree.

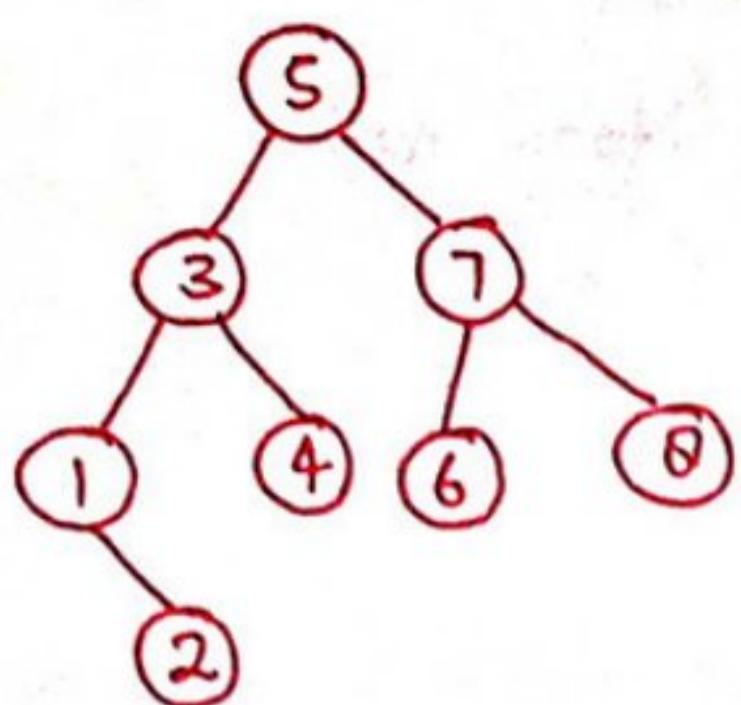
```
main(root) {
    predecessor(root, key);
    successor(root, key);
}
```

```
predecessor(root, key) {
    while (root != null) {
        if (root.data >= key) {
            root = root.left;
        } else {
            pre = root.data;
            root = root.right;
        }
    }
}
```

```
successor(root, key) {
    while (root != null) {
        if (root.data <= key) {
            // successor = root.data; //
            // root = root.left; //
            root = root.right;
        }
    }
}
```

```
else {
    successor = root.data;
    root = root.left;
}
```

## K<sup>th</sup> smallest element in a BST - / largest element



$K=3$ , we have to find the 3<sup>rd</sup> smallest solution.

**Naive solution** - take an ArrayList and do the traversal put all the elements of tree into the list and sort them and in the last get the K<sup>th</sup> element.

$$Tc = O(N) + O(N \log N)$$

$$Sc = O(N)$$

## Efficient solution -

Inorder - Left, node, Right.

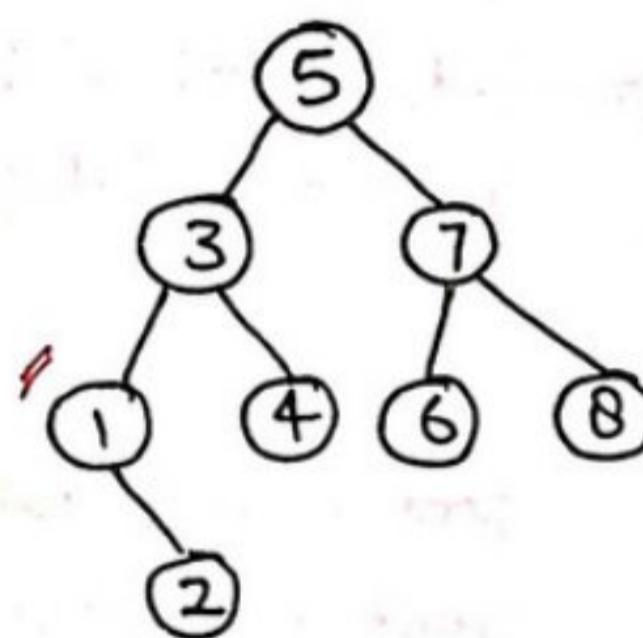
1, 2, 3, 4, 5, 6, 7, 8  
inorder traversal

If we use the inorder traversal so it gives all the elements in a sorted order.

by this we remove  $O(N \log N)$  for sorting.  
Now our task is to remove  $O(N)$  extra space.

take a cnt variable and make sure if we are visiting the node then we have to increase the cnt. by 1.

```
if (cnt == k)
    ans = node
```



$Tc = O(N)$  ] Recursion  
 $Sc = O(N)$

if we have to find  $k^{\text{th}}$  largest:

One traversal -  $O(N)$

$k^{\text{th}}$  largest =  $(N - k^{\text{th}})$  smallest.



$k^{\text{th}}$  smallest ( $\text{root}$ , int  $k$ ) {

    return help ( $\text{root}$ , new int [] { $k$ });

help ( $\text{root}$ , int  $k$  []) {

    if ( $\text{root} == \text{null}$ ) return null;

    Node left = help ( $\text{root} \cdot \text{left}$ ,  $k$ );

    if ( $\text{left} != \text{null}$ )

        return left;

$k[0]--$ ;

    if ( $k[0] == 0$ ) return  $\text{root}$ ;

    return help ( $\text{root} \cdot \text{right}$ ,  $k$ );

Logic - 1

private int cut = 0;

private int res = 0;

kth smallest ( $\text{root}$ ,  $k$ ) {

    traverse ( $\text{root}$ ,  $k$ );

    return result;

Logic - 2

traverse ( $\text{root}$ ,  $k$ ) {

    if ( $\text{root} == \text{null}$ ) return null;

    traverse ( $\text{root} \cdot \text{left}$ ,  $k$ );

    cnt++;

    if ( $cnt == k$ )

        res =  $\text{root} \cdot \text{data}$ ;

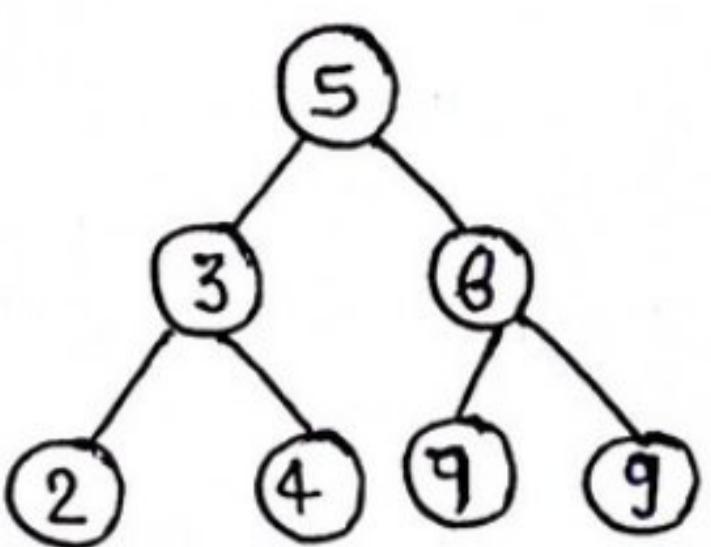
        return;

    }

    traverse ( $\text{root} \cdot \text{right}$ ,  $k$ );

}

## Pair with given sum in BST

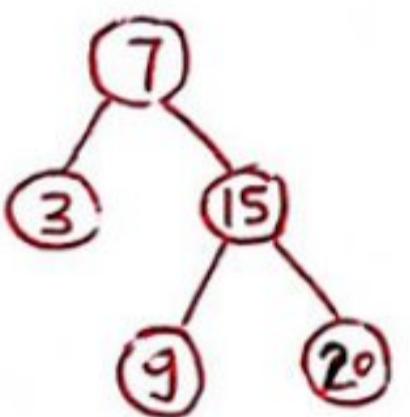


we can solve this problem using HashSet.

```

static Set<int> st;
main (root, k) {
    st = new HashSet<>();
    return help(root, k);
}
help (root, k) {
    if (root == null) return false;
    if (st.contains (root.val + k)) return true;
    st.add (root.val);
    return help (root.left, k) || help (root.right, k);
}
  
```

## Binary Search Iterator



so we have to design a class of functions so we can operate this operation.

BSTIterator(7) → root  
 next → inorder's guy.  
 next → 3  
 next → 7  
 hasNext → yes.  
 next → 9  
 hasNext → yes  
 next → 15  
 hasNext → yes  
 next → 20  
 hasNext → no

if we take an array & store all the integers so we can perform the operations easily.

3	7	9	15	20
---	---	---	----	----



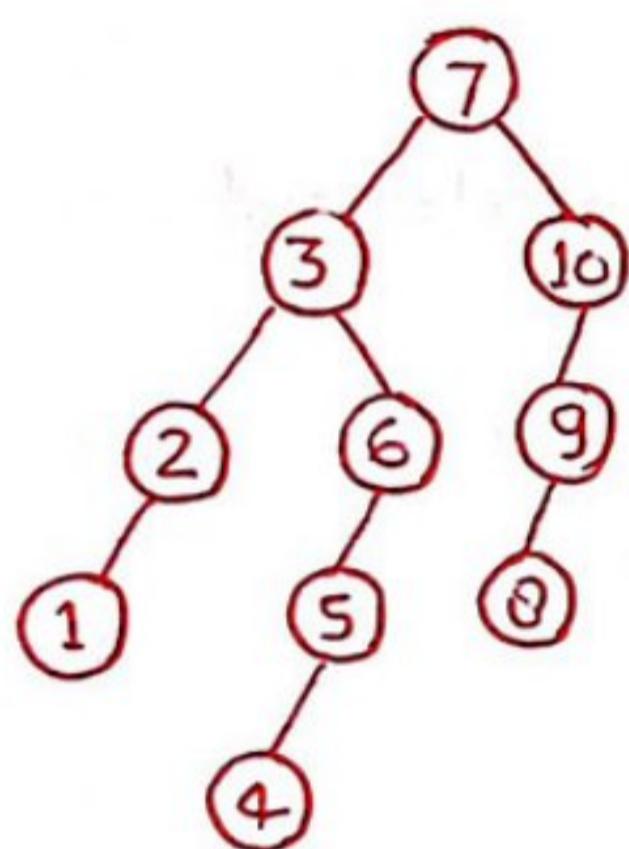
next → 3

hasnext → yes

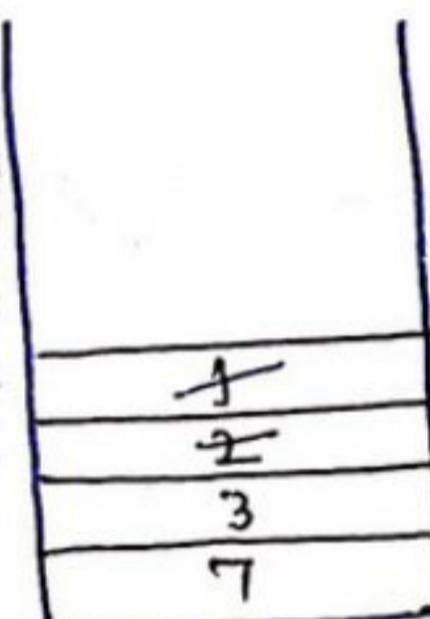
it will be easy.

But the  $Tc = O(1)$  and  $Sc = O(n)$ , because we are storing all the values in the array.

Inorder - L, N, R



class → BSTIterator(7)



Stack.

we will go in extreme left, so we will do in the construct we will go in extreme left. and put everything into stack

Now if someone calls 1's next, as it's null, so check the stack's first element.

when again next called, does 2 have a right.

check 3 has a right?

Yes it has

next - 1

next - 2

next → 3

Go to right,

Now put everything into stack.

similarly 5 and 4 added because left is called for 6.

Now 4 left will be null.

Now if some say. hasNext ?  
empty , then return true ;

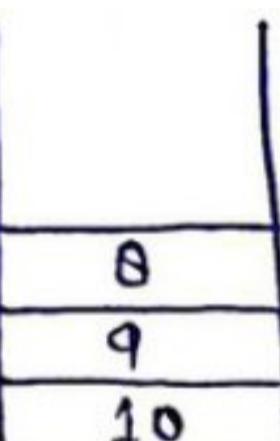
next → 4

4 doesn't have a right

next → 5    5    "    "    "

next → 6    6    "    "    "

next → 7    yes 7 have a right , put all the element



Similar things will happen -

SC - O(H)

TC - O(1)

class BSTIterator {

    private Stack<TreeNode> st = new Stack<>();

    public BSTIterator (root) {

        pushAll (root);

}

    private void pushAll (root) {

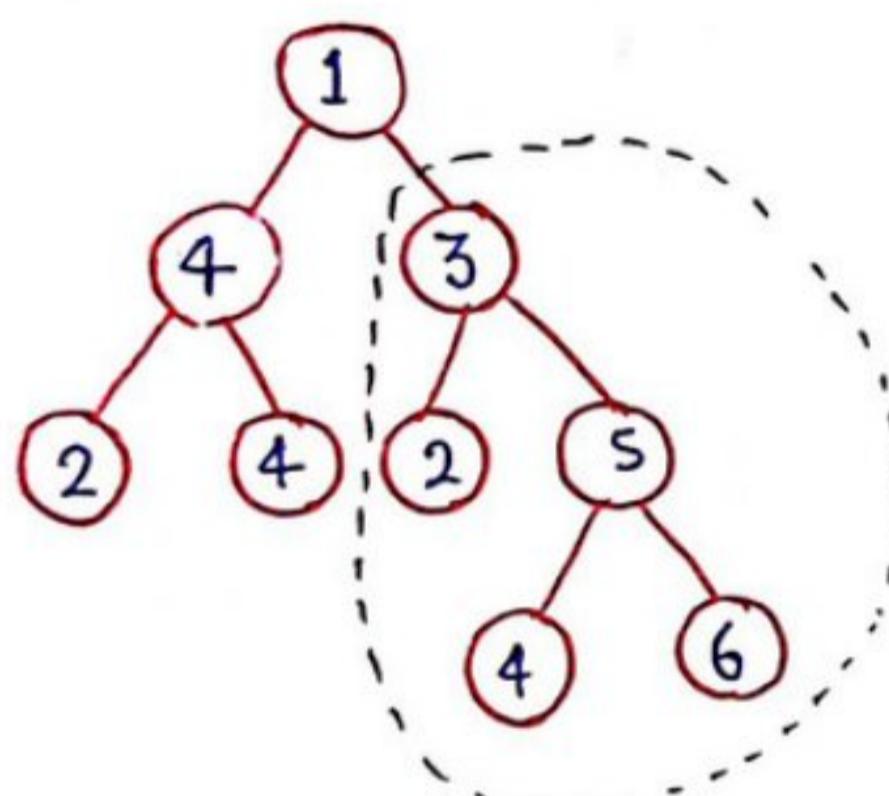
        for (; node != null; st.push (node), node = node.left);

}

```
public boolean hasNext() {  
    return !st.isEmpty();  
}
```

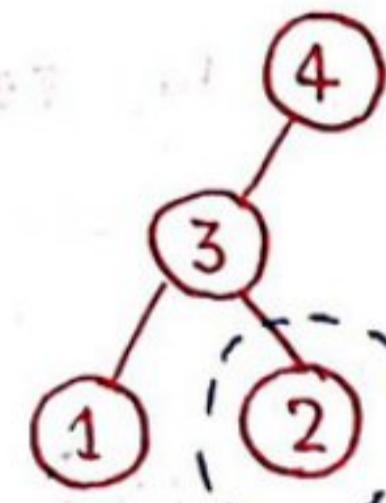
```
public int next() {  
    TreeNode node = st.pop();  
    pushAll(node.right);  
    return node.val;  
}
```

### Maximum Sum in BST in a Binary Tree



this is a BST with a max sum.

node  
everything small      everything greater

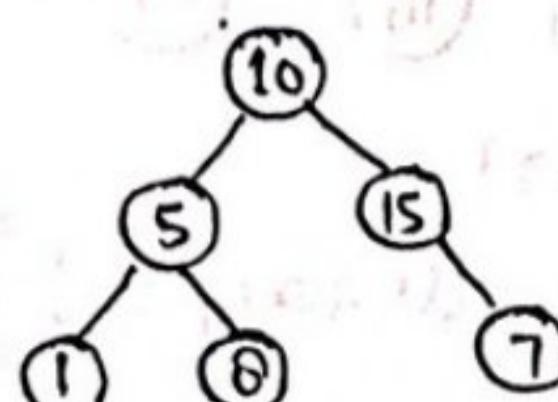


→ BST with max sum.

Bruteforce - I will validate the BST.

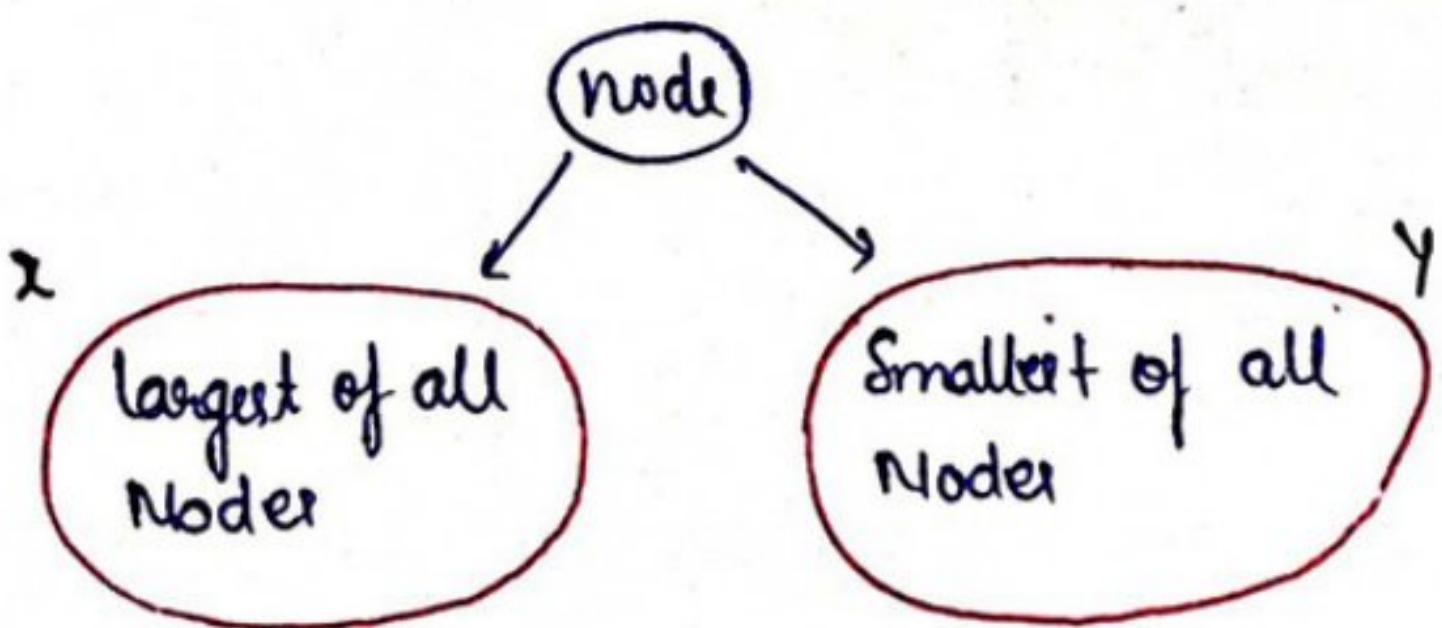
So take a min max and validate by going on every node

to → validate BST.  
is  $\leftarrow$  yes.



validate BST  $\rightarrow O(N) \times O(N) = O(N^2)$

going to every node

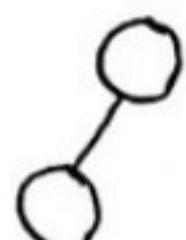


$$\text{size} = 1 + x + y$$

$\text{largest} < \text{node} < \text{smallest}$

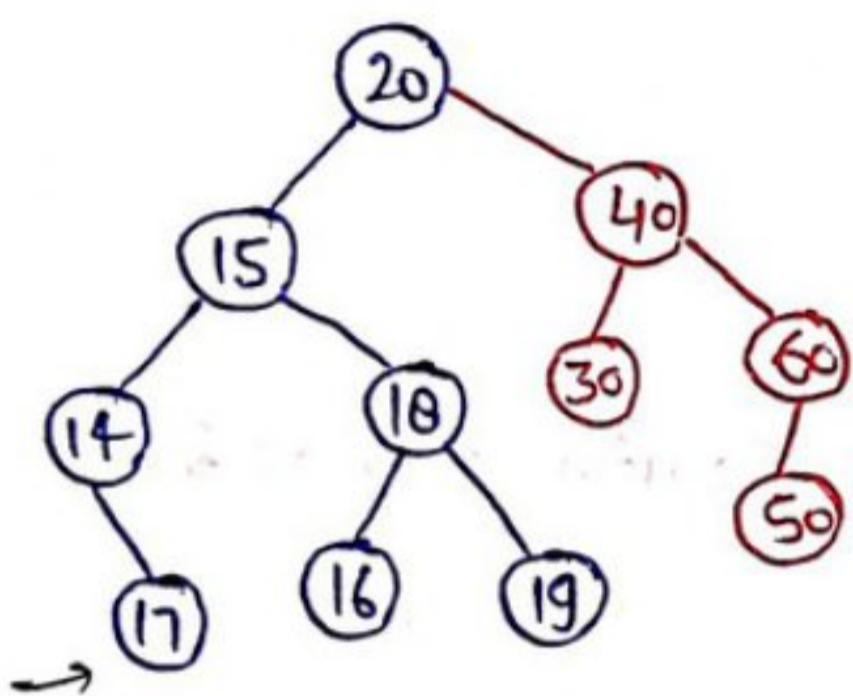
We will go to bottom and we know the bottom guy will be a BST → for sure. and it's connected to a node.

Idea - Build the things from bottom.

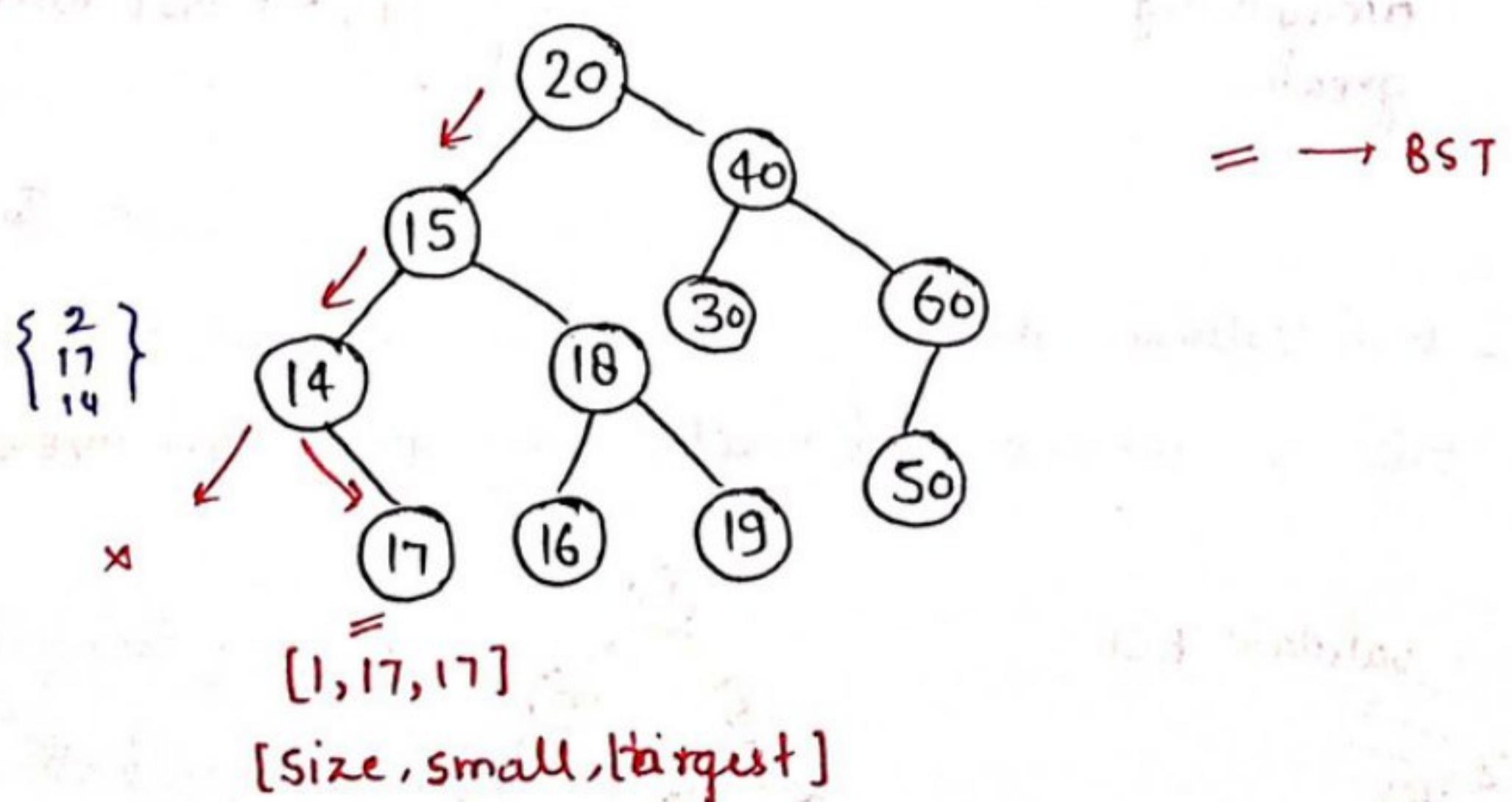


if satisfied the condition  
the BST

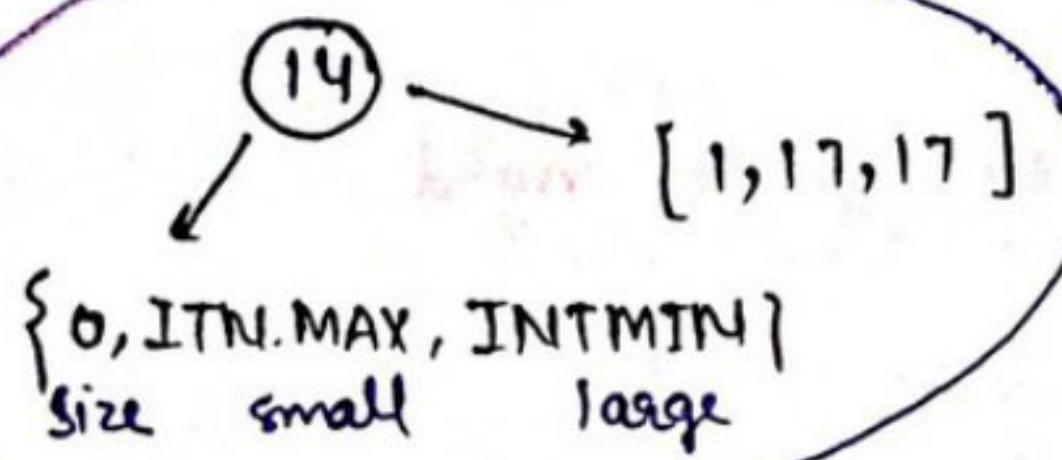
store the largest, smallest and size.



We have to do traversal L, R, N so we take all elements, L and R  
postorder - L, R, N



If there is no node on left and right so pass the dummy node of size → 0, for largest take INT-MIN and for smallest take INT-MAX.

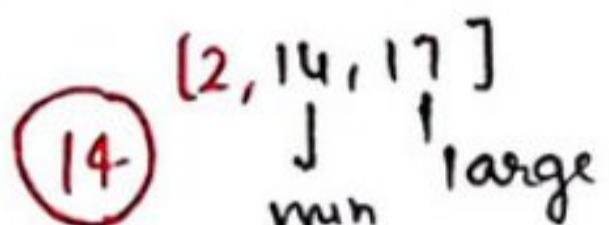


→ so when will be this will be a valid BST.

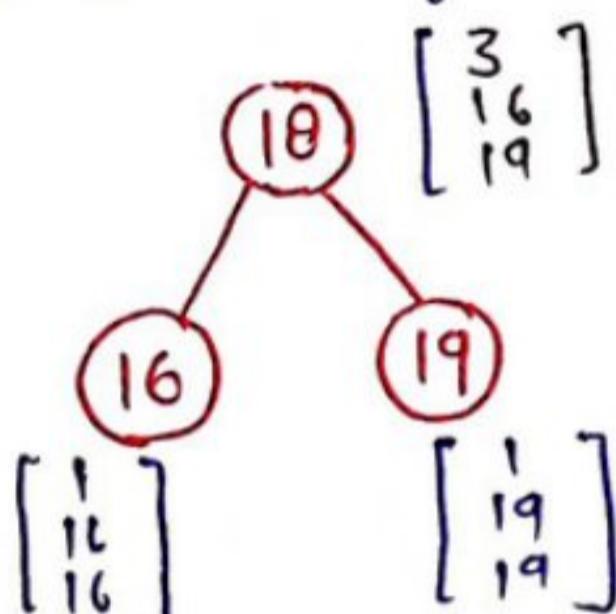
for 14 to be a valid BST.

$$\text{INTMIN} < 14 < \text{INTMAX}$$

for 14 → size will be  $1+x+y \Rightarrow 1+0+1 = 2$



Now for 15 the left is completed.



so if come to 18 so check.

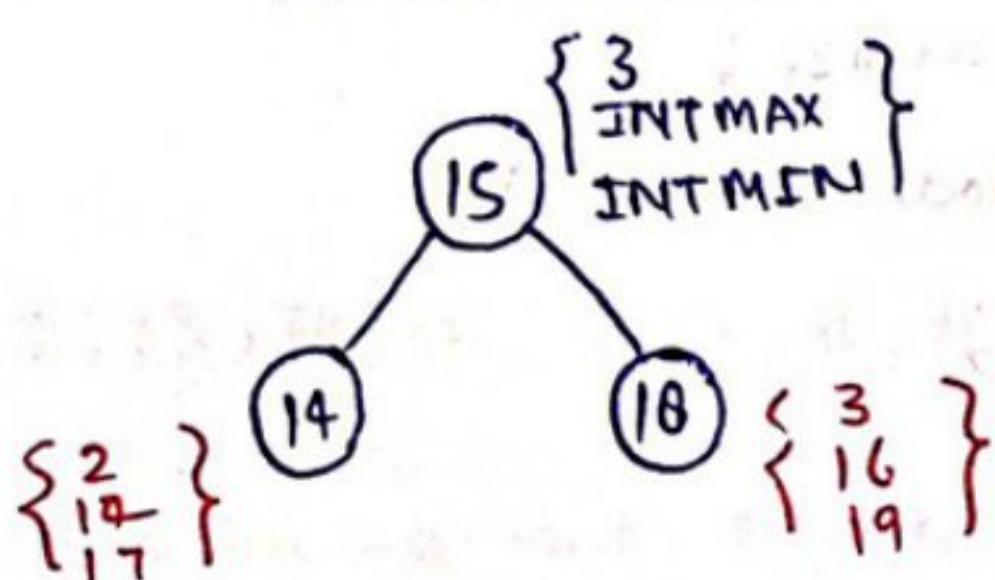
node, which is 18. is greater than largest of left subtree and smaller than the smallest of right subtree

$$16 < 18 < 19 \rightarrow \text{YES}, \text{ so } 18, \text{ will contain BST.}$$

$$\text{Size for } 18 \rightarrow 1+x+y = 1+1+1 = 3$$

Now come to 15.

At the moment we come to 15., look at largest of left 17 and minimal of right 16



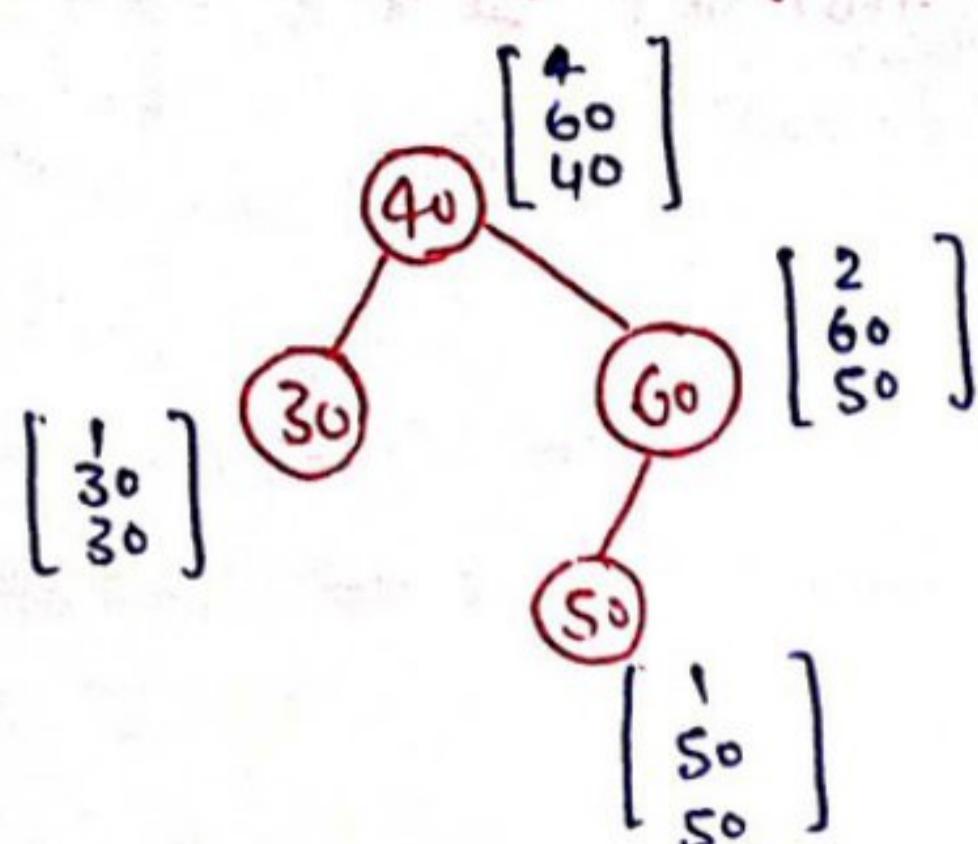
$$17 < 15 (\times \text{not following})$$

right is OK but left not

Not in that condition pass the INTMIN and MAX

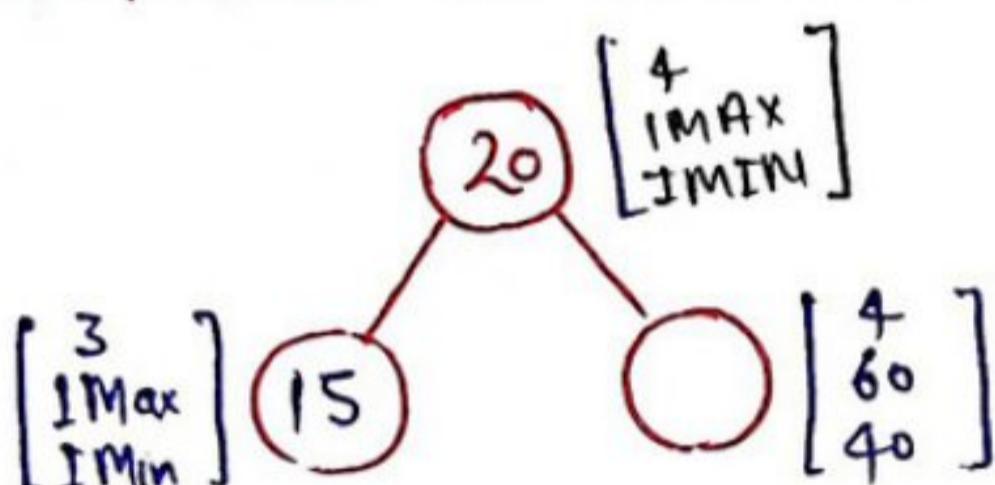
we pass MIN and MAX so comparison won't be possible in future.

Now everything in left of 20 completed go to right.



$$30 < 40 < 50 \text{ (true)}$$

At moment we came 20.



$$\underline{\text{INTMAX} \leq 20 \leq \text{INTMIN}} \quad \begin{matrix} x \\ \checkmark \end{matrix}$$

But largest BST

if BST.size = 1 + x + y otherwise = max(left, right)

Tc - O(N)

Sc → O(1)

```
class NodeValue {  
    public int maxNode, minNode, maxSize;  
    NodeValue {int maxNode, minNode, maxSize){  
        this.maxNode = maxNode;  
        this.minNode = minNode;  
        this.maxSize = maxSize;  
    }  
}
```

```

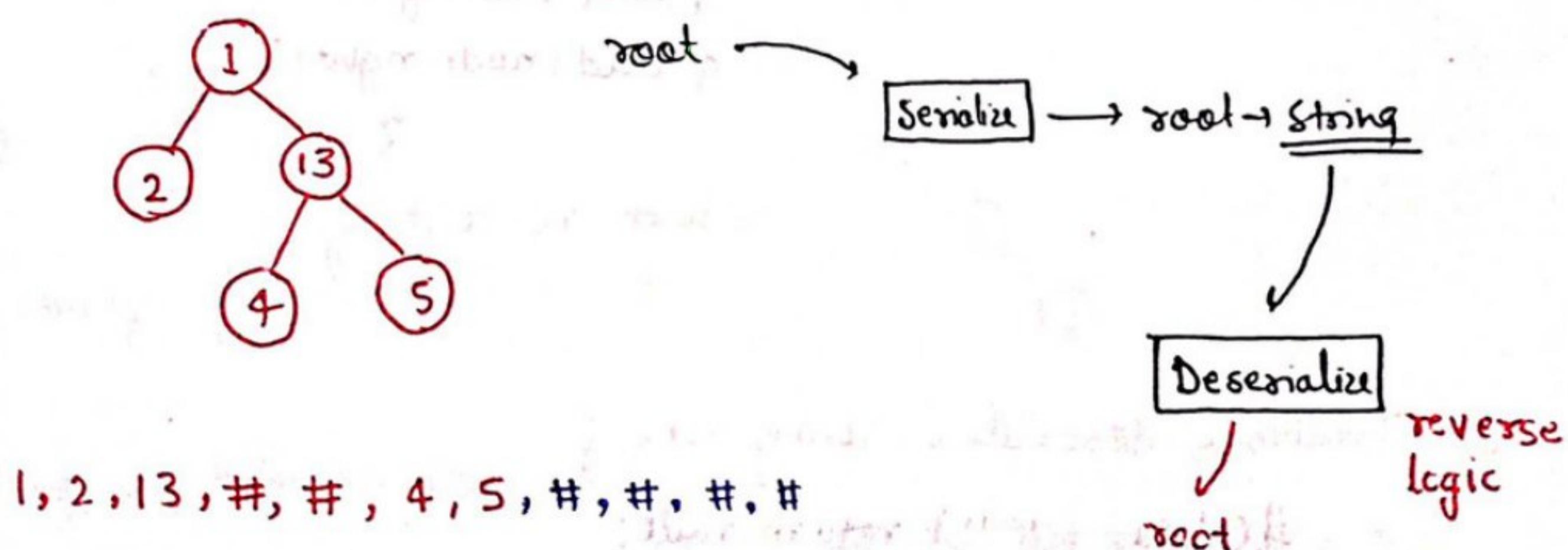
class Solution {
    main(root) {
        if (root == null) {
            return new NodeValue (INT-MAX, INT-MIN, 0);
        }
        Node left = main(root.left);
        Node right = main_(root.right);

        if (left.maxNode < root.val && root.val < right.minNode) {
            return new NodeValue (Math.min (root.val, left.minNode), Math.max (root.val,
                right.maxNode),
                (left.maxSize + right.maxSize + 1));
        }
        return new NodeValue (INTMIN, INTMAX, max (left.maxsize, right.maxsize));
    }
}

main() {
    return main_(root).maxSize;
}

```

## Serialize and Deserialize Binary Tree -



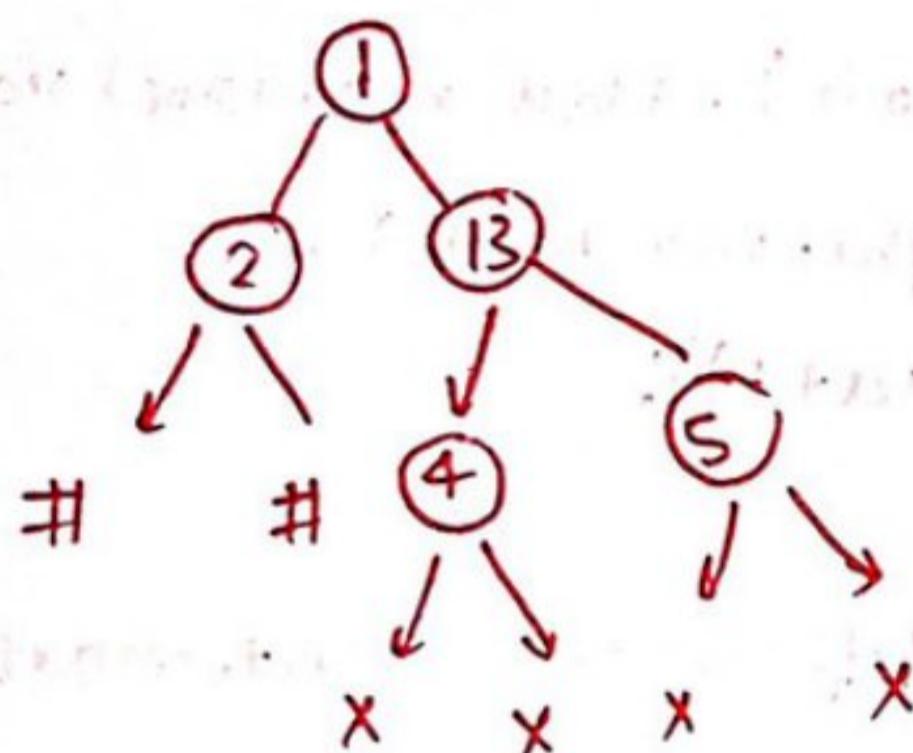
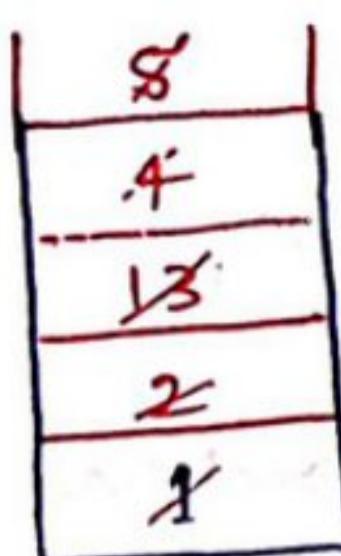
Make sure there have a null on leaf node

Now we have serialize -

the string we get. 1, 2, 13, #, #, 4, 5, #, #, #, #  
↑ ↑

Now for deserialize -

take queue ds.



```
public String serialize(TreeNode root) {
    if (root == null) return "";
    Queue<TreeNode> q = new LinkedList<>();
    StringBuilder res = new StringBuilder();
    q.add(root);
    while (!q.isEmpty()) {
        TreeNode node = q.poll();
        if (node == null) {
            res.append("n ");
            continue;
        }
        res.append(node.val + " ");
        q.add(node.left);
        q.add(node.right);
    }
    return res.toString();
}
```

```
public TreeNode deserialize(String data) {
```

```
    if (data == "") return null;
```

```
    Queue<TreeNode> q = new LinkedList<>();
```

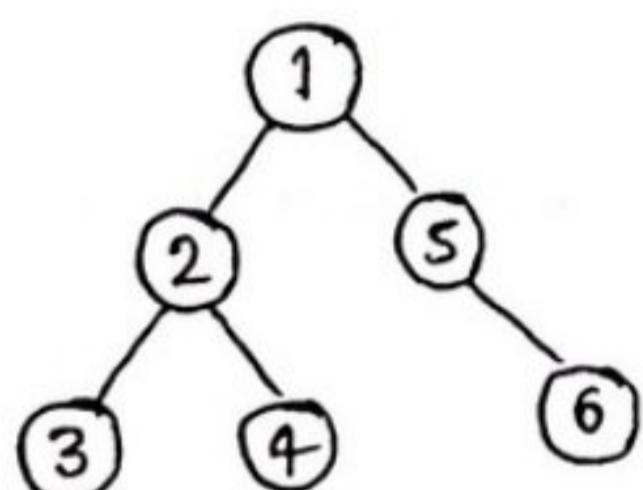
```
    String[] values = data.split(" ");
```

```

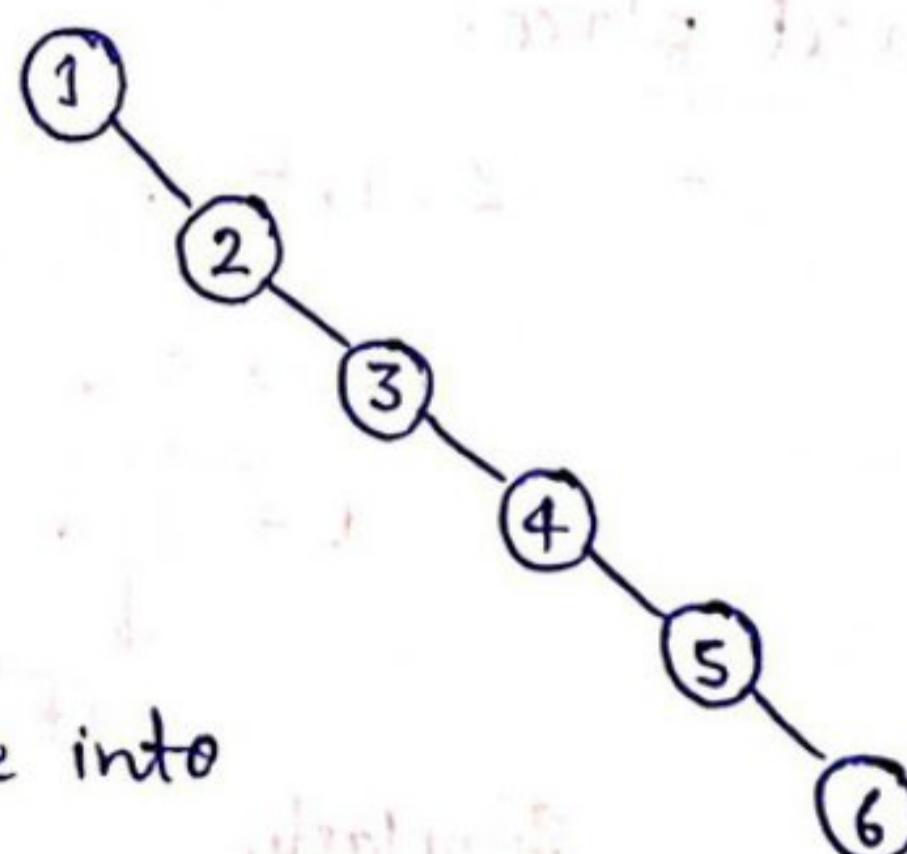
TreeNode root = new TreeNode (Integer.parseInt(values[0]));
q.add(root);
for (int i=1; i<values.length; i++) {
    TreeNode parent = q.poll();
    if (!values[i].equals("n")) {
        TreeNode left = new TreeNode (Integer.parseInt(values[i]));
        parent.left = left;
        q.add(left);
    }
    if (!values[++i].equals("n")) {
        TreeNode right = new TreeNode (Integer.parseInt(values[i]));
        parent.right = right;
        q.add(right);
    }
}
return root;
}

```

### Flatten Binary Tree to linkedlist



O/P  
→



We have to change the Binary Tree into linkedlist.

The thing we have to observe -

so 6 is at the bottom.

The conditions -

If we see the output at the left

bottom of column at binary tree

(End) about this it is no problem

- linkedlist should use the same `TreeNode`: class where the right child pointer points to the next node in the list and the left child pointer is always null.

class Solution {

```
    TreeNode pre = null;
```

```
    flatten(TreeNode root) {
```

```
        if (root == null) return;
```

```
        flatten(root.right);
```

```
        flatten(root.left);
```

```
        root.right = pre;
```

```
        root.left = null;
```

```
        pre = root;
```

```
}
```

```
}
```

## find Median from Data Stream

The Number are coming in the array stream and at any moment it can ask us for the midian.

Median → We can find the median if arr is sorted

if new Number is added so it has to be add at the exact and correct place.

3, 1, 5, 4 . . .

a → 3, median = 3

a = 3, 1

↓

sort → 1, 3 median = 2

Similarly.

a = 1, 3, 4, 5 median = 3.5

so for putting all the Number in the stream in the exact place to make sorted -

Tc - O(N) and there are N Number so it will take - O(N^2)

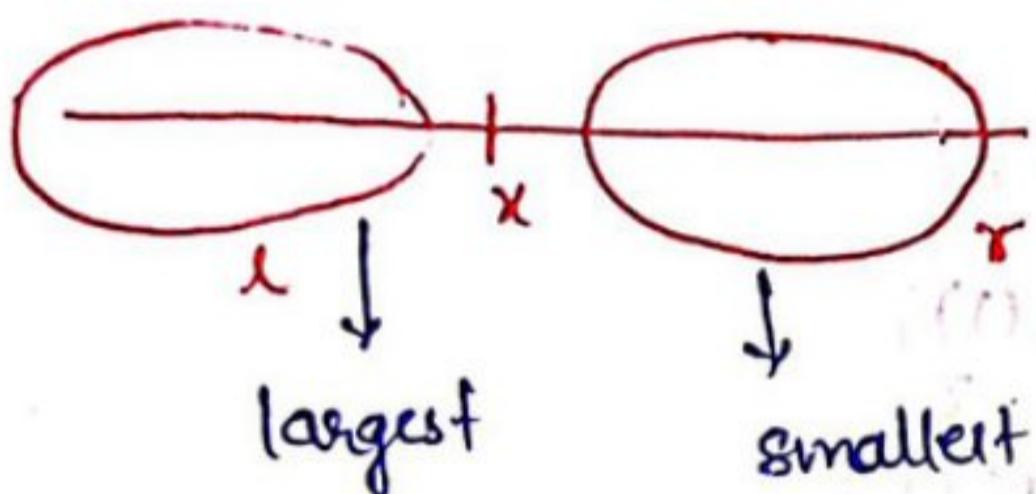
## Optimize -

Observation is we don't have to sort this we just need to return the median, and we know median the middle number.

smaller < median < greater

if x is a median so

the Number on right side will be equal to left side.



so we can find the median with the help of left part largest number and right part smallest number.

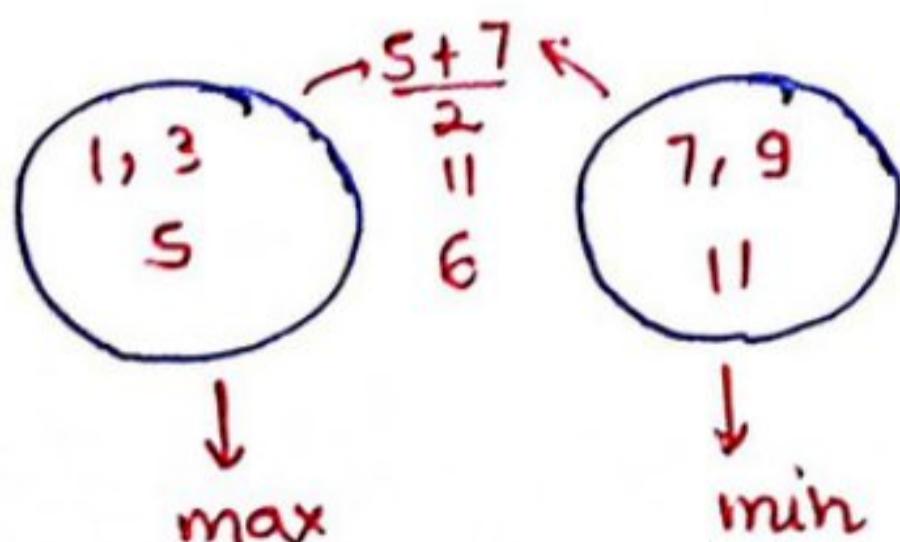
we know we can use maxHeap for getting the max Element

Similarly  
for min Heap.

if number is odd to  $\frac{n+1}{2}$  element will on left heap and other added to minHeap.

if even so works like as before.

1, 3, 5, 7, 9, 11 → even



if odd so median will largest of maxHeap.

```
class Main {
    Priority Queue<int> maxHeap = new Priority Queue<>(Collections.reverse());
    Priority Queue<int> minHeap = new Priority Queue<>();
    insertNum(int num) {
        if (maxHeap.isEmpty() || maxHeap.peek() >= num) {
            maxHeap.add(num);
        } else {
            minHeap.add(num);
        }
        if (maxHeap.size() > minHeap.size() + 1) {
            minHeap.add(maxHeap.poll());
        } else if (maxHeap.size() < minHeap.size()) {
            maxHeap.add(minHeap.poll());
        }
    }
    double findMedian() {
        if (maxHeap.size() == minHeap.size()) {
            return maxHeap.peek() / 2.0 + minHeap.peek() / 2.0;
        } else {
            return maxHeap.peek();
        }
    }
}
```