Striver

Dsa

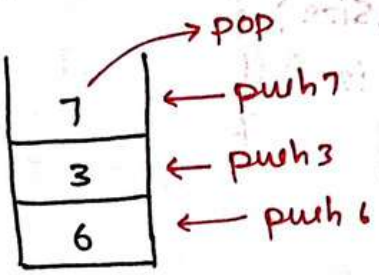
Sheet

"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;
            }
```

```
int top(){
return arr[top];
        }
int size (){
return top+1;
        }
```

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

Implement Queue using Arrays : Problem-75

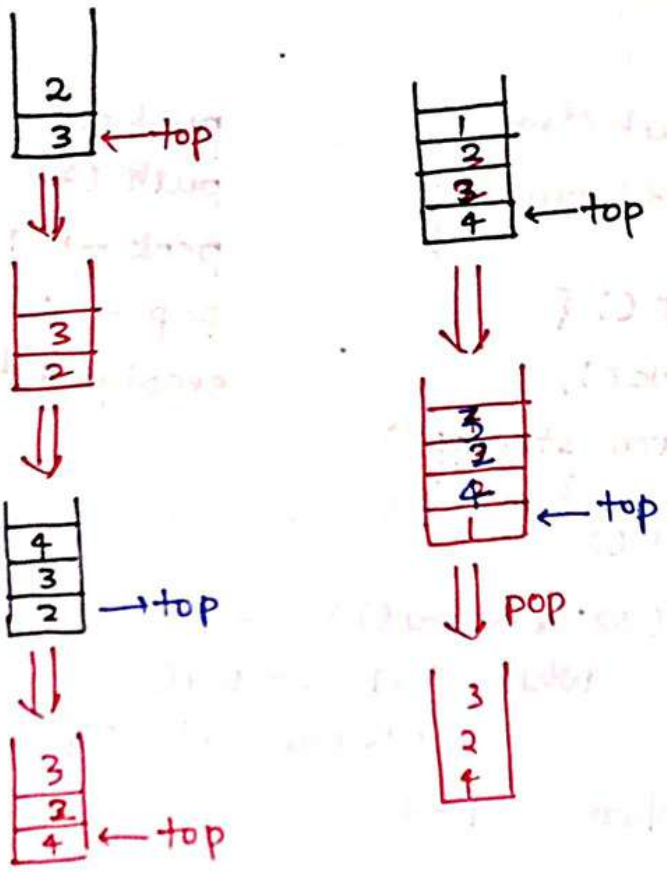
```
privat int arr[];
private int start, end, currsize, maxsize ;
public Queue (){
        arr = new int[16];
        start = -1;
        end = -1;
        currsize = 0;
            }
public Queue (int maxsize){
        this.maxsize = maxSize;
        arr = new int[maxSize];
        start = -1;
    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

Tc - O(N)

Implement stack using Queue. : Problem - 76

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



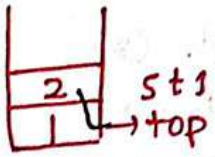
after pushing a new element remove all the element from the queue and again put into the queue.

```
Queue<Integer> ();
void push(int x){
        q.add(x);
        for(i=0;i<q.size()-1;i++){
                q.add(q.remove());
                                }
                }
int pop()
        return q.remove();
int top()
        return q.peek();
int size()
        return q.size();
```

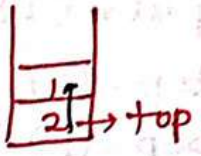
TC - O(N)
SC - O(N)

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 && it is balance otherwise no.

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

Next Greater Element : Problem-79

N = 4, arr[] = [1, 3, 2, 4]
output [3, 4, 4, -1]

traverse the array from last to 0th index.

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

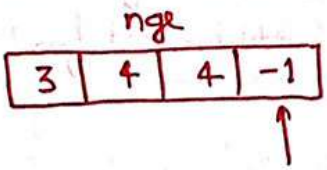
SC- O(N)
TC- O(N)

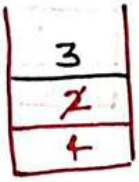
this is the optimal Approach.

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 - $O(N^2)$.

[1, 3, 2, 4]

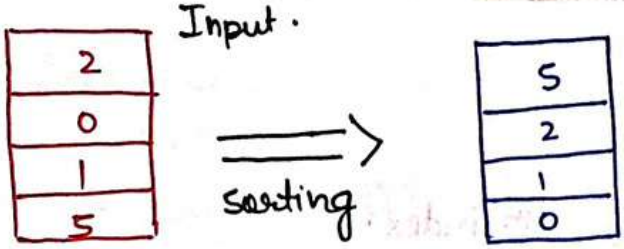




initially stack size = 0

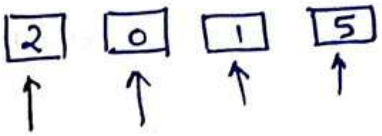
3 > 2 (false) remove 2

Sort a Stack : Problem-80

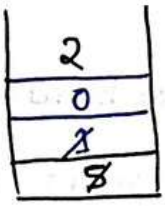


print - 5, 2, 1, 0

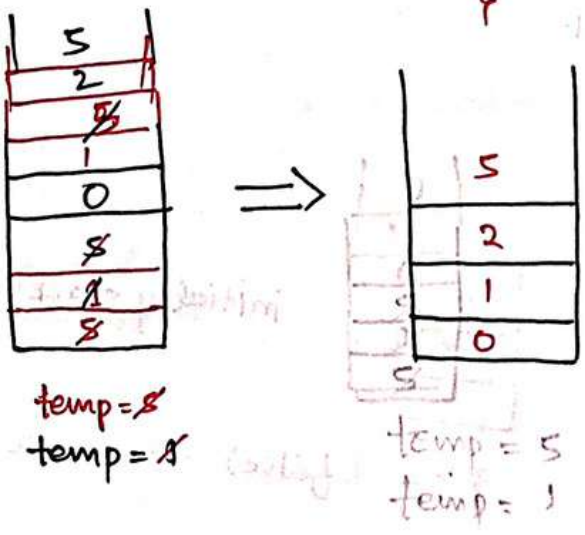
```
main (Stack<int> s) {  ⟶ fun name - sortStack
    if (!s.isEmpty()) {
        int x = s.pop();
        sortStack (s);
        sortedInsert (s, x);
    }
}
```




```
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 = 5
temp = 1



temp = 8
temp = 1

temp = 5
temp = 1

Hypothesis

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

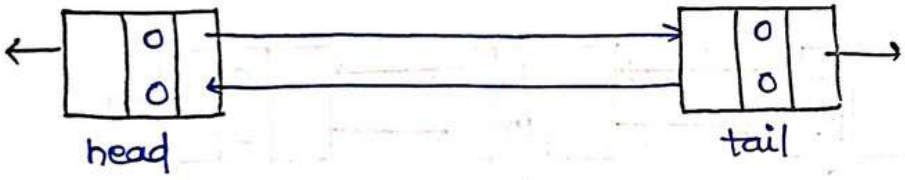
LRU cache - Problem-02

- 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.

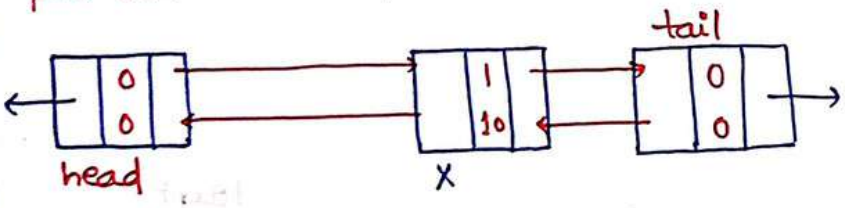
Intution - While inserting {key, val} pair into DDL make sure that we are inserting it from the back tail to head.

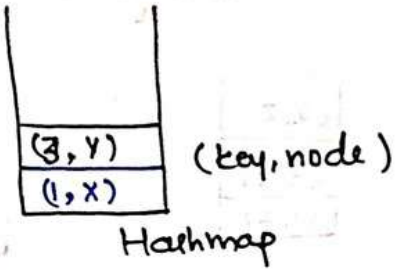
Size = 3 put(1, 10) put(3,15) put(2, 12) get(3) put(4,25)

Create a DLL and map.



put (1,10) Not present so put





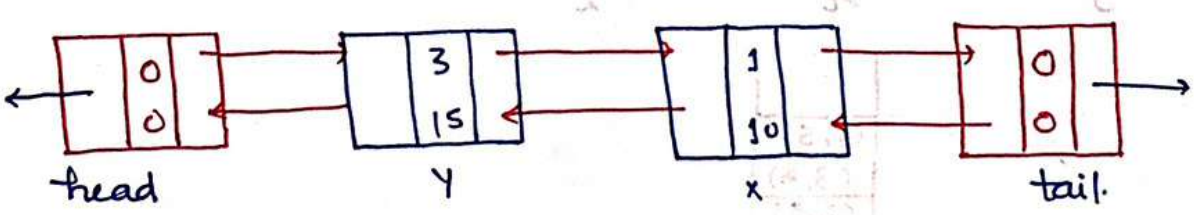
While inserting conside:

① check if {key, val} is already present

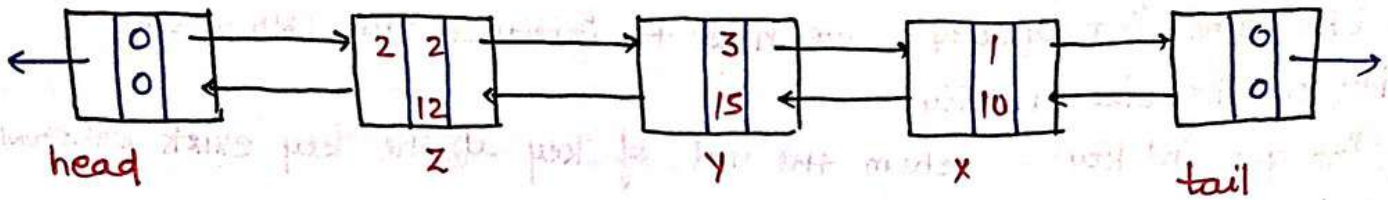
② Check for capacity if capacity == size so remove the pair ~~which is~~ and add just right after head node.

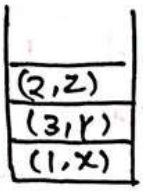
③ if key is not present return -1

put · (3, 15) Not present so put



put (2,12) Not present so put.

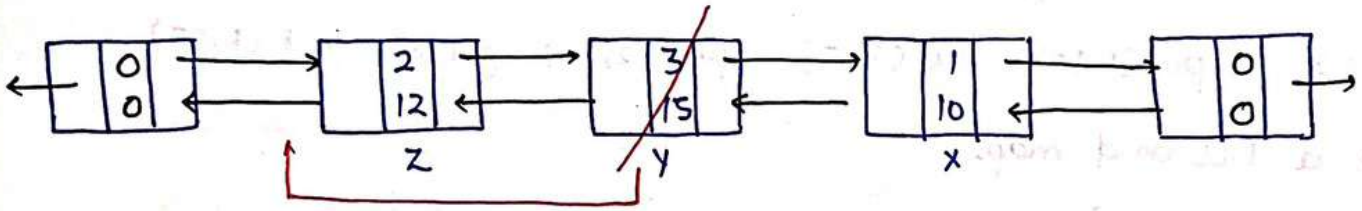


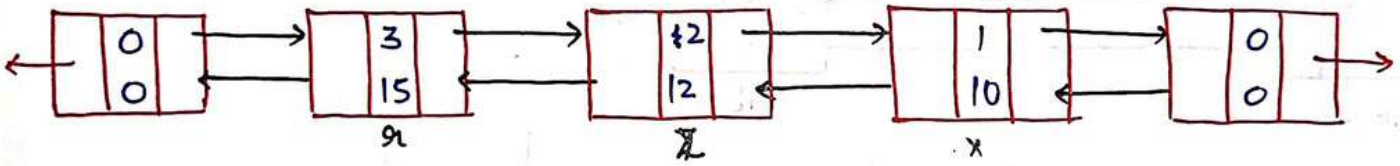


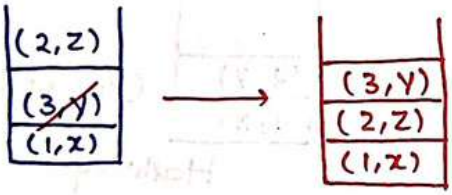
get(3), yes. so it is present.

Now most recent element should be change.

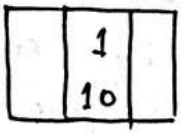
Delete the Node and add it right after head node.



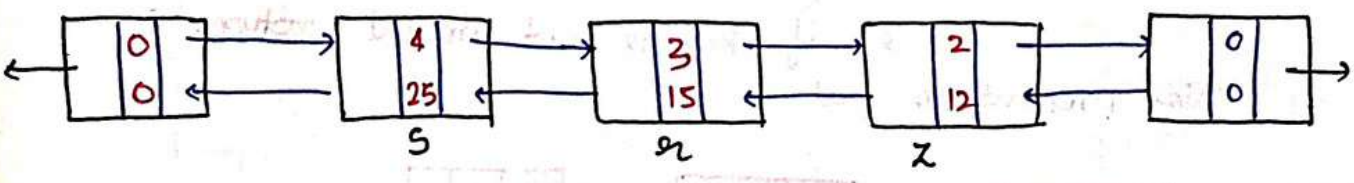


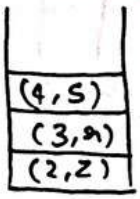


put (4,25) because capacity full. so we have to remove least recent used element.



will remove. and add (4,25)





```
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) {
                key = _key;
                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 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 cathe is full and remove (2,20)

1 → ~~(3,30)~~
2 → (1,10) (3,30)

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

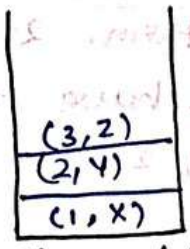
1 → ~~(4,4)~~
2 → ~~(4,4)~~ ~~(3,30)~~ (4,4)
3 → (3,30)

Map<freq, list> freqlist; → DLL

Map<key, node> keynode;

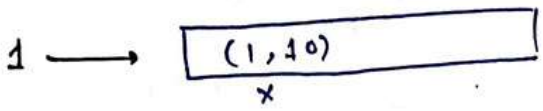
capacity = 0

freq = 0



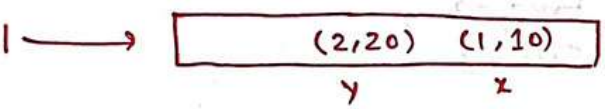
size = 3

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



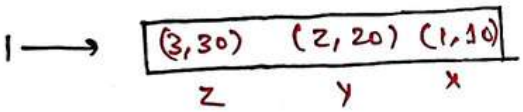
freq = 1
capacity = 1

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



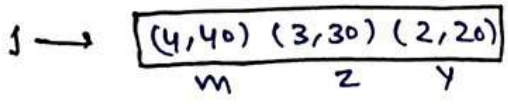
freq = 1
capacity = 2

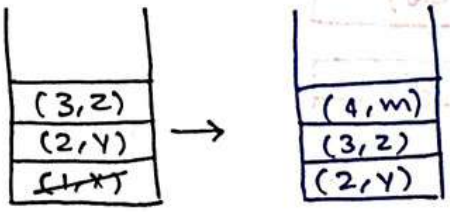
put (3,30). → verify is present in the map or not.



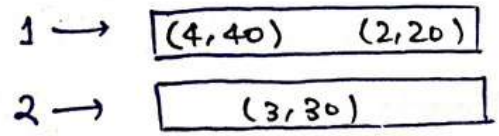
freq = 1
capacity = 3.

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)]

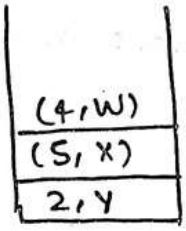
get(4) → it is exist. val = 40

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)]



freq = 1

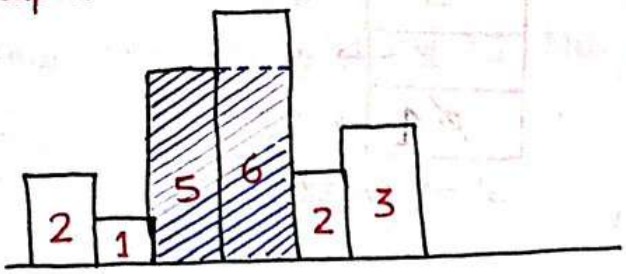
put(2,25) → 2 is exist so we update it and change freq.

1 → [(5,50)]
2 → [(4,40)]
3 → [(2,25)]

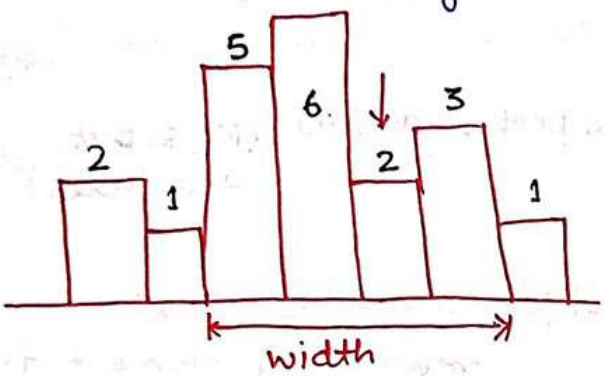
Largest Rectangle in a histogram - Problem - 04

Input => heights = [2, 1, 5, 6, 2, 3]

Output = 10



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



Brute force approach:

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

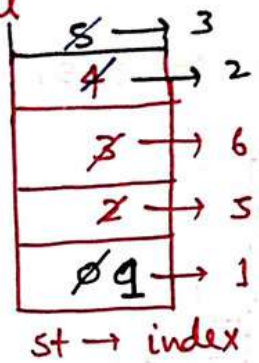
```
main (arr, n) {
    int maxArea = 0;
   for(i = 0; i<n; i++) {
    int minHeight = INT_MAX;
          for(j = i ; j<n ; j++) {
            minHeight = Math.min (minHeight, arr[j]);
            maxArea = Math.max (maxArea, minHeight*(j - i + 1));
                                    }
                                }
                            }
```

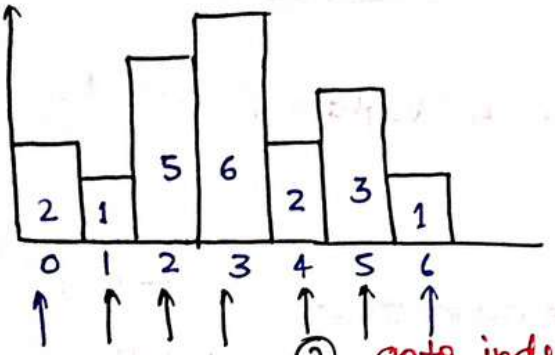
Time complexity - $O(N^2)$

Optimised Approach: The intution 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 & next smaller element.

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

Left small





① check st.peek > arr[0] (No, so put the index)

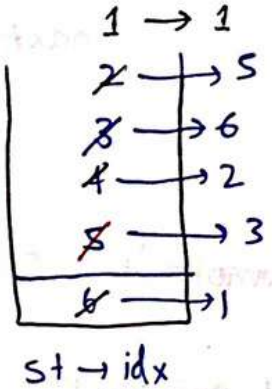
② goto index - 1 and st.peek > arr[1] (Yes, so remove the element ind and put the current idx.)

③ goto idx - 2 and
Now st.peek > arr[2] (No) and put the idx into stack.

④ goto indx → 4
st.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 idx - 6 st is empty put the curr idx in the array. and put 6 inst.

② On idx - 5
st.peek > arr[5], No
put the idx into stack and store the idx in the array.

③ No when we go to idx 4 val 2.
arr[st.peek] > arr[4] yes, so we have to remove the value from st. and put the 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.peek > arr[3] No so put it into st and arr

Now go to 2

st.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 -

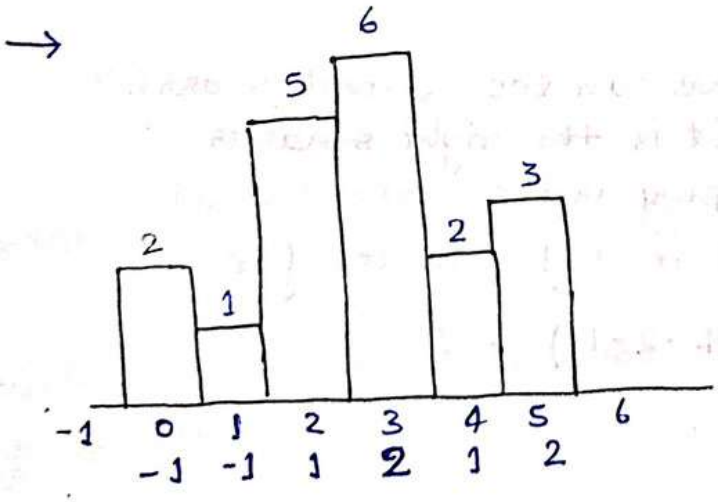
$$(\underset{\text{idx}}{\text{right smaller}} - \underset{\text{idx}}{\text{left smaller}}^{+1}) * a[i]$$

Tc - O(N) + O(N) ≈ O(N)

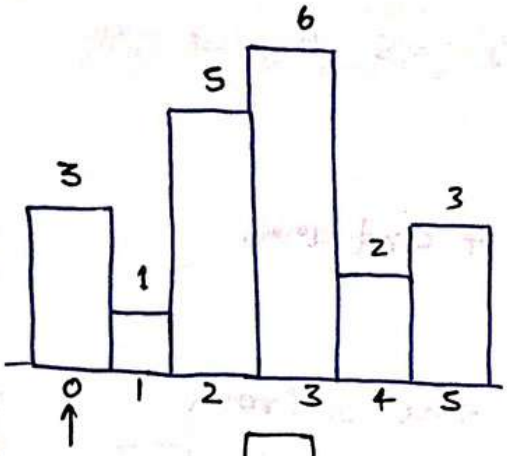
Sc - O(3N)

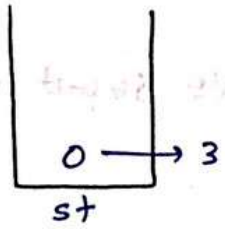
Optimsed solution - 2

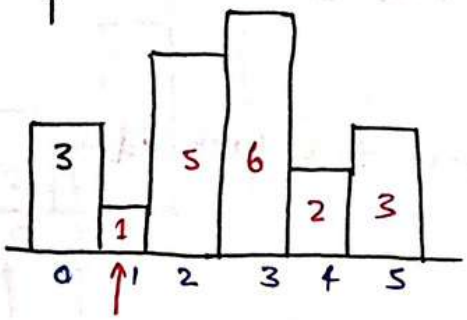
→ pop → bigger or same value
→ ans → push self indx.



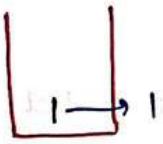
5 → 3
4 → 2
~~3~~ → 6
~~2~~ → 5
1 → 1
~~0~~ → 6
-1



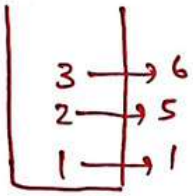


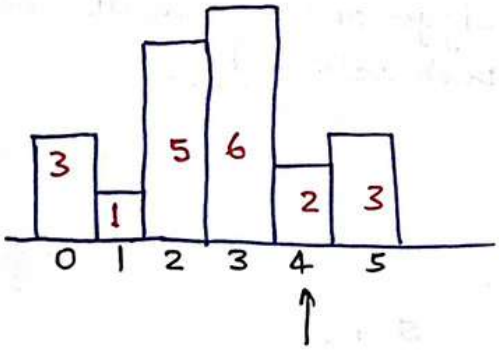


current val is greater than st.peek so pop it from st and put the curr idx.



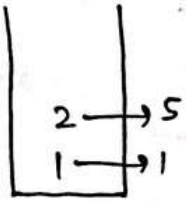
idx 2 and 3 val is greater than arr[st.peek] so simply put it into st





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.

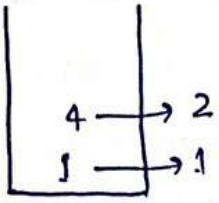
MaxArea = Max((MaxArea, (6 * (4 - 2 - 1)) = 6



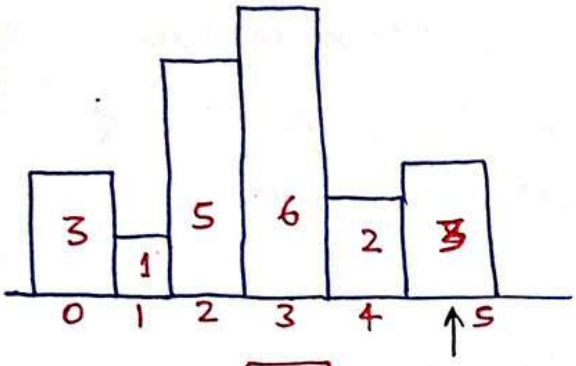
again st.peek > arr[i]

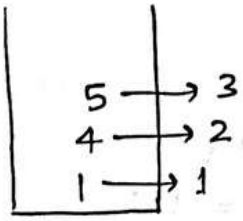
Since 2 is smaller than 5 so it means it means it is right smaller and popping out 5 from the stack, 1 is left smaller for 5

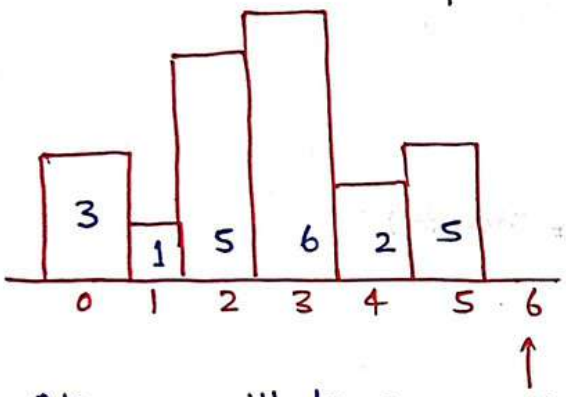
MaxArea = max (6, (5* (4-1-1)) = 10



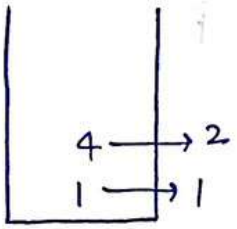
Now 1 is smaller than 2 so we directly push 2.







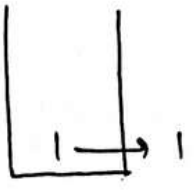
Now we will do one more iteration and consider 6 as a right smaller and so ppp out 3 and consider 2 as left smaller.



MaxArea = (10, (3* (6-4-1)) = 10

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

MaxArea = 2 * (6 - 1 - 1) = 8 , MaxArea = 10



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

MaxArea = (10, 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 * hight );
                    }

     st.push (i);
    }
return MaxA ;
  }
```

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

Space - O(N)

Sliding window Maximum : Problem - 05

We use the intution of NGE here.

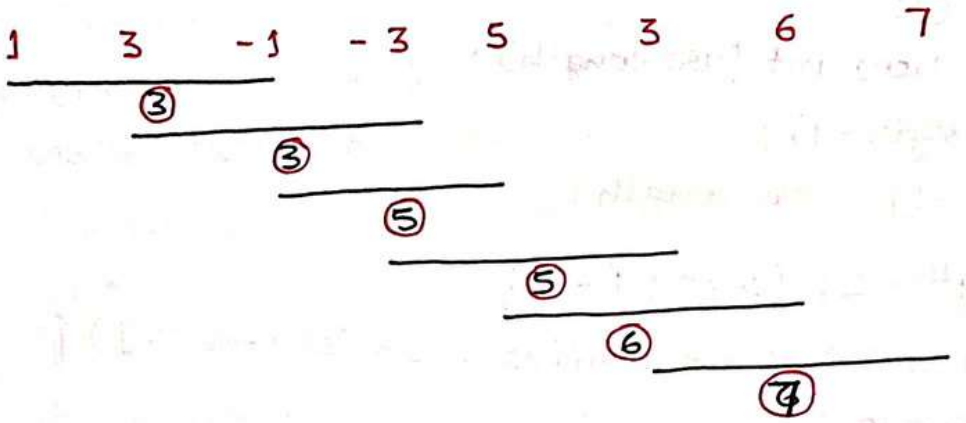
```
main(arr) {
    Stack<int> st();
    int[] nge[] = new int[arr.length];
  st.push(arr.length - 1);
  nge[arr.length-1] = arr.length;
 for(i = arr.length - 2; i>=0; i--){
        while(st.size()>0 && arr[i]>= arr[st.peek()]) {
                st.pop();
                    }
        if(st.isEmpty()){
                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]



Neive solution - traverse from 0 to five 5 (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.

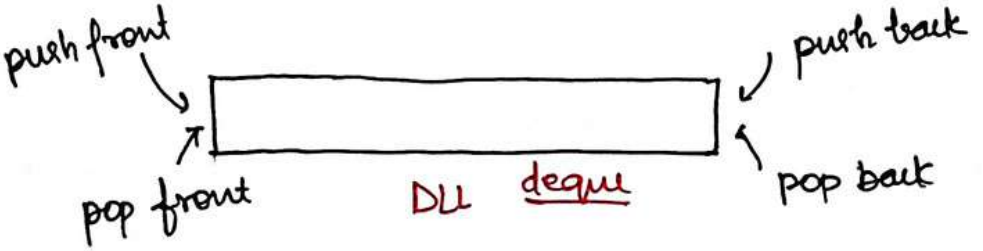
```
for (i = 0 → 5) {
        maxi = arr[i];
        for (j = i+k-1) {
                maxi = max (arr[j], maxi)
                }
        }
```

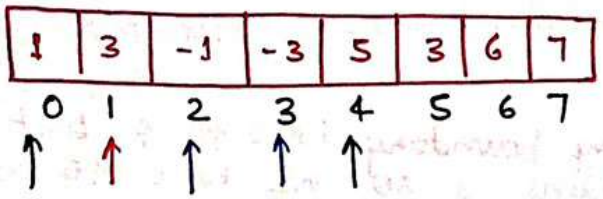
Time complexity - $O(N^2)$

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 dequeu.

It allows





In the 0th idx check does the deque has any thing → No. So push back the val 0

then go to idx 1.

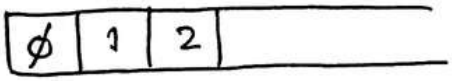
So now compare that peek element of dq. which is 1 is less than 3.

$1 < 3$ which can be my answer.

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

Move to the 2nd idx. with val -1

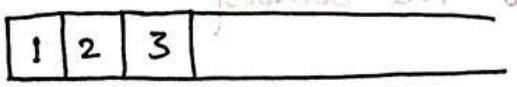
Now dq.peek > arr[i] so no need to pop because we are maintaining the decreasing order



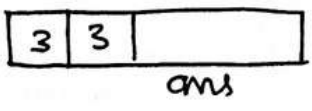
Now we can see 2 is the last index of the subarray of size K.

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

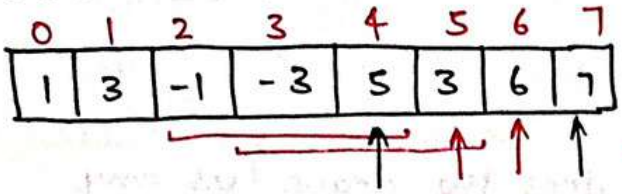
Now pointer will move on to 3rd idx and. if we check arr[dq.peek] > arr[i] so we can push into dq maintain the decreasing order.



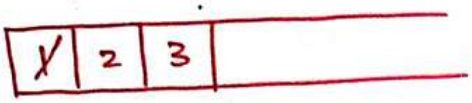
Now we have one more subarray of size 3.

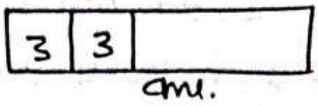


Now move to the 4th idx.

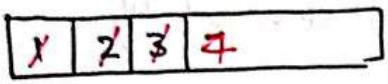


Now my boundary is 2 to 4 but dq. contains 3 so we have to pop 3.





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

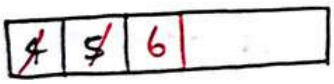


this 3 to 5 my boundary so check is there any idx in the dq out of the boundary. → No

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

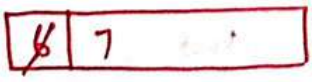
Now boundary 4-6 No element in the dq is out of boundary.

but. dq.peek < arr[i] pop both elements.



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

but. dq.peek < arr[i] → pop 6.



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

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

Sc - $O(K)$

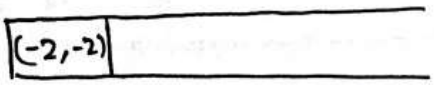
```
main (arr, k) {
  int ri = 0;
int ans[] = new int [n-k+1];
Deque<int> dq;
for (i = 0; i< a.length; i++) {
     if (!dq.isEmpty() && dq.peek == i-k) {
                  q.poll();                    // remove numbers out of range k.
                  }

// remove smaller numbers in k range as they are useless.
 while (!dq.isEmpty() && arr[dq.peekLast()] < arr[i])
              q.pollLast();

dq.offer(i);

 if (i >= k-1) {
        ans[ri++] = a[dq.peek()];
           }
                  }

return ans;
```

// 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.

Implement Min Stack :-

push(-2)
push(0)
push(-3)

getmin()
pop()
getMin()



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

(-2,-2), (0,-2), (-3,-3) we push all the elements.

Now getMin() = -3

if pop so st.

(-2,-2), (0,-2)

getMin() = -2

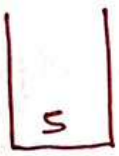
TC - O(1)
SC - O(2N)

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 ≥ minelement || 3 < minelement

↓ if Yes

current element ↓ if True

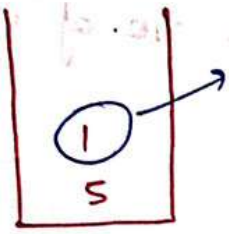
so directly push the curr val into the stack

so push 2x val - minele into stack and update the val.

3 < 5

so -

2 x 3 - 5 = 1 so we have to push 1 into the stack.



it is works like a flag (Analog)

minelement = ~~5~~ 3

Now if pop operation call so we have to check st.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.peek < minelement

⇓

minelement = 2 * minelement - st.peek

```
Stack<Long> st;
Long min;
public MinStack(){
    min = Long.MAX-VALUE;
        }
```

① void push(int val){
```
Long x = Long.valueof(val);
if (st.size==0) st.push(~~val~~x);
                min = x;
else if (val >= min) {
                st.push(val);

else {
    st.push(2 * x - min);
    min = x;
  }
        }
```

② void pop(){
```
if (st.size==0) return;
Long val = st.pop();
if (val < min){
    min = 2* min - val;
      }
  }
```

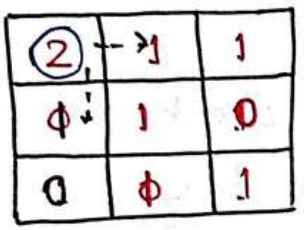
③ int top(){
```
Long val = st.peek();
if (val < min){
    return min.intValue();
  }
return val.intvalue;
  }
```

Rotting Oranges :- Problem - 07

given → m x n Matrix (m x n: size)

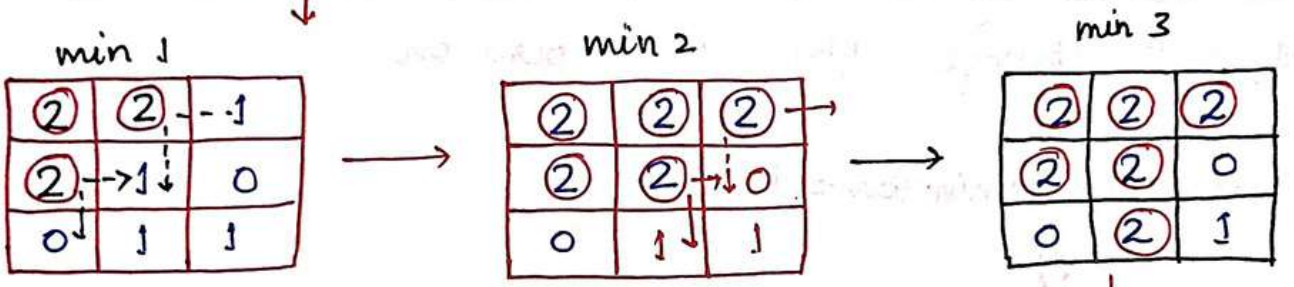
0 → represent empty cell
1 → fresh orange
2 → Rotten orange

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

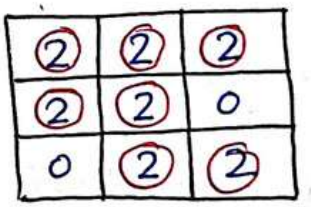


min 0

initially 2 is rotten orange



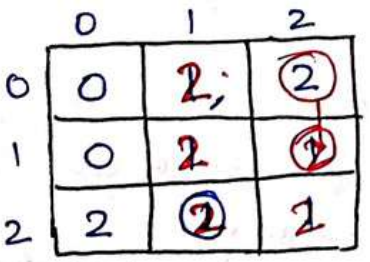
min 1 → min 2 → min 3



min 4

minTime = 4minutes.

let's take a example to think the intution -



Q

cnt = 2

x = 0 + 2 = 2	x = 2 + 0 = 2 → X
y = 1 + 0 = 1	y = 0 - 1 = -1

$x = 2 + 1 = 3$
$y = 0 + 0 = 0$] x

$x = 2 + (-1) = 1$
$y = 0 + (0) = 0$ because grid[1][0] == 0 so no change min = 1

$x = 0 + 0 = 0$
$y = 2 + 1 = 3$] x

$x = 0 + 0 = 0$
$y = 2 - 1 = 1$] ✓ push (0, 1) into q.

$x = 0 + 1 = 1$
$y = 2 + 0 = 2$] push (1, 2) into q.

$x = 0 - 1 = -1$
$y = 2 + 0 = 2$] x ∵ -1 < 0

min = 7

cnt = cnt + q.size
= 2 + 3 = 5

$x = 2 + 0 = 2$
$y = 1 + 1 = 2$] push (2, 2) into q.

$x = 2 + 0 = 2$
$y = 1 - 1 = 0$] x already vis.

$x = 2 + (1) = 3$
$y = 1 + (0) = 0$] x

$x = 2 - 1 = 1$
$y = 1 + 0 = 0$] x Zero

pop → 0, 1

$x = 0 + 0 = 0$
$y = 1 + 1 = 2$
] already present

$x = 0 + 0 = 0$
$y = 1 - 1 = 0$
] Empty cell.

$x = 0 + 1 = 1$
$y = 0 + 0 = 0$
] put it (1, 1) into q.

Similarly. (1, 2) → change pop and all the elements will pop so it will take (2 min) to rott all the oranges..

↓
ans

```
main(grid[][]){
    if(grid==null|| grid.len==0) return 0;
    int n = grid.len;
    int m = grid[0].len;
    Queue<int[]> q = new LinkedList<>();
    int cnt_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){
            cnt_fresh++;
                    }
                }
            }
```

```
if (fresh_cnt == 0) return 0;
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});
                    }
                }
        if (q.size() != 0) cntMin++;
                }
    return cnt_fresh == cnt ? cntMin : -1;
```

Time complexity - O(n*n)*4

Space complexity - O(n*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 these 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

so 5 - 1 = 4 that will be our ans.

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

Time complexity - O(N)
Space complexity - O(2N).

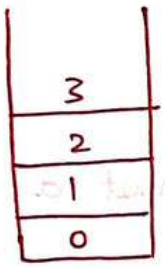
Celebrity Problem - Problem - 09

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	0✗	0✗	0✗	0	0✗
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.



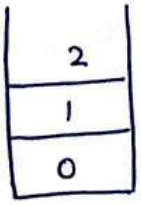
pop two elements.

	0	1	2	3
0	✗	✓	✓	✓
1	✓	✗	✓	✗
2	✗	✗	✗	✗
3	✓	✓	✓	✗

2 3

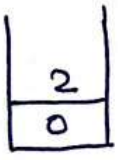
then check 2 is know by 3, No
and check 3 is know by 2, No

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.



Now pop 1 and 2

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



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 = "The sky is blue"
Output : "blue is sky The"

We can follow the Bruteforce 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 intialize str = "" again.

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

Time complexity - O(N)
Space complexity - O(N).

Optimal Approach

- Start traversing the string from the end (start to) untill 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

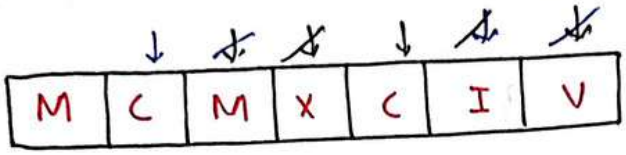
```
main fun(S){
    int left = 0, right = s.length()-1;
    String temp = "";  → for storing substring.
    String ans = "";   → for storing final string.
    while(left <= right){
        char ch = s.charAt(left);
        if(ch != ' '){
            temp += ch;
        }
        else if(ch == ' '){
            if(!ans.equals("")){
                ans = temp + " " + ans;
            }
            else{
                ans = temp;
            }
            temp = "";
        }
        left++;
    }

// for adding last word
    if(!temp.equals("")){
        if(!ans.equals(""))
            ans = temp + " " + ans;
        else ans = temp;
    }
    return ans;
```

TC - O(N)
SC - O(1)

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

res = V , so **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 substract.

res = res - val of (I)

res = 5 - 1 = 4

the go to C -

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

res = res + val of (C)
= 100 + 4
= 104

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

res = res - val of (X)
= 104 - 10
= 94

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

on M res = 1994

```
Map <Character, Int> ();
put I, V, X, L, C, D, M with its value manually.

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

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 not 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.charA(i) == ' ' || !Character.isDigit(s.charA(i))) break;
    ans = ans * 10 + s.charA(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(NlogN) + O(MN)

Space complexity - O(1)

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

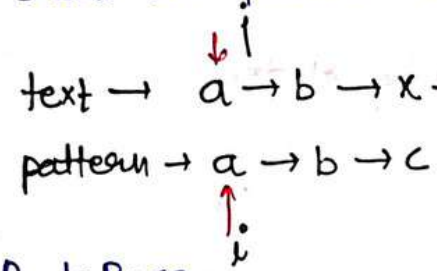
a = "abcd", b = "cdabcdab"

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 so -1.

Firstly look at Robin Karp:

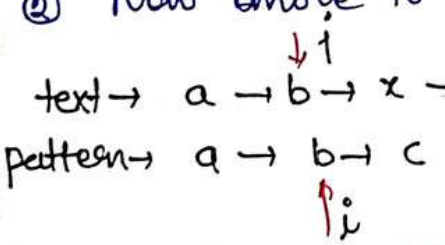
So we have a given text and pattern we have to check the pattern is a substring of text or not.



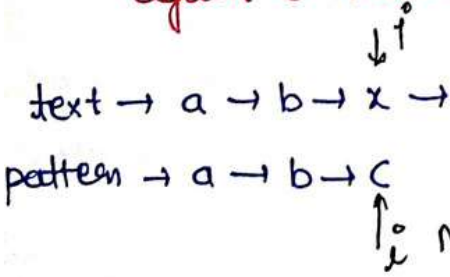
Brute force- Compare the text and pattern characters one by one.

① first both matched the character a and a.

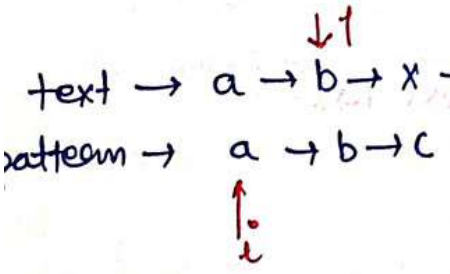
② Now move to pointer to next idx.



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



Not matched so rest the pointer i to 0 and set the pointer j to next of previous starting idx = 1



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

Test = a → b → x → a → b → c →

pattern = a → b → c

again Not match · move the idx j

test = a → b → x → a → b → c

pattern = a → b → c

① j and i value match so move to next idx

② j and i ——— ———

③ j and i ——— ———

at end i > pattern.length , so it is a substring of test.

Time complexity - O(m*n)

Space complexity - O(1).

Optimisation -

test = a b x a b c a b c a by

pattern = a b c a by

We wants 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. → rule

test = a b x a b c a b c a by

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 set to 0 and j will be increase.

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

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

Now j is on c and i is on y. → 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 idx whose value match with j. so put the i = i+1 and j = j+1

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

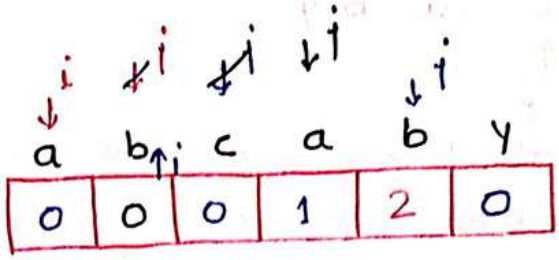
① matched i and j so move to next idx
② matched ——————
③ matched ——— ———————

Now j > 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



till 0 initially.

We create a array for setting the longest prefix substring.

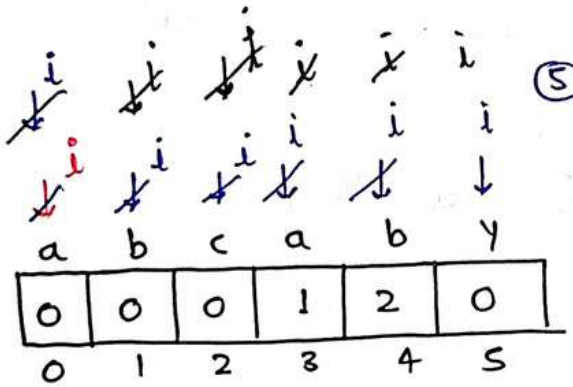
① i and j not match move j to next idx

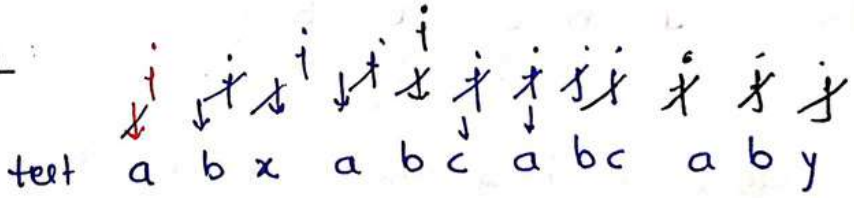
② i and j not match move j to next idx.

③ Value of i and j match so put the val of a this idx + 1
previous a = 0 + 1 = 1

④ value of i and j match. → so get the idx → which i & j and add 1 it to it and put

⑤ Now y not match to any one put 0.





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 prious location and look at the val. which having val = 0

so move to idx = 0

Now a compared with x. not equal so update the j = j+1 because i is on the first idx.
↓
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

Z-function | Find the idx of First occurance in a string

```
main fun(s1, s2){
    int n = s2.len;
    int lps[] = new int[n];
    i=0, j = 1;
    while(j<n){
        while( i>0 && s1.charA(j) != s2.charAt(j))
                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; // means matched..

    }

    return -1;
```

Move on to question :

```
main (s1, s2) {
StringBuilder sb ();
int rep = B.length() / A.length();
cnt = 1;
for (i=0 ; i< rep+2 ; i++) {
      if (sb.toString().contains(B2)
                  return cnt;

      else
          res.append(A);
          cnt++;
                                  }

return -1;
```

Tc - O(N) + O(M)] → 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 1's

n = 3 "21" two 1

n = 4 "1211"

n = 5 "111221"

Start with "1" build n-1 times.

```
main fun(int n){
 String ans = "1" ; "1"
 if(n == 1) return "1";
 if(n == 2) return "11";
for(i = 0; i<n-1; i++){
        ans = build(ans);
                }
 return ans;
            }
```

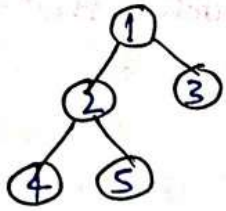
```
// function for ans. built build(ans){
 int freq = 1;
 char num = ans.charAt(0);
 String res = "";
```

```
for(i = 1; i<ans.length(); i++){
   if(curNum == ans.charA(i))
        freq++;
   else
    char freq = (char)('0'+ freq);
    res = res + freq;
    res = res + num;

    num = ans[i];
    freq = 1;
                                }
char freq = (char)('0'+ freq);
res = res + freq;
res = res + num;
```

Morris Inorder Traversal - Problem 104

Inorder - Left, root, right

I/p:



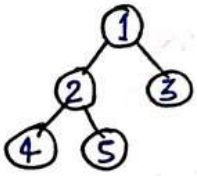
op:

4	2	5	1	3

```
Arraylist<int> lt;
Stack<inbde> 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



op:

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);
```

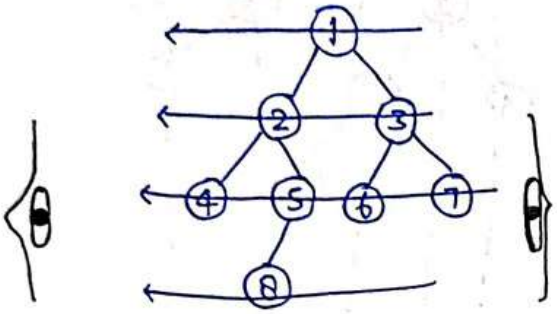
right

```
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)

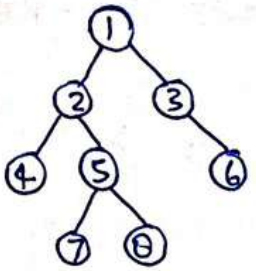
Intition: We have to do a recursive level order Traversal.

Root Right Left ⟶ for Right View

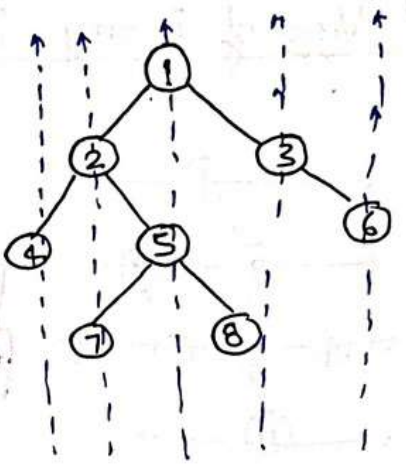
Root Left Right ⟶ for Left view

```
public class Solution {
    public List<int> rightSideView(TreeNode root) {
        List<Integer> res = new ArrayList<>();
    helpRightSideView(root, result, 0);
        return res;
                                                    }

fun helpRightSideView { TreeNode curr, List<int> res, int dep) {
                        if (curr == null) return;
        if (depth == res.size())
                    res.add(cur.val);

    helpRightView(cur.right, res, dep+1);      } swap for left view.
    helpRightView(cur.left, res, dep+1);
                                                }
```

Bottom View of Binary Tree



Bottom View - | 4 | 7 | 5 | 8 | 6 |

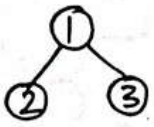



```
BottomView(root){
        ArrayList<int> res;
      if(root = = null) return res;
Map<int, int> treemap;
Queue<Node> qv;
root.hd = 0;
qv.add(root);
      while(!qv.isEmpty()){
            Node temp = qv.poll();
            int hd = qtemp.hd;
            map.put(hd, temp.data);    // for Top view check
                                          if(map.contains
                                          (hd)==false)
        if(temp.left != null)
                  temp.left.hd = hd-1;
                  qv.add(temp.left);

        if(temp.right != null)
                  temp.right.hd = hd+1;
                  qv.add(temp.right);
            }
for(Map.Entry<int, int> entry : map.entrySet()){
            res.add(entry.getValue());
                                    }
return res;
```

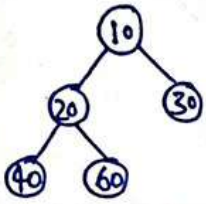
Time complexity - O(H)

Space " - O(N)

Root to leaf Paths



O/p: 1 2 # 1 3

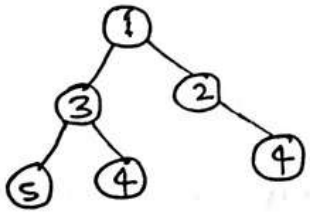


I/p: o/p: 10 20 40 # 10 20 60 # 10 30

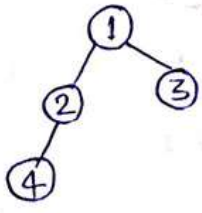
```
Arraylist < Arraylist <int>> ans;
Arraylist <int> lt;

main (Node root)
        help(root);
        return ans;

help (Node root) {
    if (root == null) return;
    lt.add(root.data);
    if (root.left == null && root.right == null)
            ans.add(new Arraylist<>(lt));
    help(root.left);
    help(root.right);
        }
```

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

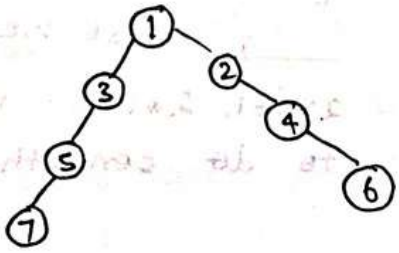
Maximum width of Binary Tree.



Max width = 4



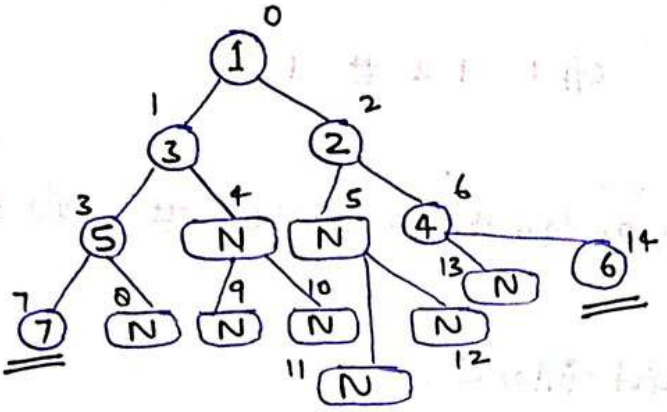
max width = 2



Max width = 8

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

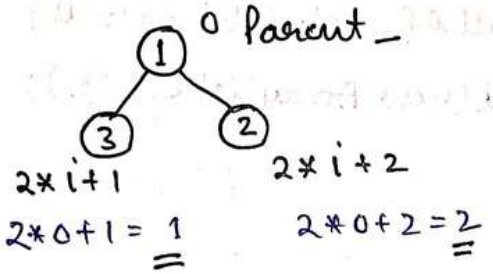
Intution:



Max width = 8

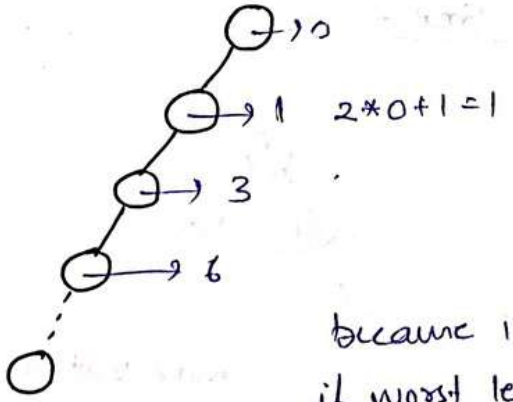
See the left most and right most node at every level. for achieve this we need unique id for every Node. Once we know the left most and right most node. width can be decide as (rightMost - left most + 1).

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



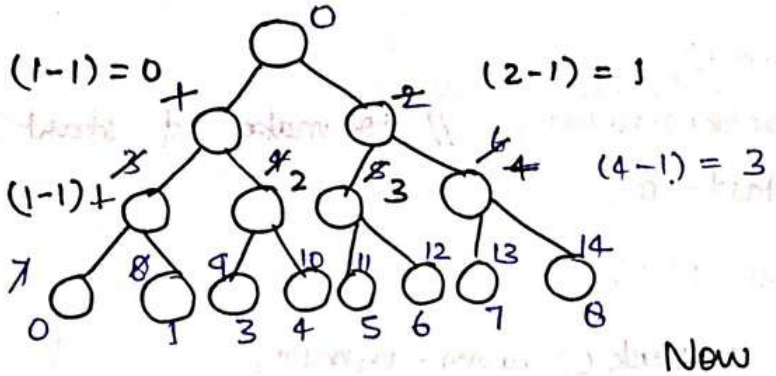
Similarly for other.

But this fails on tc.

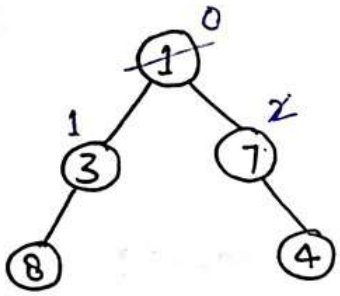


because increasing in twice if worst length 10s so it will overflow. so we can't use exactly $2*i+1$, $2*i+2$. instead we need to do something else.

How to prevent overflow



Now we can see that we handle the overflow condition.



(4, 4)
(8, 4)
~~(7, 2)~~
~~(3, 1)~~
~~(1, 0)~~

width = ~~2~~

(3, 1) → (1-1) = 0+1 = 1

(7, 2) → (2-1) = 1

(last - first + 1)

(2-1) + 1 = ~~2~~ 4

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

```
class Pair {
    TreeNode node;
    int num;
    Pair (TreeNode _node, int _num)
      {
       num = _num;
      node = _node;
      }
        }
```

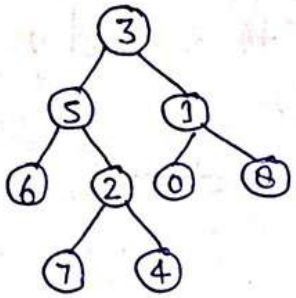
```
main (root) {
        if (root == null) return 0;
        int ans = 0;
        Queue < Pair > q = new LinkedList ();
        q.add (new Pair (root, 0));
```

```
while (!q.isEmpty()) {
        int size = q.size();
        int mmin = 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 - mmin;
            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 = max(ans, last - first + 1);
    }
return ans;
}
```

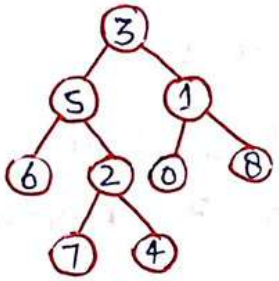
lowest common Ancestor of a Binary Tree

↓

the lowest parent of two Nodes.



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



p = 5, q = 4

lowest CA = 5

Intution - 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 → Left subtree

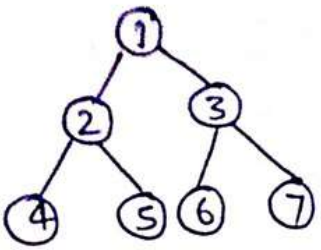
↓

if null, mean LCA not found on left subtree which means we found LCA on right subtree. so return Right.

☆ Make a recursive call → 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

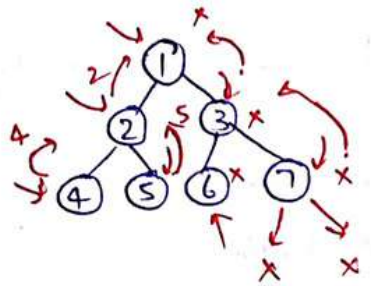
root != null & not equal to x and y
so recursion call.

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
&
Right recursive call " " "
↓
-thus 2 will return itself. to root i.e. to 1.



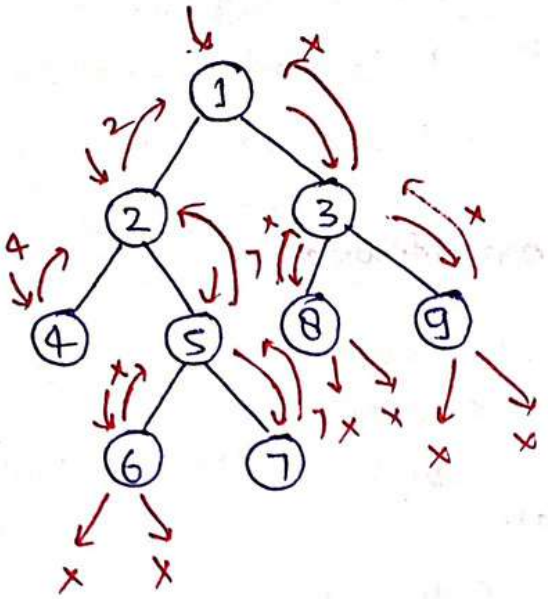
lca(4,5) = 2

Bruteforce - take the root to node path to x and y,

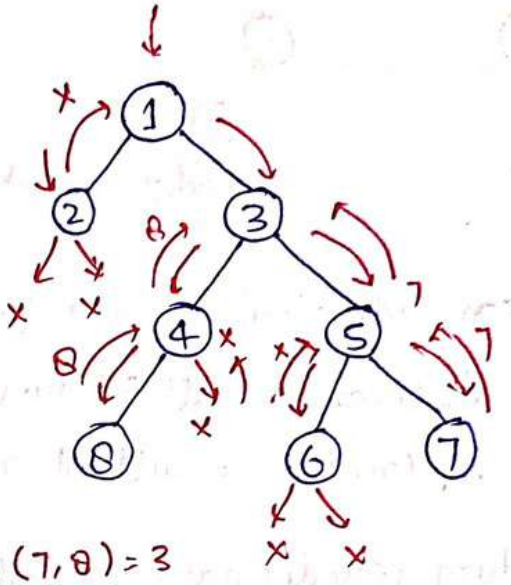
(1, 2, 4)
(1, 2, 5)

at zeroth idx match
at 1st idx match
not matched on 2nd so
last match was 2 so it is LCA.

O(N) → Space & Time



lca (4,7) = 2



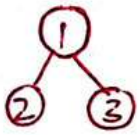
lca (7,8) = 3

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

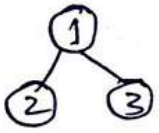
```
main LCA () {
    if (root == null || root == p || root == q) return root;
    TreeNode left = LCA (root.left, p, q);
    Tree Node right = LCA (root.right, p, q);

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

Check if two trees are identical -

1 =>



node 1

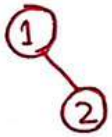


node 2

both are identical.

2 =>

node1

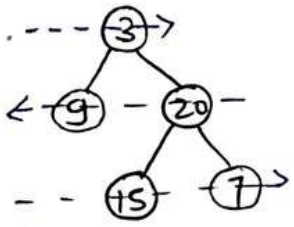


node2

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.

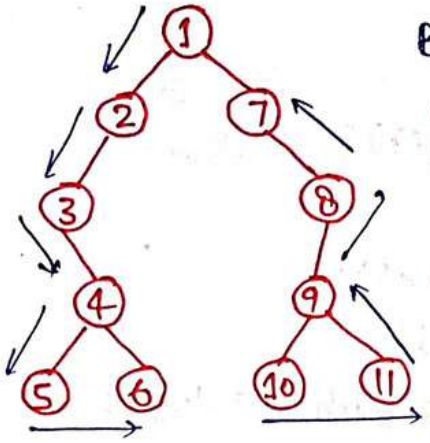


O/P: [[3], [20, 9], [15, 7]]

```
main levelordertraversal (root) {
                List< List <int>> lt;
    if (root == null)
        return lt;

    Stack < Node> st1;
    Stack < Node> st2;
    st1.push(root);
    TreeNode cure;
```

```
bodean leftfor = true;
while (!st.isEmpty() || !st2.isEmpty()) {
    List<int> level();
    if (ltor) {
        while (!st1.isEmpty()) {
            curr = st1.pop();
            ltor.add(curr.val);
level       if (cur.left != null)
                st2.push(cur.left);
            if (cur.right != null)
                st2.push(cur.right);
        }
    }
else {
    while (!st2.isEmpty()) {
        curr = st2.pop();
level   ltor = add(curr.val);
        if (curr.right != null) s1.push(cur.right);
        if (curr.left != null) s1.push(cur.left);
    }
}
    ans.add(ltor);
    ltor = !ltor;
return ans;
}
```

Boundary Traversal of Binary Tree.



Boundary traversal:

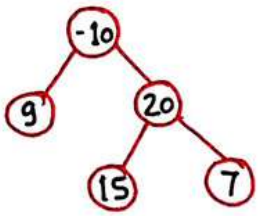
1	2	3	4	5	6	10	11	9	8	7

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.

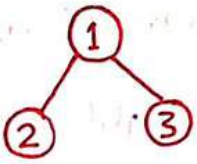
```
printBoundary(node) {
ArrayList<int> ans();
if (isLeaf(node) == false) ans.add(node.data);
addLeftBoundary(node, ans);
addLeves(node, ans);
addRightBoundary(node, ans);
return ans;                    }

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 = 42

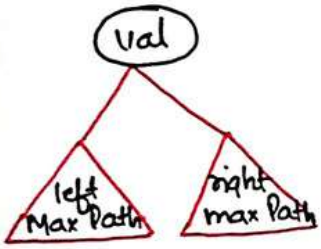


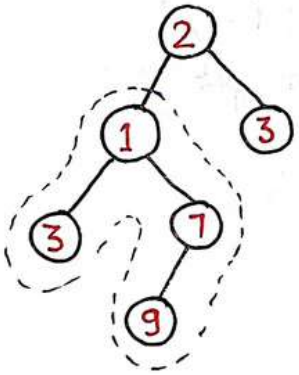
2+1+3 = 6

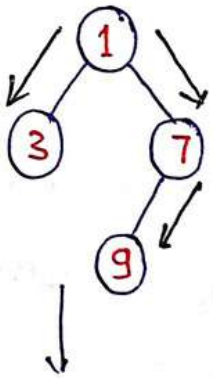
A bruteforce approach would be to generate all paths and compare them. Generating all paths will be a time costly activity therefore we need to lookup for some thing else

At a given node with a val, if we find the max left sum Path in the left subtree and right subtree,

maxPathSum = val + leftSumPath + rightSumPath







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

left path = 3

right path = 0 + 7 + 9 = 16

max Path Sum = val + left path + right path

= 3 + 1 + 16

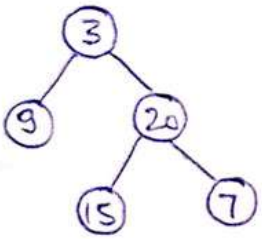
= 20

```
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;
}
```

backtrack logic.

↳ return to its parent

Construct a Binary Tree from inorder and pre order - Problem - 120



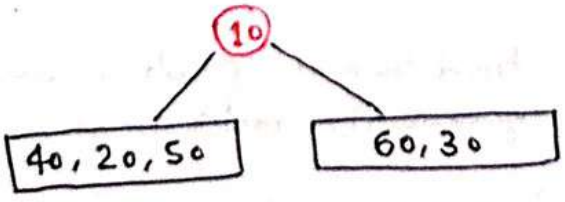
Input: [3, 9, 20, 15, 7]
output: [9, 3, 15, 20, 7]

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

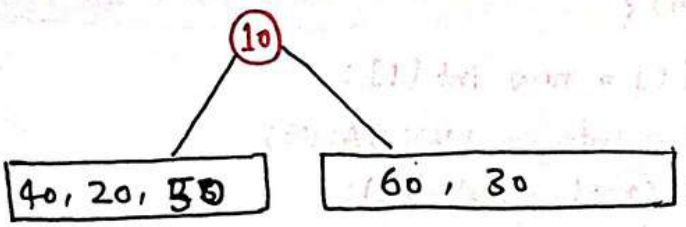
Input: inorder → [40, 20, 50, 10, 60, 30] left, root, right
preorder → [10, 20, 40, 50, 30, 60] root, left, right.

left ↓ right

root

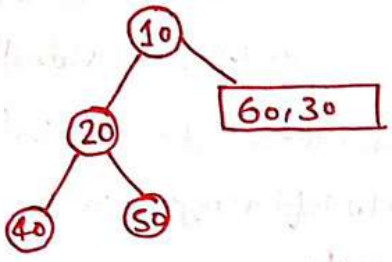


order can be anything we only consider the left and right part



Inorder → → Inorder

Preorder → 20, 40, 50 30, 60

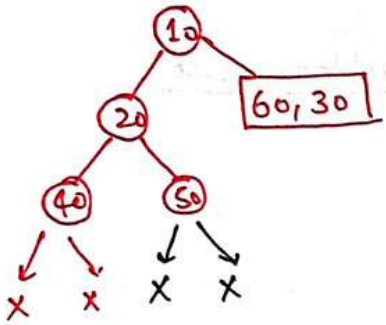


Inorder : 40 50

preorder 40 50

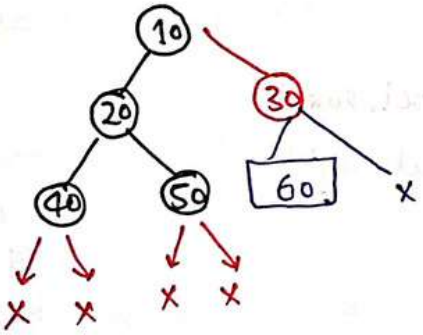
Inorder → 40, 20, 50 L, Root, R

preorder - 20, 40, 50 Root, L, R



Inorder - 40 , Inorder: 50

preoder - 40 , preorder: 50

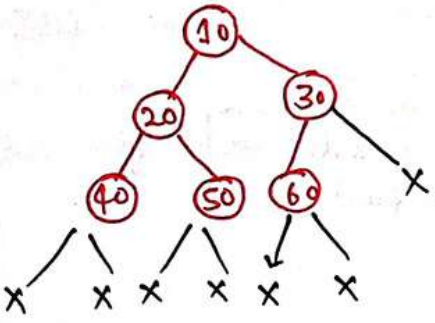


Inorder - 60, 30

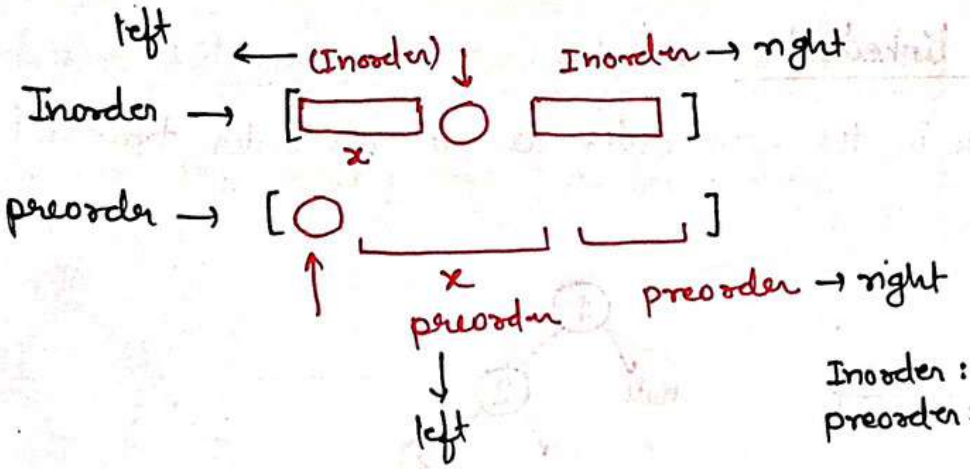
preorder - 30, 60

Inorder - 60

preorder - 60



breakdown the problem and generate the subtrees.



Inorder: 9, 3, 15, 20, 7
preorder: 3, 9, 20, 15, 7

9 → 0
3 → 1
15 → 2
20 → 3
7 → 4

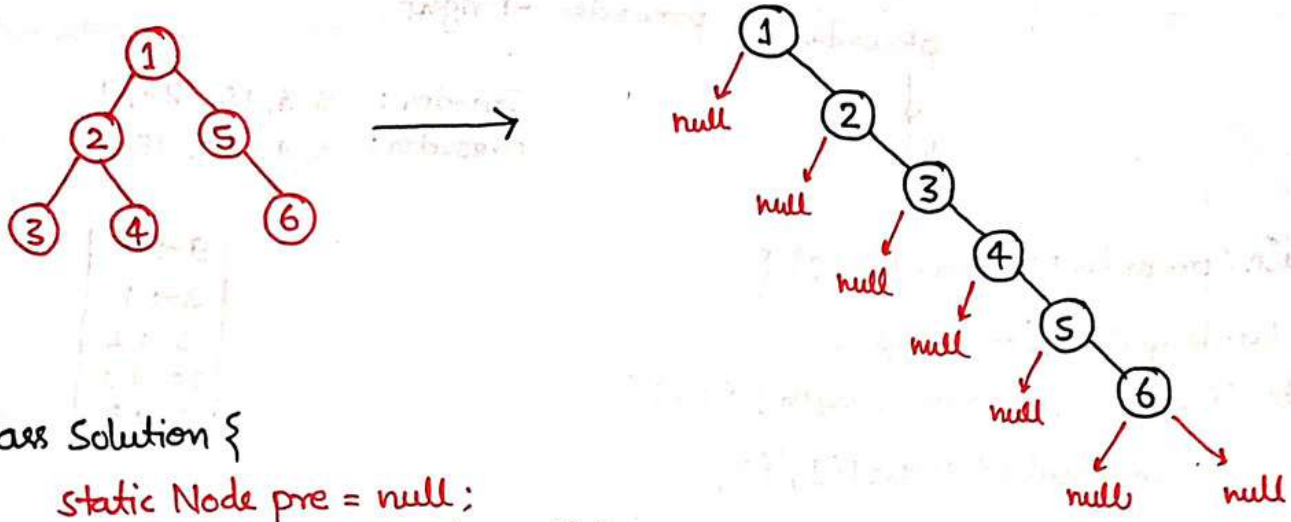
```
main (preorder[], inorder[]) {
  HashMap<int, int> map ();
  for (i = 0; i < inorder.length; i++) {
      map.put (inbrder[i], i);
                                    }
TreeNode root = buildTree(preorder, 0, preorder.length-1, inorder.0,
                                    inorder.length-1, map);
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;
                                    }
```

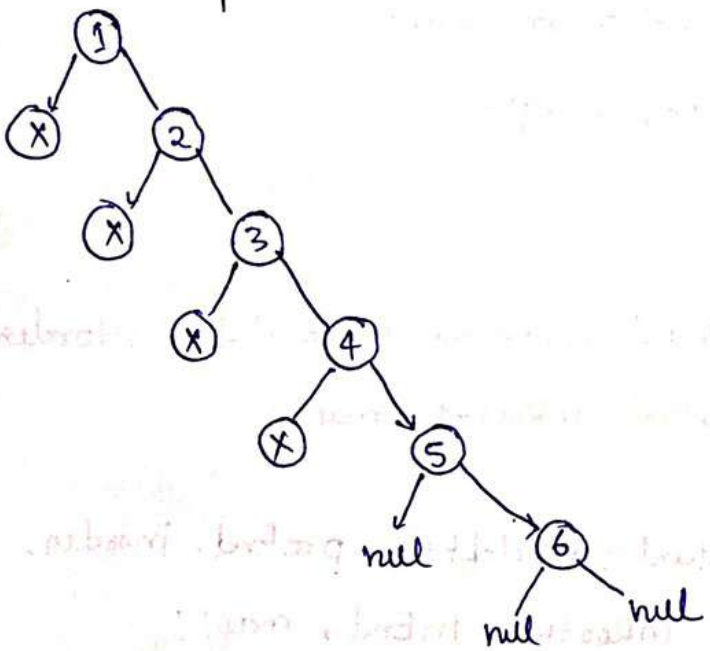
TC - O(N)
SC - O(N)

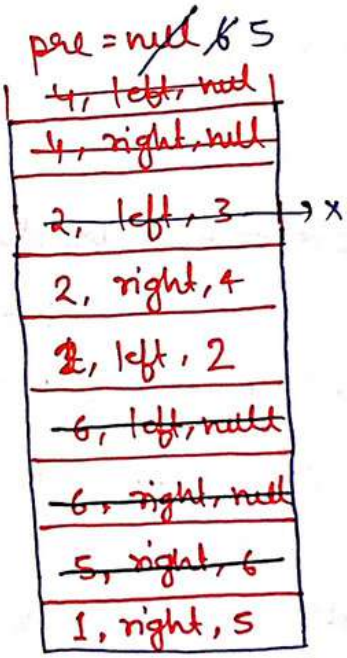
Flatten Binary Tree to linkedlist -

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




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

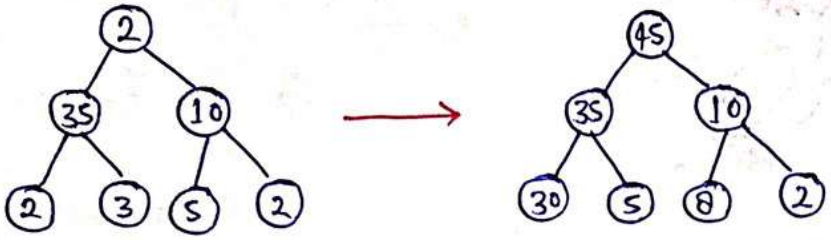




Similarly

Check for children sum property in a Binary Tree.

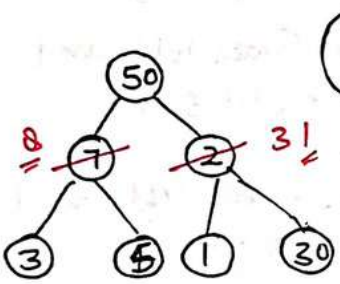
convert the binary Tree into sum tree -



node = left + right

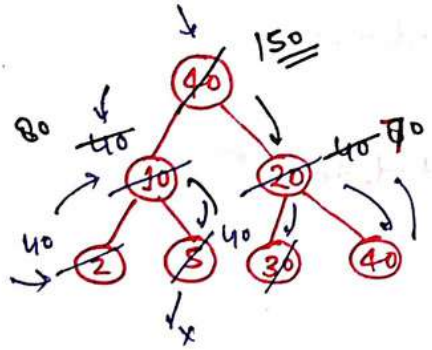
Note - 1. A val for null node can be accessed 0

2. Not allowed to change the structure.



Short II

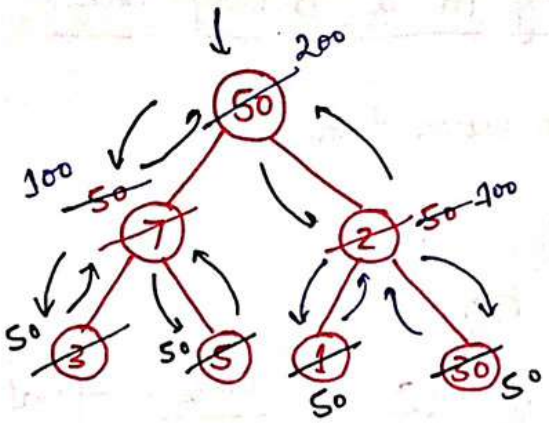
→ So we can see over here the sum of left and right 8+31 = 91, and the root is 50, so we can't minimize the root val., thats the problem this is not a simple problem.



10+20 = 30 < 40

2+5 = 7 < 40

30+40 = 70 > 40 (make root = left+right)




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

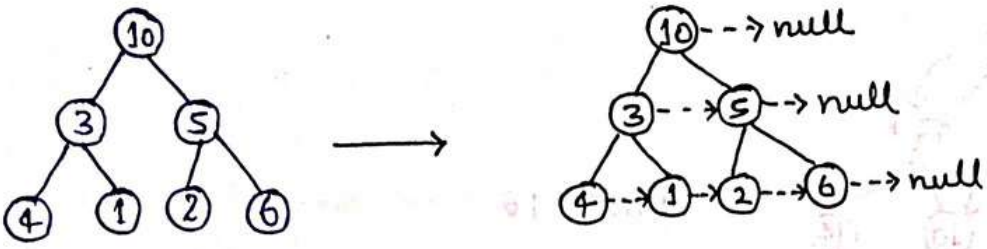
3+5 = 8 < 50 [make left and right 50]

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

TC- O(N)

SC- O(H)

Connect nodes at the same level.

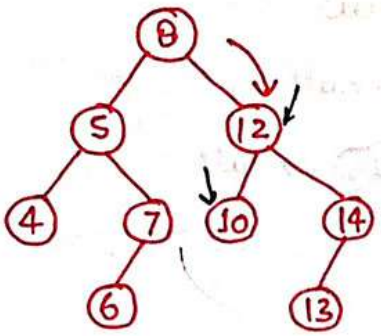



```
connect(root){

    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
                }
```

Timecomplexity - O(N)

Search In a Binary Tree -

BST → $(\log_2 N)$



node = 10

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 ✓

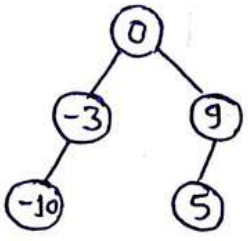
if reach null return null:

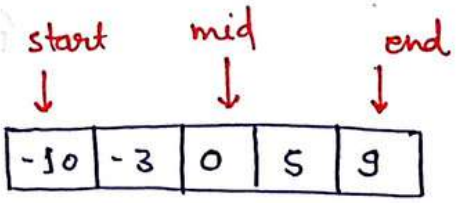
```
while (root != null && root.val != val) {
    root = val < root.val ? root.left : root.right;
}
return root;
```

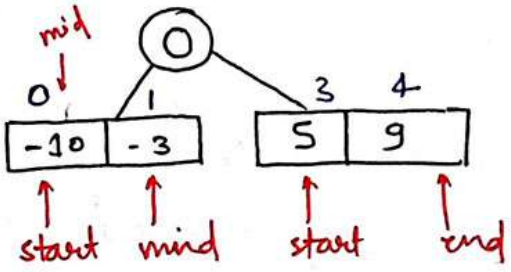
Convert sorted Array to Binary Search Tree -

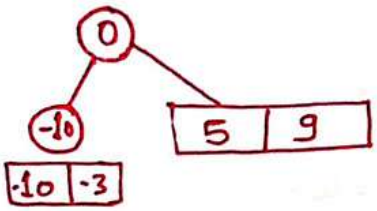
arr = [-10, -3, 0, 5, 9]

clearly show the root will be the mid element.

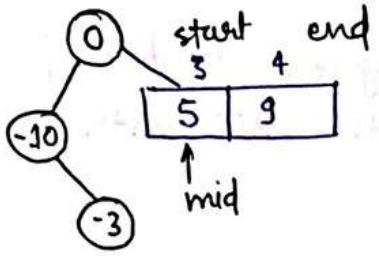


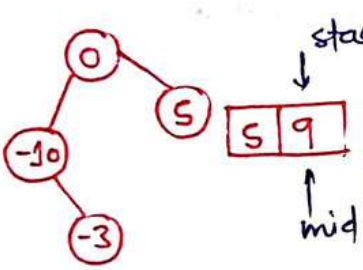




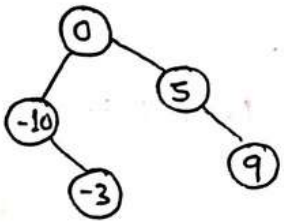


there is nothing on the left of -10 make a recursive call on right side.





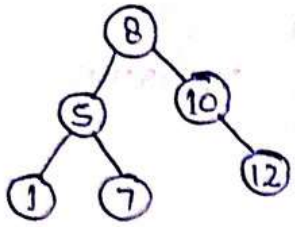
nothing on the left of 5.
So recursive call at right side.




```
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 |

} so we can creat the BST by this.

TC - O(NlogN) + O(N)
SC - O(N)

Optimal solution -

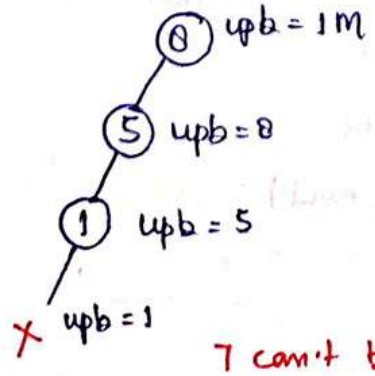
take the upperbound. initially as INT.

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

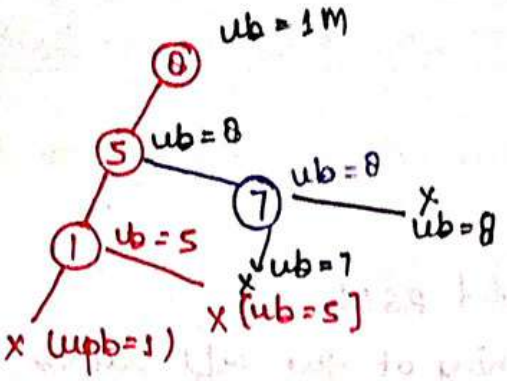
(8) upb = INT

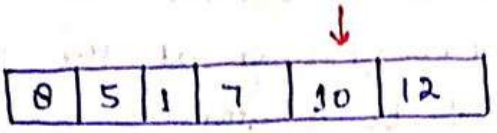
we dont know how many element that will lie at the left side of 8.

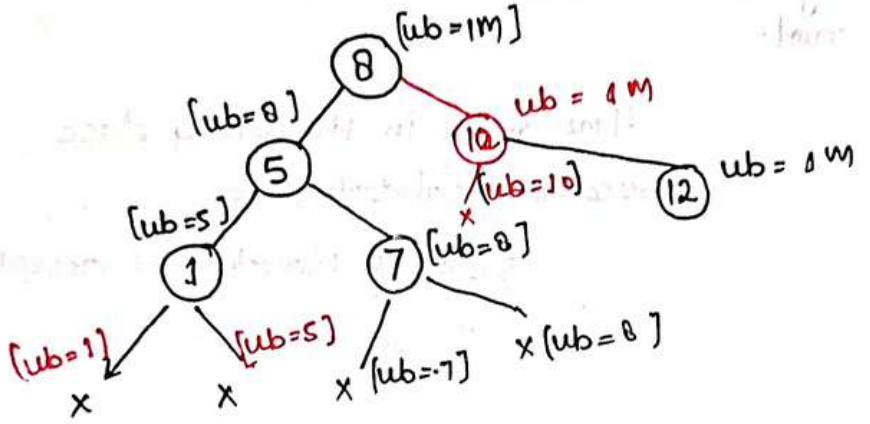
We know one thing for sure. everything at the left will always smaller than 8.



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







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).}

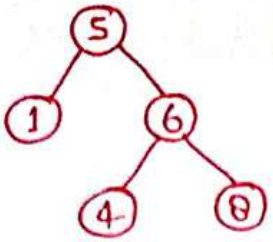
Tc - we are traversing a singal node 3 times.

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 nu
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.



The valid BST:

* every thing at the left has to be smaller than that node
* and every thing at the right has to be greater.

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

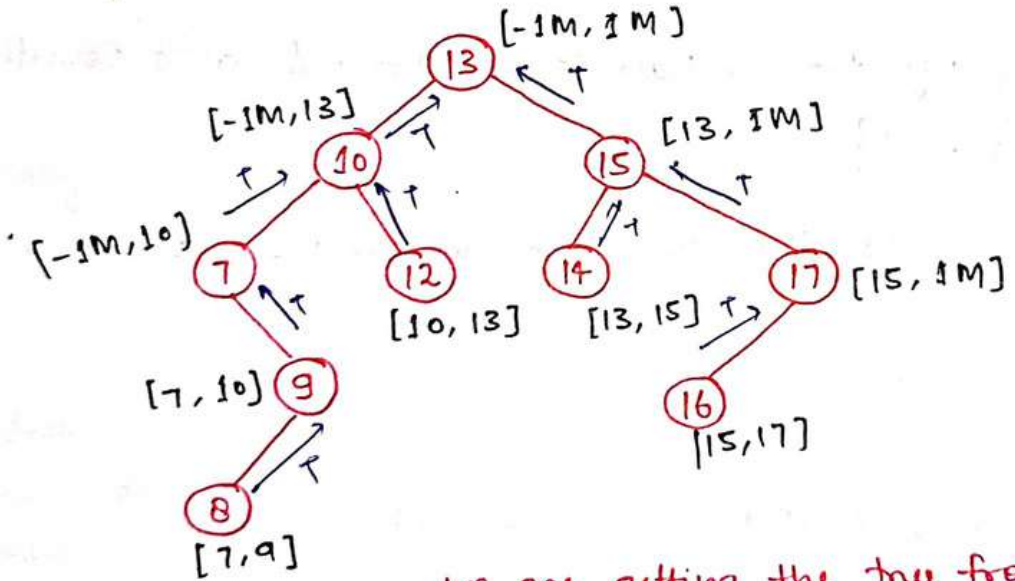
Range - [L, H]

Here 4 is in the wrong place because. validation is.

$5 \leq 4 \leq 6$ (which not accepted).

Range [5 6]

initial range [-1M, 1M]



we are getting the tree from both the subtrees. so its a valid BST.

Tc- O(N)

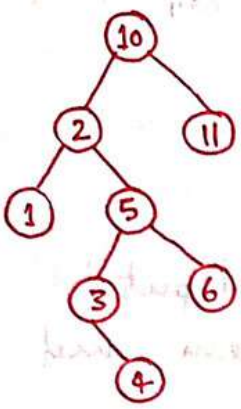
Sc- O(1)

```
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);
}
```

Predecesser and Successor:

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

if not found set it to to null.



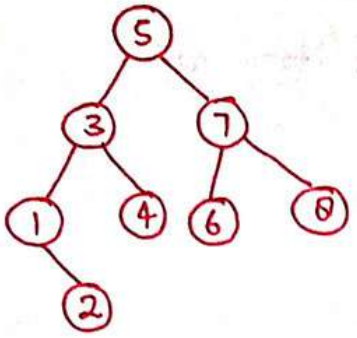
predecesor successor

key = 8 , ouput = [6, 10]

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

```
main (root) {
    predecesor (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;
            }
                }
```

Kth smallest element in a BST- / largest element



K=3, we have to find the 3rd smallest solution.

Neive solution - take an Arraylist and do the traversal put all the element of tree into the list and sort them and in the last get the Kth element.

Tc - O(N) + O(NlogN)

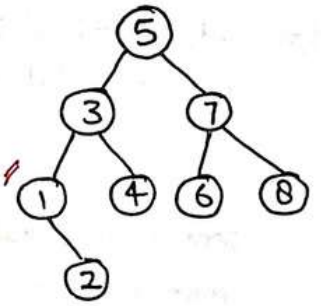
Sc - O(N)

Efficient solution -

Inorder - Left, node, Right.

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

inorder traversal



If we use the inorder traversal so it given all the element in a sorted order

by this we remove O(NlogN) 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
```

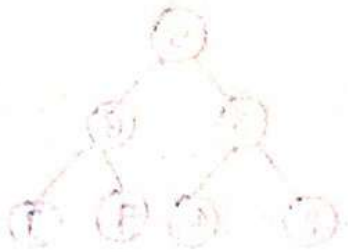
Tc - O(N)] Recursion

Sc - O(N)]

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

One traversal - O(N)

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



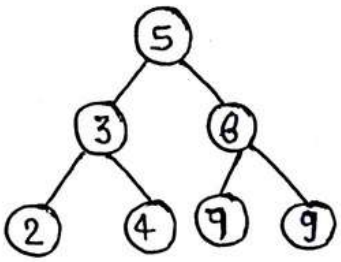
```
kth smallest(root, int k) {
    return help(root, new int[]{k});
}

help(root, int k[]) {
    if(root == null) return null;
Node left = help(root.left, k);
if (left != null)
        return left;

k[0]--;
if (k[0] == 0) return root;
return help(root.right, k);
}
```

Logic - 1

```
private int cnt = 0;
private int res = 0;

kth smallest (root, k){
      traverse (root, k);
      return result;
}

traverse (root, k){
      if (root == null) return null;
      traverse (root.left, k);
      cnt++;
      if (cnt == k)
            {res = root.data;
             return;
            }
      traverse (root.right, k);
}
```

Logic - 2

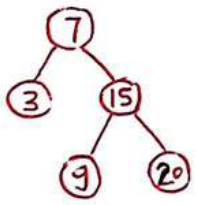
Pair with given Sum in BST.



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) || root.right, k);
                                      ^help
            }
```

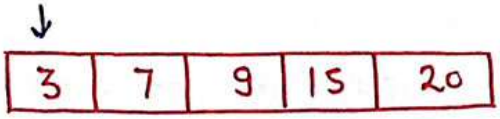
Binary Search Iterator-



So we have to disign 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.

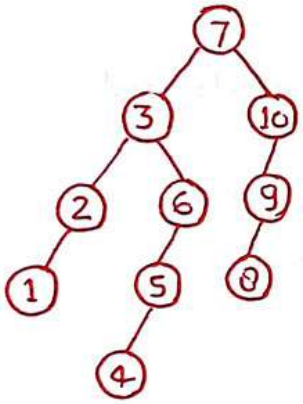


next → 3
hasnext → yes

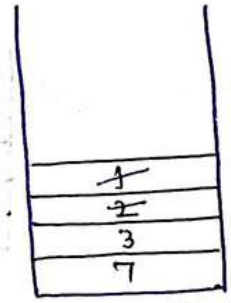
it will be easy.

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 constructor we will go in extreme left. and put everything into stack

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

next - 1.

when again next called , does 2 have a right.

next - 2

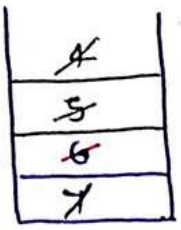
next → 3

check 3 has a right ?

Yes it has

Go to right,

Now put everthing into stack.



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

Now 4 left will be null.

Now if some say. hasNext ? Just check if stack is not empty, then return true;

next → 4

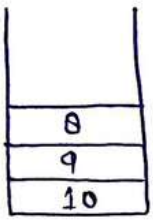
4 doesn't have a right

next → 5 5 " " "

next → 6 6 " " "

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

Similar things will happen -



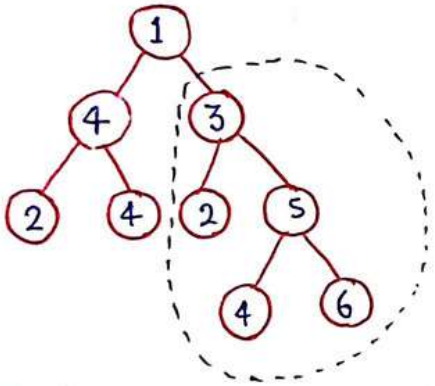
SC - O(H)

TC - O(1)

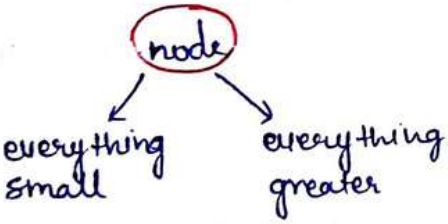
```
class BSTIterator {

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

    public BSTIterator (root) {

            pushAll (root);

                    }

    private void pushAll (root) {

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

                    }
```

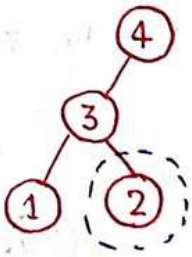
```
public boolean hasNext () {
        return !st.isEmpty ();
                }
public int next () {
        TreeNode node = st.pop();
        pushAll (node.right);
        return node.val;
                }
}
```

Maximum Sum in BST in a Binary Tree-



this is a BST with a max sum.





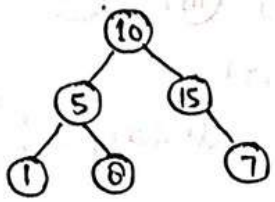
→ BST with max Sum.

Bruteforce - I will validate the BST.

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

Jo → validate BST.

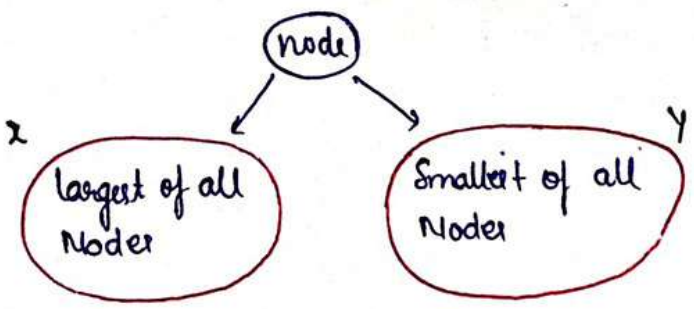
S ⇄ yes.



validate BST → O(N) x O(N) = $O(N^2)$

↓

going to every node

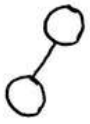


Size = 1+x+y

largut < node < smallest

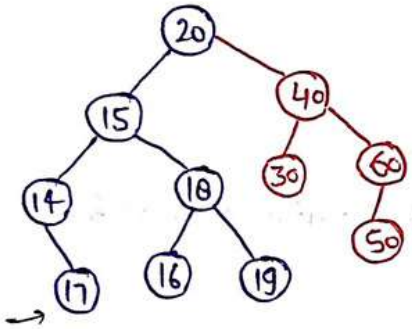
we will go to bottom and we know the bottom guy will be a BST → for sure. and it's connected to a node.

Idea - Build the things from bottom.



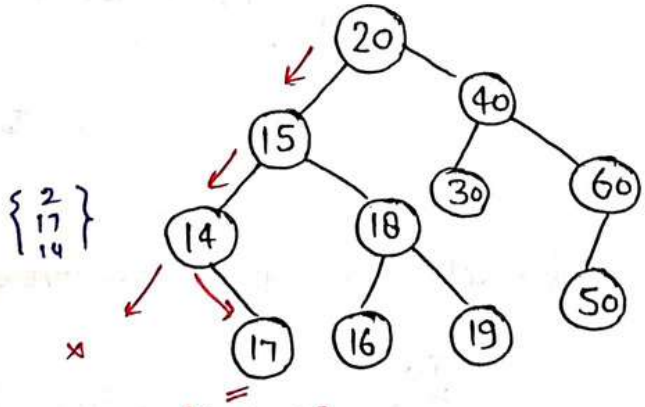
if satisfied the condition the BST

store the largut, smallest and size.



we have to traversal L, R, N so we take all element, L and R

postorder - L, R, N



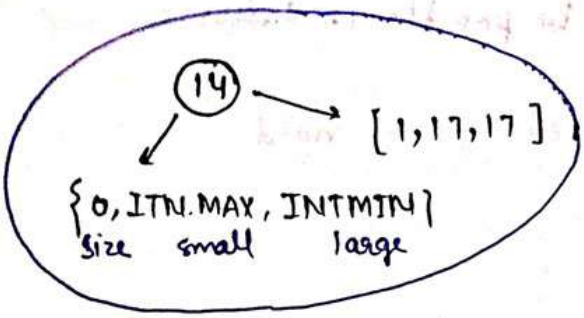
= → BST

{2, 17, 14}

[1, 17, 17]

[size, small, largest]

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.

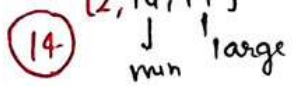


→ so when will be this will be a valid BST.

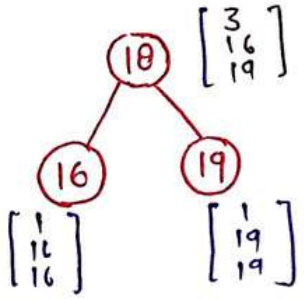
for it to be a valid BST.

INTMIN < 14 < 17

for 14 → size will be 1+x+y ⇒ 1+0+1 ⇒ 2



Now for 15 the left is completed.



so if come to 10 so check.

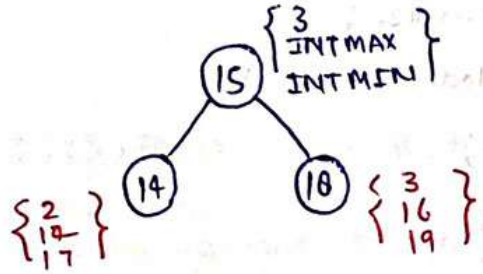
node, which is 10. is greater than largest of left subtree and smaller than the smallest of right subtree

16 < 10 < 19 → yes, so 10, will contain BST.

Size for 10 → 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



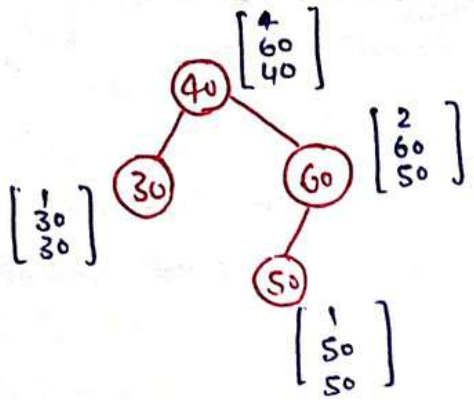
17 < 15 (x not following)

right is OK but left not

Not in that condition pass the INTMIN and MAX

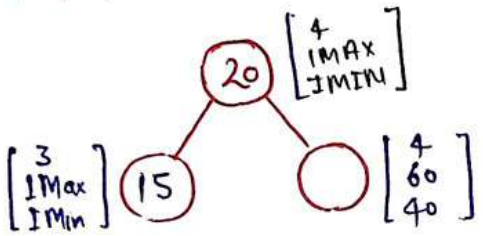
we pass MIN and MAX so comparision wont be possible in future

Now everything in left of 20 completed go to right.



30 < 40 < 50 (true)

At moment we came 20.



INTMAX < 20 < 40
x ✓

But largut BST

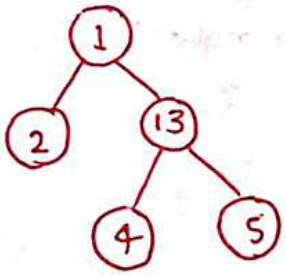
if BST. size = 1+x+y otherwise = max(left, right)

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 -



root → Serialize → root → String

String → Deserialize → root

reverse logic

1, 2, 13, #, #, 4, 5, #, #, #, #

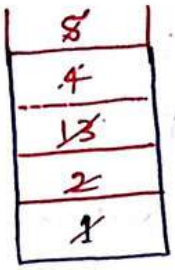
Make sure there have a null on leaf node

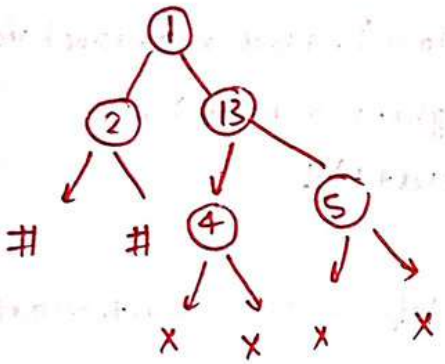
Now we have serialize -

the string we get · 1, 2, 13, #, #, 4, 5, #, #, #, #

Now for deserialize -

take queue ds.



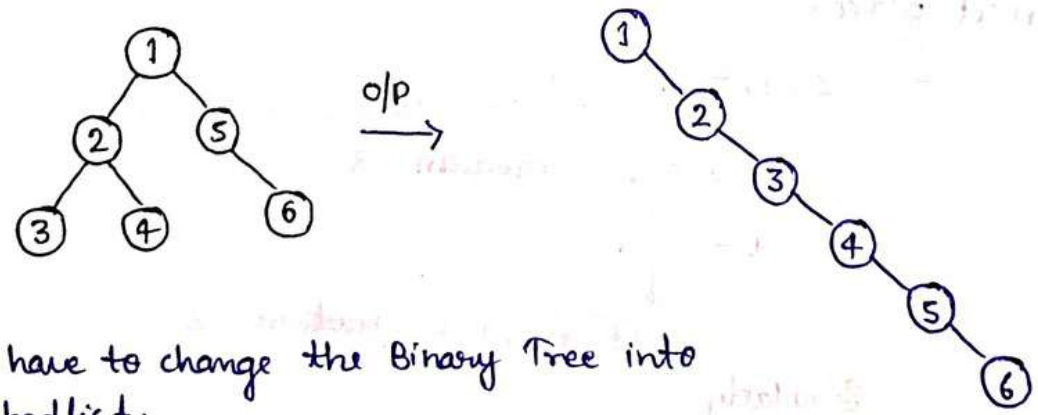



```
public String serialize(root) {
if(root == null) return "";
Queue<TreeNode> q = new LinkedList<>();
StringBuilder res = new StringBuilder();
q.add(root);
while(!q.isEmpty){
    TreeNode node = q.poll();
    if(node == null){
        res.append("n ");
        continue;
            }
res.append(node.val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



we have to change the Binary Tree into linkedlist.

The thing we have to observe - If we see the output at the left so 6 is at the bottom.

The conditions -

- 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 (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 mean 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 -

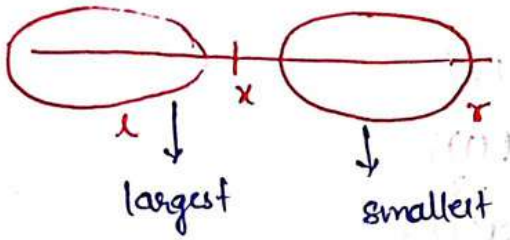
Tc - O(N) and there are N Number so it will take - $O(N^2)$

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

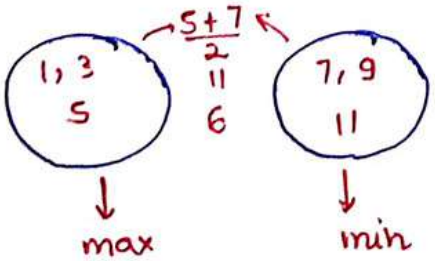
Similarly

for min Heap.

if number is odd to $\frac{n}{2}+1$ 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 {
    PriorityQueue<int> maxHeap = new PriorityQueue<>(Collections.reverseOrder());
    PriorityQueue<int> minHeap = new PriorityQueue<>();

    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());
        }
        elseif (maxHeap.size() < minHeap.size()) {
            maxHeap.add(minHeap.poll());
        }
    }

    double FindMedian() {
        if (maxHeap.size() = minHeap.size()) {
            return maxHeap.peek/2.0 + minHeap.peek()/2.0;
        }
        return maxHeap.peek();
    }
```