Stacks

Q what is stack?

→ Stack is a data structure which follow last in first out rule, it store memory in homogenous form.

Stack Question

1) Nearest greater element to left
2) Nearest greater element to right
3) Nearest smaller element to left.
4) Nearest smaller element to right
5) stack span problem
6) Maximum area of Histogram
7) Max area of rectangle in binary search
8) Rain trapping
9) implementing a min stack
10) implementing stack using heap
11) The celebrity problem
12) Longest valid Paranthesis
13) interactive Toh (# Tower of Hanoi)

How to identify stack question?

I) Array/Heap → Most probability stack lgega

II) Brute force → n^2

Note

if there are two loops present in the array and the second loop is dependent on i so in that condition we can use stack

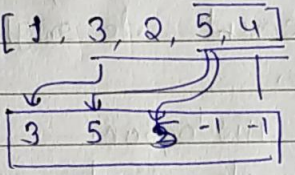
j → 0 to i
j → i to 0
j → i to n
j → n to i

→ in this senerio we can use stack.

Question 1

Nearest greater element to right → || Next largest elem

input → [1, 3, 2, 5, 4]



input → 1 3 0 0 1 4

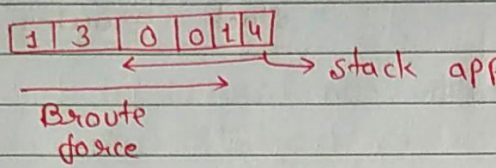
3 4 1 1 4 -1 → op

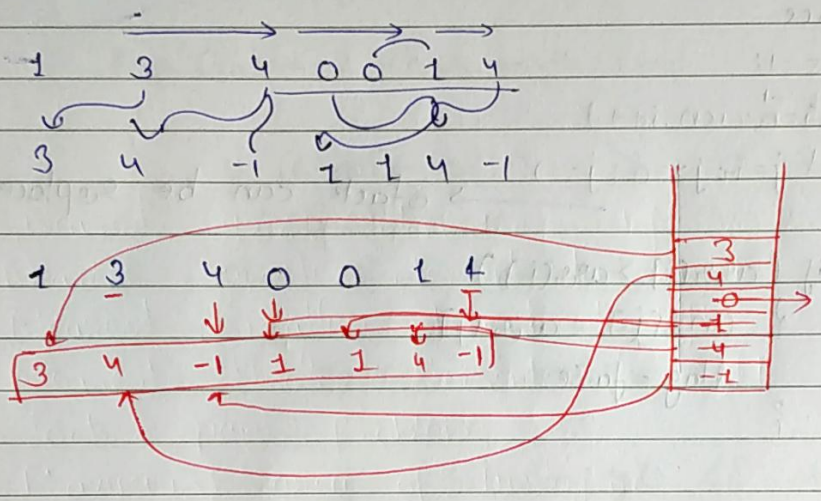
Brout force

```
int ans[] = new int[n]
for (int i=0 ; i<n; i++)
{ flag = true;
    for (int j=j+1 ; j<n ; j++)
      { if (arr[i]< arr[j]
          { ans[i]= arr[j]
            flag = false;
          }
      }
    if (flag)
    { ans[i] = -1; }
}
```

→ this can be changed into stack

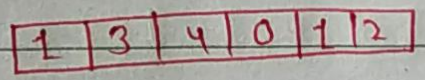
Note → if our dependent loop is in i to n-1 and at that time we will traule in n-1 to i

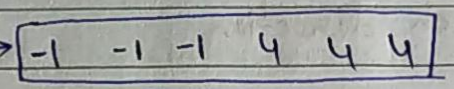




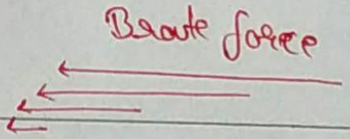

```
int next_greater_to_right (int [] arr)
{
  stack <Integer> st = new stack <Integer> ();
  int [] ans = new int [arr.length];
  for( int i = arr.length-1 ; i>=0 ; i--)
    {     if (st.isEmpty())
          { ans[i] = -1; }
          if (st.isEmpty() && st.peek() > arr[i])
            {
              ans[i] = st.peek();
            }
          else if (!st.isEmpty() && st.peek() <= arr[i])
          { while(!st.isEmpty() && st.peek() <= arr[i])
              { st.pop(); }
            if (st.isEmpty()) { ans[i] = -1; }
            else { ans[i] = st.peek(); }
    } st.push(arr[i]); DOMS
```

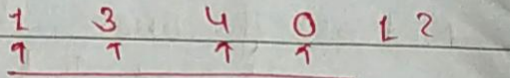
Next - greater to left

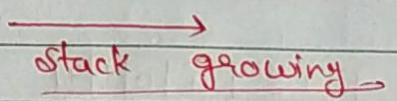




Brute force

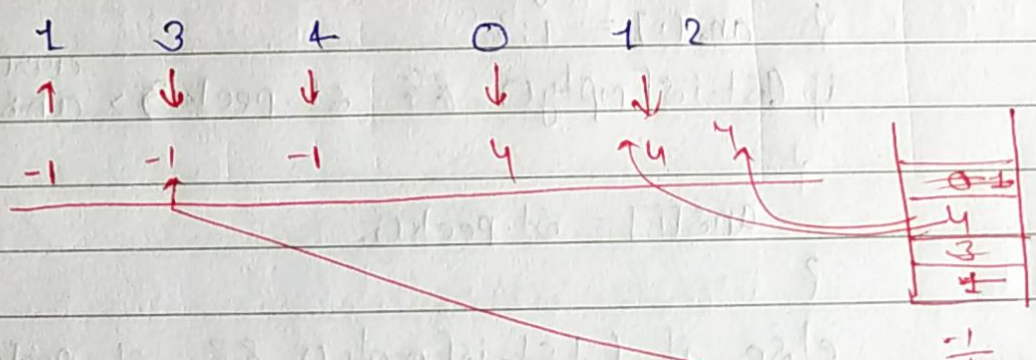







```
Brute force
    ans[0] = -1;
for (int i=0; i<n; i++)
    for(int j=i; j>=0; j--)  ---> stack can be replaced
    {
        if (arr[j] > arr[i])
        { ans[j] = arr[j]
          flag = false;
        }
    }
    if (flag) { ans[i]=-1; }
}
```

This Problem is similar to Next greater element but the loop flow will be opposite

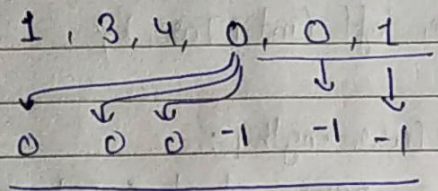


Next - greater element to left

```
~~void~~ int [] next_greater_left (int [] arr) {

    int [] ans = new int [arr.length];
    stack <Integer> st = new stack <Integer> ();

for (int i=0; i<arr.length; i++)
{
      if (st.isEmpty())
       { ans[i] = -1; }
      else if (!st.isEmpty() && st.peek() > arr[i])
       {
          ans[i] = st.peek();
       }
      else if ( !st.isEmpty() && st.peek()<= arr[i])
         { while ( !st.isEmpty() && st.peek() <= arr[i])
             { st.pop(); }
           if (!st.isEmpty())
             { ans[i] = -1; }
           else
             { ans[i] = st.peek(); }
         }
      st.push(arr[i]); // Pushing the curr element
}
```

Next smaller element to Right

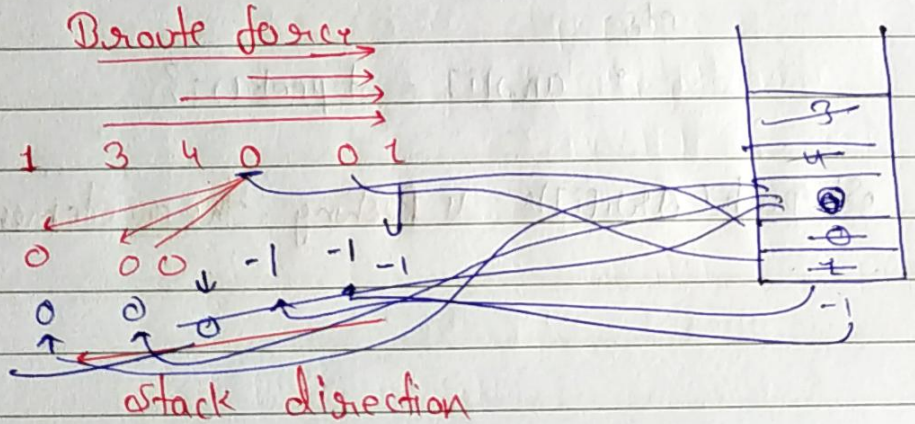
input → 1, 3, 4, 0, 0, 1



Braute force

```
for (int i=0; i<arr.length; i++)
{ flag = true;
    for (int j = i+1; j<arr.length; j++)
    {
        if (arr[j] < arr[i])
        { ans[i] = arr[j];
          flag = false;
        }
    }
    if (flag)
    { ans[i] = -1; }
}
```

Braute force



stack direction

Final Code for next smaller to right

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

Next smaller to left

Note → this question is exactly same as Next smaller to right only loop flow will be opposite

Final code for next smaller to left

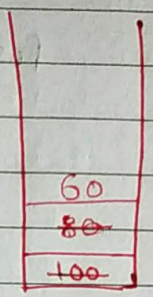
```
int [] next_smaller_toleft (int [] arr)
{
    int [] ans= new int [arr.length];
    Stack<Integer> st = new stack<~~Integer~~ Integer> ();

    for(int i=0; i<arr.length; i++)
    {
        if (st.isEmpty())
        { ans[i] = -1; }
        else if (!st.isEmpty() && st.peek()<arr[i])
        { ans[i] = st.peek(); }
        else if (!st.isEmpty() && st.peek()>=arr[i])
        { while (!st.isEmpty() && st.peek()>=arr[i])
            { st.pop(); }
            if (st.isEmpty())
            { ans[i] = -1; }
            else
            { ans[i] = st.peek(); }
        }
        st.push (arr[i]);
    }
    return ans;
}
```

Stack Span Problem

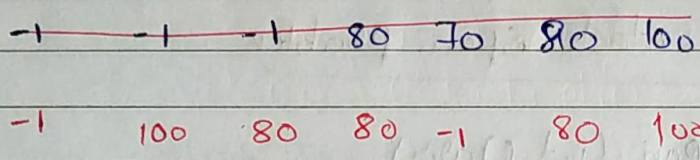
→ 100, 80, 60, 70, 60, 75, 85

↓ ↓ ↓ ↓ ↓ ↓ ↓

1 1 1 2 1 4 6



next ~~greater~~ smaller left



-1 100 80 80 -1 80 100

-1 -1 -1

(4)

we need to store index of NGL element so that we can find our ans

```
int ans[] = new int [length+1];
Stack<Integer> st = new Stack<Integer>();

for (int i=0; i< arr.length; i++)
{   if(st.isEmpty())
    { ans[i] = -1;}
  else if ( !st.isEmpty() && arr[st.peek()] > arr[i])
    {
      ans[i] = st.peek();
    }
  else if (!st.isEmpty() && ans[st.peek()]<=arr[i])
    {
        while (!st.isEmpty() && ans[st.peak()]<=arr[i]);
          { st.pop();}
        if (st.isEmpty())
        { ans[i] = -1;}
        else { ans[i] = st.peek();}
    }
st.push(i);
}
```

when we have index of NGL after that we need a simple loop to small calculation

```
for (int i=0; i<n; i++)
{ ans[i] = i - ans[i]; }
```

this will store our ans.

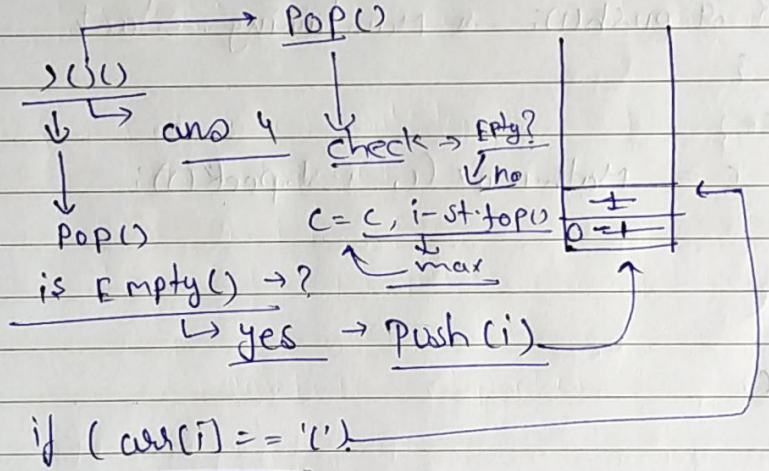
final code for stock span problem

```
int [] stock_span (int ()arr)
{ Finding Nearest greater to left index

    int [] ans = new int [arr.length];
    Stack <Int> st = new Stack <Int> ();

    for (int i=0; i<arr.length; i++)
    { if (st.isEmpty())
        { ans[i] = -1; }
      else if (!st.isEmpty() && arr[st.peek()] > arr[i])
        { ans[i] = st.peek(); }
      else if (!st.isEmpty() && arr[st.peek()] <= arr[i])
        {
          while (!st.isEmpty() && arr[st.peek()] <= arr[i])
            { st.pop(); }
          if (st.isEmpty())
            { ans[i] = -1; }
          else
            { ans[i] = st.peek(); }
        }
      st.push(i);
    }
```

```
for (int i = 0; i < arr.length; i++)
{ int temp = arr[i];
    arr[i] = i - temp; }

return ans;
}
```

Longest Valid Paranthisis

in this question we will maintian a stack of containing index of opening bracket and the last index || Bottom position of stack will contain a invalid index, or -1
↳ which is not valid



Note
↳ stack k Bottom ko ek invalid index se maintain krege
→ initially → -1
→ is kisi condition m stack empty ho jata hai which means ki vo ek valid paranthisis nhi hai, that's why stack maintain krne k liye
→ Push(i)

Final code for Longest Paranthisis :-

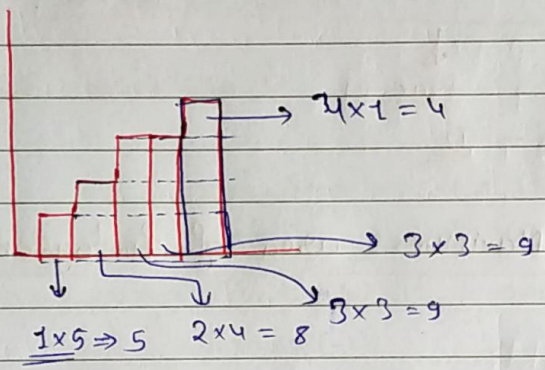
```
int  Longest-Para (int Steing ste)
{
 stack<Integer> st = new stack <Integea>();
   int   c=0;
   for (int  i=0; i<ste.length() ;i++)
   {
      if (ste.charAt(i) == '(')
        {
            st. push(i);
        }
      else{
           st. pop()  // removing opening bracket
         if (St: isEmpty())
           { st. push(i);  // Maintaining Stack
             }
         else{
             c =  Math.max (c, i-st.peek());
           }
      }
  }
  return  c;
}
```

Maximum Area Histogram

input → n number of building and array height[] which represent the height of each building

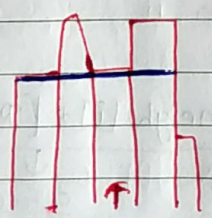
op:→ Maximum area of rectangle.

n⇒ 5
height → 1, 2, 3, 3, 4



Maximum area of histogra
⇒ 9 Ans

Note → A building can be extendend when the height of the other building is >= current build



Here we need to find of out nearest smaller to left, nearest smaller to right [# indexes]

	0	1	2	3	4	5	6
arr →	6	2	5	4	5	1	6
Right NSR index	1	5	3	5	5	7	7
left NSL index	-1	-1	1	1	3	-1	5
(R-(-1)-1)	(5+1-1)	(3-1-1)	5-1-1	5-3-1	7-1	7-5-1	
b ⇒ R-L+1 ⇒	1	5	1	3	1	1	1
	×6	×2	×5	×4	×5	×1	×6
	6	10	5	12	5	6	6

(R-L+1)
+1-1

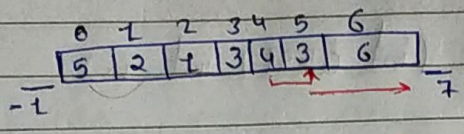
↓ max

ans = 12

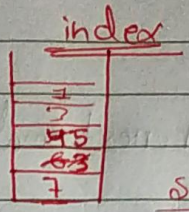
steps for solving Maximum area of histogram :→

1) Find of Nearest smaller to Right index
2) find of the nearest smaller to left index
3) iterate loop 0 to n.
4) maximum value of height[i] * [R[i] - L[i] - 1]
   ↓ area
5) return max

NSR



1 2 7 7 3 7 7



st.push(n)

i = n-1

while (i >= 0)

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

NGR NSL

-1 -1 -1 2 3 2 5

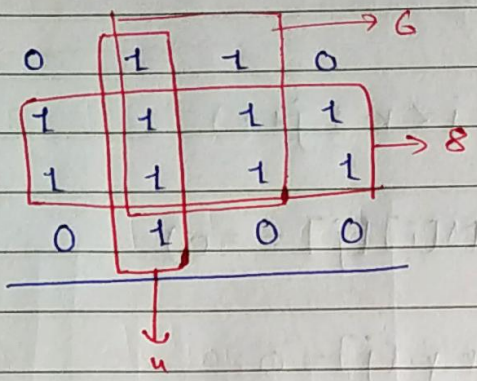
Final code

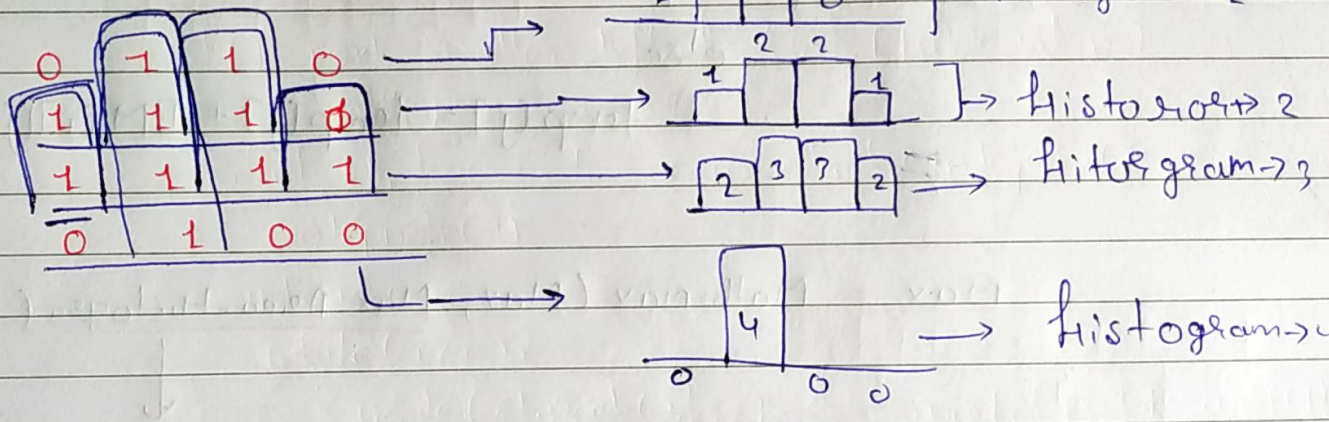
```
class Solution {

    int[] next_smaller_left_idx (int [] arr, int n)
    { int Left[] = new int [n];
      Stack <Integer> st = new Stack <Integer> ();
         st.push(-1);
      int i=0;
      while(i<n){
         while (st.peek() != -1 && arr[st.peek()] >= arr[i])
            { st.pop(); }
       Left[i] = st.peek();
        st.push(i);
        i++;
      }
     return left;

    }
```

```
int [] next_smaller_Right_index (int arr[], int n)
{
    int [] right = new int [n];
    stack <Integer> st = new stack<Integer>();
    st.push(n);
    int i = n-1;

    while (i>=0)
    { while ( st.peek()!=n && st arr[st.peek()]>= arr
        { st.pop();}
      right[i] = st.peek();
      st.push (i);
      i--;
    }

    return right;
}

int Max_Area_Histogram (int []arr, int n) {
{
    int[] R= next_smaller_Right_index (arr, n);
    int[] L = next_smaller_left_index (arr, n);

    int Max = Integer.MIN_VALUE;
    for(int i=0 ; i<n; i++)
    { int h = arr(i);
      int b = R[i]-L[i]-1;
      max = Math.max (max, h*b);
    }
    return max;
  }
}
```

Maximum area rectangle in binary Matrix:-

You are given an binary matrix you need to find out the maximum area rectangle which can be formed with 1.





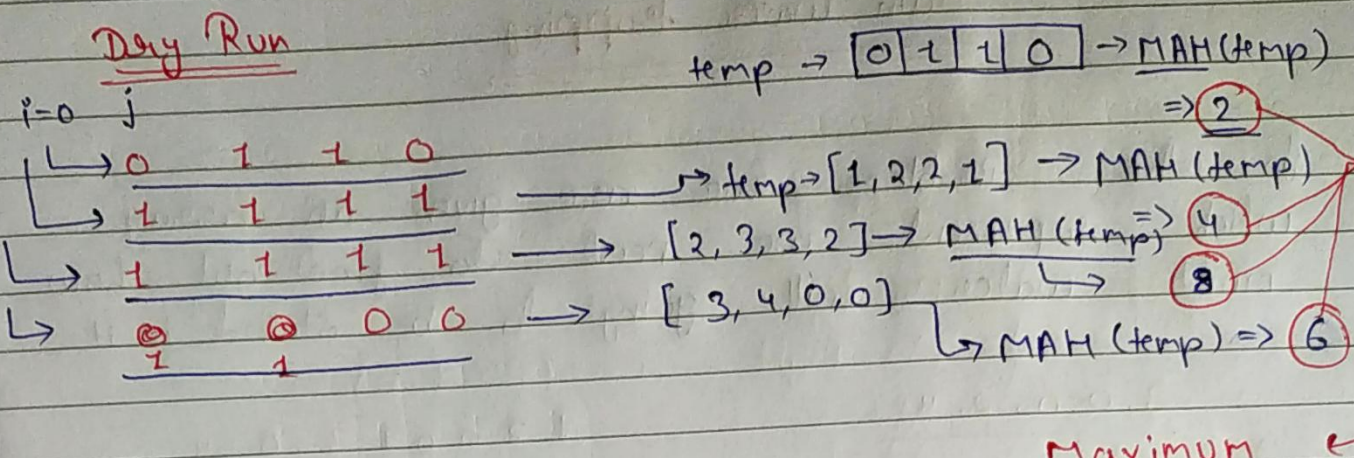
M

ans ⇒ Max (Max_Area_Histogram(1), (2), (3), (4))

```
int Max_area_of_BM (int [][] BM , int n)
{
    int Max = Integer.MIN_VALUE;
    int temp[] = new int[n];
for(int i=0; i<n; i++)
{
    for (int j = 0; j<n ; j++)
    {
      temp[ if (BM[i][j] == 0)
            {
              temp[j]=0;
            }
          else
           { temp[j] = temp[j] + BM[i][j]; }
    }

    Max = Math.max (Max, Max_Area_Histogra ( temp, n)
}
return Max;
}
```

function calling for getting Max areas histograms
Solve in previous question

Dry Run



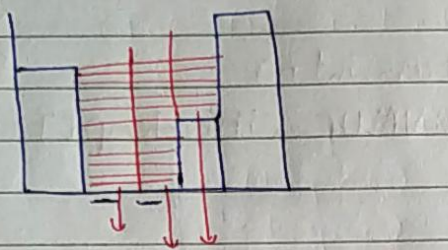
code

```
int     Max_Area( int[][] arr, int n)
{ int  ans= Integer.MIN_VALUE; int temp[] = new int[n]
  for(int i=0; i<n; i++)
    {
      for(int j=0; j<n; j++)
      {
        if (arr[i][j] == 0)
         { temp[j] = 0; }
        else
         { temp[j] = temp[j] + arr[i][j]; }
      }

      ans = Math.max ( max, Max_area_Histogram (temp,n))
    }
    return ans;
}
```

Rain water trapping

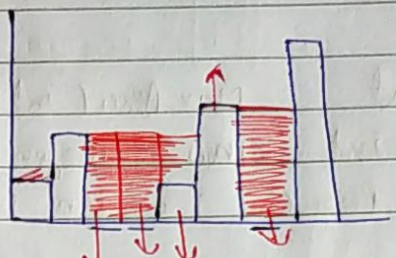
In this problem we are given an array which represent we need to find out how much water will going to be stored in it.

3, 0, 0, 2, 4



3 3 1 → 7 ans

1 2 0 0 1 3, 0, 4



0 2 1 3 ⇒ 8

(2) 4

⇒

2-3 ⇒ -1

1	2	0	0	1	3	0	4

0 0

above water in this building

temp = min (MaxLeft, MaxRight) - heigh[i]

if (temp < 0)

temp = 0

ans += temp;

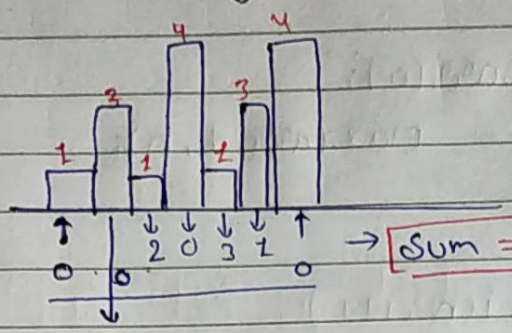
Steps

→ here each

Note

Steps

→ here we need to find out water unit on each building



→ Sum ⇒ 6 → Array

Note → first and last building have zero water unit

Yha par find out krne k liye Do chize required hai

1) Max element in left array
2) Max element in right part of array → Min value

1 4

min

1

min value - heigh

temp = 1 - 3 ⇒ -2

condition if (temp < 0)
temp = 0

Note → we need to find out le Max element in right part of array, and Max element of left Part of array

→ than other Part going to solve from a min calculation

Min (Max_left, Max_Right) - h[i] ⇒ temp

Code

```
int Rain_trapping ( int [] arr, int n )
{
    int Max_left = arr[0];
    int Max_Right = max (arr, 1, n);
    int ans = 0;

    for (int i = 1 ; i < n ; i++ )
    {
      int temp = Min (Max_Left, Max_Right ) - arr [i];
      if (temp < 0)
        { temp = 0; }
      ans = ans + temp;
      Max_left = Math.max (Max_left, arr[i]);
      Max_Right = N
      if (Max_Right == arr[i])
        { Max_Right = max (arr, i, n); }
    }

    return ans;
}
```

string Paranthisis checker

You are given ano string, you need to find out whether the given string is having valid paranthisis or not.

string → "{[()]}" → true → this is a valid string

"{[()]" → this is a invalid

use stack

→ if you get opening bracket

→ Push

→ if closing bracket

→ check is top of stack is having ~~minimum~~ no corresponding ~~open~~ bracket

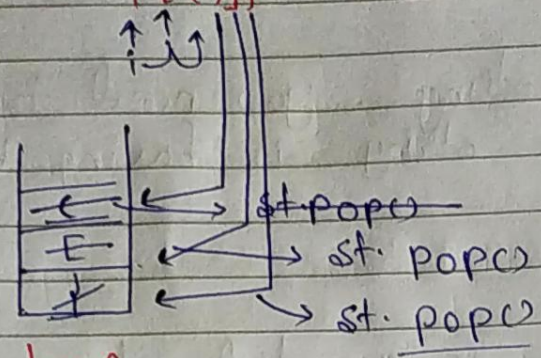
→ if yes

→ st.pop

→ false

→ return false

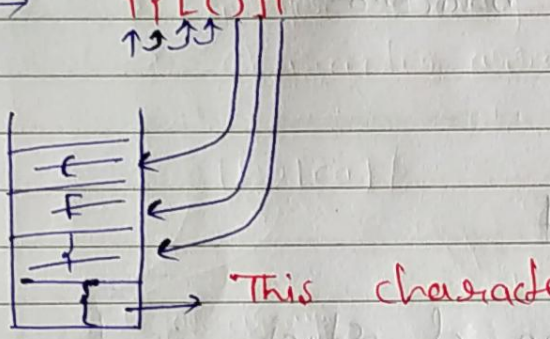
str → "{[()]}"



after loop

```
if (!st.isEmpty())
{ return false; } // if stack is not empty which are some invalid opening bracke
return true;
```

str → "{{[()]}"



This character [opening bracket is occuring invalid condition] thats why

return false;

Note :→ there can be three types of bracket

'{', '(', '[' → considered as opening

'}', ')', ']' → considered as closing

Final code for ~~stac~~ Paranthesis checking

```
class Solution {

    is Boolean is_Valid_Paranthisis ( String str)
    {                  character                  character
        stack <Integer> st = new stack<Integer> ();

        for (int i=0 ; i< str.length() ; i++)
        {
            char t = st.charAt (i);

            if (t=='[' || t=='{' || t=='(') //for opening brack
              { st.push (t); }
            else                          → Empty stack invalid
            { is (st.isEmpty()) { return false}}

                char t1 = st.pop();

same invalid Brackets ←  if ( t==')' && t1 != '(') return false;
                         else if (t==']' && t1 != '[') return false;
                         else if (t=='}' && t1 != '{') return false;
            }

        }

        if (! st.isEmpty ())
        { return false; }  → extra opening bracket return false.

        return true;
    }
```

Decode the string

you are ~~going~~ given an string you need to ~~remove~~ decode it

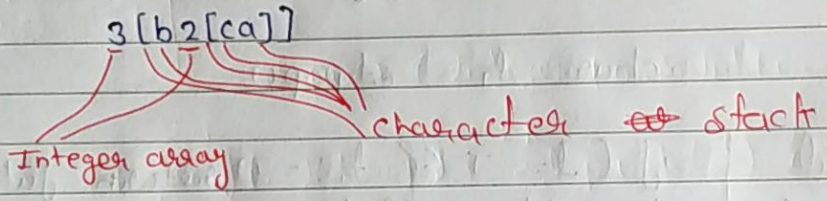
input → "3[b2[ca]]"

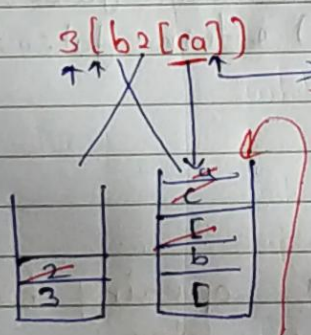
3[bcaca]

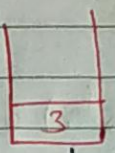
output → bcaca bcaca bcaca

for ~~doing~~ this problem we need to ~~store~~ manage two stacks

→ Integer array
→ character array



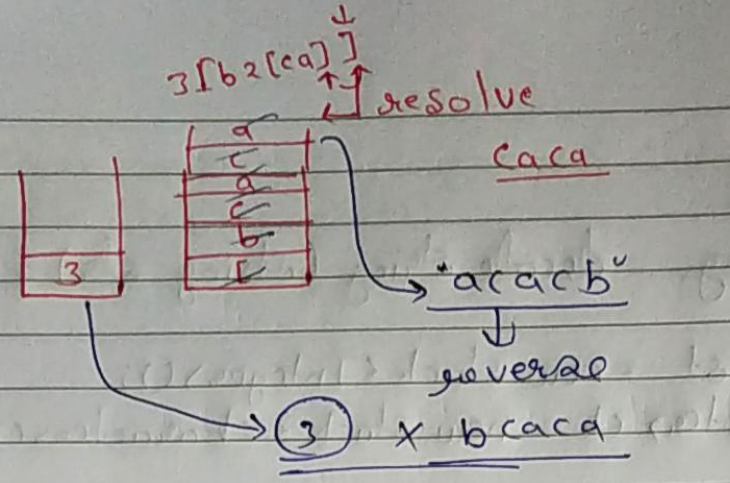




final

Note →

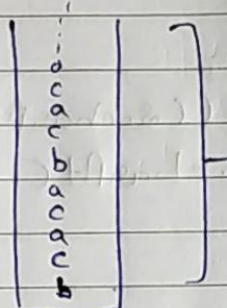
DOMS



3 (bcaca)

final ap bcacabcaca bcaca

stack

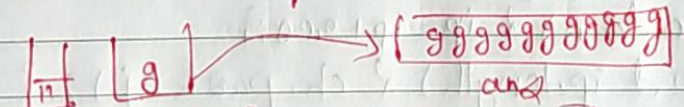


at the end of loop we will be having this stack now we need to convert into string and reverse and that will be over final ans.

Note → we may have ~~integer~~ more than one digit, we need to manage that

11[g] (11) is considered as single intege

in this example



```
class Solution {
    static string decodeString(string str) {

        Stack<Integer> st = new Stack<Integer>();
        Stack<Character> ch = new Stack<Character>();

        string ip = "";
        string res = "";

        for (int i=0; i<str.length(); i++){
            char t = str.charAt(i);
            if(Character.isDigit(t)){
                string num = t + "";
                while (Character.isDigit((s.charAt(i+1)))
                { num = num + str.charAt(i+1);
                    i++;
                }
                st.push(Integer.parseInt(num));
            }
            else if( t!=']'){ ch.push(t);}
            else {
                res = "";
                while(!ch.isEmpty() && ch.peek()!='[') {
                    res = ch.pop()+res;}
                ch.pop();
                ip="";
                int n = st.pop();
                for(i=0 to n) { ip=ip+res;}
                for(j=0 t ip.length) { ch.push(j); }
            }
        }
```

converting into digit and pushing in integer stack

character push

resolving

string and
while (!ch.
return
}
}

```
String ans = "";
while (!ch.isEmpty()){ ans = ch.pop() + ans; } //getting ans
return ans;
    }
}
```