**STACK**

**Aditya Verma**

|  |  |
| --- | --- |
|  |  |
|  | [2 NGR | Nearest Greater to right](#stack_2) |
|  | [3 NGL | Nearest Greater to left](#stack_3) |
|  | [4 NSL | Nearest SMALLER to left](#stack_4) |
|  | [5 NSR | Nearest SMALLER to RIGHT](#stack_5) |
|  | [6 Stock Span Problem](#stack_6) |
|  | [7 Maximum area in histogram](#stack_7) |
|  | [8 Max Area Rectangle in binary matrix](#stack_8) |
|  | [9 Rain Water Trapping](#stack_9) |
|  | [10 Minimum Element in Stack with Extra space](#stack_10) |
|  |  |

**2 NG****R | Nearest Greater to right**

Java solution stack|| [GFG](https://practice.geeksforgeeks.org/problems/next-larger-element-1587115620/1)

Int wala

public int[] NGR(int arr[], int n)  
{  
 //FOR ADDING THE ELEMENTS  
 ArrayList<Integer> list = new ArrayList<>();  
 Stack<Integer> s = new Stack<>();  
  
 //REVERSER LOOP  
 for(int i = n-1; i>=0; i--)  
 {  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(-1);  
 }  
 //IF STACK IS NOT EMPTY AND GREATER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek() > arr[i])  
 {  
 list.add(s.peek());  
 }  
 //IF STACK IS NOT EMPTY AND SMALLER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek() <= arr[i])  
 {  
 //REMOVING THE SMALLER ELEMENTS  
 while(s.size()>0 && s.peek() <= arr[i])  
 {  
 s.pop();  
 }  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(-1);  
 }  
 //IF GREATER THEN ADD  
 else{  
 list.add(s.peek());  
 }  
 }  
  
 //ADDING THE ELEMENT  
 s.push(arr[i]);  
 }  
  
 //REVERSING THE LIST  
 Collections.*reverse*(list);  
  
 //ANS ARRAY DECLEARATION  
 int ans[] = new int[list.size()];  
 //COUNTER FOR ARRAY  
 int a = 0;  
  
 //FOR EACH LOOP FOR COPYING THE ELEMENETS  
 for(int x : list)  
 {  
 ans[a++] = x;  
 }  
  
 //RETURING THE ARRAY  
 return ans;  
  
}

Long wala

class Solution  
{  
 //Function to find the next greater element for each element of the array.  
 public static long[] nextLargerElement(long[] arr, int n)  
 {  
 // Your code here  
 //FOR ADDING THE ELEMENTS  
 ArrayList<Long> list = new ArrayList<>();  
 Stack<Long> s = new Stack<>();  
  
 //REVERSER LOOP  
 for(int i = n-1; i>=0; i--)  
 {  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(Long.*valueOf*(-1));  
 }  
 //IF STACK IS NOT EMPTY AND GREATER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek() > arr[i])  
 {  
 list.add(s.peek());  
 }  
 //IF STACK IS NOT EMPTY AND SMALLER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek() <= arr[i])  
 {  
 //REMOVING THE SMALLER ELEMENTS  
 while(s.size()>0 && s.peek() <= arr[i])  
 {  
 s.pop();  
 }  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(Long.*valueOf*(-1));  
 }  
 //IF GREATER THEN ADD  
 else{  
 list.add(s.peek());  
 }  
 }  
  
 //ADDING THE ELEMENT  
 s.push(arr[i]);  
 }  
  
 //REVERSING THE LIST  
 Collections.reverse(list);  
  
 //ANS ARRAY DECLEARATION  
 long ans[] = new long[list.size()];  
 //COUNTER FOR ARRAY  
 int a = 0;  
  
 //FOR EACH LOOP FOR COPYING THE ELEMENETS  
 for(Long x : list)  
 {  
 ans[a++] = x;  
 }  
  
 //RETURING THE ARRAY  
 return ans;  
 }  
}

Given an array **arr[ ]** of size **N** having distinct elements, the task is to find the next greater element for each element of the array in order of their appearance in the array.  
Next greater element of an element in the array is the nearest element on the right which is greater than the current element.  
If there does not exist next greater of current element, then next greater element for current element is -1. For example, next greater of the last element is always -1.

**Example 1:**

**Input**:

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

**Output**:

3 4 4 -1

**Explanation**:

In the array, the next larger element

to 1 is 3 , 3 is 4 , 2 is 4 and for 4 ?

since it doesn't exist, it is -1.

**3 NGL | Nearest Greater to** **Left**

Java solution stack

Int wala

public int[] NGL(int arr[], int n)  
{  
 //FOR ADDING THE ELEMENTS  
 ArrayList<Integer> list = new ArrayList<>();  
 Stack<Integer> s = new Stack<>();  
  
 //REVERSER LOOP  
 for(int i = 0; i< n; i++)  
 {  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(-1);  
 }  
 //IF STACK IS NOT EMPTY AND GREATER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek() > arr[i])  
 {  
 list.add(s.peek());  
 }  
 //IF STACK IS NOT EMPTY AND SMALLER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek() <= arr[i])  
 {  
 //REMOVING THE SMALLER ELEMENTS  
 while(s.size()>0 && s.peek() <= arr[i])  
 {  
 s.pop();  
 }  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(-1);  
 }  
 //IF GREATER THEN ADD  
 else{  
 list.add(s.peek());  
 }  
 }  
  
 //ADDING THE ELEMENT  
 s.push(arr[i]);  
 }  
  
  
 //ANS ARRAY DECLEARATION  
 int ans[] = new int[list.size()];  
 //COUNTER FOR ARRAY  
 int a = 0;  
  
 //FOR EACH LOOP FOR COPYING THE ELEMENETS  
 for(int x : list)  
 {  
 ans[a++] = x;  
 }  
  
 //RETURING THE ARRAY  
 return ans;  
  
}

Long wala

package sample;  
import java.util.\*;  
  
  
class Solution  
{  
 //Function to find the next greater element for each element of the array.  
 public long[] nextLargerElement(long[] arr, int n)  
 {  
 // Your code here  
 //FOR ADDING THE ELEMENTS  
 ArrayList<Long> list = new ArrayList<>();  
 Stack<Long> s = new Stack<>();  
  
 //REVERSER LOOP  
 for(int i = 0; i < n; i++)  
 {  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(Long.*valueOf*(-1));  
 }  
 //IF STACK IS NOT EMPTY AND GREATER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek() > arr[i])  
 {  
 list.add(s.peek());  
 }  
 //IF STACK IS NOT EMPTY AND SMALLER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek() <= arr[i])  
 {  
 //REMOVING THE SMALLER ELEMENTS  
 while(s.size()>0 && s.peek() <= arr[i])  
 {  
 s.pop();  
 }  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(Long.*valueOf*(-1));  
 }  
 //IF GREATER THEN ADD  
 else{  
 list.add(s.peek());  
 }  
 }  
  
 //ADDING THE ELEMENT  
 s.push(arr[i]);  
 }  
  
 long ans[] = new long[list.size()];  
 //COUNTER FOR ARRAY  
 int a = 0;  
  
 //FOR EACH LOOP FOR COPYING THE ELEMENETS  
 for(Long x : list)  
 {  
 ans[a++] = x;  
 }  
  
 //RETURING THE ARRAY  
 return ans;  
 }  
}  
  
  
  
public class Main {  
 public static void main(String args[])  
 {  
 Solution s = new Solution();  
  
 long arr[] = {1,3,2,4};  
  
 long ans[] = s.nextLargerElement(arr, arr.length);  
  
 for(long x: ans)  
 {  
 System.*out*.println(x);  
 }  
  
 }  
}

**Example 1:**

**Input**:

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

**Output**:

-1 -1 3 -1

**4 NSL | Nearest** **SMALLER to Left**

Java solution stack

Int wala

public int[] NSL(int[] arr, int n)  
{  
 // Your code here  
 //FOR ADDING THE ELEMENTS  
 ArrayList<Integer> list = new ArrayList<>();  
 Stack<Integer> s = new Stack<>();  
  
 //REVERSER LOOP  
 for(int i = 0; i < n; i++)  
 {  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(-1);  
 }  
 //IF STACK IS NOT EMPTY AND SMALLER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek() < arr[i])  
 {  
 list.add(s.peek());  
 }  
 //IF STACK IS NOT EMPTY AND GREATER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek() >= arr[i])  
 {  
 //REMOVING THE GREATER ELEMENTS  
 while(s.size()>0 && s.peek() >= arr[i])  
 {  
 s.pop();  
 }  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(-1);  
 }  
 //IF GREATER THEN ADD  
 else{  
 list.add(s.peek());  
 }  
 }  
  
 //ADDING THE ELEMENT  
 s.push(arr[i]);  
 }  
  
 int ans[] = new int[list.size()];  
 //COUNTER FOR ARRAY  
 int a = 0;  
  
 //FOR EACH LOOP FOR COPYING THE ELEMENETS  
 for(int x : list)  
 {  
 ans[a++] = x;  
 }  
  
 //RETURING THE ARRAY  
 return ans;  
}

long wala

package sample;  
import java.util.\*;  
  
class Solution  
{  
 //Function to find the next greater element for each element of the array.  
 public long[] NSL(long[] arr, int n)  
 {  
 // Your code here  
 //FOR ADDING THE ELEMENTS  
 ArrayList<Long> list = new ArrayList<>();  
 Stack<Long> s = new Stack<>();  
  
 //REVERSER LOOP  
 for(int i = 0; i < n; i++)  
 {  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(Long.*valueOf*(-1));  
 }  
 //IF STACK IS NOT EMPTY AND SMALLER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek() < arr[i])  
 {  
 list.add(s.peek());  
 }  
 //IF STACK IS NOT EMPTY AND GREATER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek() >= arr[i])  
 {  
 //REMOVING THE GREATER ELEMENTS  
 while(s.size()>0 && s.peek() >= arr[i])  
 {  
 s.pop();  
 }  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(Long.*valueOf*(-1));  
 }  
 //IF GREATER THEN ADD  
 else{  
 list.add(s.peek());  
 }  
 }  
  
 //ADDING THE ELEMENT  
 s.push(arr[i]);  
 }  
  
 long ans[] = new long[list.size()];  
 //COUNTER FOR ARRAY  
 int a = 0;  
  
 //FOR EACH LOOP FOR COPYING THE ELEMENETS  
 for(Long x : list)  
 {  
 ans[a++] = x;  
 }  
  
 //RETURING THE ARRAY  
 return ans;  
 }  
}  
  
  
  
public class Main {  
 public static void main(String args[])  
 {  
 Solution s = new Solution();  
  
 long arr[] = {4,5,2,10,8};  
  
 long ans[] = s.nextLargerElement(arr, arr.length);  
  
 for(long x: ans)  
 {  
 System.*out*.println(x);  
 }  
  
 }  
}

**Example 1:**

**Input**:

N = 4, arr[] = [4,5,2,10,8]

**Output**:

-1 4 -1 2 2

**5 NSR | Nearest SMALLER to RIGHT**

Java solution stack

Int wala

//NEAREST SMALLER TO RIGHT  
public int[] NSR(int[] arr, int n)  
{  
 // Your code here  
 //FOR ADDING THE ELEMENTS  
 ArrayList<Integer> list = new ArrayList<>();  
 Stack<Pair<Integer,Integer>> s = new Stack<>();  
  
 //REVERSER LOOP  
 for(int i = n-1; i >= 0; i--)  
 {  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(-1);  
 }  
 //IF STACK IS NOT EMPTY AND SMALLER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek() < arr[i])  
 {  
 list.add(s.peek());  
 }  
 //IF STACK IS NOT EMPTY AND GREATER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek() >= arr[i])  
 {  
 //REMOVING THE GREATER ELEMENTS  
 while(s.size()>0 && s.peek() >= arr[i])  
 {  
 s.pop();  
 }  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(-1);  
 }  
 //IF GREATER THEN ADD  
 else{  
 list.add(s.peek());  
 }  
 }  
  
 //ADDING THE ELEMENT  
 s.push(arr[i]);  
 }  
  
 //REVERSING THE LIST  
 Collections.*reverse*(list);  
  
 //ANS ARRAY DECLARATION  
 int ans[] = new int[list.size()];  
 //COUNTER FOR ARRAY  
 int a = 0;  
  
 //FOR EACH LOOP FOR COPYING THE ELEMENETS  
 for(int x : list)  
 {  
 ans[a++] = x;  
 }  
  
 //RETURING THE ARRAY  
 return ans;  
}

Long wala

package sample;  
import java.util.\*;  
  
class Solution  
{  
 //Function to find the next greater element for each element of the array.  
 public long[] NSR(long[] arr, int n)  
 {  
 // Your code here  
 //FOR ADDING THE ELEMENTS  
 ArrayList<Long> list = new ArrayList<>();  
 Stack<Long> s = new Stack<>();  
  
 //REVERSER LOOP  
 for(int i = n-1; i >= 0; i--)  
 {  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(Long.*valueOf*(-1));  
 }  
 //IF STACK IS NOT EMPTY AND SMALLER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek() < arr[i])  
 {  
 list.add(s.peek());  
 }  
 //IF STACK IS NOT EMPTY AND GREATER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek() >= arr[i])  
 {  
 //REMOVING THE GREATER ELEMENTS  
 while(s.size()>0 && s.peek() >= arr[i])  
 {  
 s.pop();  
 }  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(Long.*valueOf*(-1));  
 }  
 //IF GREATER THEN ADD  
 else{  
 list.add(s.peek());  
 }  
 }  
  
 //ADDING THE ELEMENT  
 s.push(arr[i]);  
 }  
  
 //REVERSING THE LIST  
 Collections.*reverse*(list);  
  
 //ANS ARRAY DECLARATION  
 long ans[] = new long[list.size()];  
 //COUNTER FOR ARRAY  
 int a = 0;  
  
 //FOR EACH LOOP FOR COPYING THE ELEMENETS  
 for(Long x : list)  
 {  
 ans[a++] = x;  
 }  
  
 //RETURING THE ARRAY  
 return ans;  
 }  
}  
  
  
  
public class Main {  
 public static void main(String args[])  
 {  
 Solution s = new Solution();  
  
 long arr[] = {4,5,2,10,8};  
  
 long ans[] = s.NSR(arr, arr.length);  
  
 for(long x: ans)  
 {  
 System.*out*.println(x);  
 }  
  
 }  
}

**Example 1:**

**Input**:

N = 4, arr[] = [4,5,2,10,8]

**Output**:

2 2 -1 8 -1

**6 | Stock Span Problem**

package sample;  
import java.util.\*;  
  
class Pair<A, B> {  
 private A first;  
 private B second;  
  
 public Pair(A first, B second) {  
 super();  
 this.first = first;  
 this.second = second;  
 }  
  
 public A getValue() {  
 return first;  
 }  
  
 public B getIndex() {  
 return second;  
 }  
  
}  
  
class Solution{  
 public int[] NGL(int arr[], int n)  
 {  
  
 //FOR ADDING THE ELEMENTS  
 ArrayList<Integer> list = new ArrayList<>();  
  
 //USED PAIR TO ADD ELEMENT OF ARR AND INDEX  
 Stack<Pair<Integer, Integer>> s = new Stack<>();  
 //REVERSER LOOP  
 for(int i = 0; i< n; i++)  
 {  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(-1);  
 }  
 //IF STACK IS NOT EMPTY AND GREATER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek().getValue() > arr[i])  
 {  
 //ADDING THE INDEX OF THE ELEMENT  
 list.add(s.peek().getIndex());  
 }  
 //IF STACK IS NOT EMPTY AND SMALLER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek().getValue() <= arr[i])  
 {  
 //REMOVING THE SMALLER ELEMENTS AND INDEX PAIR  
 while(s.size()>0 && s.peek().getValue() <= arr[i])  
 {  
 s.pop();  
 }  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(-1);  
 }  
 //IF ELEMENT IS GREATER THEN ADD ITS INDEX  
 else{  
 list.add(s.peek().getIndex());  
 }  
 }  
  
 //ADDING THE PAIR  
 s.push(new Pair(arr[i], i));  
 }  
  
  
 //ANS ARRAY DECLEARATION  
 int ans[] = new int[list.size()];  
  
  
 //subtracting the index with list OF INDEX VAL  
 for(int i =0; i<list.size(); i++)  
 {  
 ans[i] = i - list.get(i);  
 }  
  
 //RETURING THE ARRAY  
 return ans;  
  
 }  
  
}  
  
  
public class Main {  
 public static void main(String args[])  
 {  
 Solution s = new Solution();  
  
 int arr[] = {100, 80,60,70,60,75,85};  
  
 int ans[] = s.NGL(arr, arr.length);  
  
 for(int x: ans)  
 {  
 System.*out*.println(x);  
 }  
  
 }  
}

The stock span problem is a financial problem where we have a series of **n** daily price quotes for a stock and we need to calculate the span of stocks price for all **n** days.   
The span **Si** of the stocks price on a given day **i** is defined as the maximum number of consecutive days just before the given day, for which the price of the stock on the current day is less than or equal to its price on the given day.  
For example, if an array of 7 days prices is given as {100, 80, 60, 70, 60, 75, 85}, then the span values for corresponding 7 days are {1, 1, 1, 2, 1, 4, 6}.

**Example 1:**

**Input**:

N = 7, price[] = [100 80 60 70 60 75 85]

**Output**:

1 1 1 2 1 4 6

**Explanation**:

Traversing the given input span for 100

will be 1, 80 is smaller than 100 so the

span is 1, 60 is smaller than 80 so the

span is 1, 70 is greater than 60 so the

span is 2 and so on. Hence the output will

be 1 1 1 2 1 4 6.

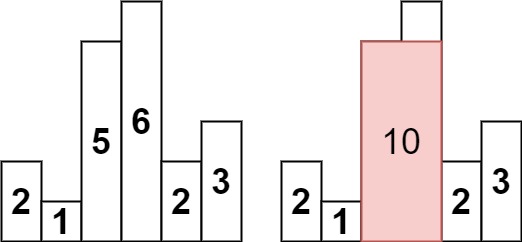
**7 |Maximum Area in Histogram**

[Leet code 84. Largest Rectangle in Histogram JAVA](https://leetcode.com/problems/largest-rectangle-in-histogram)

/\*STEPS TO APPROACH THE QUESTION  
//(Nearest smaller element to right & Nearest smaller element to Left)  
  
 Calculate Index of NSR and NSL and store it in arrays Left and Right  
 Calculate Width by Width[i] = Left[i] - Right[i] - 1  
 Calculate Area by Area = Height[i] \* Width[i]  
\*/  
  
//FOR STORING TWO VALUSE IN HE STACK  
class Pair<A, B> {  
 private A first;  
 private B second;  
  
 public Pair(A first, B second) {  
 super();  
 this.first = first;  
 this.second = second;  
 }  
  
 public A getValue() {  
 return first;  
 }  
  
 public B getIndex() {  
 return second;  
 }  
  
}  
class Solution {  
  
 //NEAREST SMALLER TO LEFT  
 public int[] NSL(int[] arr, int n)  
 {  
 // Your code here  
 //FOR ADDING THE ELEMENTS  
 ArrayList<Integer> list = new ArrayList<>();  
 Stack<Pair<Integer,Integer>> s = new Stack<>();  
  
 //REVERSER LOOP  
 for(int i = 0; i < n; i++)  
 {  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(-1);  
 }  
 //IF STACK IS NOT EMPTY AND SMALLER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek().getValue() < arr[i])  
 {  
 list.add(s.peek().getIndex());  
 }  
 //IF STACK IS NOT EMPTY AND GREATER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek().getValue() >= arr[i])  
 {  
 //REMOVING THE GREATER ELEMENTS  
 while(s.size()>0 && s.peek().getValue() >= arr[i])  
 {  
 s.pop();  
 }  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(-1);  
 }  
 //IF GREATER THEN ADD  
 else{  
 list.add(s.peek().getIndex());  
 }  
 }  
  
 //ADDING THE ELEMENT  
 s.push(new Pair(arr[i], i));  
 }  
  
 int ans[] = new int[list.size()];  
 //COUNTER FOR ARRAY  
 int a = 0;  
  
 //FOR EACH LOOP FOR COPYING THE ELEMENETS  
 for(int x : list)  
 {  
 ans[a++] = x;  
 }  
  
 //RETURING THE ARRAY  
 return ans;  
 }  
  
  
  
  
 //NEAREST SMALLER TO RIGHT  
 public int[] NSR(int[] arr, int n)  
 {  
 // Your code here  
 //FOR ADDING THE ELEMENTS  
 ArrayList<Integer> list = new ArrayList<>();  
 Stack<Pair<Integer,Integer>> s = new Stack<>();  
  
 //REVERSER LOOP  
 for(int i = n-1; i >= 0; i--)  
 {  
 //IF STACK IS EMPTY  
 //ADDING n  
 if(s.size() == 0)  
 {  
 list.add(n);  
 }  
 //IF STACK IS NOT EMPTY AND SMALLER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek().getValue() < arr[i])  
 {  
 list.add(s.peek().getIndex());  
 }  
 //IF STACK IS NOT EMPTY AND GREATER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek().getValue() >= arr[i])  
 {  
 //REMOVING THE GREATER ELEMENTS  
 while(s.size()>0 && s.peek().getValue() >= arr[i])  
 {  
 s.pop();  
 }  
 //IF STACK IS EMPTY  
 //ADDING n  
 if(s.size() == 0)  
 {  
 list.add(n);  
 }  
 //IF GREATER THEN ADD  
 else{  
 list.add(s.peek().getIndex());  
 }  
 }  
  
 //ADDING THE ELEMENT  
 s.push(new Pair(arr[i], i));  
 }  
  
 //REVERSING THE LIST  
 Collections.*reverse*(list);  
  
 //ANS ARRAY DECLARATION  
 int ans[] = new int[list.size()];  
 //COUNTER FOR ARRAY  
 int a = 0;  
  
 //FOR EACH LOOP FOR COPYING THE ELEMENETS  
 for(int x : list)  
 {  
 ans[a++] = x;  
 }  
  
 //RETURING THE ARRAY  
 return ans;  
 }  
  
 //MAIN FUNCTION  
 public int largestRectangleArea(int[] arr) {  
  
 //FOR CALCULATING THE WIDTH AND AREA  
 int width[] = new int[arr.length];  
 int area[] = new int[arr.length];  
  
  
 //FOR STORING THE INDEX OF NSR AND NSL  
 // NSR = NEARST SMALLET TO RIGHT  
 // NSR = NEARST SMALLET TO LEFT  
 int right[] = NSR(arr, arr.length);  
 int left[] = NSL(arr, arr.length);  
  
  
  
 //CALLCULATING THE WIDTH BY  
 // WIDTH = INDEX OF NSR - INDEX OF NSL - 1  
 for(int i =0; i< arr.length; i++)  
 {  
 width[i] = right[i]-left[i]-1;  
 }  
  
 //CALLCULATING THE AREA BY  
 // AREA = GIVEN HEIGHT(ARR) \* WIDTH  
 for(int i =0; i< arr.length; i++)  
 {  
 area[i] = arr[i]\*width[i];  
 }  
  
 //SORTING FOR EASY  
 Arrays.*sort*(area);  
  
 //Retuning the largest area  
 return area[area.length-1];  
 }  
}

Given an array of integers heights representing the histogram's bar height where the width of each bar is 1, return *the area of the largest rectangle in the histogram*.

**Example 1:**



**Input:** heights = [2,1,5,6,2,3]

**Output:** 10

**Explanation:** The above is a histogram where width of each bar is 1.

The largest rectangle is shown in the red area, which has an area = 10 units.

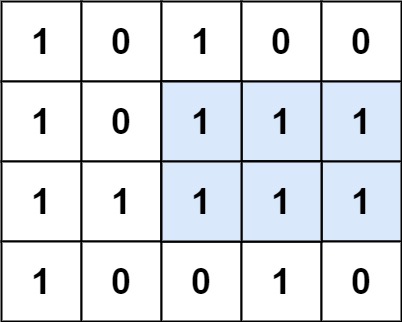
**8 |****Maximum Area in Rectangle in a Binary Matrix**

[**85. Maximal Rectangle leetcode**](https://leetcode.com/problems/maximal-rectangle)

//FOR STORING TWO VALUSE IN HE STACK  
class Pair<A, B> {  
 private A first;  
 private B second;  
  
 public Pair(A first, B second) {  
 super();  
 this.first = first;  
 this.second = second;  
 }  
  
 public A getValue() {  
 return first;  
 }  
  
 public B getIndex() {  
 return second;  
 }  
  
}  
  
class Solution {  
  
 //NEAREST SMALLER TO LEFT  
 public int[] NSL(int[] arr, int n)  
 {  
 // Your code here  
 //FOR ADDING THE ELEMENTS  
 ArrayList<Integer> list = new ArrayList<>();  
 Stack<Pair<Integer,Integer>> s = new Stack<>();  
  
 //REVERSER LOOP  
 for(int i = 0; i < n; i++)  
 {  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(-1);  
 }  
 //IF STACK IS NOT EMPTY AND SMALLER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek().getValue() < arr[i])  
 {  
 list.add(s.peek().getIndex());  
 }  
 //IF STACK IS NOT EMPTY AND GREATER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek().getValue() >= arr[i])  
 {  
 //REMOVING THE GREATER ELEMENTS  
 while(s.size()>0 && s.peek().getValue() >= arr[i])  
 {  
 s.pop();  
 }  
 //IF STACK IS EMPTY  
 //ADDING -1  
 if(s.size() == 0)  
 {  
 list.add(-1);  
 }  
 //IF GREATER THEN ADD  
 else{  
 list.add(s.peek().getIndex());  
 }  
 }  
  
 //ADDING THE ELEMENT  
 s.push(new Pair(arr[i], i));  
 }  
  
 int ans[] = new int[list.size()];  
 //COUNTER FOR ARRAY  
 int a = 0;  
  
 //FOR EACH LOOP FOR COPYING THE ELEMENETS  
 for(int x : list)  
 {  
 ans[a++] = x;  
 }  
  
 //RETURING THE ARRAY  
 return ans;  
 }  
  
  
  
  
 //NEAREST SMALLER TO RIGHT  
 public int[] NSR(int[] arr, int n)  
 {  
 // Your code here  
 //FOR ADDING THE ELEMENTS  
 ArrayList<Integer> list = new ArrayList<>();  
 Stack<Pair<Integer,Integer>> s = new Stack<>();  
  
 //REVERSER LOOP  
 for(int i = n-1; i >= 0; i--)  
 {  
 //IF STACK IS EMPTY  
 //ADDING n  
 if(s.size() == 0)  
 {  
 list.add(n);  
 }  
 //IF STACK IS NOT EMPTY AND SMALLER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek().getValue() < arr[i])  
 {  
 list.add(s.peek().getIndex());  
 }  
 //IF STACK IS NOT EMPTY AND GREATER THAN THE ELEMENT  
 else if(s.size() > 0 && s.peek().getValue() >= arr[i])  
 {  
 //REMOVING THE GREATER ELEMENTS  
 while(s.size()>0 && s.peek().getValue() >= arr[i])  
 {  
 s.pop();  
 }  
 //IF STACK IS EMPTY  
 //ADDING n  
 if(s.size() == 0)  
 {  
 list.add(n);  
 }  
 //IF GREATER THEN ADD  
 else{  
 list.add(s.peek().getIndex());  
 }  
 }  
  
 //ADDING THE ELEMENT  
 s.push(new Pair(arr[i], i));  
 }  
  
 //REVERSING THE LIST  
 Collections.*reverse*(list);  
  
 //ANS ARRAY DECLARATION  
 int ans[] = new int[list.size()];  
 //COUNTER FOR ARRAY  
 int a = 0;  
  
 //FOR EACH LOOP FOR COPYING THE ELEMENETS  
 for(int x : list)  
 {  
 ans[a++] = x;  
 }  
  
 //RETURING THE ARRAY  
 return ans;  
 }  
  
 //MAXIMUM AREA IN A HISTOGRAM  
 public int MAH(int[] arr) {  
  
 //FOR CALCULATING THE WIDTH AND AREA  
 int width[] = new int[arr.length];  
 int area[] = new int[arr.length];  
  
  
 //FOR STORING THE INDEX OF NSR AND NSL  
 // NSR = NEARST SMALLET TO RIGHT  
 // NSR = NEARST SMALLET TO LEFT  
 int right[] = NSR(arr, arr.length);  
 int left[] = NSL(arr, arr.length);  
  
  
  
 //CALLCULATING THE WIDTH BY  
 // WIDTH = INDEX OF NSR - INDEX OF NSL - 1  
 for(int i =0; i< arr.length; i++)  
 {  
 width[i] = right[i]-left[i]-1;  
 }  
  
 //CALLCULATING THE AREA BY  
 // AREA = GIVEN HEIGHT(ARR) \* WIDTH  
 for(int i =0; i< arr.length; i++)  
 {  
 area[i] = arr[i]\*width[i];  
 }  
  
 //SORTING FOR EASY  
 Arrays.*sort*(area);  
  
 //Retuning the largest area  
 return area[area.length-1];  
 }  
  
  
 //Main function  
 public int maximalRectangle(char[][] a) {  
  
 //row and coloumn of the char 2D array  
 int r = a.length;  
 int c = a[0].length;  
  
 //Array named list to save the values  
 int list[] = new int[a[0].length];  
  
  
 //BASE CASE  
 if(a.length == 1 && a[0].length == 1)  
 {  
 if(a[0][0] == '1')  
 {  
 return 1;  
 }else{  
 return 0;  
 }  
 }  
  
 //FOR SAVING INT FORM OF THE 2D ARRAY  
 int arr[][] = new int[a.length][a[0].length];  
  
 //CONERTING THE CHAR ARRAY INTO INT ARRAY   
 // AND SAVING IT IN arr[][]  
 for(int i =0; i<a.length; i++)  
 {  
 for(int j = 0; j<a[0].length; j++)  
 {  
 if(a[i][j] == '1')  
 {  
 arr[i][j] = 1;  
 }else{  
 arr[i][j] = 0;  
 }  
 }  
 }  
  
 //COPYING 1ST ROW OF THE MATRIX  
 for(int j =0; j<c; j++)  
 {  
 list[j] = arr[0][j];  
 }  
  
 //CALCULATING MAX AREA OF HISTOGRAM   
 //AND SAVING IT IN MAX VARIABLE  
 int max = MAH(list);  
  
  
 for(int i =1;i <r; i++)  
 {  
 for(int j = 0; j<c; j++)  
 {  
 //IF 0 THE SET THE HEIGHT TO 0  
 if(arr[i][j] == 0)  
 {  
 list[j] = 0;  
 }//ELSE ADD THE HEIGHT TO THE EXISTING ARRAY  
 else  
 {  
 list[j] += arr[i][j];  
 }  
 }  
 //CALCULATING MAX AREA OF HISTOGRAM   
 //AND SAVING IT IN MAX VARIABLE  
 max = Math.*max*(max, MAH(list));  
 }  
  
  
 //RETURNING THE ANS  
 return max;  
  
 }  
}

Given a rows x cols binary matrix filled with 0's and 1's, find the largest rectangle containing only 1's and return *its area*.

**Example 1:**



**Input:** matrix = [["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]

**Output:** 6

**Explanation:** The maximal rectangle is shown in the above picture.

**9 |****Rain Water Trapping**

[**42. Trapping Rain Water**](https://leetcode.com/problems/trapping-rain-water/)

class Solution {  
 public int trap(int[] arr) {  
 //LENGTH OF THE ARRAY  
 int n = arr.length;  
  
 //STORING THE ABSOLUTE   
 //GREATER TO RIGHT AND LEFT  
 //OF THE ARRAY AT ith POSITION  
 int maxL[] = new int[n];  
 int maxR[] = new int[n];  
  
 //COPYING THE FIRST ELEMENT  
 maxL[0] = arr[0];  
  
 //FOR CALCULATING THE ABSOLUTE   
 //GREATER TO LEFT AT ith POSTION  
 for(int i =1; i<n; i++)  
 {  
 maxL[i] = Math.*max*(maxL[i-1], arr[i]);  
 }  
  
 //COPYING THE LAST ELEMENT  
 maxR[n-1] = arr[n-1];  
  
 //FOR CALCULATING THE ABSOLUTE   
 //GREATER TO RIGTH AT ith POSTION  
 for(int i =n-2; i >= 0; i--)  
 {  
 maxR[i] = Math.*max*(maxR[i+1], arr[i]);  
 }  
  
 //FOR STORING THE VALUE OF WATER ABOVE THE BUILDINGS  
 int water[] = new int[n];  
 for(int i =0; i<n; i++)  
 {  
 water[i] = Math.*min*(maxL[i], maxR[i])-arr[i];  
 }  
  
 //FOR SOTRING SUM  
 int sum = 0;  
  
 //ADDING ALL THE VAL OF WATER STORED ABOVE THE BUILDING  
 for(int i =0; i<n; i++)  
 {  
 sum += water[i];  
 }  
  
 //RETURNING TOTAL WATER TRAPPED  
 return sum;  
 }  
}

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

**Example 1:**



**Input:** height = [0,1,0,2,1,0,1,3,2,1,2,1]

**Output:** 6

**Explanation:** The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

**10 |****Minimum Element in Stack with Extra Space**

[Get minimum element from stack  gfg](https://practice.geeksforgeeks.org/problems/get-minimum-element-from-stack/1/)

class GfG  
{  
 int minEle;  
 Stack<Integer> s = new Stack<Integer>();  
 Stack<Integer> ss= new Stack<Integer>();  
  
  
  
 /\*returns min element from stack\*/  
 int getMin()  
 {  
 // Your code here  
 if(ss.size() == 0)  
 {  
 return -1;  
 }  
 return ss.peek();  
 }  
  
  
  
  
 /\*returns poped element from stack\*/  
 int pop()  
 {  
 // Your code here   
 if(s.size() == 0)  
 {  
 return -1;  
 }  
 int ans = s.peek();  
 s.pop();  
 if(ss.peek() == ans)  
 {  
 ss.pop();  
 }  
  
 return ans;  
 }  
  
  
  
  
 /\*push element x into the stack\*/  
 void push(int x)  
 {  
 // Your code here  
 s.push(x);  
 if(ss.size() == 0 || ss.peek() >= x)  
 {  
 ss.push(x);  
 };  
  
 }  
}

You are given **N** elements and your task is to Implement a Stack in which you can get minimum element in O(1) time.

**Example 1:**

**Input:**

push(2)

push(3)

pop()

getMin()

push(1)

getMin()

**Output:** 3 2 1

**Explanation:** In the first test case for

query

push(2)  Insert 2 into the stack.

  The stack will be {2}

push(3)  Insert 3 into the stack.

  The stack will be {2 3}

pop() Remove top element from stack

  Poped element will be 3 the

  stack will be {2}

getMin() Return the minimum element

  min element will be 2

push(1) Insert 1 into the stack.

The stack will be {2 1}

getMin() Return the minimum element

  min element will be 1