

# Stack

*-By Nikhil Kumar*

<https://www.linkedin.com/in/nikhilkumar0609/>

# STACK

Q Reverse a string using stack

Program :-

```
#include<iostream>
#include<stack>
using namespace std;

int main() {
    string str = "babbar";
    stack<char> s;
    for(int i=0; i<str.length(); i++) {
        char ch = str[i];
        s.push(ch);
    }

    string ans = "";
    while(!s.empty()) {
        char ch = s.top();
        ans.push_back(ch);
        s.pop();
    }

    cout << "answer is : " << ans << endl;
    return 0;
}
```

T.C  $\rightarrow O(n)$ , S.C  $\rightarrow O(n)$

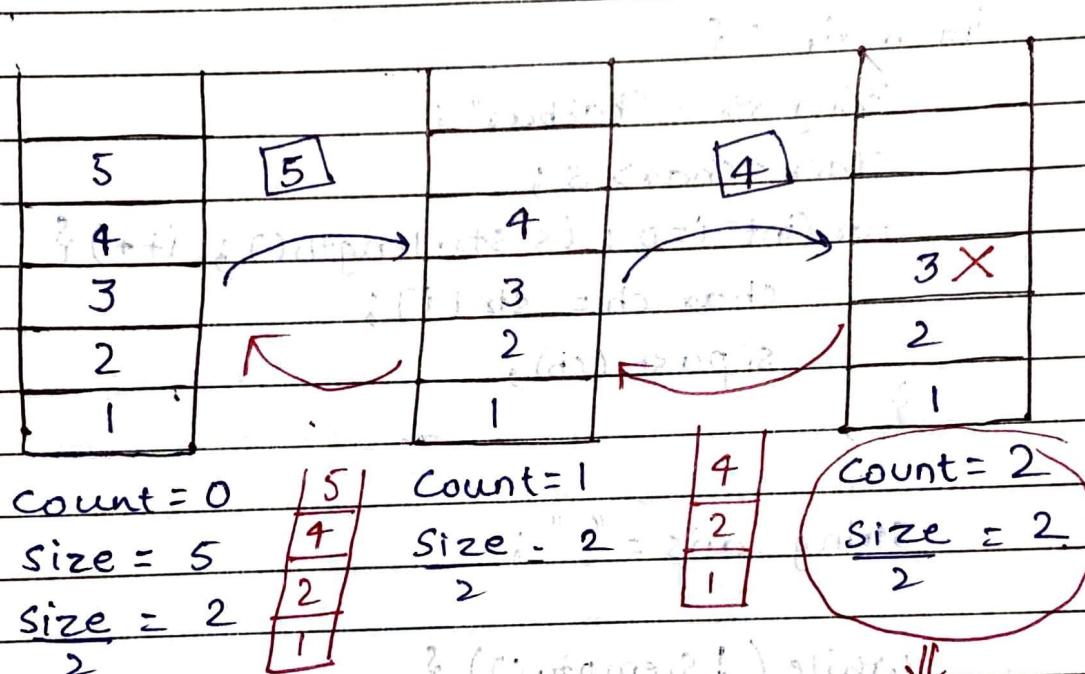
<https://www.linkedin.com/in/nikhilkumar0609/>

Q) Delete middle element from stack

i/p :  $[1, 2, 3, 4, 5] \Rightarrow o/p = [1, 2, 4, 5]$

i/p :  $[5, 6, 7, 8] \Rightarrow o/p = [5, 7, 8]$

Approach :



∴ (Count = 2) is the Base case

∴ (Size = 2) is the Base case

∴ (Size = 2)

Program :→

```
void solve(stack<int>& inputStack, int count, int size) {  
    //base case  
    if (count == size / 2) {  
        return;  
    }  
    int num = inputStack.top();  
    inputStack.pop();  
    solve(inputStack, count + 1, size);  
    inputStack.push(num);  
}
```

```
void deleteMiddle(stack<int>& inputStack, int N) {
```

```
    int count = 0;  
    solve(inputStack, count, N);  
}
```

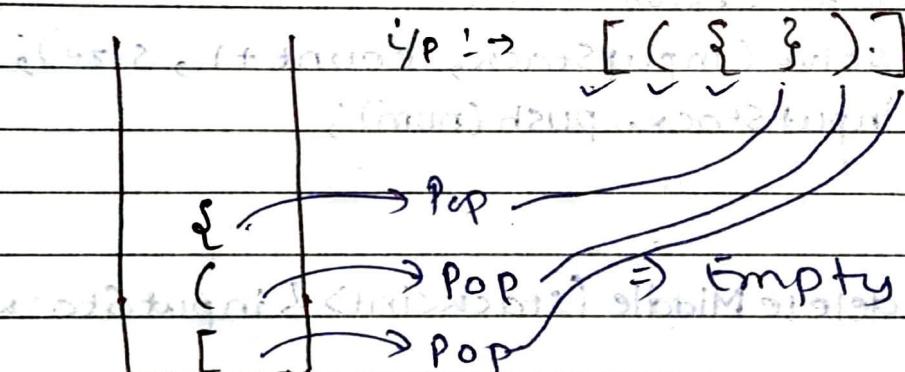
## Q Valid Parenthesis

Approach: -

i/p :  $\left[ \left( \{ \} \right) \right]$

Agar koi open bracket aaye toh usko stack mein push krdo aur jb koi closed bracket aaye toh dekho uske corresponding koi open bracket hai stack mein, aur agar ho toh pop kr do.

Aise krke agar stack empty ho jaaye mtlb valid Parenthesis hai.



## Program :

```
bool isValidParenthesis(string expression) {
    stack<char> s;
    for (int i=0; i<expression.length(); i++) {
        char ch = expression[i];
        // if opening bracket, stack push
        if (ch=='(' || ch=='{' || ch=='[') {
            s.push(ch);
        }
        else {
            // if close bracket, stack top check and pop
            if (!s.empty()) {
                char top = s.top();
                if ((ch==')' && top=='(') ||
                   (ch=='}' && top=='{') ||
                   (ch==']' && top=='[')) {
                    s.pop();
                }
                else {
                    return false;
                }
            }
            else {
                return false;
            }
        }
    }
    if (s.empty())
        return true;
    else
        return false;
}
```

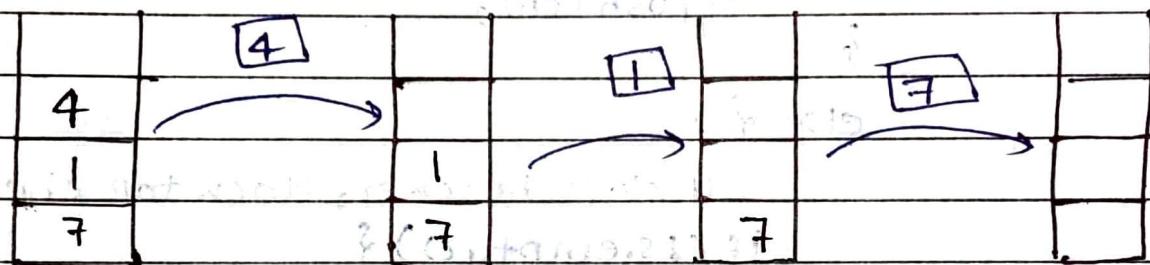
Q Insert an element at its bottom in a given stack.

i/p  $\Rightarrow [7, 1, 4]$ ,  $x = 5$

o/p  $\Rightarrow [5, 7, 1, 4] \rightarrow$



Approach :-



: (0) s.empty()      s.empty()      s.empty()

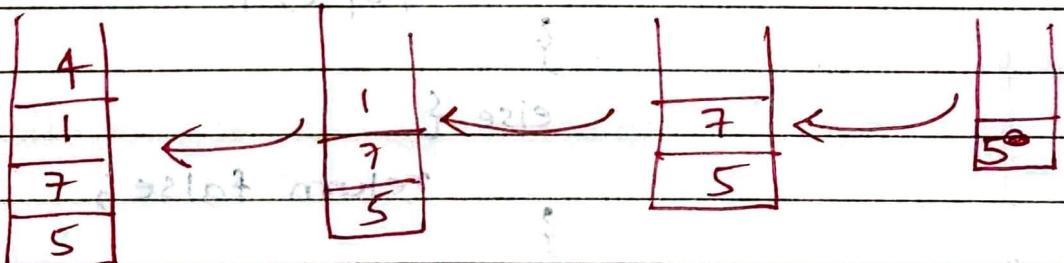
1 (0) == got 88 '1' == ADL      ↓

1 (1) == got 88 'False' == False

True

4 (1) == got 88 '4' == ADL

↓  
s.push(x)



Program :-

```
void solve(stack<int> &s, int x) {
```

// base case

```
if(s.empty()) {  
    s.push(x);  
    return;  
}
```

```
int num = s.top();  
s.pop();
```

// recursive call

```
solve(s, x);  
s.push(num);
```

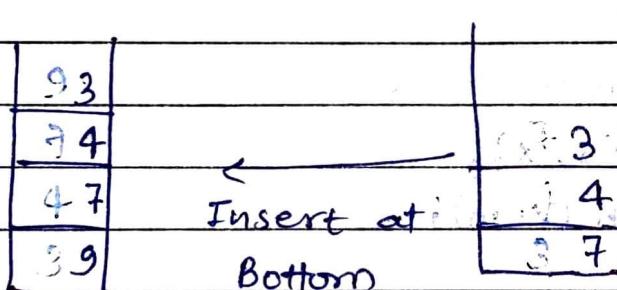
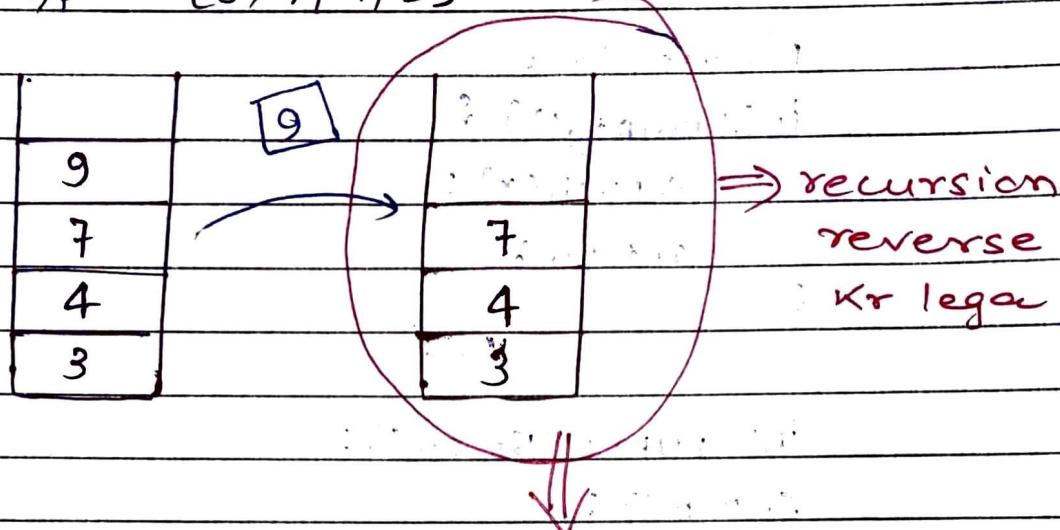
```
stack<int> pushAtBottom(stack<int> &myStack, int x)
```

```
{  
    solve(myStack, x);  
    return myStack;  
}
```

## Reverse Stack using Recursion

i/p : [9, 7, 4, 3]

o/p = [3, 4, 7, 9]



### Algorithm

- Stack Top ko side mein rkh lo.
- Use recursion to reverse remaining stack.
- Wapas aate hue, insertAt Bottom kr dena stackTop ko.

T.C  $\rightarrow O(n^2)$

## Program :-

```
void insertAtBottom(stack<int>&s, int element){  
    //base case  
    if (s.empty()) {  
        s.push(element);  
        return;  
    }  
  
    int num = s.top();  
    s.pop();  
    //recursive call  
    insertAtBottom(s, element);  
    s.push(num);  
}
```

```
void reverseStack(stack<int>&stack) {
```

```
    //base case
```

```
    if (stack.empty()) {
```

```
        return;
```

```
}
```

```
int num = stack.top();
```

```
stack.pop();
```

```
//recursive call
```

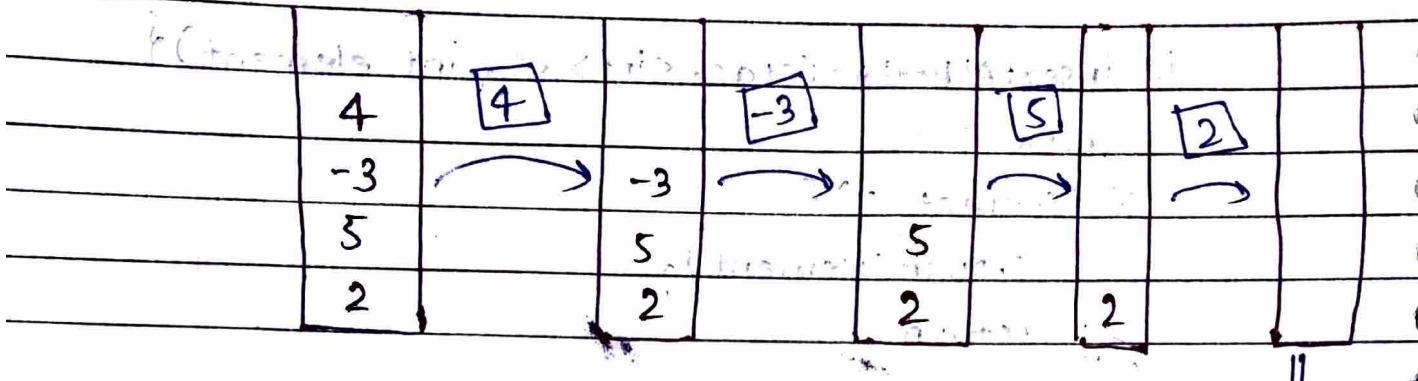
```
reverseStack(stack);
```

```
insertAtBottom(stack, num);
```

```
}
```

0

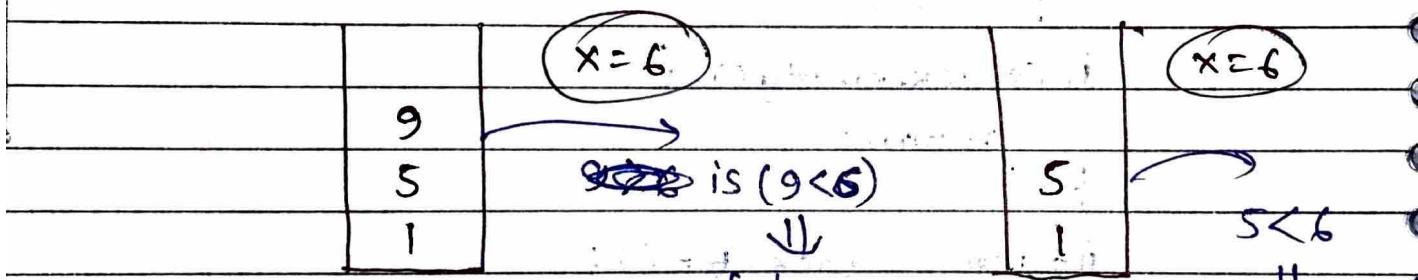
## Sort a Stack



s.empty()

wapas jaate time sorted True  
way mein insert ↗  
krna hai

## Sorted Insert.



toh 9 side

akh Inge

True

↓

6 insert krne

dnge aur

T.C  $\rightarrow O(n^2)$

wapas jaate

time 9 ko

insert kr Inge

## Program :-

```
void sortedInsert(stack<int>& s, int num){  
    //base case  
    if(s.empty() || (!s.empty() & s.top() < num)){  
        s.push(num);  
        return;  
    }  
    int n = s.top();  
    s.pop();
```

### //recursive call

```
    sortedInsert(s, num);  
    s.push(n);  
}
```

```
void sortStack(stack<int>& s){
```

```
    //base case  
    if(s.empty()) {  
        return;  
    }
```

```
    int num = s.top();  
    s.pop();
```

```
    //recursive call
```

```
    sortStack(s);
```

```
    sortedInsert(s, num);
```

```
}
```

## Q Redundant Brackets

( (a+b) )

For 1st pop  
closing bracket, pop +  
we find operator  
 $\downarrow$

Toh operator  
ko pop kije  
aur fir open  
bracket  
ko  
pop  
kije

Koi bhi open bracket ya operator  
toh stack mein push kr do, aur  
Jab closing bracket aaye toh  
check kro ki stack mein  
open bracket se phle koi operator  
hai ya nahi.

Found      Not Found

Redundant Pair

Not Redundant Pair

) → Is baixi koi operator nahi  
mila toh ye redundant  
pair hai.

→ A pair of brackets is said to be redundant  
when a subexpression is surrounded by  
needless/useless brackets.

e.g. ( (a+b) ), (a) , ((a)+(b\*c)), etc.

→ T.C = O(n)

Program :-

```
#include<stack>
bool findRedundantBrackets(string &s) {
    stack<char> st;
    for(int i=0; i<s.length(); i++) {
        char ch = s[i];
        if(ch == '(' || ch == '+' || ch == '-' || ch == '*' || ch == '/') {
            st.push(ch);
        } else { // ch ya toh ')' hai or lowercase letter
            if(ch == ')') {
                bool isRedundant = true;
                while(st.top() != '(') {
                    char top = st.top();
                    if(top == '+' || top == '-' || top == '*' || top == '/') {
                        isRedundant = false;
                    }
                    st.pop();
                }
                if(isRedundant == true)
                    return true;
                st.pop();
            }
        }
    }
    return false;
}
```

## Q Minimum cost to make string valid

## Algorithm :-

Step 1: If length is odd then it is invalid  
            ⇒ return -1

Step 2: → i/p string ⇒ Remove valid part

↓  
Remaining part is of below 3 types

(i) { } { } { } - - -

(ii) गग्गु गग्गु गग्गु - - -

(iii) - - 2 2 2 2 2 - -

Step 3: Count a & b

$a \rightarrow$  count of close braces,  $b \rightarrow$  count of open braces.

even  $a=0$

$$\cdot \quad \{ \quad \} \quad \{ \quad \} = (2) \rightarrow \frac{b}{2}$$

$$(ii) \quad \frac{3333}{\text{even}} \Rightarrow a=4 \quad b=0$$

$$\{ \} \{ \} = 2 \rightarrow \frac{a}{2}$$

(iii)

$$\frac{\{ \quad \} \quad \{ \quad \} \quad \{ \quad \} \quad \{ \quad \} \quad \{ \quad \}}{\text{odd} \qquad \text{odd}} \Rightarrow a=3, b=3$$

$$\overbrace{\{ \quad \} \quad \{ \quad \}}^{\text{even}} \Rightarrow a=24$$

$b=4$

$$\{ \{ \} \{ \} \{ \} = 4$$

$$\{ \} \{ \} \{ \} \{ \} = 4$$

$$\text{ans} = \left( \frac{a+1}{2} \right) + \left( \frac{b+1}{2} \right)$$

Program :-

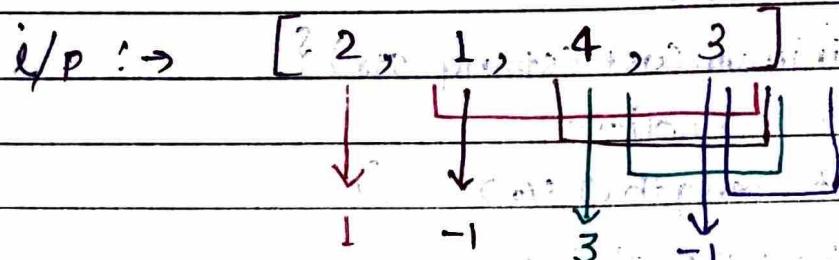
```
#include<stack>
int findMinimumCost(string str) {
    //odd condition
    if (str.length() % 2 == 1)
        return -1;

    stack<char> s;
    for (int i=0; i< str.length(); i++) {
        char ch = str[i];
        if (ch == '{') // ch is open brace
            s.push(ch);
        else { // ch is closed brace
            if (!s.empty() && s.top() == '{')
                s.pop();
            else
                s.push(ch);
        }
    }

    //stack contains invalid expression
    int a=0, b=0;
    while (!s.empty()) {
        if (s.top() == '{')
            b++;
        else
            a++;
        s.pop();
    }

    int ans = (a+1)/2 + (b+1)/2;
    return ans;
}
```

## Q) Next Smaller Element



$$O/P = \{1, -1, 3, -1\}$$

Approach 1:  $2 \rightarrow \{1, 4, 3\} \rightarrow (n-1)$  Comparison

$1 \rightarrow \{4, 3\} \rightarrow (n-2)$  "

$4 \rightarrow \{3\} \rightarrow (n-3)$  "

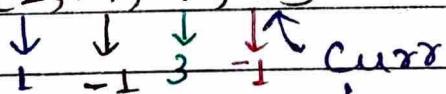
$3 \rightarrow \{\}$   $\rightarrow O(n-4) = 0$  "

$$T.C = O(n^2).$$

$\Rightarrow$  2 for loop chlega, EK main element ke liye, aur dusra, uske aage ke element se compare krne ke liye.

Approach 2:

i/p :  $[2, 1, 4, 3]$



|            |    |
|------------|----|
| Stacks;    | 2  |
| s.push(-1) | 1  |
|            | X  |
|            | X  |
|            | -1 |

chhota element lao

(pop()  $\downarrow$ )

s.top()

Bada.

chhota dhundh ke lao  
while (s.top() >= curr)

{ s.pop(); }

s.top()  $\rightarrow$  chhota

ans = s.top()

s.push(curr)

T.C  $\rightarrow O(n)$

chhota

ans store

s.push(curr)

Program :-

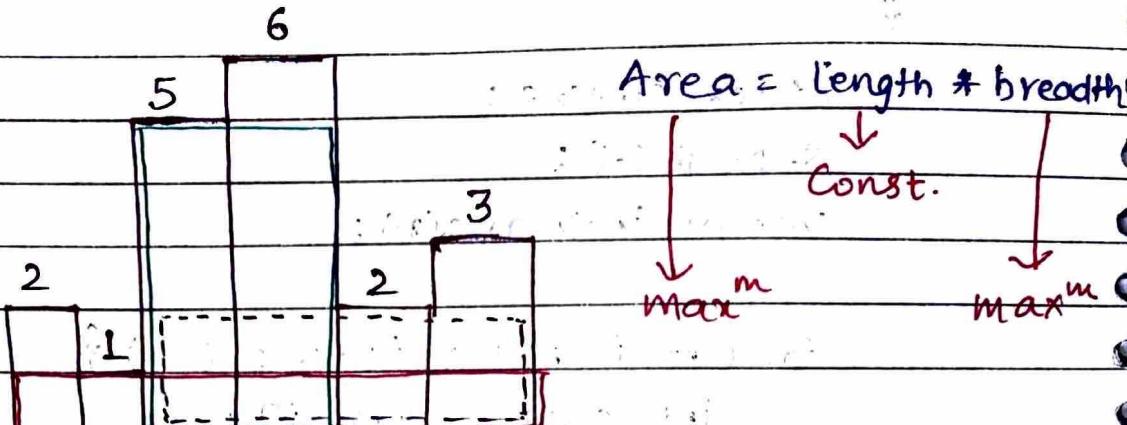
```
#include <stack>
vector<int> nextSmallerElement(vector<int>& arr, int n)
{
    stack<int> s;
    s.push(-1);
    vector<int> ans(n);

    for (int i=n-1 ; i>=0 ; i--) {
        int curr = arr[i];
        while (s.top() >= curr) {
            s.pop();
        }
        //ans is stack ka top
        ans[i] = s.top();
        s.push(curr);
    }
    return ans;
}

Approach 2:
vector<int> ans;
for (int i=0; i<n; i++) {
    int count = 0;
    for (int j=i+1; j<n; j++) {
        if (arr[j] < arr[i]) {
            count++;
            ans.push_back(arr[j]);
            break;
        }
    }
    if (count == 0) {
        ans.push_back(-1);
    }
}
return ans;
```

## Q) Largest Rectangle in Histogram

Approach 1: Brute-force Approach



for 2 → we can't shift either in left/right.

$$\text{so, max}^m \text{ area} = 1 \times 2 = 2$$

$$\text{for } 1 \rightarrow 1 \times 6 = 6$$

$$\text{for } 5 \rightarrow 2 \times 5 = 10$$

for 6 → can't shift in either dir<sup>n</sup>

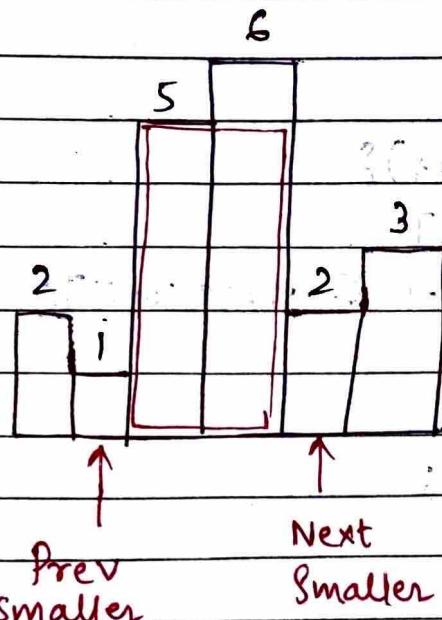
$$\text{so, } 1 \times 6 = 6$$

$$\text{for } 2 \rightarrow 4 \times 2 = 8$$

$$\text{for } 3 \rightarrow 1 \times 3 = 3$$

$$\text{T.C} \rightarrow O(n^2)$$

# Approach 2: Using stack to find next smaller element



like, for 5

prevSmaller index = 1

nextSmaller index = 4

width = next - prev - 1

$$\Rightarrow 4 - 1 - 1$$

length = 2

$$Area = 5 \times 2 = 10.$$

S.C  $\rightarrow O(n)$ , T.C  $\rightarrow O(n)$

Program :

```
private:
    vector<int> nextSmallerElement(vector<int> arr, int n) {
        stack<int> s;
        s.push(-1);
        vector<int> ans(n);
        for (int i = n - 1; i >= 0; i--) {
            int curr = arr[i];
            while (s.top() != -1 && arr[s.top()] >= curr) {
                s.pop();
            }
            // ans is stack ka top
            ans[i] = s.top();
            s.push(i);
        }
        return ans;
    }
```

```

vector<int> prevSmallerElement(vector<int>arr, int n) {
    stack<int> s;
    s.push(-1);
    vector<int> ans(n);
    for (int i=0; i<n; i++) {
        int curr = arr[i];
        while (s.top() != -1 && arr[s.top()] >= curr) {
            s.pop();
        }
        ans[i] = s.top();
        s.push(i);
    }
    return ans;
}

```

```

public:
    int largestRectangleArea(vector<int> &heights) {
        int n = heights.size();
        vector<int> next(n);
        next = nextSmallerElement(heights, n);
        vector<int> prev(n);
        prev = prevSmallerElement(heights, n);
        int area = INT_MIN;
        for (int i=0; i<n; i++) {
            int l = heights[i];
            if (next[i] == -1)
                next[i] = n;
            int b = next[i] - prev[i] - 1;
            int newArea = l * b;
            area = max(area, newArea);
        }
        return area;
    }
}

```

## Q The Celebrity Problem

$\Rightarrow$  Celebrity doesn't know anyone.  
Everyone knows celebrity.

Person

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

$\rightarrow$  Ignore Diagonal

$\rightarrow$  '1' knows No-one.

Everyone Knows '1'

'1' is a celebrity.

### Approach 1: Brute-Force

$\Rightarrow$  celebrity  $\rightarrow$  rows  $\rightarrow$  all 0's.

$\Rightarrow$  Celebrity  $\rightarrow$  col  $\rightarrow$  all 1's except diagonal

element.

$$TC = O(n^2)$$

Optimized Solution

Q

Optimized Solution

Time Complexity

$$\begin{matrix} & 0 & 1 & 2 \\ 0 & \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \\ 1 & & & \\ 2 & & & \end{matrix}$$

# Approach 2 :-

|                |   |   | → Put all element inside stack $(0, 1, 2)$   |
|----------------|---|---|--|
|                |   |   | → Jab tak stack size $b = 1$ .   |
|                |   |   | $\rightarrow A \rightarrow s.top() \rightarrow s.pop()$<br>$\rightarrow B \rightarrow s.top() \rightarrow s.pop()$ |
| <del>A=2</del> | 1 | X |  |
| <del>B=1</del> | X |   | → if (A knows B)   |
| <del>A=1</del> | X |   | $A \times, B \rightarrow wapas push Krd$   |
| <del>B=0</del> | X | X |  |
|                |   |   | → if (B knows A)   |
|                |   |   | $B \times, A \rightarrow wapas push Krd$   |

→ Jo single element bcha hua hai, wo ek "potential celebrity" ho skta hai.

↓

Verifying a candidate for celebrity

→ Celebrity knows No one

↓

row check → all 0's

→ Everyone Knows Celebrity

↓

col check → all 1's except diagonal element

T.C  $\rightarrow O(n)$

Program :-

```
class Solution {
private:
    bool knows(vector<vector<int>>& M, int a, int b, int n) {
        if (M[a][b] == 1)
            return true;
        else
            return false;
    }

public:
    int celebrity(vector<vector<int>>& M, int n) {
        stack<int> s;
        // step 1 :- push all element in stack
        for (int i = 0; i < n; i++) {
            s.push(i);
        }
        // step 2 :- get 2 elements and compare them
        while (s.size() > 1) {
            int a = s.top();
            s.pop();
            int b = s.top();
            s.pop();
            if (knows(M, a, b, n))
                s.push(b);
            else
                s.push(a);
        }
        return s.top();
    }
}
```

<https://www.linkedin.com/in/nikhilkumar0609/>

int ans = s.top();

// step 3 :- single element in stack is potential celebrity, so verify it

int zeroCount = 0;

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

// all zeroes

```
if (zeroCount != n)  
    return -1;
```

// column check

int oneCount = 0;

```
for (int i=0; i<n; i++) {  
    if (M[i][ans] == 1)  
        oneCount++;  
}
```

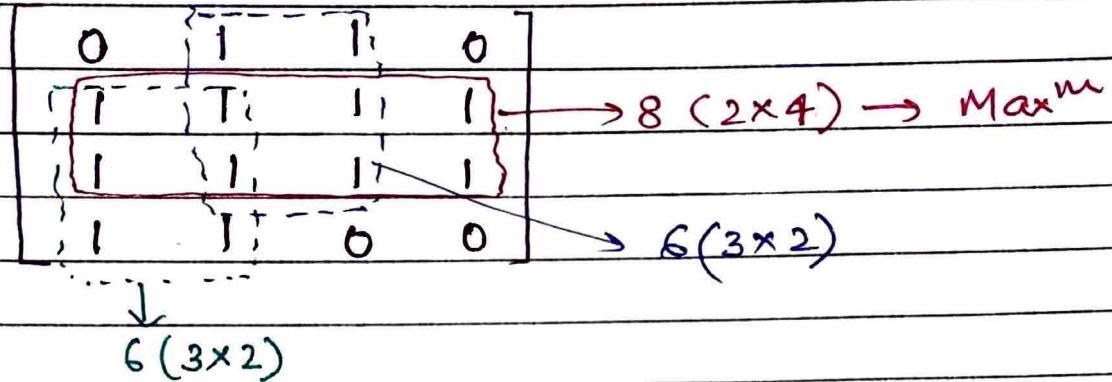
```
if (oneCount == n-1)  
    return ans;
```

return ans;

}

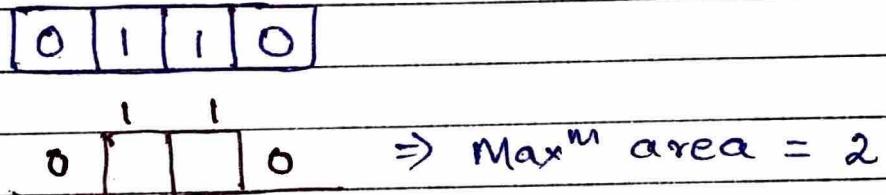
};

## Q Max Rectangle in a Binary Matrix with all 1's

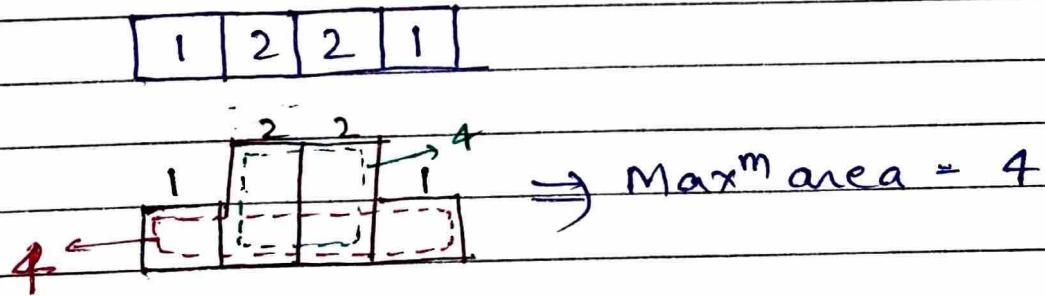


### Approach :-

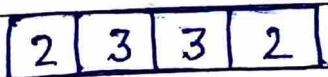
Step 1 : Take only first row as base

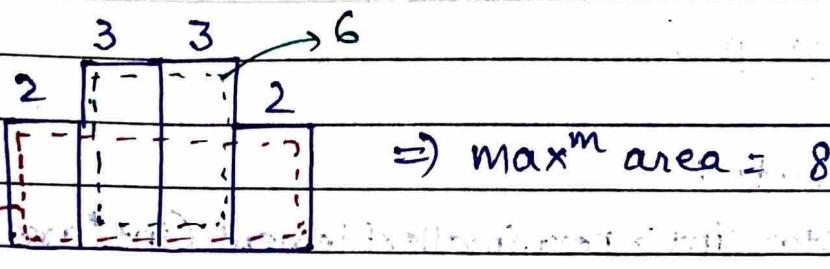


Step 2 : Take second row as base and consider all row above second row.



Step 3 : Take third row as base and consider all row above third row.





Step 4: Take 4th row as base and consider all row above it.

|   |   |   |   |
|---|---|---|---|
| 3 | 4 | 0 | 0 |
|---|---|---|---|

Yahan base hi zero hai, Toh uspe koi ~~rectangle~~ rectangle nahi bnegai.



Max<sup>m</sup> area = 6

$\Rightarrow T.C \rightarrow O(n \times m)$ ,  $S.C \rightarrow O(m)$

no. of rows    no. of col.

no. of col.

## Program :-

private:

```
vector<int> nextSmallerElement(int *arr, int n) {
```

```
    stack<int> s;
```

```
    s.push(-1);
```

```
    vector<int> ans(n);
```

```
    for (int i = n - 1; i >= 0; i--) {
```

```
        int curr = arr[i];
```

```
        while (s.top() != -1 && arr[s.top()] >= curr) {
```

```
            s.pop();
```

```
}
```

```
//ans is stack ka top
```

```
ans[i] = s.top();
```

```
s.push(i);
```

```
}
```

```
return ans;
```

```
}
```

```
vector<int> prevSmallerElement(int *arr, int n) {
```

```
    stack<int> s;
```

```
s.push(-1);
```

```
vector<int> ans(n);
```

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

```
    int curr = arr[i];
```

```
    while (s.top() != -1 && arr[s.top()] >= curr) {
```

```
        s.pop();
```

```
}
```

```
// ans is stack ka top
ans[i] = s.top(); m[Ax[i]] = i
s.push(i);
}
return ans;
}
```

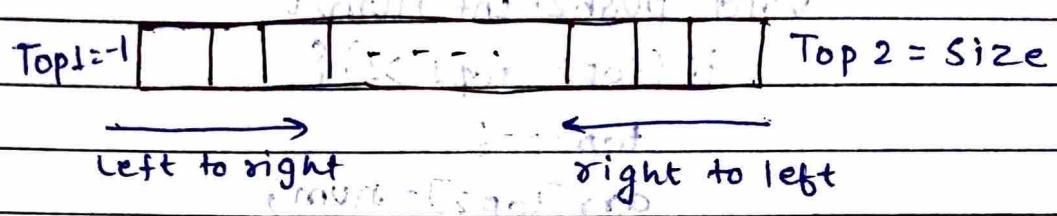
```
int largestRectangleArea (int* heights, int n) {
    vector <int> next (n);
    next = nextSmallerElement (heights, n);
    vector <int> prev (n);
    prev = prevSmallerElement (heights, n);
    int area = INT_MIN;
    for (int i=0; i<n; i++) {
        int l = heights[i];
        if (next [i] == -1) {
            next [i] = n;
        }
        int b = next [i] - prev [i] - 1;
        int newArea = l * b;
        area = max (area, newArea);
    }
    return area;
}
```

```
public:
int maxArea(int M[MAX][MAX], int n, int m) {
    // compute area for first row
    int area = largestRectangleArea(M[0], m);

    for (int i=1; i<n; i++) {
        for (int j=0; j<m; j++) {
            // row update : by adding previous row's value
            if (M[i][j] != 0)
                M[i][j] = M[i][j] + M[i-1][j];
            else
                M[i][j] = 0;
        }
        // entire row is updated now
        area = max(area, largestRectangleArea(M[i], m));
    }
    return area;
}
```

Q

## Two Stacks in an array



Program:-

```
class TwoStack {  
    int* arr;  
    int top1;  
    int top2;  
    int size;  
  
public:  
    //initialize TwoStack  
    TwoStack(int s) {  
        this->size = s;  
        top1 = -1;  
        top2 = s;  
        arr = new int[s];  
    }  
  
    //push in stack1  
    void push1(int num) {  
        //atleast an empty space present  
        if (top2 - top1 > 1) {  
            top1++;  
            arr[top1] = num;  
        }  
    }
```

<https://www.linkedin.com/in/nikhilkumar0609/>

//push in stack2

void push2(int num){

if (top2 - top1 > 1) {

top2--;

arr[top2] = num;

}

}

//pop from stack1 and return popped element

int pop1() {

if (top1 >= 0) {

int ans = arr[top1];

top1--;

return ans;

}

else {

return -1;

}

}

//pop from stack2 and return popped element

int pop2() {

if (top2 < size) {

int ans = arr[top2];

top2++;

return ans;

}

else {

return -1;

}

}

};

Fragmentation  
Memory Management  
Technique } Operating  
System.

## D) "N" stacks in an array

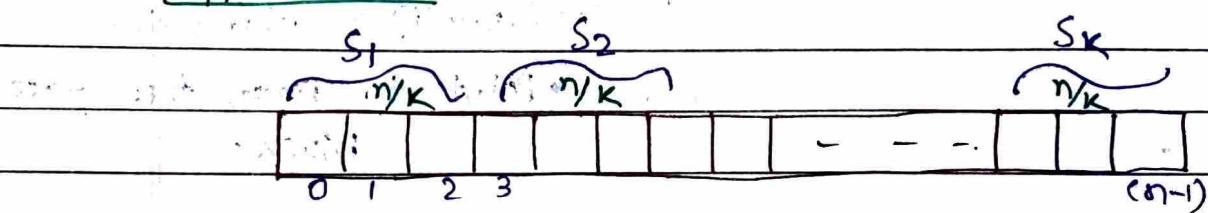
Qsn Statement :-

$\text{push}(X, M)$  :- Pushes an element X in M<sup>th</sup> stack.

Return true if element is pushed in the stack, otherwise false.

$\text{pop}(M)$  :- Pops the top element from the M<sup>th</sup> stack. Return -1 if the stack is empty, otherwise returns the popped element.

Approach 1 :-



Suppose there are 'k' stacks.

entire array length  $\rightarrow n$ .

1 part length =  $\frac{n}{k}$

Drawbacks :- Space optimized solution nhi hai.

$\Rightarrow$  Kyuki koi ek stack bhi jaaye aur usme push opr kro toh "false" btayega kyuki us stack bhi chuka hai, Joki baaki khaali bhi ho skte hai.

↓  
Stack Overflow

### Approach 2:⇒

Let No. of stacks = 3

Size of array = 6

imp → top[] → Represent index of top element  
K Size

next[] → if (arr[i] → stores an element)

- Points to next element after stack top.

→ else (arr[i] → nothing stored)

- Points to next freespace/block.

arr → [ ]  
0 1 2 3 4 5

top → [-1] -1 -1  
0 1 2

next → [1] 2 3 4 5 -1  
0 1 2 3 4 5

freespot = [0]

constructor mein jayega

→ sare top starting mein (-1) honge kyuki  
inme koi element nhi hai

→ starting mein next[] point krega next free  
space kyuki abhi nothing stored hai.

next[5] = -1, kyuki iske aage  
kuch nhi hai.

→ variable freespot starting m zero hogा  
kyuki ~~arr~~ arr empty hai toh phla free  
spot arr[0] hi hogा.

→ freespot mtlb freelist (next[]) ka starting index kyun sa hai?

1st Query → push(10,1)

Step 1: → //find index

int index = freespot;  
↓  
0

Step 2: → //update freespot

freespot = next[index];  
⇒ next[0] = 1  
freespot = 1

Step 3: → //Insert in array

arr[index] = x;

arr → 

|    |   |   |   |   |   |
|----|---|---|---|---|---|
| 10 |   |   |   |   |   |
| 0  | 1 | 2 | 3 | 4 | 5 |

Step 4: → //update next

next[index] = top[m-1]

→ next index hoga purana wala top

→ (m-1) gile kyuki zero-based index hai.

next[0] = top[0], M=1 ⇒ m-1=0

⇒ next[0] = -1

top → 

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

next → 

|     |   |   |   |   |    |
|-----|---|---|---|---|----|
| *-1 | 2 | 3 | 4 | 5 | -1 |
| 0   | 1 | 2 | 3 | 4 | 5  |

Step 5:  $\rightarrow$  // update top

$\text{top}[m-1] = \text{index};$

$\Rightarrow \text{top}[0] = 0$

|                          |   |   |    |    |  |
|--------------------------|---|---|----|----|--|
| $\text{top} \rightarrow$ | X | 0 | -1 | -1 |  |
|                          | 0 | 1 | 2  |    |  |

Query 2  $\rightarrow$  push (20, 1)

Step 1:  $\rightarrow$  index = freespot = 1

Step 2:  $\rightarrow$  freespot = next[index] = next[1]

freespot = 2

Step 3:  $\rightarrow$  arr[index] = X;

arr[1] = 20;

|                   |    |    |   |   |   |   |  |
|-------------------|----|----|---|---|---|---|--|
| arr $\rightarrow$ | 10 | 20 |   |   |   |   |  |
|                   | 0  | 1  | 2 | 3 | 4 | 5 |  |

Step 4:  $\rightarrow$  next[index] = top[m-1];

next[1] = top[0],  $m=1 \Rightarrow m-1=0$

|                   |   |    |    |  |
|-------------------|---|----|----|--|
| top $\rightarrow$ | 0 | -1 | -1 |  |
|                   | 0 | 1  | 2  |  |

|                    |    |   |   |   |   |   |    |
|--------------------|----|---|---|---|---|---|----|
| next $\rightarrow$ | -1 | X | 0 | 3 | 4 | 5 | -1 |
|                    | 0  | 1 | 2 | 3 | 4 | 5 |    |

Step 5:  $\rightarrow$  top[m-1] = 'index';

$\Rightarrow \text{top}[0] = 1$

|                   |   |   |    |    |  |
|-------------------|---|---|----|----|--|
| top $\rightarrow$ | X | 1 | -1 | -1 |  |
|                   | 0 | 1 | 2  |    |  |

Program :-

```
class NStack {  
    int* arr;  
    int* top;  
    int* next;  
    int n, s;  
    int freespot;  
  
public:  
    // Initialize your data structure.  
    NStack(int N, int S) {  
        n = N;  
        s = S;  
        arr = new int[s];  
        top = new int[n];  
        next = new int[s];  
  
        // top initialize  
        for (int i=0; i<n; i++) {  
            top[i] = -1; // initial value  
        }  
  
        // next initialise  
        for (int i=0; i<s; i++) {  
            next[i] = i+1; // initial value  
        }  
  
        next[s-1] = -1; // update last index value to -1  
        // initialize freespot  
        freespot = 0;  
    }
```

```

    bool push(int x, int m) {
        // check overflow condn
        if (freespot == -1)
            return false;
        int index = freespot; // step 1
        freespot = next[index]; // step 2
        arr[index] = x; // step 3
        next[index] = top[m-1]; // step 4
        top[m-1] = index; // step 5
        return true;
    }

```

```

int pop(int m) {
    // check underflow condn
    if (top[m-1] == -1)
        return -1;
    int index = top[m-1];
    top[m-1] = next[index];
    next[index] = freespot;
    freespot = index;
    return arr[index];
}

```

$$\begin{aligned}
 T.C &\rightarrow O(1), S.C \rightarrow O(s+n+s) = O(2s+n) \\
 &= O(s+n).
 \end{aligned}$$

Q Design a stack that supports `getMin()` in  $O(1)$  time and  $O(1)$  extra space.

Approach 1 :-

I/P = [ 5, 3, 8, 2, 4 ]  
curr -

|   |   |   |
|---|---|---|
| 4 | → | 2 |
| 2 | → | 2 |
| 8 | → | 3 |
| 3 | → | 3 |
| 5 | → | 5 |

int mini = INT\_MAX;  
mini = min(mini, curr);

2nd stack

push()

- (i) I/P → Normal push
- (ii)  $\min^m$  value in 2nd stack.

pop() → 2nd stack se pop krdo.

getMin() → return 2nd stack top.

T.C  $\rightarrow O(1)$

S.C  $\rightarrow O(N) \rightarrow$  because of extra stack.

## Approach 2:-

→ Push Operation

[5, 3, 8, 2, 4]

|                    |   |
|--------------------|---|
| 2                  |   |
| X                  |   |
| X                  | 1 |
| <del>INT-MAX</del> | 8 |
| mini               | 1 |

$$\text{mini} = \min(\text{mini}, \text{curr})$$

Step 1: → check overflow cond<sup>n</sup>

Step 2: → for 1st element

→ Normal Push

→ update mini

•  $3 < 5 \rightarrow T$

$$\text{val} = 2 * 3 - 5 = 1, \text{mini} = 3$$

for baaki element

→ if ( $\text{curr} < \text{mini}$ )

$$\text{val} = 2 * \text{curr} - \text{mini};$$

push(val);

update mini

•  $8 < 3 \rightarrow F$

•  $2 < 3 \rightarrow T$

$$\text{val} = 2 * 2 - 3 = 1$$

$$\text{mini} = 2$$

→ Else

normal push

→ Pop Operation

|                                 |   |   |
|---------------------------------|---|---|
|                                 |   | curr = s.top()                                |
|                                 |   | mini.   |
| X                               | 2 | Step 1: → check Underflow cond <sup>n</sup>   |
| 1                               |   | Step 2: → if ( $\text{curr} > \text{mini}$ )  |
| 8                               |   | Normal pop()                                  |
|                                 |   | → else  |
| $\text{val} = 2 * 2 - 1$<br>= 3 |   | $\text{val} = 2 * \text{mini} - \text{curr};$ |
| 1                               |   | $\text{mini} = \text{val};$                   |
| 5                               |   | Pop()   |

→ getMinimum()

return mini;

<https://www.linkedin.com/in/nikhilkumar0609/>

$\text{push} \rightarrow \text{val} = 2 * \text{curr\_mini}$  } To access previous  
 $\text{pop}() \rightarrow \text{val} = 2 * \text{mini} - \text{curr}$  } min<sup>m</sup> using current minimum.

Program :-

```
#include <stack>
#include <limits.h>
class SpecialStack {
    // Define the data members.
    stack<int> s;
    int mini = INT_MAX;

public:
    void push(int data) {
        // for first element
        if (s.empty()) {
            s.push(data);
            mini = data;
        }
        else {
            if (data < mini) {
                s.push(2 * data - mini);
                mini = data;
            }
            else
                s.push(data);
        }
    }
}
```

```
int pop() {
    if (s.empty())
        return -1;
    }

    int curr = s.top();
    s.pop();
    if (curr > mini)
        return curr;
    }

    else {
        int prevMin = mini;
        int val = 2 * mini - curr;
        mini = val;
        return prevMin;
    }

}
```

```
int top() {
    if (s.empty())
        return -1;
    }

    int curr = s.top();
    if (curr < mini)
        return mini;
    }

    else {
        return curr;
    }

}
```

<https://www.linkedin.com/in/nikhilkumar0609/>

```
bool isEmpty () {  
    return s.empty ();  
}
```

```
int getMin () {  
    if (s.empty ()) {  
        return -1;  
    }  
    return mini;  
};
```