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;
}
```

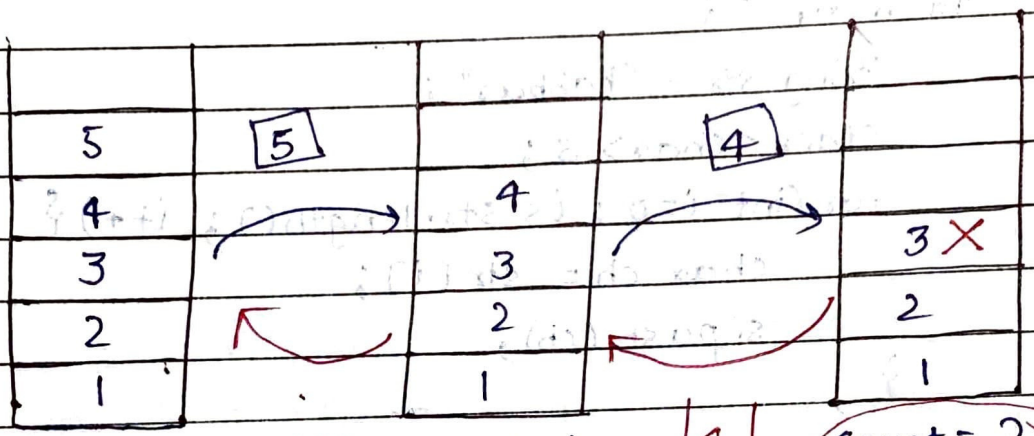
T.C → O(n) , S.C → O(n)

Q Delete middle element from stack

i/p :→ [1, 2, 3, 4, 5] ⇒ o/p = [1, 2, 4, 5]

i/p :→ [5, 6, 7, 8] ⇒ o/p = [5, 7, 8]

Approach :⇒



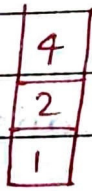
Count = 0

Size = 5

$\frac{Size}{2} = 2$

Count = 1

$\frac{Size}{2} = 2$



Count = 2

$\frac{Size}{2} = 2$

⇓

Base case

Program :→

```
void solve(stack<int>& inputStack, int count, int size) {

    //base case
    if(count == size/2) {
        inputStack.pop();
        return;
    }

    int num = inputStack.top();
    inputStack.pop();

    //Recursive call
    solve(inputStack, count+1, size);
    inputStack.push(num);
}

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

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

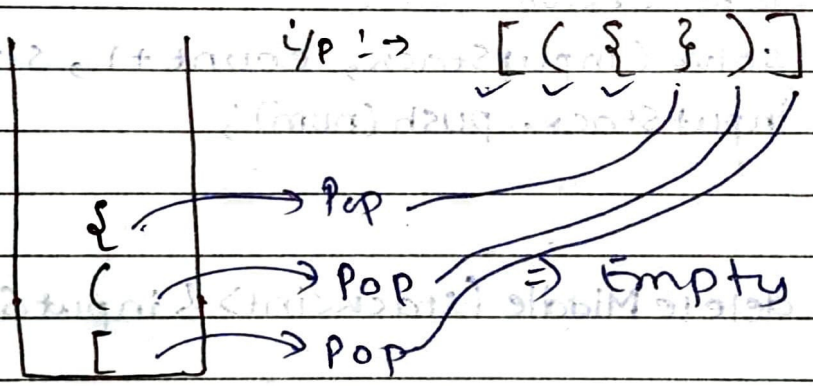
Q Valid Parenthesis

Approach :-

i/p :→ [({})]

Agar koi open bracket aaye toh usko stack mein push kro aur Jb koi closed bracket aaye toh dekho uske corresp-onding koi open bracket hai stack mein, aur agar ho toh pop kr do.

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



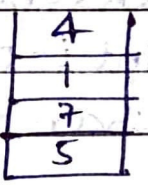
Program :->

```
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;
}
```

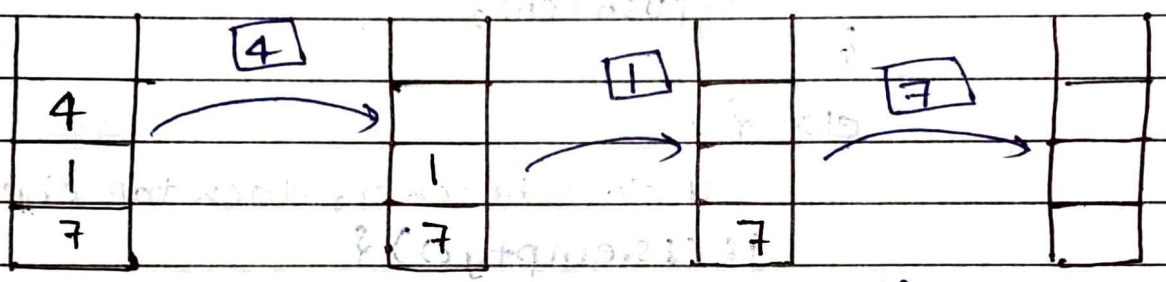
Insert an element at its bottom in a given stack.

i/p ⇒ [7, 1, 4], X = 5

o/p ⇒ [5, 7, 1, 4]



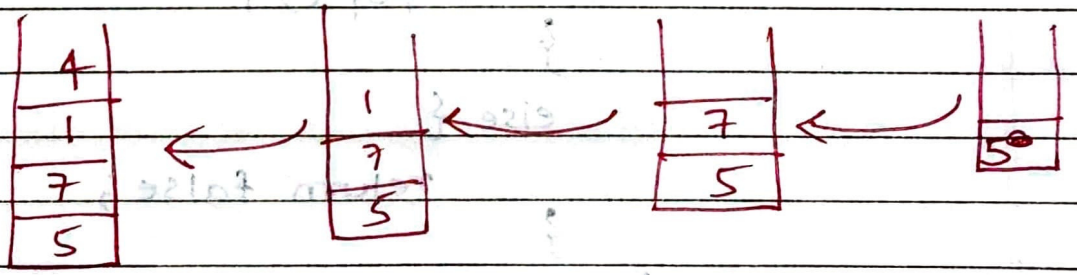
Approach :-



S.empty() ⇓ False

S.empty() ⇓ False

S.empty ⇓ True ⇓ S.push(x)



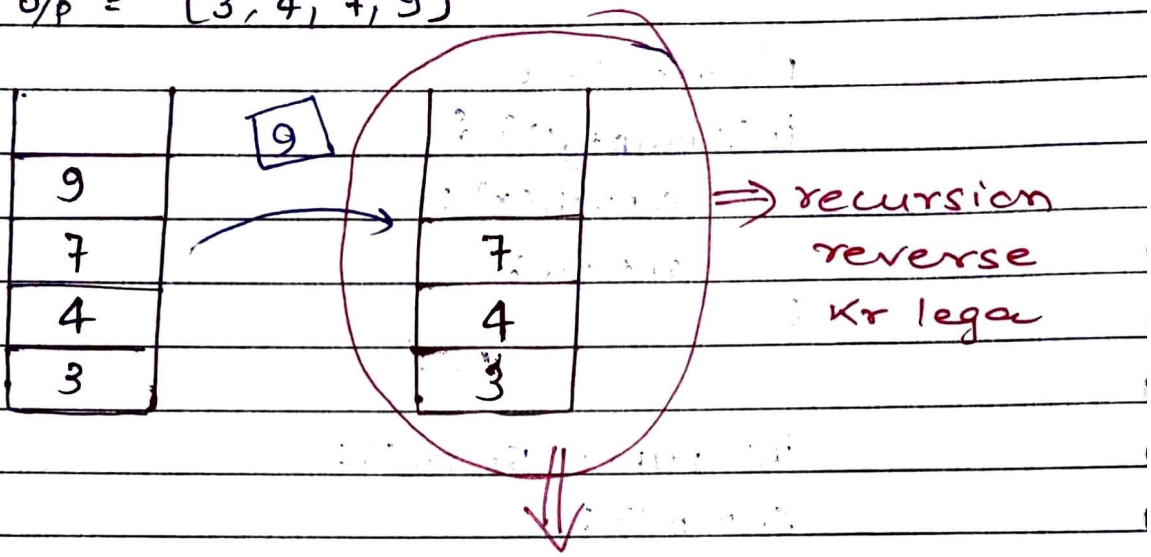
Program :->

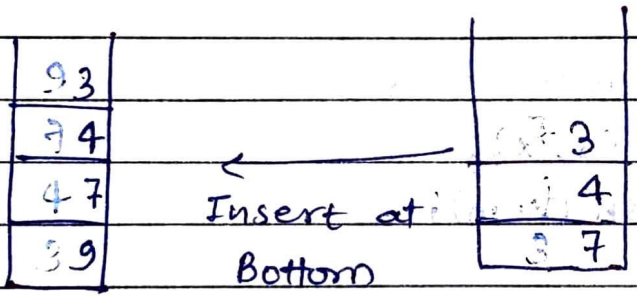
```
void solve (stack<int> &s, int x) {

    // 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]





Algo :→

→ Stack Top ko side mein rkh lo.

→ Use recursion to reverse remaining stack.

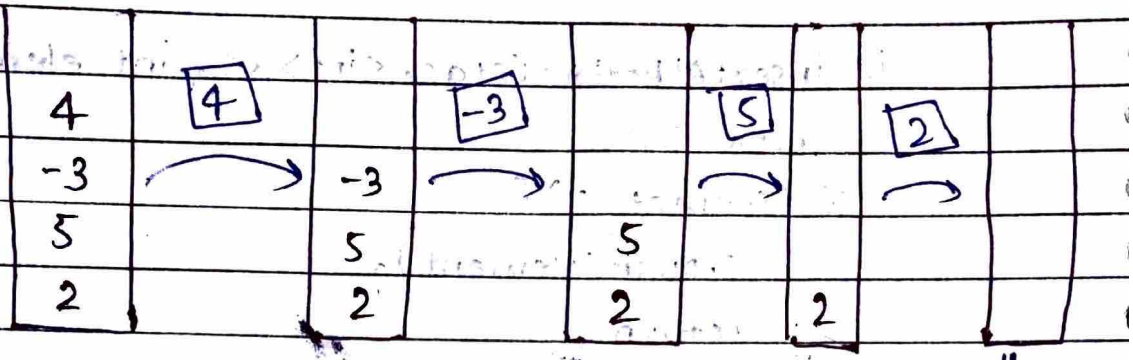
→ Wapas aate hue, insertAt Bottom Kr dena stackTop ko.

T.C → $O(n^2)$

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);
}
```

Q Sort a stack

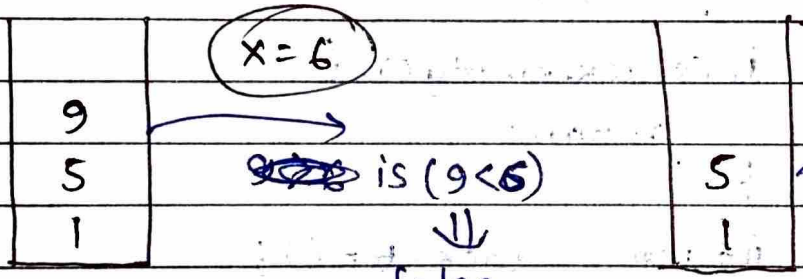


s.empty()

True

Wapas jaate time Sorted way mein Insert krna hai

Sorted Insert



6 insert kr dnge aur wapas jaate time 9 ko insert kr lnge.

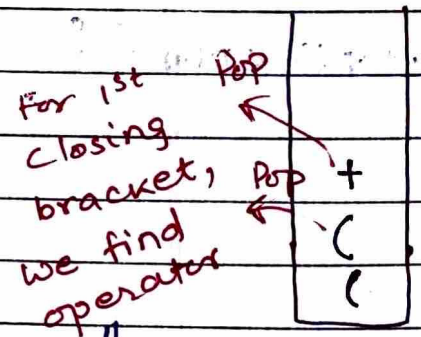
T.C → $O(n^2)$

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);
}
```

Redundant Brackets

((a+b))



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

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

Found → Not Redundant Pair

Not Found → Redundant Pair

')' → is bair koi operator nhi 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;
}
```

Minimum cost to make string valid

(Coding Ninzas → Codestudio)

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) }}}}}} - - -

(iii) - - }}}{{{ - - -

Step 3 :⇒ count a & b :

a → count of close braces, b → count of open braces.

(i) {{{{ ⇒ b=4, a=0 (even)
{{ {{ = (2) → b/2

(ii) }}}} ⇒ a=4, b=0 (even)
{{ {{ = (2) → a/2

(iii)
}}} {{{ ⇒ a=3, b=3 (odd, odd)
{ } { } { } = 4

}}}} {{{{ ⇒ a=4, b=4 (even, even)
{ } { } {{ }{ } = 4

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

* Step 3 : (i) & (ii) main sirf ek check kr rha kyuki odd even case mein invalid ho jayga.

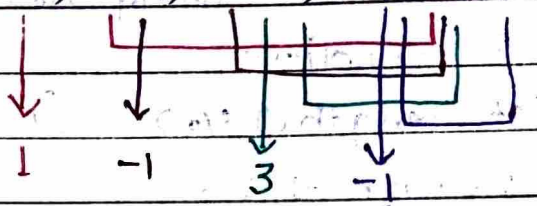
~~Step (ii)~~

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;
}
```

Next Smaller Element

i/p :→ [2, 1, 4, 3]



o/p = {1, -1, 3, -1}

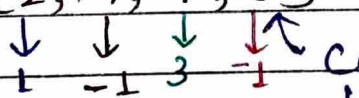
Approach 1 :⇒ 2 → {1, 4, 3} → (n-1) comparison
1 → {4, 3} → (n-2) "
4 → {3} → (n-3) "
3 → { } → ! (n-4) = 0 "

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

⇒ 2 for loop chlega, Ek main element ke liye, aur dusra, uske aage ke element se compare krne ke liye.

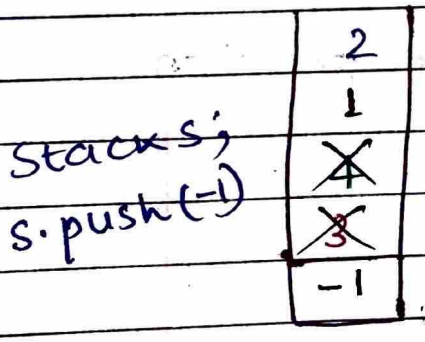
Approach 2 :⇒

i/p :→ [2, 1, 4, 3]



chhota element lao

s.top()



chota
↓
ans store
s.push(curr)

Bada
↓
Chota dhundh ke lao
while (s.top() >= curr)
{ s.pop();
}
s.top() → Chota
ans = s.top()
s.push(curr)

T.C → O(n)

Program :-

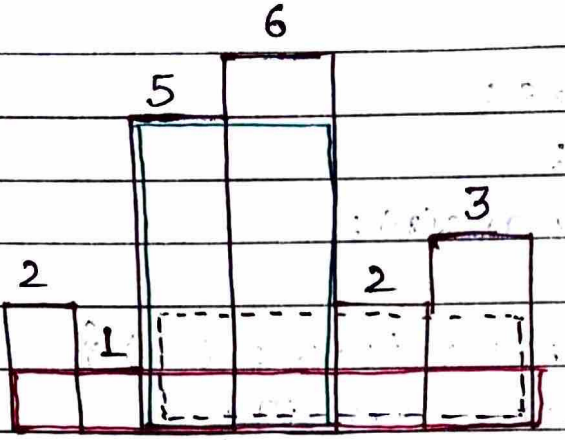
```
#include <stack>
vector<int> nextSmallerElement(vector<int>& arr, int n)
{
    stack<int> s;
    s.push(-1);
    vector<int> ans(n);

    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 1 :→

```
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;
```

Largest Rectangle in Histogram

Approach 1 :→ Brute-force Approach



Area = length * breadth

↓ Const.

↓ max^m

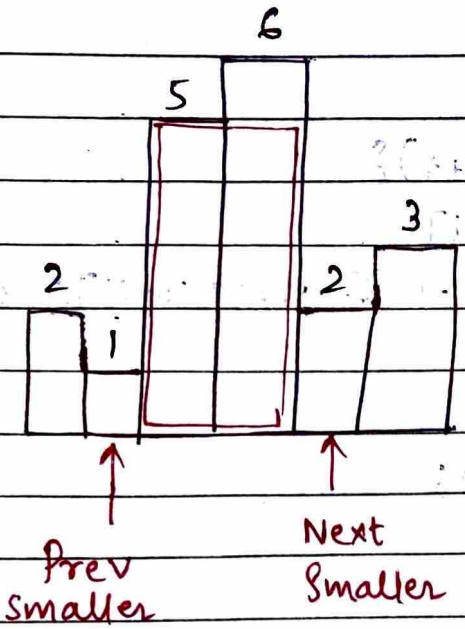
↓ max^m

for 2 → we can't shift either in left/right.
So, max^m area = 1×2 = 2

for 1 → 1×6 = 6
for 5 → 2×5 = 10
for 6 → can't shift in either dir^n
So, 1×6 = 6
for 2 → 4×2 = 8
for 3 → 1×3 = 3

T.C → $O(n^2)$

Approach 2 :→



Like, for 5

prevSmaller index = 1

nextSmaller index = 4

width = next - prev - 1

= 4 - 1 - 1

= 2

length = 5

Area = 5 x 2 = 10.

S.C → O(n), T.C → 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;
}
```

The Celebrity Problem

Q" :→ Celebrity doesn't know anyone.
Everyone knows celebrity.

Person	0	1	2
0	0	1	0
1	0	0	0
2	0	1	0

→ Ignore Diagonal
→ '1' knows No-one.
Everyone knows '1'
⇓
'1' is a celebrity.

Approach 1 :→ Brute-Force

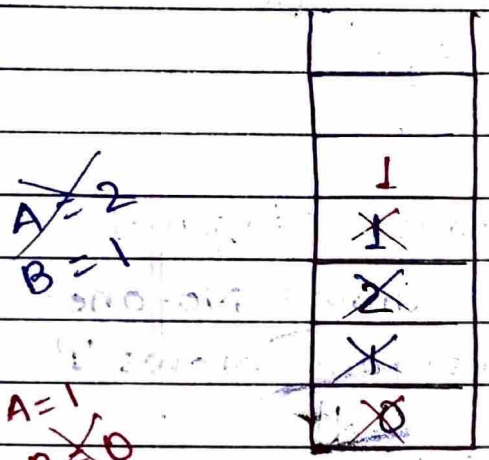
⇒ Celebrity → rows → all 0's.

⇒ Celebrity → Col → all 1's except diagonal element.

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

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

Approach 2 :-



→ Put all element inside stack (0,1,2)

→ Jab tak stack size != 1

→ A → S.top() → S.pop()

→ B → S.top() → S.pop()

→ if (A knows B)

A X , B → wapas push Kro

→ if (B knows A)

B X , A → wapas push Kro

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

⇓

Verify

→ Celebrity knows Noone

⇓

row check → all 0's

→ Everyone knows Celebrity

⇓

Col check → all 1's except diagonal element.

T.C → O(n)

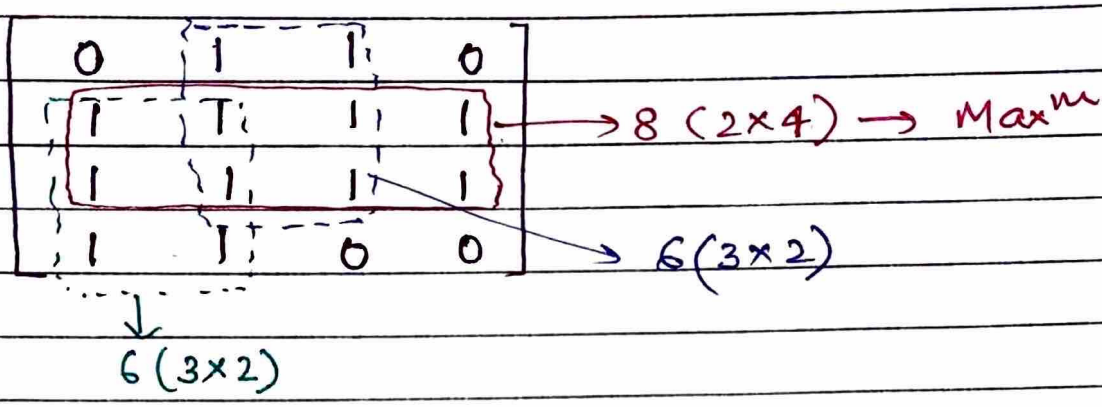
Program :-

```
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);
    }
```

```
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 -1;

return ans;
}
};
```

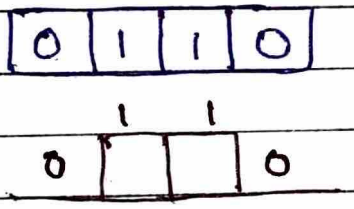
Leetcode 85.

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



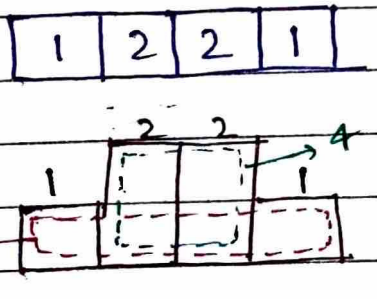
Approach :-

Step 1 :→ Take only first row as base



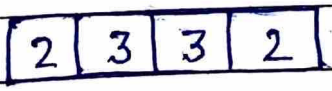
⇒ Max^m area = 2

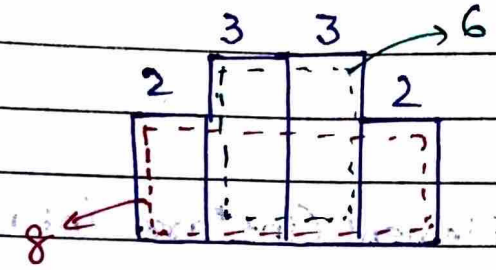
Step 2 :→ Take second row as base and consider all row above second row.



⇒ Max^m area = 4

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



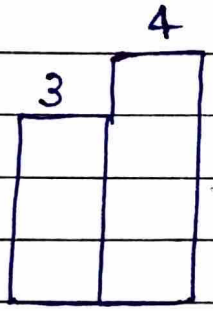


⇒ Max^m area = 8

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 ~~area~~ rectangle nhi bnega.



⇒ Max^m area = 6

⇒ T.C → $O(n \times m)$, S.C → $O(m)$

n → no. of row, m → no. of col. (for T.C); m → no. of col. (for S.C)

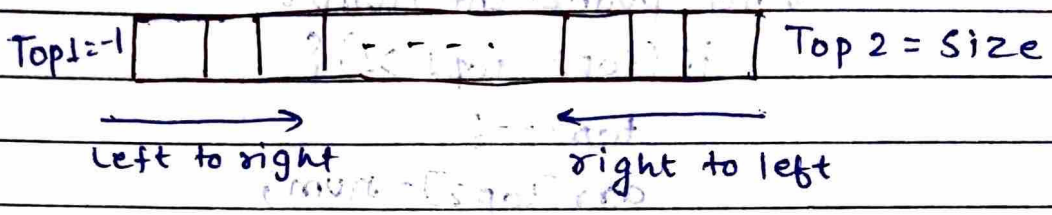
Program :-

```
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();
        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;
}
```

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 stack 1.
        void push1(int num) {
            //atleast an empty space present
            if (top2 - top1 > 1) {
                top1++;
                arr[top1] = num;
            }
        }
```

```
//push in stack 2:
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.

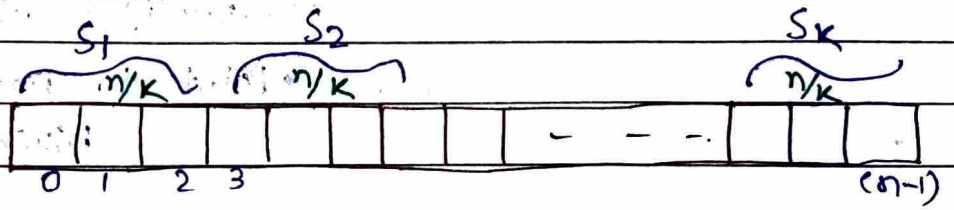
"N" stacks in an array

Qstn Statement :⇒

push(X, M) :- Pushes an element X in Mth stack. Return true if element is pushed in the stack, otherwise false.

pop(M) :- Pops the top element from the Mth stack. Return -1 if the stack is empty, otherwise returns the popped element.

Approach 1 :⇒



Suppose there are 'k' stacks.

entire array length → n.

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

Drawbacks :⇒ Space optimized soln nhi hai.

→ Kyuki koi ek stack bhr jaaye aur usme push opr^n kre toh "false" btayega kyuki us stack bhr chuka hai, Jbki baaki khaali bhi ho skte hai.

⇓

Stack Overflow

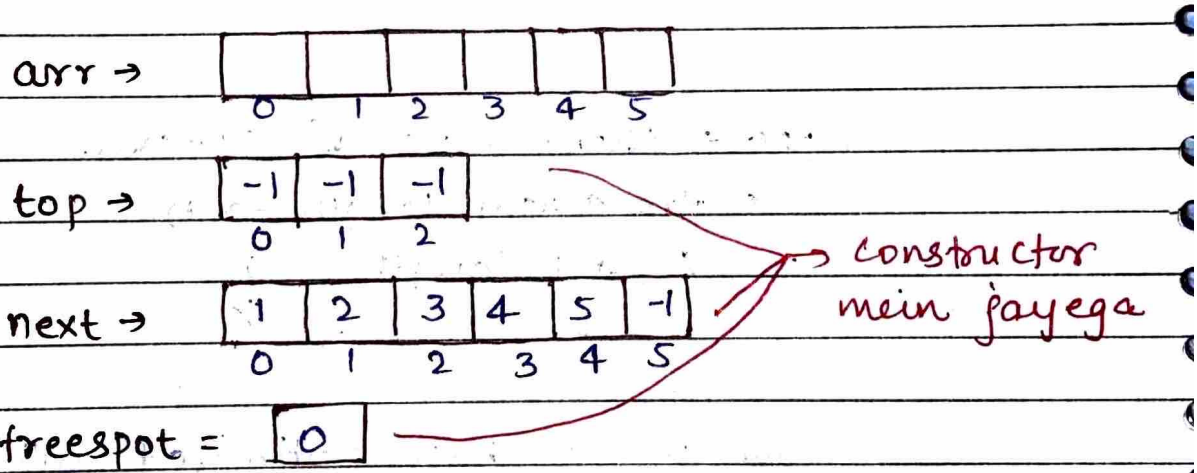
Approach 2:→

Let No. of stacks = 3
Size of array = 6

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

next[] → If (arr[i] → stores an element)
- Points to next element after stack top.

→ else (arr[i] → nothing stored)
- Points to next free space/ block.



→ Saare 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 hoga kyuki ~~arrr~~ arr empty hai toh phla free spot arr[0] hi hoga.

→ freespot mtlb freelist (next[]) ka starting index kaun sa hai.

1st Query → push (10,1)
x M

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) isliye 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 :→ // update top

top [m-1] = index ;

⇒ top [0] = 0

top →

0	-1	-1
0	1	2

Query 2 → push (20, 1)

Step 1 :→ index = freespot = 1

Step 2 :→ freespot = next [index] = next [1]

freespot = 2

Step 3 :→ arr [index] = X ;

arr [1] = 20 ;

arr →

10	20				
0	1	2	3	4	5

Step 4 :→ next [index] = top [m-1] ;

next [1] = top [0] , m = 1 ⇒ m-1 = 0

top →

0	-1	-1
0	1	2

next →

-1	0	3	4	5	-1
0	1	2	3	4	5

Step 5 :→ top [m-1] = index ;

⇒ top [0] = 1

top →

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;
        }
        // next initialise
        for(int i=0; i<s; i++){
            next[i] = i+1;
        }
        next[s-1] = -1; // update last index value to -1.
        // initialize freespot
        freespot = 0;
    }
```

```
bool push(int x, int m){
    // check overflow cond^n
    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 cond^n
    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];
}
};
```

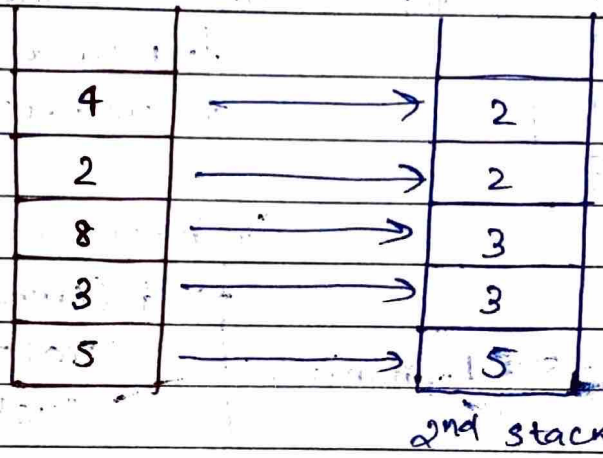
T.C → O(1), S.C → O(S+n+S) = O(2S+n)
= O(S+n).

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



int mini = INT_MAX;
mini = min(mini, curr);

push()
- (i) i/p → Normal push
- (ii) min^m value in 2nd stack.

pop() ⟶ Dono stack se pop krdo.

getMin() ⟶ return 2nd stack top.

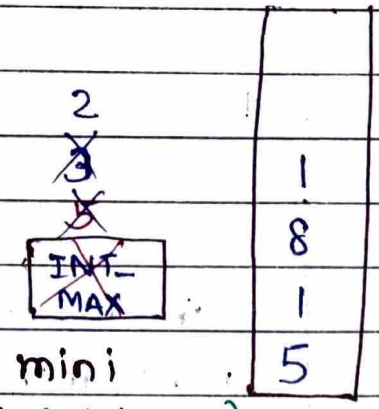
T.C → O(1)

S.C → O(N) → because of extra stack.

Approach 2 :-

→ Push Operation

[5, 3, 8, 2, 4]



Step 1 :→ check overflow cond^n

Step 2 :→ for 1st element

→ Normal Push

→ update mini

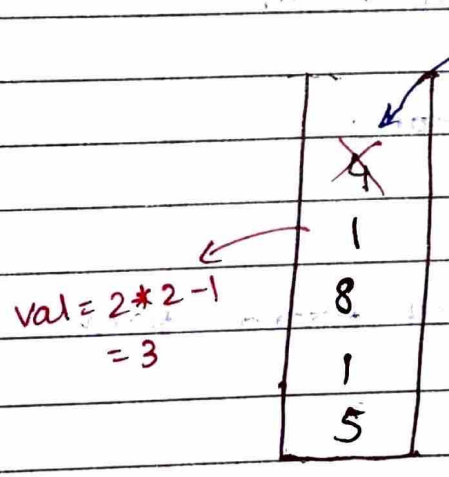
mini = min(mini, curr)

for baaki element

- 3 < 5 → T
  val = 2 * 3 - 5 = 1, mini = 3
- 8 < 3 → F
- 2 < 3 → T
  val = 2 * 2 - 3 = 1
  mini = 2

→ if (curr < mini)
  val = 2 * curr - mini;
  Push (val);
  update mini
→ else
  normal push

→ Pop Operation



Step 1 :→ check Underflow cond^n

Step 2 :→ if (curr > mini)
  Normal pop()
→ else
  val = 2 * mini - curr;
  mini = val;
  Pop ()

→ getMinimum()
  Return mini;

push → val = 2* curr - mini
pop() → val = 2*mini - curr
} To access previous minm 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;
    }
}
```

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

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