श्री

DATA STRUCTURES AND ALGORITHMS

INDEX

NAME. : Smit Sekhadia Std. : Div. :

Roll No. : School / College. :

Sr. No	TITLE	Date	Teacher's Sign / Remark
①	LINKED LIST.		
②	STACKS		

Stacks

DSA 450.

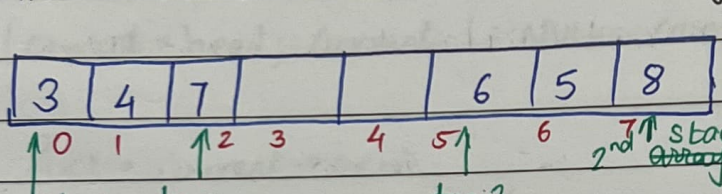
2] Implement 2 stack in an array.

Procedure:

→ Start both stacks from each extreme end of the array. So top1 will be on first index i.e a[0] & top2 will be at a[n-1].

→ Now push & pop elements as you like till top1 is not equal to top2.

→ if the become full our array is completely filled.



if top1 = top2 when we push in either of the stack, then our array will be completely filled.

TIME: O(1)

SPACE: O(1)

Code:

```
void push1 (int x)
{   if (top1 != top2)
    {   top1++;
        arr[top1] = x;
    }
}

void push2 (int x)
{   if (top1 != top2)
    {   top2--;
        arr[top2] = x;
    }
}
```

```
int pop1 (int)
{   if (top1 >= 0)
    {   return arr[top1--];
    }
    else { return -1};
}

int pop2 ()
{   if (top2 <= size-1)
    {   return arr[top2++];
    }
    else { return -1;
}
```

DSA 45°

Q. 3] Find middle element of a stack. ∴ We use DLL.

→ Which data structure will you use ? ⇒ Doubly Linked List.

Operations : push(), pop(), find_middle(), delete_middle().

Procedure :

→ When you push element make its prev NULL, its next as head, heads prev as the element pushed and bring head on the element pushed.

→ if no of nodes are even make mid to go on its prev

→ if no of nodes are odd return mid.

→ When you pop element ; make a temp node as head ^ heads prev NULL ; decrease size by one,

→ if no of nodes are even return mid.

→ if no of nodes are odd take mid to mid's next & return mid.

PUSH()

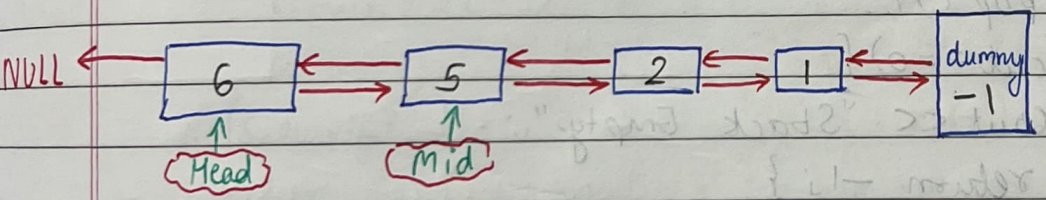
count is odd → No operation

count is even → mid = mid → prev

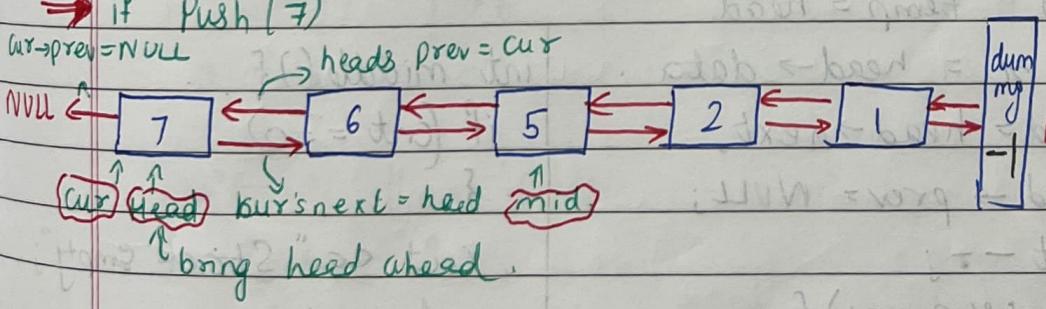
POP()

count is odd → mid = mid → next.

count is even → No operation



⇒ if Push(7)

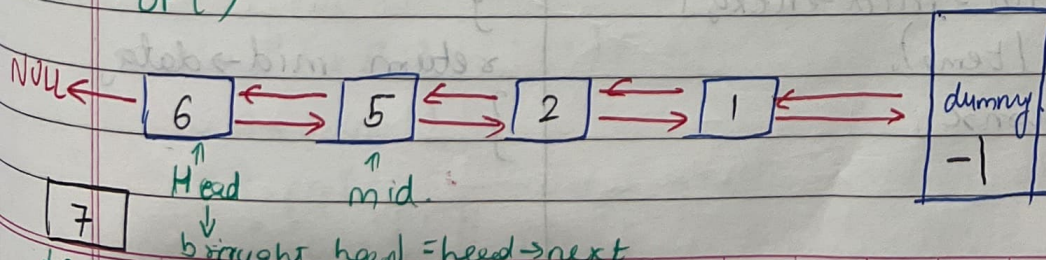


Count = 5 (odd)

Odd in push

↳ No operation on mid

Pop()



Count = 4 (EVEN)

Even in Pop

↳ No operation on mid.

Page No.
Date

Code:

```
Node* dummy = new Node(-1);
Node* head, mid = dummy;
int cnt = 0;

void push(int data){
Node* cur = new Node(data);
cur->prev = NULL;
cur->next = head;
cnt++;
head->prev = cur;
head = cur;
if (cnt == 1) mid = cur
else if (cnt % 2 == 0){
    mid = mid->prev;
  }
}
int
void pop(){
  if(cnt == 0){
    cout << "Stack Empty";
    return -1; }
  Node* temp = head
  int x = head->data.
  head = head->next;
if(head != Null) head->prev = NULL;
  cnt--;
  if (cnt % 2 == 1){
    mid = mid->next; }
  free(temp);
  return x;
}
```

Time: O(1) ↔ of push(), pop(), middle()

```
int middle(){
 if (cnt == 0)
 {
 cout << "Stack empty";
 return -1;
 }
 return mid->data;
}
```

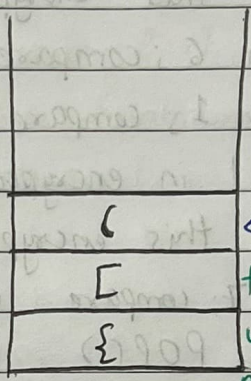
Page No.
Date

Qs] Balanced Parenthesis

Procedure:

→ Whenever a Paranthesis is found push in stack.
→ if closing Paranthesis is found check if the top of stack has similar opening bracked.
→ if it is similar pop that element from stack and move ahead
→ if it is different return false.

{ [(a + b)] - [(c + d)] }



Code:

```
bool ispar (string x){
  stack <char> s;
  char a;
  for (int i=0; i<x.length(); i++){
    if (x[i]=='(' || x[i]=='{' || x[i]=='['){
      s.push (x[i]);
      continue;
    if (s.empty()){
      return false;
    if (x[i]==')'){
      a=s.top();
      s.pop();
      if(a=='{' || a=='['){
        return false;
      } break;
    }
    else if (x[i]=='}'){
      a=s.top();
      s.pop();
      if (a=='(' || a=='[')
        { return false;
      } break;
    else if (x[i]==']'){
      a=s.top();
      s.pop();
      if(a=='(' || a=='{'){
        return false;
        break
      }
    }
    return (s.empty());
```

Time: O(n) as iterating string once
Space: O(n) as stack < > is used.

Page No.
Date

***. Q7] Design Stack that supports getMin() in O(1) Time and O(1) Space.
DSA 450.

Procedure:

→ Elements: {5, 8, 4, 6, 1, 7}

→ Push 5 and update min = 5;

→ Push 8, compare with 5; it is greater so dont update 5

→ Next 4, compare with 5, it is SMALLER; min = 4 (is smaller than 4)
Store 4 in encrypted format as $2\overset{4}{x} - \overset{5}{min} = 3$
Push this encrypted value 3 in stack & update min = 4.

→ Push 6; compare with 4; it is greater.

→ Next 1; compare with 4; it is SMALLER.; min = ~~5~~ 1
Push 1 in encrypted format as $2\overset{1}{x} - \overset{4}{min} = -2$.
Push this encrypted value -2 in stack.

→ Push 7; compare with 1; it is greater.

→ Now POP().

→ Pop 7; it is greater than min = 1

→ next in stack is -2; We know that this value is encrypted as it is smaller than min.
Decrypt it by $2 \times \overset{1}{min} - \overset{-2}{V} = \overset{min}{4}$ (V is topmost element) So when we pop -2 that is encrypted value of 1 we get minimum as 4

→ Pop 6; it is greater than min = 4.

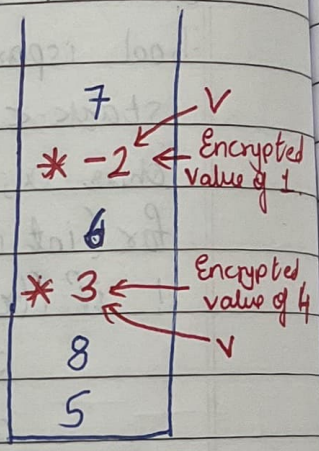
→ next in stack is 3 This value is encrypted as it is smaller than min.

∴ How to know wether value is encrypted
↳ if it is smaller than minimum.
Decrypt it by $2 \times \overset{4}{min} - \overset{3}{V} = 5 = min$

→ Pop 8

→ Pop 5

→ stack empty.



To encrypt a value	To Decrypt the value
$2 \times x - min$ ↳ value to be pushed	$2 \times min - V$ ↳ top most element
Check with above examples.	

Page No.
Date

Code:

```
int min = 99999;
void push(stack<int>& s, int a){
  if (s.empty()){
    min = a;
    s.push(a);
  }
  else{
    int x = a
    if (a < min){
      x = 2*a - min;
      min = a;
    }
    S.push(x);
  }
}
int pop(stack<int>& s){
  int v = s.top()
  if(v >= min) s.pop();
  else{
    int y = 2*min - v;
    s.pop();
    min = y;
  }
}
int getMin(stack<int>& s){
  return min;
}
```

Pushing first element in stack.

Encrypting value if it is smaller than minimum & updating min

Time: O(1)
Space: O(1)

decrypting value if y is smaller than minimum & updating min

DSA-450 Q9] The Celebrity Problem

Method 1 [Time: $O(n^2)$ Space: $O(n)$]

→ if a[i][j] = 1 then i → j ko janta hai

→ Make 2 arrays in[ ] and out[ ].

in[i] → Kitne log i ko jante hai

out[i] → i kitne log ko janta hai.

if everyone knows i and i knows no one then i is celebrity.

Code for Method 1.

```
int celebrity (vector<vector<int>>& a, int n) {
    int in[n] = {0};
    int out[n] = {0};
    for (int i = 0; i<n; i++) {
        for (int j = 0; j<n; j++) {
            if (a[i][j] == 1){
                in[j]++;
                out[i]++;
            }
        }
    }
    for (int i = 0; i<n; i++) {
        if ( int i = in[i] == n-1 && out[i] == 0) {
            return i;
        }
    }
    return -1;
}
```

Time: $O(n^2)$ → 2 nested for loop
Space: $O(n)$ ↓ 2 n size array created in & out.

i → j ko janta hai
So in[j]++ & out[i]++.

Sab i ko jante hai

i kisi ko nahi janta
∴ Celebrity.

Method 2 [Time: $O(n)$ Space: $O(1)$]

→ Take celebrity c as 0^{th} person.

→ Check if 0^{th} person knows ~~somebody~~ 1st person if yes make c = 1 & check if 1 knows 2^{nd} person & so on

∴ (for if (a[c][i] == 1) → agar c, next vale ko janta hai { c = i; }) ~~if~~ it cant be celeb so update c to next & check again

Page No.
Date

Now you got a c who dosent know ~~anyone~~ the next person.
Now check if that c knows ~~eve~~ anyone or someone exist who dosent know c

Code for Method 2.

```
int c = 0;
for (int i = 1; i < n; i++) {
    if (a[c][i] == 1) {
        c = i;
    }
}
for (int i = 0; i < n; i++) {        // dont compare celeb with himself.
    if (i != c && (a[c][i] == 1 || a[i][c] == 0)) {
        return -1;
    }
else return c;
}
}
```

Time: O(n)
Space: O(1)

if c knows someone he is not celeb so return -1

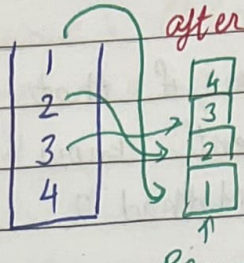
if someone exist who dosent know c, E is not celeb so return -1.

DSA 450

Q13] Reverse Stack using Recursion. [See video of Code Library to refer]

Procedure:

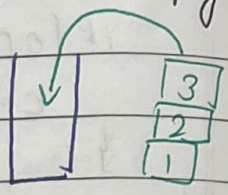
→ Pop all the elements and store them in recursion stack. till our stack becomes empty.



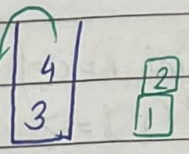
→ Now insert 4 again in stack. only if stack is empty.

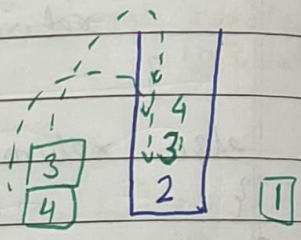
→ Pop 4 and store it in diff recursion stack.

→ insert 3 in stack and then insert 4 again after 3 ∴ 3 & 4 are reversed.



→ Similarly you wants to insert 2 so check stack is empty or not We can see stack is not empty So pop 4 and store in diff recursion stack. Now still it is not empty so pop 3 and store in ~~diff~~ recursion stack





→ Insert 3 and 4 again after 2 is inserted.

→ Repeat same procedure for 1.

Time: $O(n^2)$

Code:

```
void insertatbottom(char x)
{
  if(st.size()==0){
    st.push(x);
  }else{
    char a = st.top();
    st.pop();
    insertatbottom(x);
    st.push(a);
  }
}
```

DSA 450.

Sort a Stack using Recursion

if(s.size==0 || s.top()<x)

only this is the change in Code.

```
void reverse(char x)
{
  if(st.size>0){
    int x = st.top()
    st.pop()
    reverse();
    insertatbottom(x);
  }
}
```

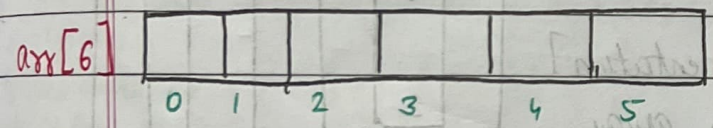
will be called after all elements of stack are stored in backstack.

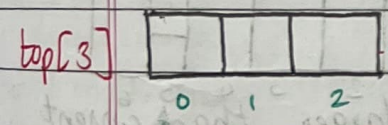
Page No.
Date

2.4] Implement 'N' stacks in Array. (Hard)
Refer youtube video of code Library OP Sum!.
or Love Babbar

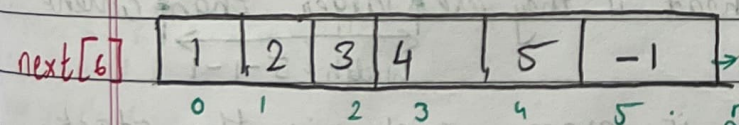
Approach-1 [Brute Force].
→ Create ~~Vector~~ array of n size and if you have k stacks divide array in n/k parts ~~and~~ for each stack.
Drawback: If you want to push element & that part of stack in array is full it will not be pushed & stackoverflow.

Approach 2. [Stacks].
→ Create 3 arrays: arr[n], top[no of stacks], next[n].
arr[n] → n sized array to store elements of n stacks.
top[k] → k = no of stacks; stores the index of topmost element of k^{th} stack.
next[n] → if corresponding index in arr[] is empty; it stors the next available free space (index) for element to be insert
→ if corresponding index is occupied; it stores the index of the element prev to the top element of k^{th} stack.





→ Stores index of topmost element of S0, S1 & S2



→ as stack is empty stores the next available free space's index.

To understand dry run; see video of code library.

DSA 450

[Q16] Maximum Area in a Histogram.

Method-I [Brute Force]

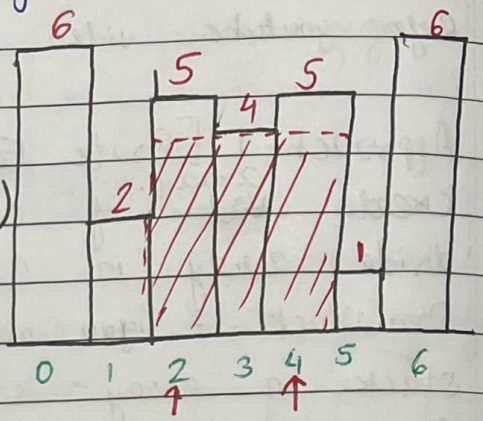
→ 2 nested for loops

→ width = (i - j + 1)

→ height = minimum (height between i & j)

→ Code:

```
int max_area = 0;
for (i = 0; i < heights.size(); i++){
    for(j = i; j < heights.size(); j++) {
        int min_height = INT_MAX;
        for(k = i; k < j; k++) {
            min_height = min(min_height, heights[k]);
        }
        max_area = max(max_area, min_height * (i - j + 1));
    }
}
return max_area;
}
```



width = i - j + 1 = (4 - 2) + 1 = 3 = width

height = minimum (height bet^n i & j) = 4

Area = width x height = 4 x 3 = 12.

Method II [Stack Implementation]

→ Traverse through histogram array.

→ push 1st index of hist[] in stack.

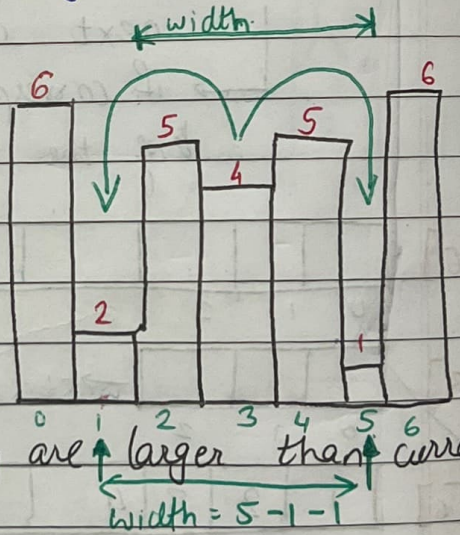
→ now check the elements ahead if they are larger than current height or smaller.

→ If the next element is larger push in the stack.

→ Now compare again with top of the stack.

→ if element is smaller, pop the top of stack and update height to that corresponding height of hist bar.

→ Basically we compare top of the stack with elements if they are higher we push them; if lower we pop the top and see the lower element on both sides that will be width of rectangle.



Page No.
Date

Code:

```
stack<long long> s;
long long ma = 0;
long long i = 0;
while (i<n){
    if (s.empty() or hist[s.top()] <= hist[i])
        s.push(i++);
    else{
        long long tp = s.top();
        s.pop();
        long long ans = hist[tp] * (s.empty() ? i : i - s.top() - 1);
        ma = max(ma, ans);
    }
}
while (!s.empty()){
    long long tp = s.top();
    s.pop();
    long long ans = hist[tp] * (s.empty() ? i : i - s.top() - 1);
    ma = max(ma, ans);
}
return ma;
}
```

* In stack we will store the index's of histogram bars.

if first element push; if next element is bigger, push

if we find element smaller than our hist bar pop & calc the width

else see eg on left: 5 - 1 - 1 = 3 = width

ans = hist[tp] * (s.empty() ? i : i - s.top() - 1);
hist[tp] → height; width of rectangle

if stack is empty we will take i as width i.e full width of histogram

if our first loop ends and there are bars whose area is not calculated i.e in left ex, 1's area won't be calc as there is no element smaller than 1 on the right & same for 6 so we need to run while loop again to check. While loop will be same.

Time: O(n)
Space: O(n)

DSA 450.

Q 8] Next Greater Element [Similarly next Smaller element also]

Method 1: [Brute Force; Time: $O(n^2)$]

→ Use 2 loops and check each element if it is greater than current element.

Method 2: [Stack Approach]

→ Create vector Ans.

→ Iterate the given array from right to left.

→ at first stack is empty; ∴ return -1

→ push first element in stack.

→ move ahead; compare if current element is greater or smaller than top of stack.

→ if it is smaller then push back s.top() in ans as that will be next greatest element.

→ You need to push that element irrespective it is small or big than top of stacks

→ If it is bigger than s.top; pop the top and push the curr element in stack.

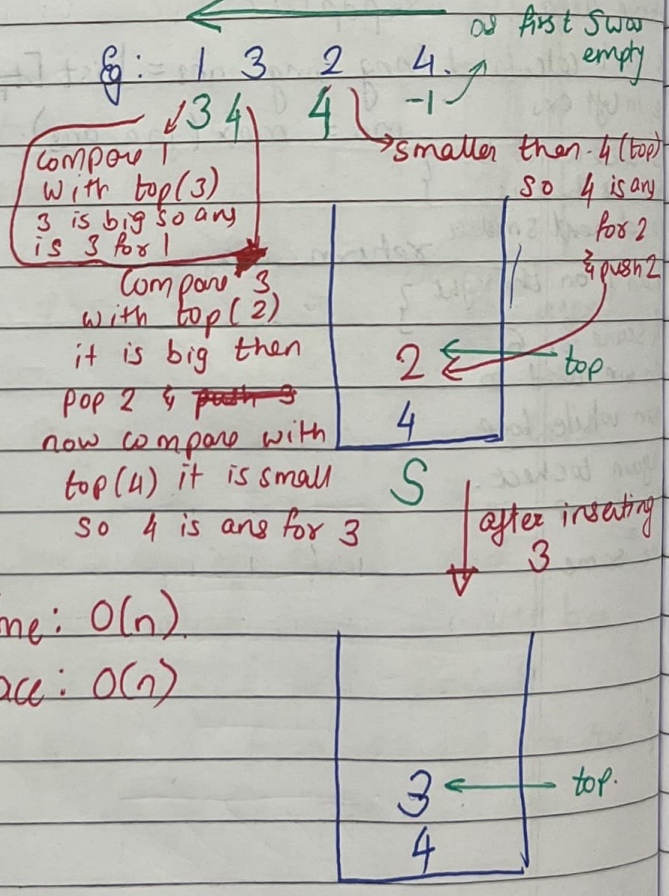
→ reverse the ans array.

Code:

```
stack<int> s;
int i;
vector<int> ans;
for(i=n-1; i>=0; --i){
    while(s.size() != 0){
        if(s.top() <= a[i])
            s.pop()
        else break;
    }
    if(s.size() == 0){
        ans.push_back(-1);
    else{
        ans.push_back(s.top());
    s.push(a[i]);
} reverse(ans.begin(), ans.end()); return ans;}
```

Time: $O(n)$

Space: $O(n)$



Page No.
Date

3 things you need to do:

→ Stack Empty → push_back -1

→ s.top() > arr[i] → s.top()

→ s.top() < arr[i] → pop → Stack is Empty.
→ s.top() > arr[i].

Code:

```
{ vector<int> v;
  stack<int> s;
 for(int i=n-1; i>=0; i--){
    if(s.size()==0){
      v.push_back(-1);
    }
    else if(s.size()>0 && s.top()>arr[i]){
      v.push_back(s.top());
    }
    else if(s.size()>0 && s.top()<=arr[i]){
      while(s.size()>0 && s.top()<=arr[i]){
        s.pop();
      }
      if(s.size==0){
        v.push_back(-1);
      }else{
        v.push_back(s.top())
      }
    }
    s.push(arr[i]);
 } reverse(ans.begin(), ans.end());
 return ans;
}
```

if Q was find next greater element to LEFT.

for(int i=0; i<=n-1; i++){

body this change & in end no need to reverse ans

if in ques next smaller element to right is asked, just change the sign

Time: O(n)

Space: O(n)

Here same code with minor changes 4 Q's are possible.

1) Next Greater Element
2) Next Smaller Element.
3) Nearest Greater to left.
4) Nearest Smaller to left.

Page No.
Date

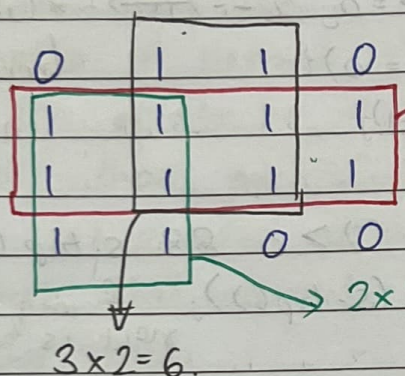
V Good Sum ***

Q] Maximum Area Rectangle in Binary Matrix.

In this you should know maximum Area of Histogram [MAH]

```
MAH(int arr[], int size)
{
    right[] ⟶ NSR(arr, size) → 2D vector, stores value & index of NSR
    left[] ⟶ NSL(arr, size) → 2D vector, stores value & index of NSL
    width[i] = right(i) - left(i) - 1
    area[i] = arr[i] * width[i].
    return max(area[i]);
}
```

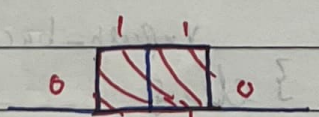
Binary Matrix:



Here we dont know how to find max area of 2D matrix
But we know to find of 1D array [MAH()]
So we convert this into 1D array.
↳ break

① Select first row:

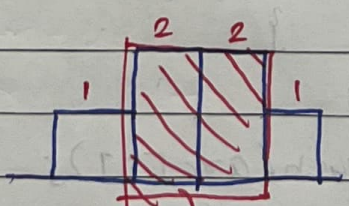
0 1 1 0



↳ MAH = 1×2 = 2.

② Add first & second row.

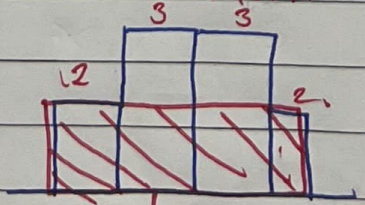
```
  0 1 1 0
+ 1 1 1 1
-------
  1 2 2 1
```



↳ MAH = 2×2 = 4.

③ Add third & ~~se~~ upper wala ②.

```
  1 2 2 1
+ 1 1 1 1
-------
  2 3 3 2.
```

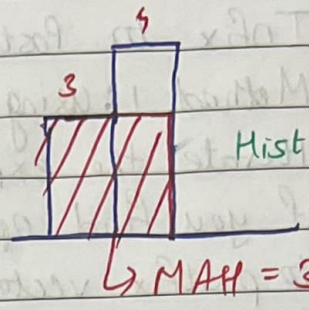


↳ MAH = 2×4 = 8

Page No.
Date

④ Add fourth row & ③ wala.

```
  2  3  3  2
+ 1  1  0  0
-------------
  3  4  0  0
```



↳ MAH = 3×2 = 6.

Here 0 will come as if you see matrix this cannot be included in area i.e if you find any lower row that has 0; dont add and write then 0.

Our ans will now be Max (Hist ①, Hist ②, Hist ③, Hist ④)

Code:

```
vector<int> v;
for(int j=0; j<m; j++){
    v.push_back(arr[0][j]);
}
```

i \ j	0	1	1	0
	1	1	1	1
	1	1	1	1
	1	1	0	0

```
int mx = MAH(v) → MAH will return max area of all Hist ①, ②, ③, ④.
```

n×m matrix → Code = max area of Histogram.

```
for(int i=1; i<n; i++){
    for(int j=0; j<m; j++){
        {
            if(arr[i][j]==0){
                v[j] = 0;
            }else{
                v[j] = v[j] + arr[i][j];
            }
            mx = max(mx, MAH(v));
        }
        return mx;
}
```

Time: O(n×m)
Space: O(m)

Page No.
Date

Q Infix to Postfix Conversion.

Method 1: Using stack.

→ Iterate the expression.

→ if you find operand, add it to postfix vector

→ if you find operator then
- Compare with top of stack if it's precedence is higher or equal than top of stack, push it into stack.
- if its precedence is lower than top of stack, pop the top & push_back it top into postfix vector. Again check with top & repeat procedure.
- if Stack is empty then push operator in stack.

Symbol	Precedence	Associativity
+, −	1	L-R
*, /	2.	L-R
^	3	R-L
− (unary minus)	4	R-L
()	5	L-R

Eg: a + b * c − d / e.

Symbol	Stack	Postfix
a	—	a
+	+,	a
b	+,	ab
*	+, *2	ab
c	+, *2	abc
−	−,	abc*+
d	−,	abc*+d
/	/3 −,	abc*+d
e	/2, −,	abc*+de.
−		abc*+de/−

Time: O(n)
Space: O(n).

Method 2: Using Stack.

→ Same as above method but you push operands & operators both operands have precedence 3 & Repeat the exact same procedure as Method I

Symbol	Precedence	Associativity
+, −	1	L-R
*, /	2	L-R
a, b, c	3	L-R

Code: look on VS code. (Method 1)