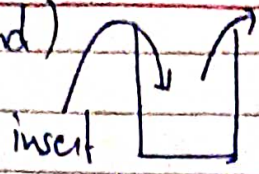
22-07-2022

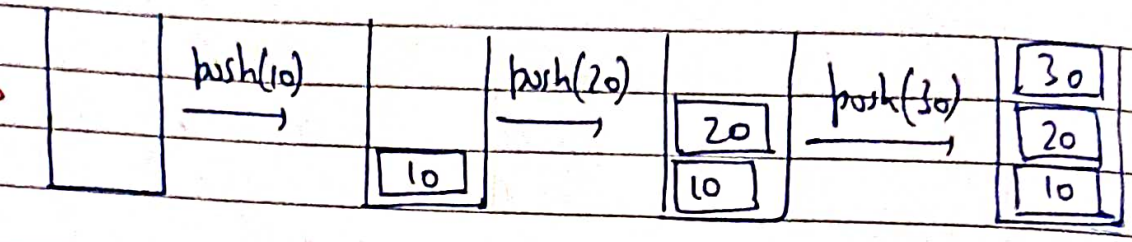
STACK

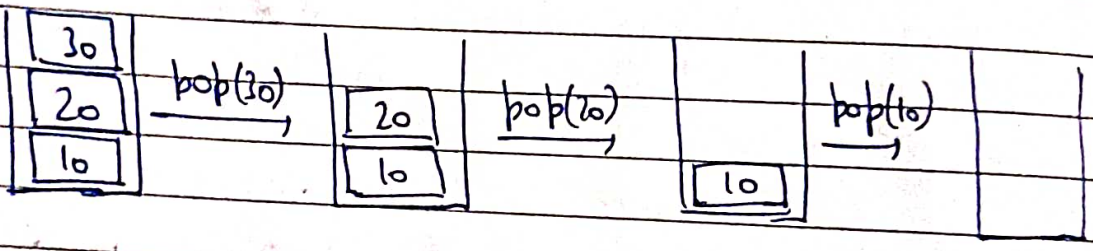
LIFO ← (Last in First out)

- Stack is a linear DS (works like a box which is closed at one end)



- Insertion operation is called push operation
- Removal ——— pop ———





TC of all is O(1)

Stack Operation

① ~~X~~Empty() returns true if stack is empty else false.
② push(x) insert an item on top of stack
③ pop() remove an item from the top
top() ④ ~~peek~~ return the top item → don't modify stack
⑤ size() return the size of stack →

→ **Two Corner Condition on Stack Operation:**

① **Underflow:**
occurs when pop() or top() is called on an empty stack

② **Overflow:**
- when push called on full stack
- Removed when using Dynamic sized Array or linked list.

peek() → Java में
top() → C++ ——

isEmpty() → Java
empty() → C++

Not Dynamic

→ Array Implementation of Stack

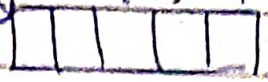
- We use an Stack class or Structure

```
struct MyStack
{
    int *arr;
    int cap, top;
    MyStack(int c)
    {   cap = c;
        arr = new int[cap];
        top = -1;
    }
    void push(int x)
    {   if(top == cap-1) { ... }
        top++;
        arr[top] = x;
    }
    int pop()
    {   if(top == -1) { ..... }
        int rs = arr[top];
        top--;
        return rs;
    }
    int peek()
    {   if(top == -1) { ..... }
        return arr[top];
    }
    int size()
    {   return (top+1);
    }
    bool Empty()
    {   return (top == -1);
    }
};
```

return लिख तो जो इसके return करना है।

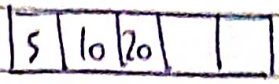
Example

```
MyStack s(5);
```



top = -1

```
s.push(5); s.push(10); s.push(20);
```



now top = 2

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

top = 1

s.size() → (top+1) = 2

s.Empty() → false

black pen जो लिखा है वो Overflow & Underflow को manage कर रहा।

Dynamic

→ Vector Implementation of the Stack

```
struct MyStack
{
    vector<int> v;
    void push(int x)
    {
        v.push_back(x);
    }

    int pop()
    { int ans = v.back();
      v.pop_back();
      return ans;
    }

    int size()
    {  return v.size();
    }

    bool isEmpty()
    {  return v.empty();
    }

    int peek()
    {
       return v.back();
    }
};
```

→ No Overflow problem

→ Linked list Implementation of Stack:

We insert & delete from head to maintain O(1) complexity.

```
struct Node
{ int data;
  Node * nxt;
  Node(int d)
  { data = d;  nxt = NULL; }};
```

Node structure

```
class MyStack
{ Node *head;
  int sz;
  MyStack()
  { head = NULL; sz = 0; }

  void push(int x)
  { Node *temp = new Node(x)
    temp->next = head;
    head = temp;
    sz++;
  }

  int pop()
  { if(head == NULL)                    ] edge case Underflow
      return INT_MAX;
    int ans = head->data;
    Node *temp = ~~data~~ head;
    head = head->next;
    delete(temp);   // to avoid memory leakage
    sz--;
    return ans;
  }
  int size() { return sz; }
  bool Empty() { return (head == NULL) }

  int peek()
  { if(head == NULL)                    ] edge case Underflow
      return INT_MAX;
    return head->data;
  }
};
```

→ Application of Stack :

① Function call (LIFO)
- when we use recursion, last function call is finished first
- Most programming langugs use stack for function call called function call stack.

② Checking for Balanced paranthesis
③ Reversing item
④ Evaluation of Postfix / Prefix
⑤ Infix to Postfix / Prefix

Infix is the expression We generally write in Program
Prefix: operator first then operand.
Postfix: operand ——— operator.

** Infix expression are difficult for Compiler to evaluate bz it follow operator precedence, Associativity, brackets.

but in prefix & postfix expression we don't have these precedence, associativity.

- Every operator has fixed no of Operand) Rule for evaluating writing postfix & prefix expr
- Compiler convert infix to postfix / prefix then evaluate prefix/postfix.
  convertion is based on Stack

⑥ Stock Span Problem & its variations.
⑦ Undo/Redo or Forward/Backward (like in Web Browser)
   - last opened URL

Priority Queue & Queue are also Container Adaptor

→ Stack in C++ STL

- C++ STL has a class STL
- s.push(), s.pop(), s.top(), s.size(), s.empty() → all O(1)
- Stack can be implemented on any underlying container that provides following operations.
  - back()
  - size()
  - empty()
  - push_back()
  - pop_back()

→ vector, deque, list → these all provide all the above functionality

* By default it is implemented internally using Deque container
* Since stack is implemented on another container therefore it is called Container Adaptor.

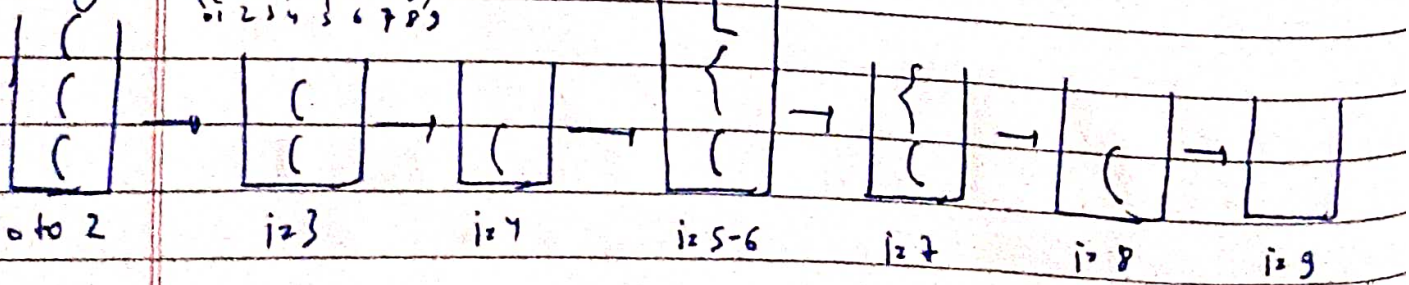
- Open bracket must be closed by same type of brackets.
- Open bracket must be ——— in the correct order.

(I) Valid Parenthesis :

s="()[]{}"	"(]"	"(())[]"	"(()){[]}{([])}"
o/p - True	false	True	True

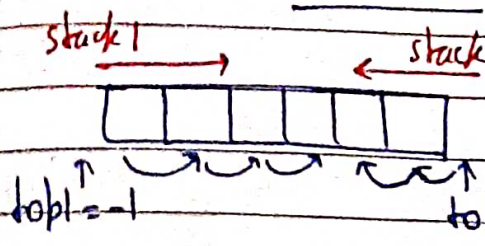
```
bool isValid(string s)
{
    stack<int> st;
    for(int i=0; i<s.size(); i++)
    {
        if(s[i]=='(' || s[i]=='{' || s[i]=='[')
            st.push(s[i]);
        else
        {   if(!st.empty() && s[i]=='}' && st.top()=='{')
                st.pop();
            else if(!st.empty() && s[i]==')' && st.top()=='(')
                st.pop();
            else if(!st.empty() && s[i]==']' && st.top()=='[')
                st.pop();
            else
                return 0;
        }
    }
    return (st.size()==0);
}
```

eg. ((()){[]})
0 1 2 3 4 5 6 7 8 9



(II) Implement two Stack in an Array

Logic : we maintain first stack from front & second ——— back




```
void push1(int x)
{   if(top1 < top2 - 1)
        top1++;
    arr[top1] = x;
}
```

```
void push2(int x)
{   if(top1 < top2 - 1)
        top2++;
    arr[top2] = x;
}
```

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

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

(III) Stock Span Problem

a[] = {100, 80, 60, 70, 60, 75, 85}

o/p = [1, 1, 1, 2, 1, 4, 6]

- The Span S_i of the stock price on a given day i is defined as the consecutive day (starting) price less than or equal to given day i.

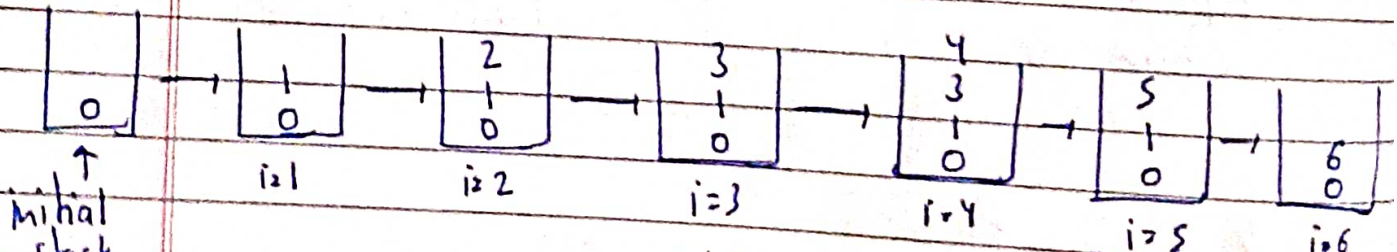
stack<pair<int,int>> st;
To access last element st.top().first;
st.top().second;

```
vector<int> f(int a[], int n)
{
    stack<int> st;
    vector<int> v;
    st.push(0);
    v.push_back(1);
    for(int i=1; i<n; i++)
    {
        while(!st.empty() && a[st.top()]<=a[i])
            st.pop();
        int span = st.empty()? i+1 : i-st.top();
        v.push_back(span);
        st.push(i);
    }
    return v;
}
```

index push कर रहे हैं stack में

0 1 2 3 4 5 6

eg: [100 80 60 70 60 75 85]



initial stack i=0 | i=1 | i=2 | i=3 | i=4 | i=5 | i=6

2 pop | 4,3 pop | 1,5 pop

vector v: 1 | 1 | 1 | 2 | 1 | 4 | 6

- Stack में क्या है?

→ Say किसी i पे है हम (say i=6), तो i=5 वाला index stack के top पे होगा। दी और है पे जो बड़ा element है उसका index stack के 2nd top पे होगा & so on.

(V) Score of Parenthesis

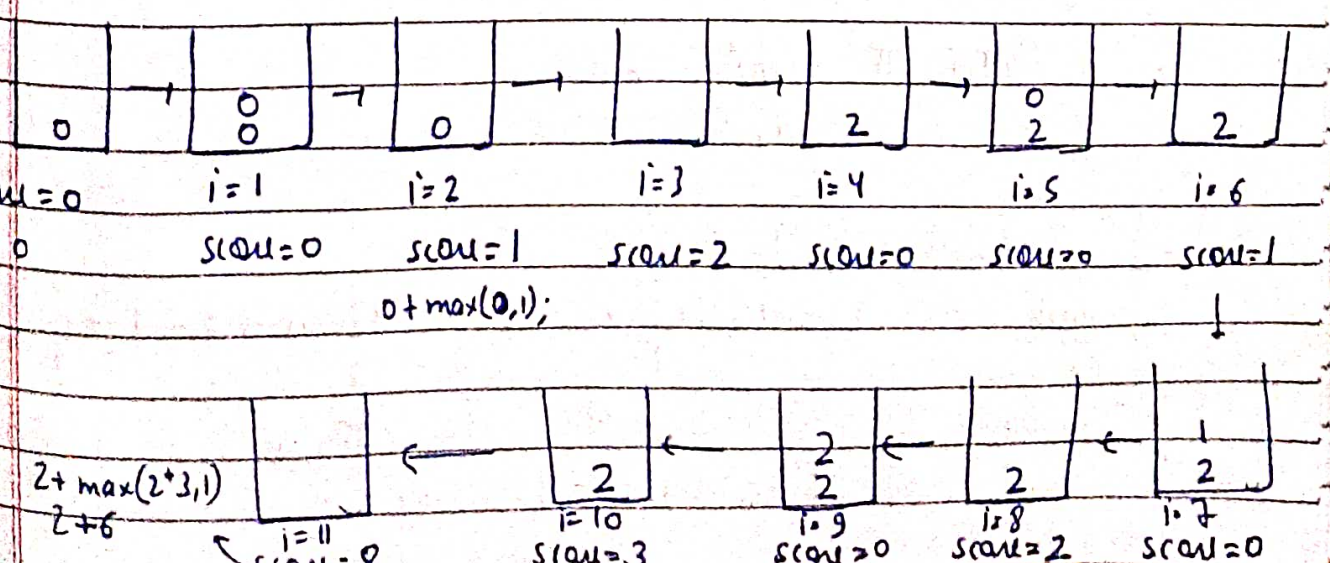
() → score 1
AB → ()() → A+B = 2
(A) → score = 2 * A

} find score of given string

a)
```
int f(string s)
{
    stack<int> st;
    int score=0;
    for(int i=0; i<s.size(); i++)
    {
        if(s[i] == '(')
        {   st.push(score);
            score=0;
        }
        else
        {   score = st.top() + max(2*score,1);
            st.pop();
        }
    }
    return score;
}
```

0 1 2 3 4 5 6 7 8 9 10 11

b) (()) (()()())



o एक और way से करना होगा (Java mapping)

b) int f(string s)
{
int depth=0, ans=0;
char prev;
for(const char &ch:s) // for(auto ch:s)
{
if(ch == '(')
depth++;
else
{ depth--;
if(prev == '(')
ans += pow(2, depth);
}
prev = ch; // prev को हर बार change होना।
}
return ans;
}

Logic: (((()))) = $2^3 = 8$
I I I 1

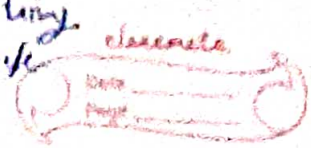
(((()()))) = 8+8 = 16

. () → अगर ये एक साथ है तभी ans को update करेंगे।
. ')' मिलेगा तो depth-- होगा
. '(' ——— depth++ ——

. ((()()) (())) ()
depth = 0 → 1 → 2 → 3 → 2 → 1 → 2 → 3 → 2 → 1 → 0 → 1 → 0
if () exist () exist () exist () exist

$2^2 + 2^2 + 2^2 + 2^0 = 13$

for circular array we simply iterate over string from i=n-1 to 0 & store next NGE in stack



⑤ Next Greater Element II

Given circular integer array, return NGE for every element in nums.

nums[] = [1, 2, 3, 4, 3]
o/p = [2, 3, 4, -1, 4]

```
vector<int> f(vector<int> &nums)
{
    int n = nums.size();
    vector<int> nge(n, -1);
    stack<int> st;
    for(int i=2*n-1; i>=0; i--)
    {
        while(!st.empty() && st.top() <= nums[i%n])
            st.pop();
        if(i<n)
        {
            if(!st.empty())
                nge[i] = st.top();
        }
        st.push(nums[i%n]);
    }
    return nge;
}
```

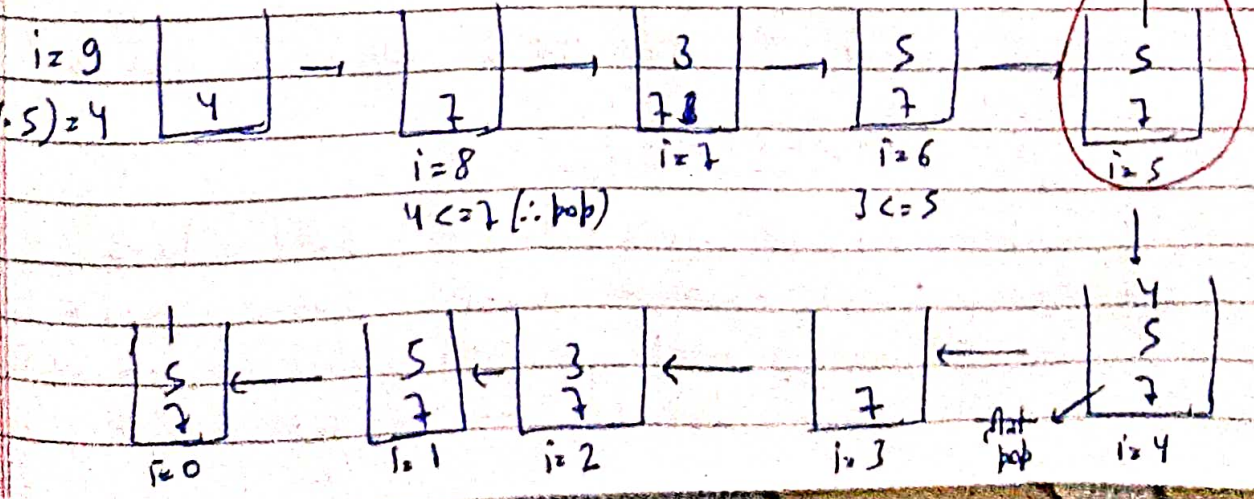
(i<n) वाली nge vector में push करनी है

for imagine

1	5	3	7	4	1	5	3	7	4
0	1	2	3	4	5	6	7	8	9

eg. [1, 5, 3, 7, 4] n=5

जो stack में अंत nge में NGE store होती



• Revise it (if not clear) see Striver video

(VI) Trapping Rainwater:

T.C → O(n) S.C → O(1)

```
int f(vector<int> &v)
{
    int l=0, r=v.size()-1, leftmax=0, rightmax=0, ans=0;
    while(l<r)
    {
        if(v[l]<=v[r])
        {   if(leftmax<=v[l])
                leftmax=v[l];
            else
                ans+=leftmax-v[l];
            l++;
        }
        else
        {   if(rightmax<=v[r])
                rightmax=v[r];
            else
                ans+=rightmax-v[r];
            r--;
        }
    }
    return ans;
}
```

• किसी index पे left और right में से leftmax और rightmax

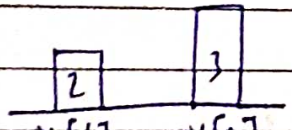
0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1

0<=1	1<=1	0<=1	2<=1 x	2<=2	1<=2
lm, rm=0	0<=1 ✓	1<=0 x	0<=1	1<=2	2<=1 x
l++	lm=1	ans=1	rm=1	lm=2	ans=1+(2-1)=2
	l++	l++	r--	l++	l++

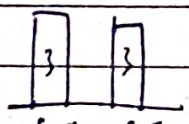
0 <= 2 2 <= 0 x ans = 2 + (2-0) = 4 l++	1 <= 2 2 <= 1 x ans = 4 + (2-1) = 5 l++	3 <= 2 x 1 <= 2 ans = 2 r--	3 <= 1 x 2 <= 1 x ans = 5 + (2-1) = 6 r--

3 <= 2 x 2 <= 2 ans = 2 r--;	Stop

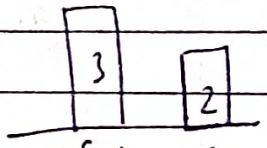
ध्यान से देखो v[l] <= v[r] से line confirm कर रहा कि जो v[l] का मान है less than होगा या equal होगा v[r] से मतलब से line confirm कर रहा right में जो height of block है वो लंबा है या equal है।



v[l] < v[r]

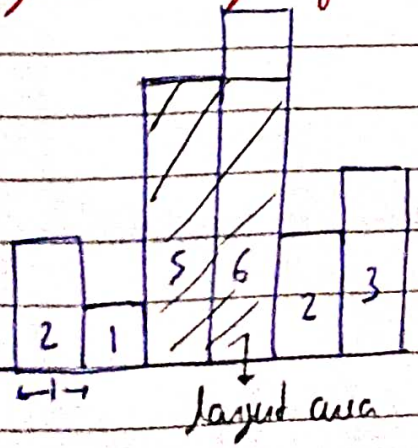


v[l] == v[r]



leftmax changes

VII. Largest Rectangle of Histogram :



v[] = [2, 1, 5, 6, 2, 3] Given
o/p = 10

Storing index in lsmallest & rsmallest

@ int f(vector<int> &v) O(n) + S(n) O(4N)

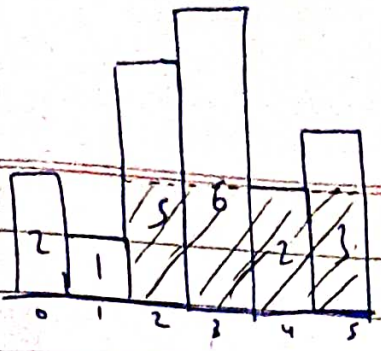
3N

```
{
    int n = v.size();
    vector<int> lsmallest(n, 0), rsmallest(n, n-1);
    stack<int> st1, st2;
    for(int i=0; i<n; i++)
    {
        while(!st1.empty() && v[st1.top()] >= v[i])
            st1.pop();
        int ans = st1.empty() ? 0 : st1.top()+1;
        lsmallest[i] = ans;
        st1.push(i);
    }

    for(int i=n-1; i>=0; i--)
    {
        while(!st2.empty() && v[st2.top()] >= v[i])
            st2.pop();
        int ans = st2.empty() ? n-1 : st2.top()-1;
        rsmallest[i] = ans;
        st2.push(i);
    }
    int ans=0;
    for(int i=0; i<n; i++)
        ans = max(ans, (rsmallest[i] - lsmallest[i] + 1) * v[i]);

    return ans;
}
```

Formula : (rsmallest[i] - lsmallest[i] + 1) * v[i] area of height v[i]



i=4 lsmallest[4] = 2 (st.top() + 1)
rsmallest[4] = 5 (n-1)

(5-2+1) * v[4] = 4 * 2 = 8 ✓

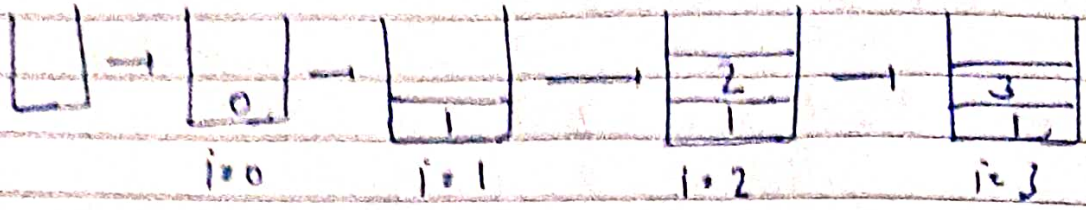
lsmallest[] = [0, 0, 2, 3, 2, 5]
rsmallest[] = [0, 5, 3, 3, 5, 5]

(b) TC → O(n) SC → O(n) ← Optimal

```
int largestRectangle(vector<int> &v)
{   int n=v.size();
    stack<int> st;
    int ans=0, curr=0;
    for(int i=0;i<n;i++)
    {
        while(!st.empty() && v[st.top()]>=v[i])
        {
            int ind = st.top();
            st.pop();
            curr = v[ind] * (st.empty()?i : (i-st.top()-1));
            ans = max(ans, curr);
        }
        st.push(i);
    }
    while(!st.empty())
    {   int ind=st.top();
        st.pop();
        curr = v[ind]*(st.empty()?n: n-st.top()-1);
        ans = max(ans, curr);
    }
}   return ans;
```

stack में index push

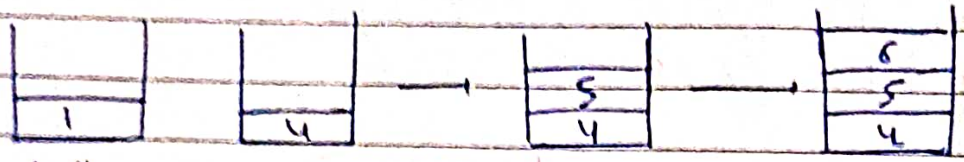
Q. a[] = { 6, 2, 5, 4, 1, 5, 6 }



i=0 | i=1 | i=2 | i=3

i=1: sum=6, ans=6

i=3: sum=5, ans=6



i=4 | i=5 | i=6

i=4: sum=8, sum=8, ans=8 (ans updated)

i=5: sum=8

i=6: sum=8

i=4

element in stack after for loop complete

- जो index stack में है उसी इनका left smallest नहीं है

→ now while loop execute

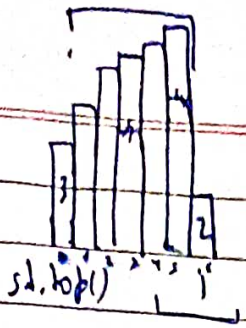
sum=6 | sum=10 | sum=1+7=7

update ans=10

logic

** जब भी element निकालो stack से (st.pop()) तब एक sum calculate होगा उसके पीछे जो index निकलेगा वो उसी निकलेगा जब v[st.top()] >= v[i] तात्पर्य st.top() का ~~next~~ right smallest index i होगा और left smallest इसको पता है st.top() के नीचे वाला

stack
when i=6



Question में कोई दिक्कत नहीं simply stack में push

जबतक ही v[sl.top()] >= v[i]

ans exist करता होगा अगर समझो

right smallest i होगा और

left smallest sl.top() के नीचे वाला

इससे भी एक ans calculate होगा।

for loop के सबसे वाला while loop कब तक चलेगा क्योंकि

v[sl.top()] >= v[i] when v[i] = 2

VIII Maximal Rectangle O(r×c) S.C. O(c)

Given r×c of binary matrix find largest rectangle containing only 1's.

1	0	1	0	0
1	0	1	1	1
1	1	1	1	1
1	0	0	1	0

logic

0	1	0	1	0	0
1	2	0	2	1	1
2	3	1	3	2	2
3	4	0	0	3	0

→ सब एक एक vector लेलो previous Question के func. में & calculate MR

```
int f(vector<vector<char>>&v)
{   int r=v.size(), c=v[0].size();
    vector<int> vl(c,0);
    int ans=0;
    for(int i=0;i<r;i++)
    {
        for(int j=0; j<c;j++)
        {
            vl[j] = (v[i][j]=='0') ? 0 : vl[j]+=1;
        }
        int ans = f(vl);        → largestRectangle(v)
        ans = max(ans,ans);          ↓ previous Question function optimal wala
    }
    return ans;
}
```