01 FRIDAY
JULY 2-0-2-2 Week-27 (182-183)

July 2022

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Stack

Stack :- is a type of container adaptors with LIFO (last in First out) type of working, where a new element is added at one end (top) and an element is removed from that end only.

Functions / Operations :-

1) Push → Insert element to top
2) Pop → Remove element from top.
3) Peek → Prints Top element.
4) empty → Returns if stack is empty or not.
5) Emplace → Same as Push but instead of copy new Constr. is formed
6) Swap :- Swap elements of one stack to another.

Reverse a String using Stack :-

String = Hello Abhi

1.) Push each character of String in Stack

I
H
B
A

→ S[i] = st.top();
S = "IHBA".

Easy Peasy!

Design a Stack with 'min_element' function to get minimum element in O(1) complexity.

For this we'll use a vector of pair of type int.

vector<pair< int, int >> st;

Sunday 03

Now while inserting (Push function) we'll store the minimum value in second part of pair.

If Push function {
→ if stack is empty
p = { val, val }
as first element is minimum one.

July 2022
S M T W T F S S M T W T F S
1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31

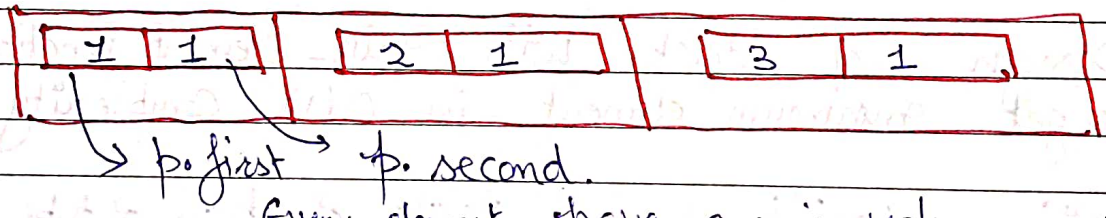
otherwise

min (val, st.back().second) is minimum

↳ Same as arr[n-1].

for Popping out an element

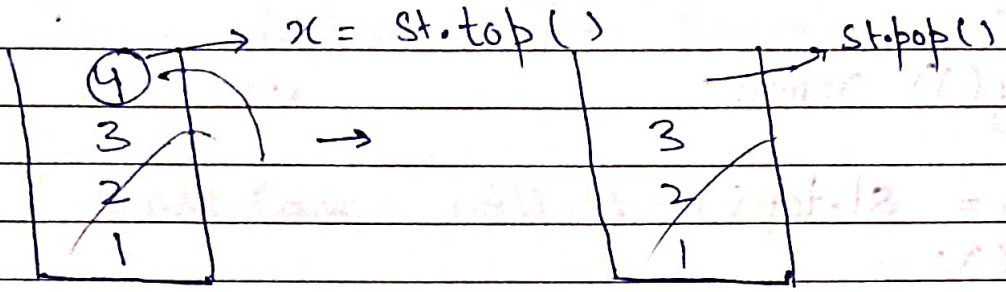
```
{
    st.pop_back();
}
// Checkout code for Better Understanding.
```



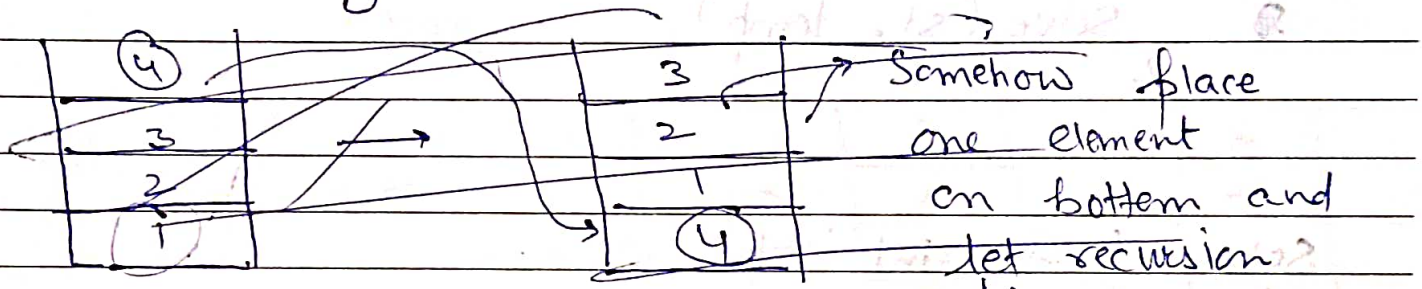
Every element have a min value associated.

Reverse a Stack - Recursion (Backtracking).

(temp = 4)

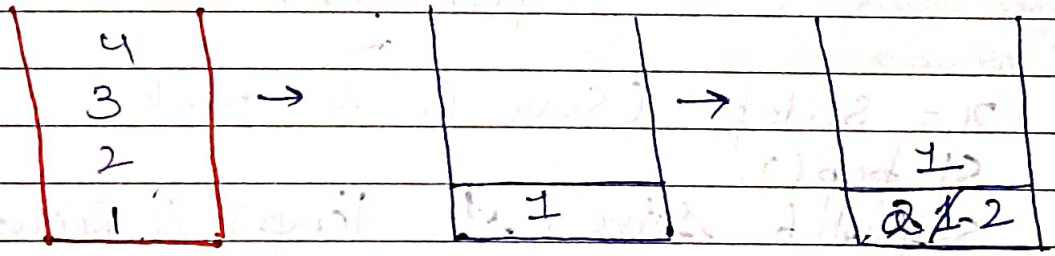


Basically Here we'll ~~not~~ solve 1 case and we'll rely on recursion.

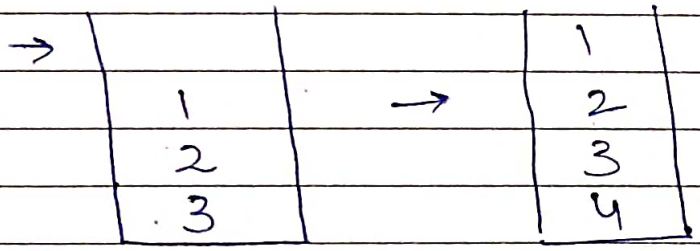


Real Dry + Run

→



→



July 2022
S M T W T F S S M T W T F S
1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31

First traverse through Stack Using Recursion

```
if (st.empty()) return;

int temp = st.top();
st.pop();

Reverse(st); // Recursion

Solve(st, temp);
}

Solve (st, int temp)
{
    Base Case :- if (stack is empty)
                    push(temp) & return

    x = st.top (Save for Backtrack)
    st.pop();
    Solve (st, temp). // Recursion
    st.push(x) // Backtrack.
```

August 2022

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Week-28 (188-177)

THURSDAY 07 JULY

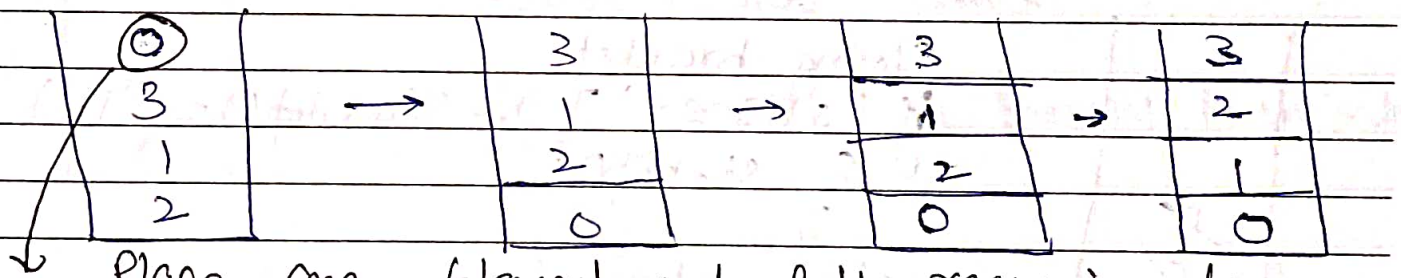
2-0-2-2

Sort a Stack :-

Same Approach as Reverse a Stack But here

Base Case will be .

```
if ( st.empty() || st.top() <= num )
{
    st.push ( num);
    return;
}
```



Place one Element and let recursion play it's role #Easy Life #Bhupendra Jogi #RecursionOP #RecursionHai TohKyaChahiye.

July 2022

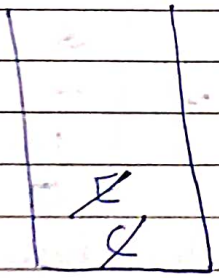
S	M	T	W	T	F	S	S	M	T	W	T	F	S
					1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31						

Valid - Parenthesis:-

if it's an opening bracket push it into Stack.

Else if it's a closing bracket & stack is Empty return false (as no closing bracket comes first).

Example :- ([ ])



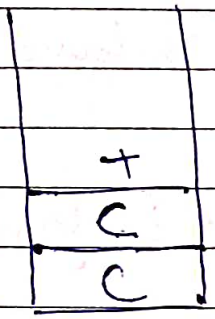
```
Closing bracket
if (s[i] == ']' && stack.top() == '[')
  { st.pop();
  }
else return false;

Return true;
```

Reduclant Bracket :-

Reduclant Bracket Basically means useless bracket.

((a+b)) → (a+b); Here two ~~bca~~ useless brackets are present hence return ~~fal~~ True.

To check so,



→ If an opening bracket is encountered push it into Stack.

→ If an operator is present push into Stack.

→ Now if a closing bracket is found initialize a bool flag with false value

Now until an ~~operating~~ opening bracket is found pop the Stack

Check if

if an operator is found in mid put flag = true;

Now if (St.top == 'C' && flag == false)
return true
else pop();

11 MONDAY
JULY 2-0-2-2
Week-29 (192-173)

July 2022

S	M	T	W	T	F	S	S	M	T	W	T	F	S
					1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	.	.	.	.	.	.

Longest Valid Parenthesis :-

→ first Initialise Stack with (-1)

0 1 2 3 4 5 6 7
) ( ) ( )) ( )

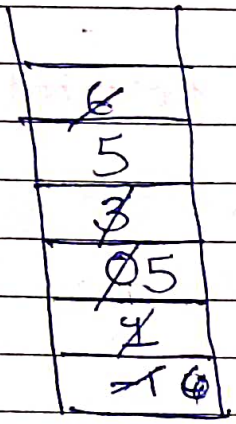
→ if(S[i] == '(')
{
S.push(i);
}
// If an opening bracket is found push its index to stack

→ Else // if closing bracket is found
{
→ pop the stack.

→ if (stack is empty)
{
st.push[i];
}

→ else
{
len = i - st.top();
ans = ans (max, len);
}

For String :- 0 1 2 3 4 5 6 7
( ) ( )) ( )



len = 2-0 = 2 len = 7-5 = 2

len = 4-0 = 4

→ 1234 = ( ) ( ) Right ans.

Count - Reversals :-

// Edge Case :-

→ for odd length there can't be a valid balanced brackets
Simply return -1;

→ Now create a Stack
→ push opening brackets
→ pop them when their corresponding bracket occurs.
→ Example :-

} { { } } { } { {

{
{
{
}

Now with resultant Stack.

if { { → two same pairs are found simply increment Count by 1 as it'll take one Reversal to make it balanced { }.

Else :- } { , Count += 2 as two ~~int~~ reversals are require.

Next - Smaller Element :-

Here we're given an array

0	1	2	3	4
3	8	5	2	25

→ find the first smaller element present in next part of array and store it. ~~in same~~

→ For 3, Smaller 1st number is 2, similarly.

3	8	5	2	25
2	5	2	-1	-1

incase no. number (smallest number) is present after that number store (-1).

Approach :-

→ Take a Stack and push (-1) by default.

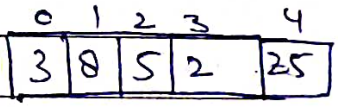
→ Start from End of Array & iterate till first element.

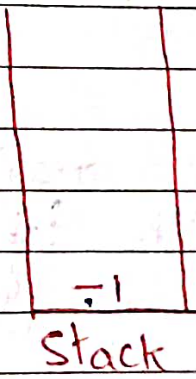
→ if (top element of stack > arr[i]

{

ans[i] = st.top();

}



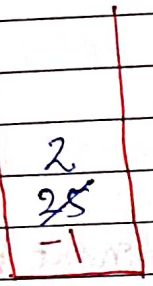


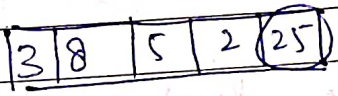
15 FRIDAY
JULY 2-0-2-2 Week-29 (196-169)

July 2022
S M T W T F S S M T W T F S
1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31

// What that condition Even means?

Example:- Here in Stack we' have (-1)
and we're iterating from back of array.





if (-1) > 25, it basically means next element smaller than 25 is -1
So, store -1 in ans.
→ Also push arr[i] in Stack

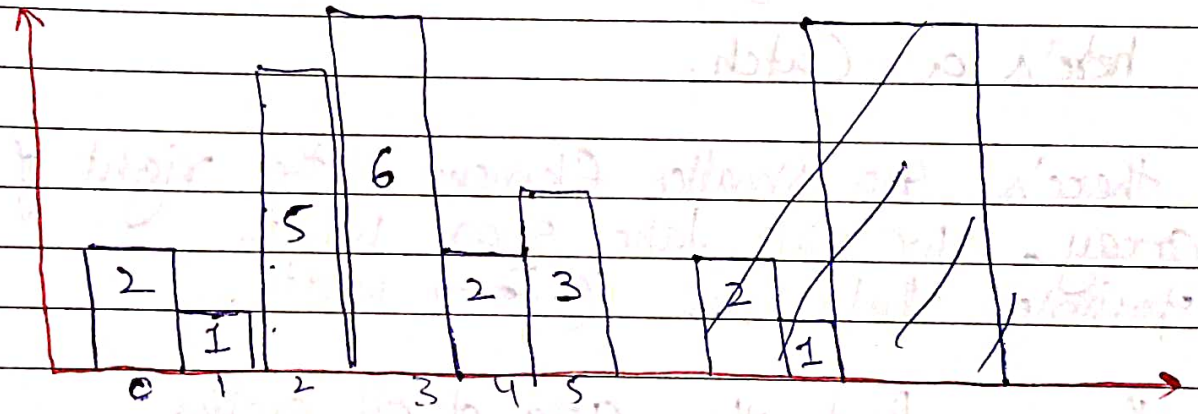
```
else if (arr[i] < st.top())
{
    while (arr[i] <= st.top()) st.pop();

    ans[i] = st.top();
    st.push(arr[i]);
}
```

Basically if array's element is smaller than top element, pop the stack until smaller element is found (& there'll be atleast -1) here.

// Previous Smaller element :-

Same as Next Smaller Element But just iterate from beginning of array

Largest Histogram Problem (Very Important).



Here arr[i] = height of (length of bar). width of Each

We have to return the maximum area of a particular rectangle.

Sunday 17

→ firstly let's see how we can create a rectangle.

→ for bar '0' → length = 2

width will be :- ~~- Previous Smaller + NextSmaller - 1~~

:- ~~Else~~

width is $n - p - 1$

$1 - (-1) - 1$

$1 + 1 - 1 = 1$

So area, will be 2.

Basically we need previous & next smaller values for array as well.

But here's a Catch.

if there's no smaller Element in the right of array, we can take max width possible that is (n) → width.

Then find the area of all graphs by multiplying $l \times b$;

and store the max of ans

and return ans;

Check if Word is Valid After Substitution.

Here if S string is made by splitting string into parts and then putting abc in middle. Then.

"abc" → [a] abc [bc]
left right

or "abc" → [ab] abc [c]
left right.

So, in Every Valid Case there should be groups of abc & popping all the groups we should have an empty string if not so, it means it's not valid.

Take a Stack

abcabcababcc

Now if first word is not a... simply return false

else if a is found push into stack.

b
a
b
a
b b
a a

→ Empty ⇒ valid

if s[i] == 'b' & st.top() == a → push b
else return false.

if s[i] == 'c' & st.top() == 'b' pop 2 times
else return false.

return st.empty();

22 FRIDAY
JULY 2-0-2-2

Week-30 (203-162)

July 2022

S	M	T	W	T	F	S	S	M	T	W	T	F	S
					1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	.	.	.	.	.	.

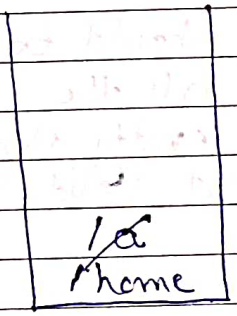
Simplify Path :-

Take a Stack of type string.

Dry Run on Test Case

/home/../a/../

Take a stack of type string.



Now take a string & till next '/' is found
that is /home

if (str == '/' or '.') continue;

else if (str != "..") push(str);
else st.pop();

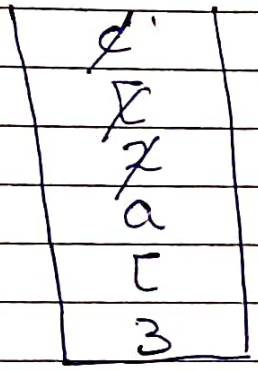
if stack is empty return "/";

Else reverse stack using recursion & put ans to string and all set!

Decode - String

3 [a2 [c]]

Simply push into stack until closing bracket



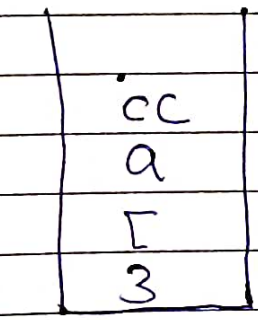
Now when s[i] is ']'
store alphabets in alphabet string
and numeric in numeric string & pop
~~until isdi~~

```
if (st.top() != "[")
while (not a digit) alphabets += st.top();
```

after that while (isDigit)
numstring += st.top()
st.pop();

alphabets = c, numstring = 2
now ~~result~~, result = 2 times c = cc
push result in stack

Sunday 24



alphabets = cca, numstring = 3
→ cca cca cca, result.
reverse the result
and return boom.