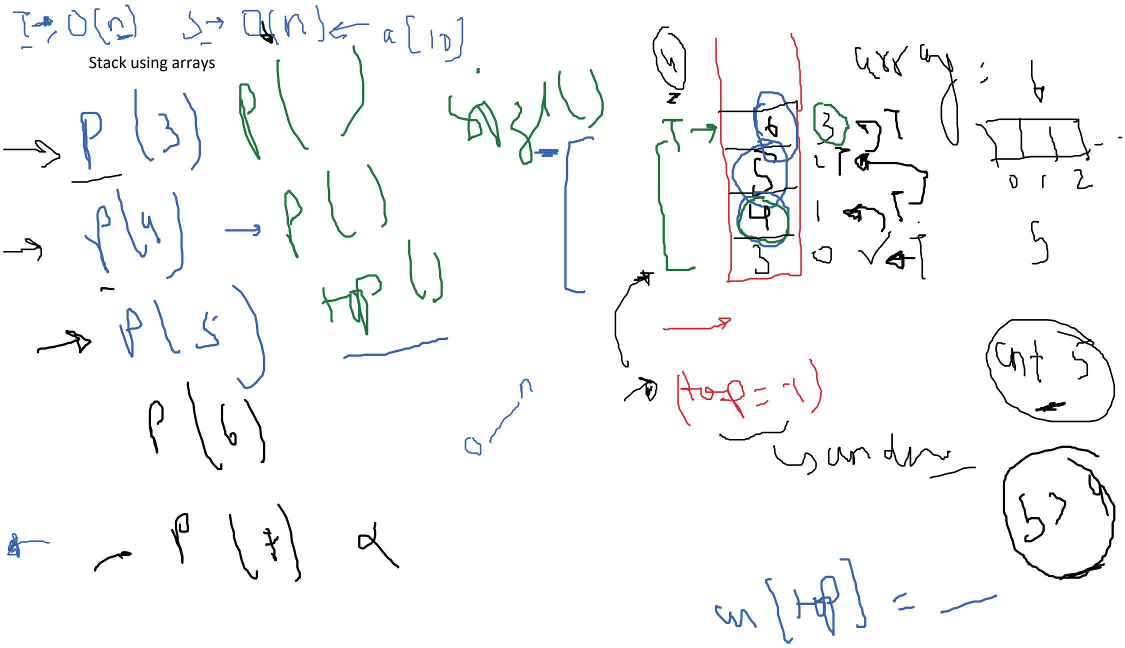
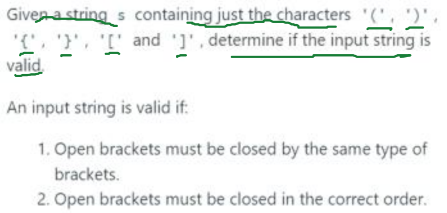
Stack And Queues 1

18 July 2022 09:11

Stack using arrays



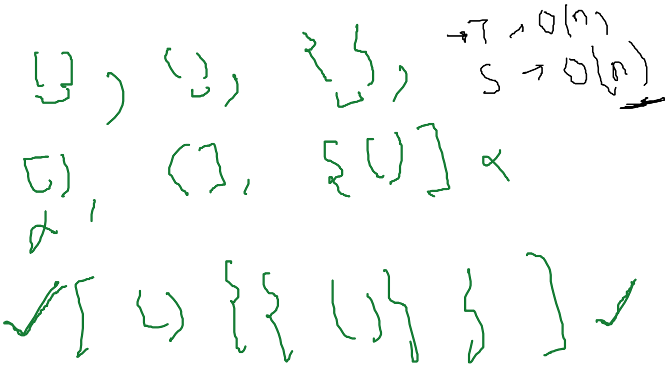


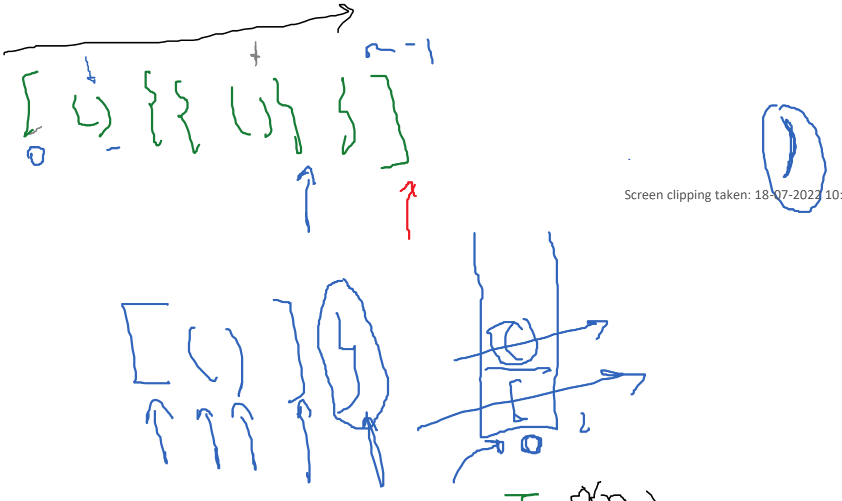
Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

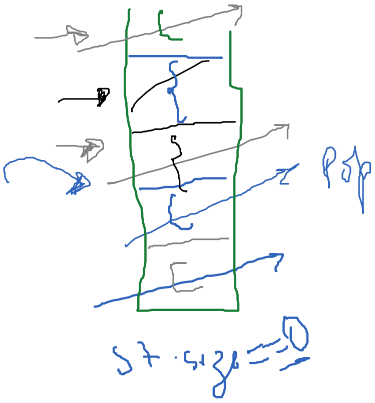
1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

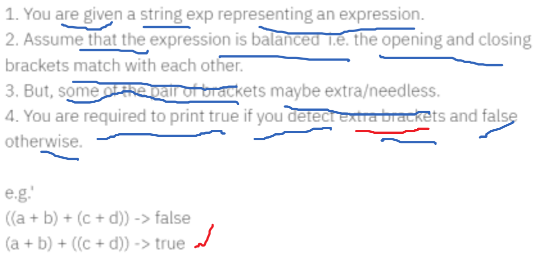
Screen clipping taken: 18-07-2022 10:18





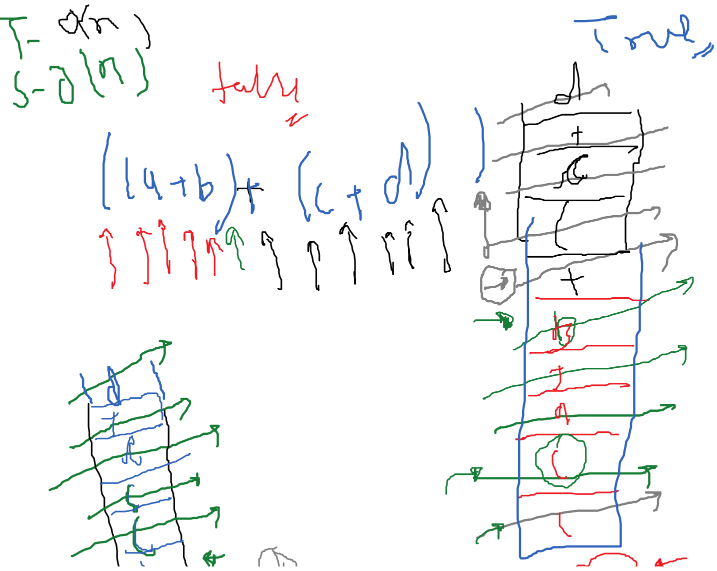
Screen clipping taken: 18-07-2022 10:45

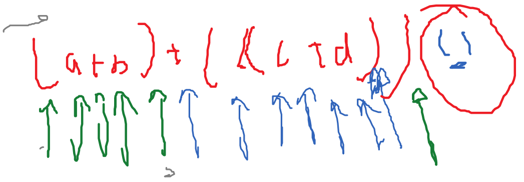


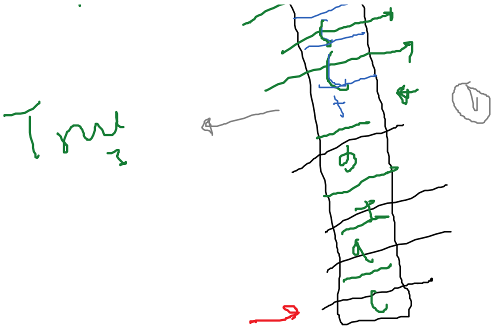


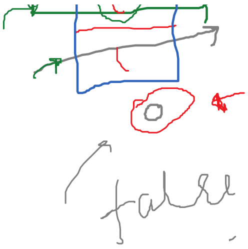
1. You are given a string exp representing an expression.
2. Assume that the expression is balanced i.e. the opening and closing brackets match with each other.
3. But, some of the pair of brackets maybe extra/needless.
4. You are required to print true if you detect extra brackets and false otherwise.

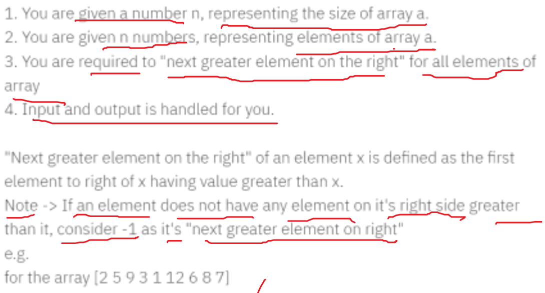
e.g.'
((a + b) + (c + d)) -> false
(a + b) + ((c + d)) -> true





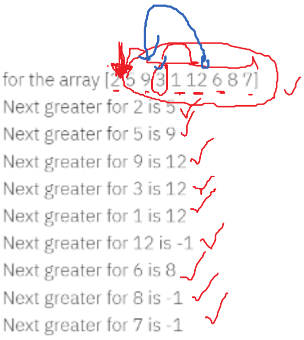






1. You are given a number n, representing the size of array a.
2. You are given n numbers, representing elements of array a.
3. You are required to "next greater element on the right" for all elements of array
4. Input and output is handled for you.

"Next greater element on the right" of an element x is defined as the first element to right of x having value greater than x.
Note -> If an element does not have any element on it's right side greater than it, consider -1 as it's "next greater element on right"
e.g.
for the array [2 5 9 3 1 12 6 8 7]



for the array [2 5 9 3 1 12 6 8 7]
Next greater for 2 is 5
Next greater for 5 is 9
Next greater for 9 is 12
Next greater for 3 is 12
Next greater for 1 is 12
Next greater for 12 is -1
Next greater for 6 is 8
Next greater for 8 is -1
Next greater for 7 is -1

[5, 9, 12, 12, 12, -1, 8, -1, -1]

[5, 3, 9, -2, 7]

[9, 9, -1, 7, -1]

7 > -2

-2 < stk.top

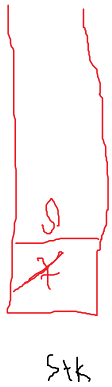
9 < 7

3 8 7 3

8 7 5

Ans. [-1, 7, -1, 9, 9]

[9, 9 -1, 7, -1]



[5, 3, 9, -2, 7]

-1	7	8	9	9

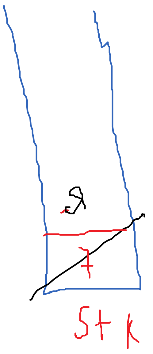
8 8 9 7 -1 ✓

7 > -2

9 > 7

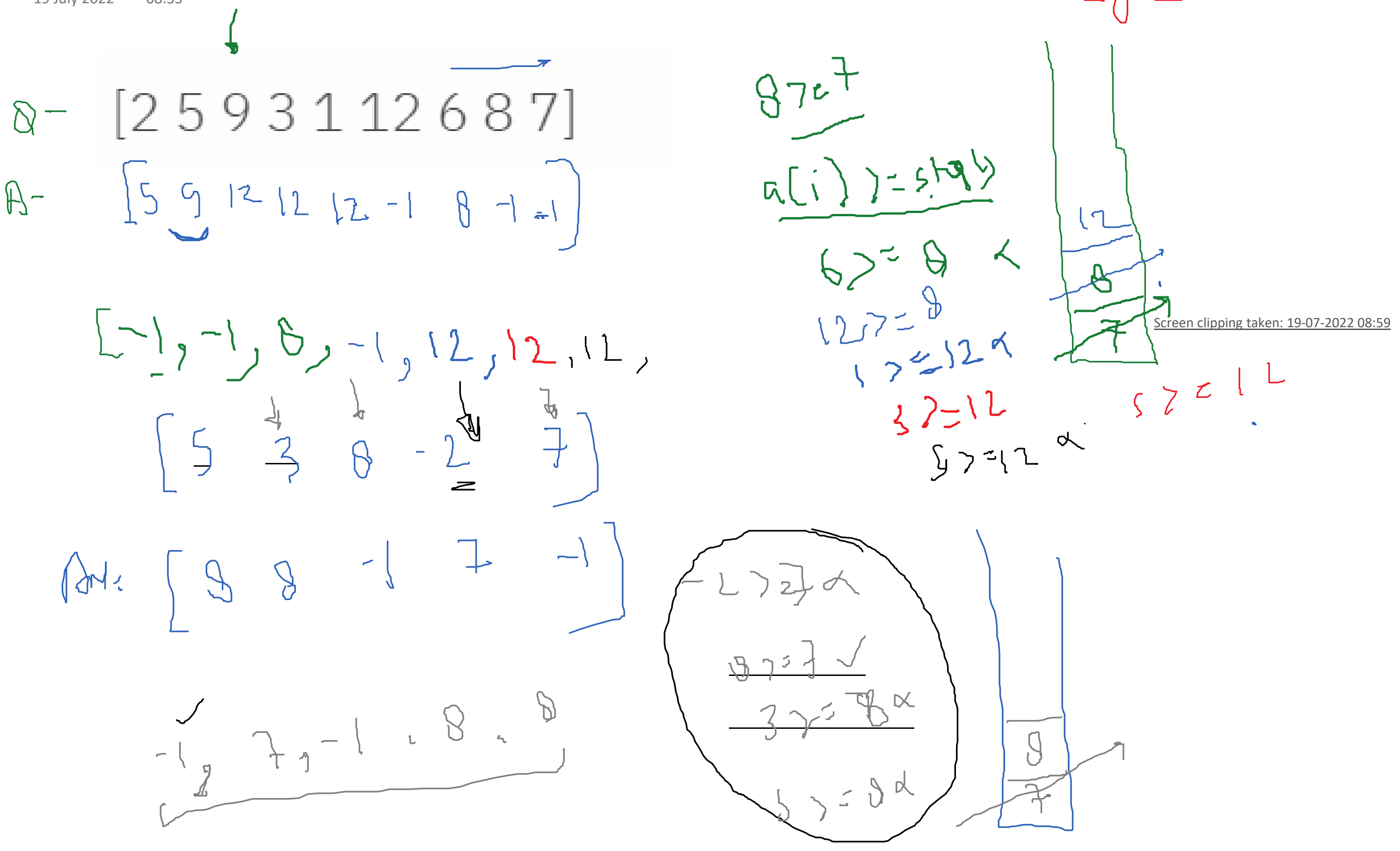
8 7 3

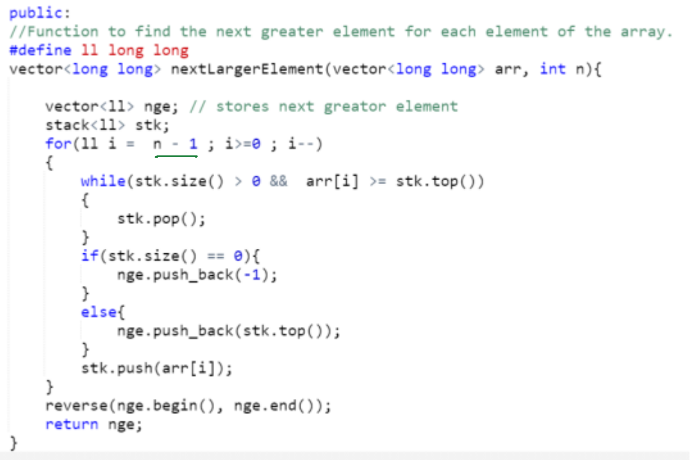
8 7 5



Stack and Queue 2

19 July 2022 08:33



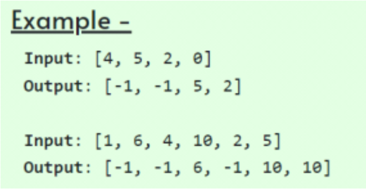


```
public:
//Function to find the next greater element for each element of the array.
#define ll long long
vector<long long> nextLargerElement(vector<long long> arr, int n){

    vector<ll> nge; // stores next greator element
    stack<ll> stk;
    for(ll i =  n - 1 ; i>=0 ; i--)
    {
        while(stk.size() > 0 &&  arr[i] >= stk.top())
        {
            stk.pop();
        }
        if(stk.size() == 0){
            nge.push_back(-1);
        }
        else{
            nge.push_back(stk.top());
        }
        stk.push(arr[i]);
    }
    reverse(nge.begin(), nge.end());
    return nge;
}
```

2 5 9 3 1 12 6 8 7

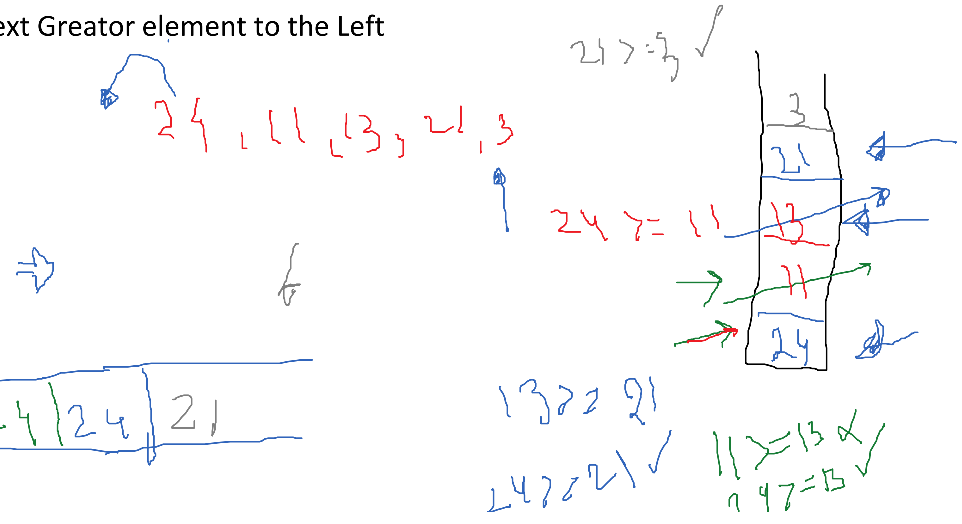
Next Greator element to the Left



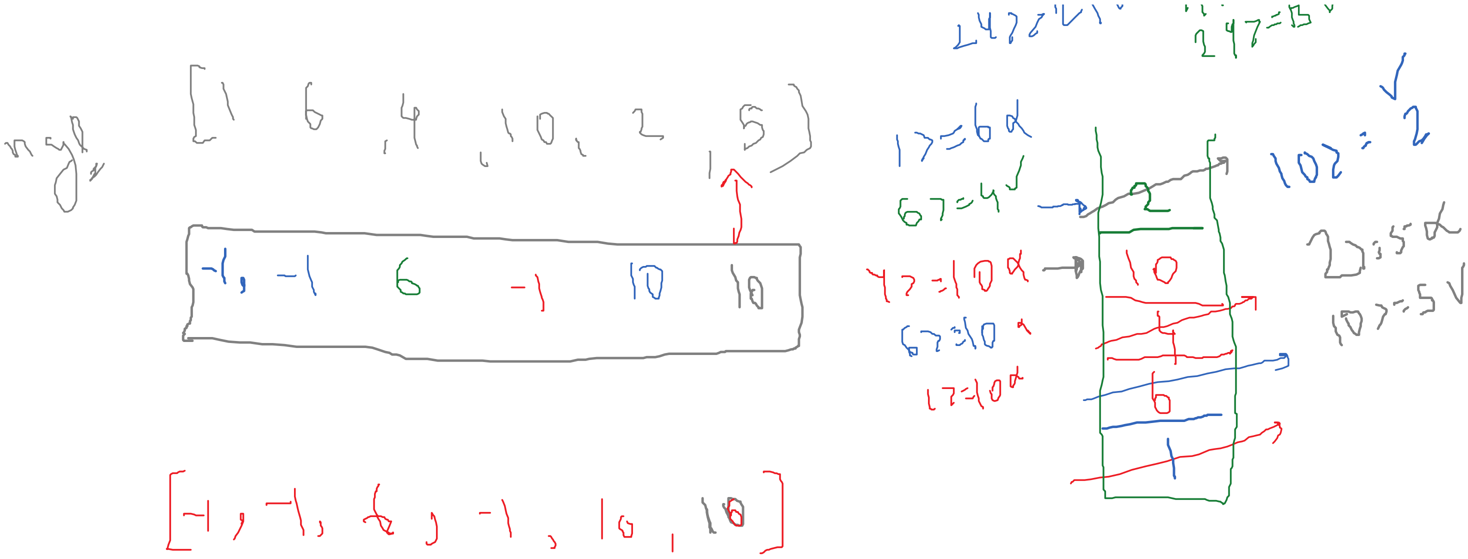
Example -

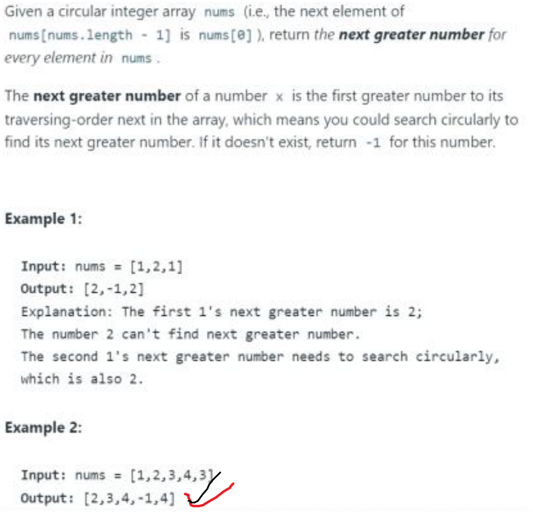
```
Input: [4, 5, 2, 0]
Output: [-1, -1, 5, 2]

Input: [1, 6, 4, 10, 2, 5]
Output: [-1, -1, 6, -1, 10, 10]
```



Ans – -1 | 24 | 24 | 24 | 21





Given a circular integer array `nums` (i.e., the next element of `nums[nums.length - 1]` is `nums[0]`), return *the* ***next greater number*** *for every element in* `nums`.

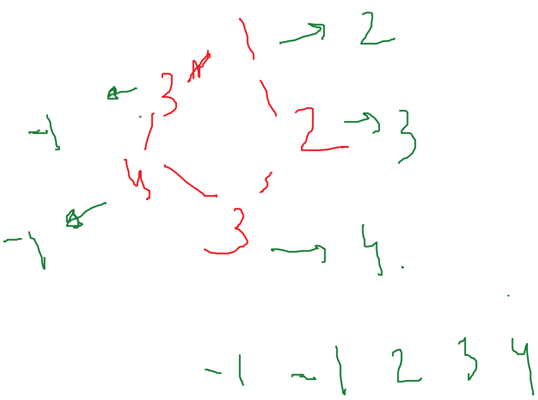
The **next greater number** of a number `x` is the first greater number to its traversing-order next in the array, which means you could search circularly to find its next greater number. If it doesn't exist, return `-1` for this number.

Example 1:

```
Input: nums = [1,2,1]
Output: [2,-1,2]
Explanation: The first 1's next greater number is 2;
The number 2 can't find next greater number.
The second 1's next greater number needs to search circularly,
which is also 2.
```

Example 2:

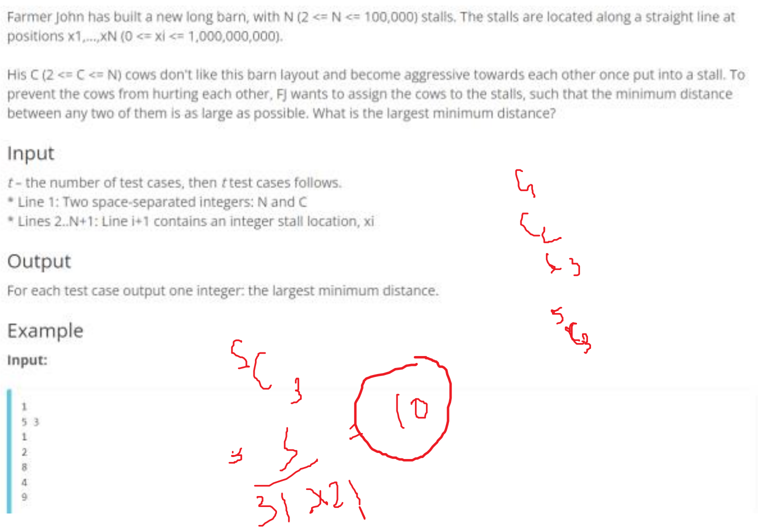
```
Input: nums = [1,2,3,4,3]
Output: [2,3,4,-1,4]
```



Stack and queue 3

20 July 2022 09:16

Screen clipping taken: 20-07-2022 09:16



Farmer John has built a new long barn, with N (2 <= N <= 100,000) stalls. The stalls are located along a straight line at positions x1,...,xN (0 <= xi <= 1,000,000,000).

His C (2 <= C <= N) cows don't like this barn layout and become aggressive towards each other once put into a stall. To prevent the cows from hurting each other, FJ wants to assign the cows to the stalls, such that the minimum distance between any two of them is as large as possible. What is the largest minimum distance?

Input

t – the number of test cases, then *t* test cases follows.
* Line 1: Two space-separated integers: N and C
* Lines 2..N+1: Line i+1 contains an integer stall location, xi

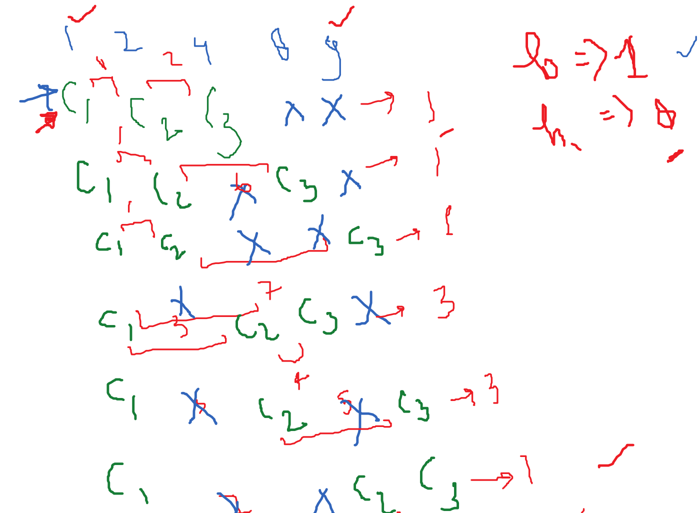
Output

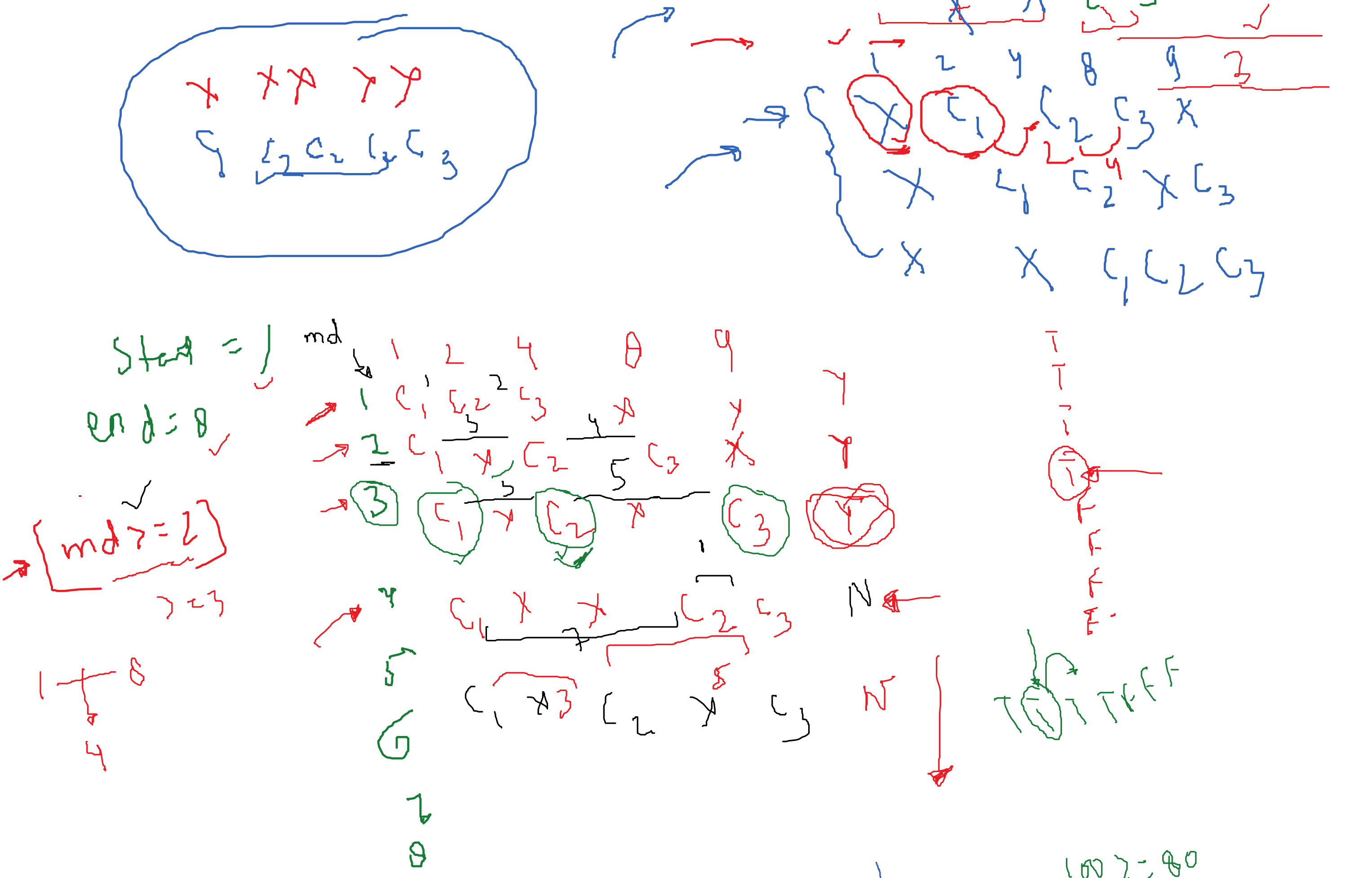
For each test case output one integer: the largest minimum distance.

Example

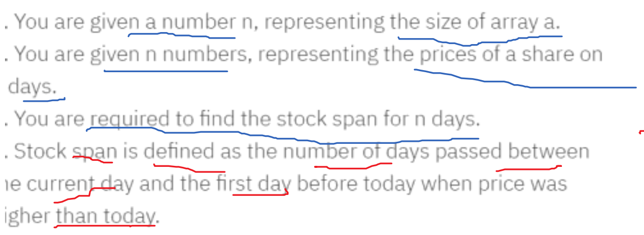
Input:

```
1
5 3
1
2
8
4
9
```

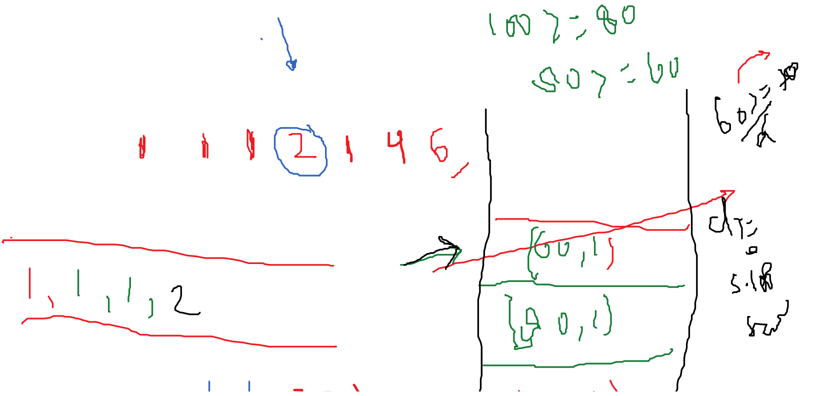


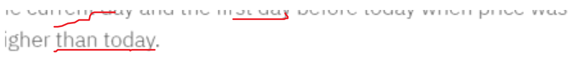


stock span



. You are given a number n, representing the size of array a.
. You are given n numbers, representing the prices of a share on days.
. You are required to find the stock span for n days.
. Stock span is defined as the number of days passed between ıe current day and the first day before today when price was igher than today.

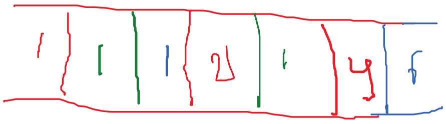


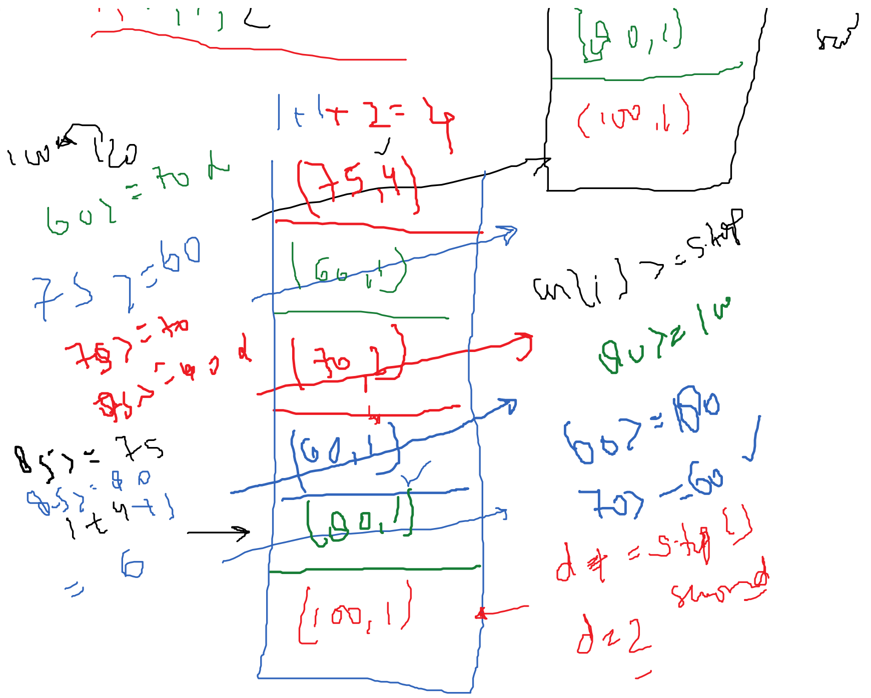


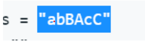
...e current day and the first day before today when price was higher than today.

d=1

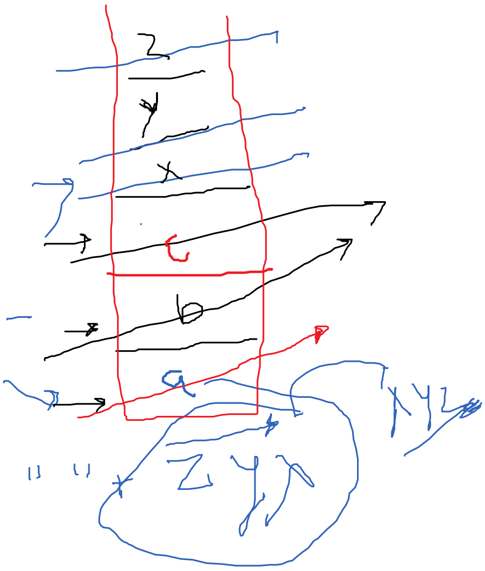
100 , 80, 60, 70, 60 , 75, 85





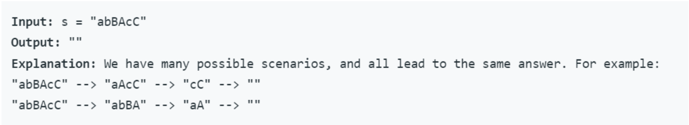


s = "abBAcC"

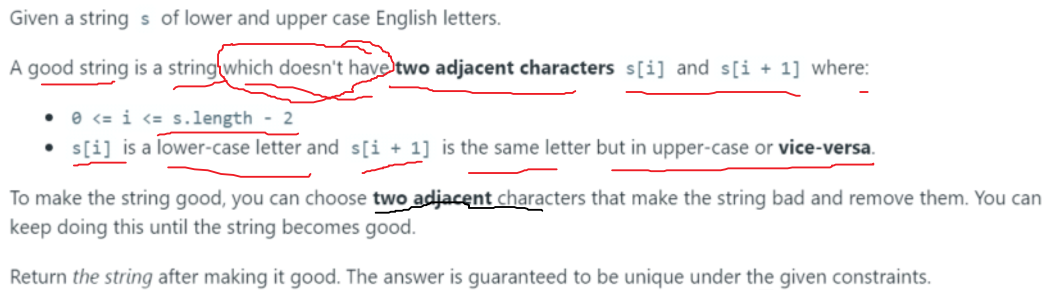


Stack and queue 4

21 July 2022 14:11



```
Input: s = "abBAcC"
Output: ""
Explanation: We have many possible scenarios, and all lead to the same answer. For example:
"abBAcC" --> "aAcC" --> "cC" --> ""
"abBAcC" --> "abBA" --> "aA" --> ""
```



Given a string `s` of lower and upper case English letters.

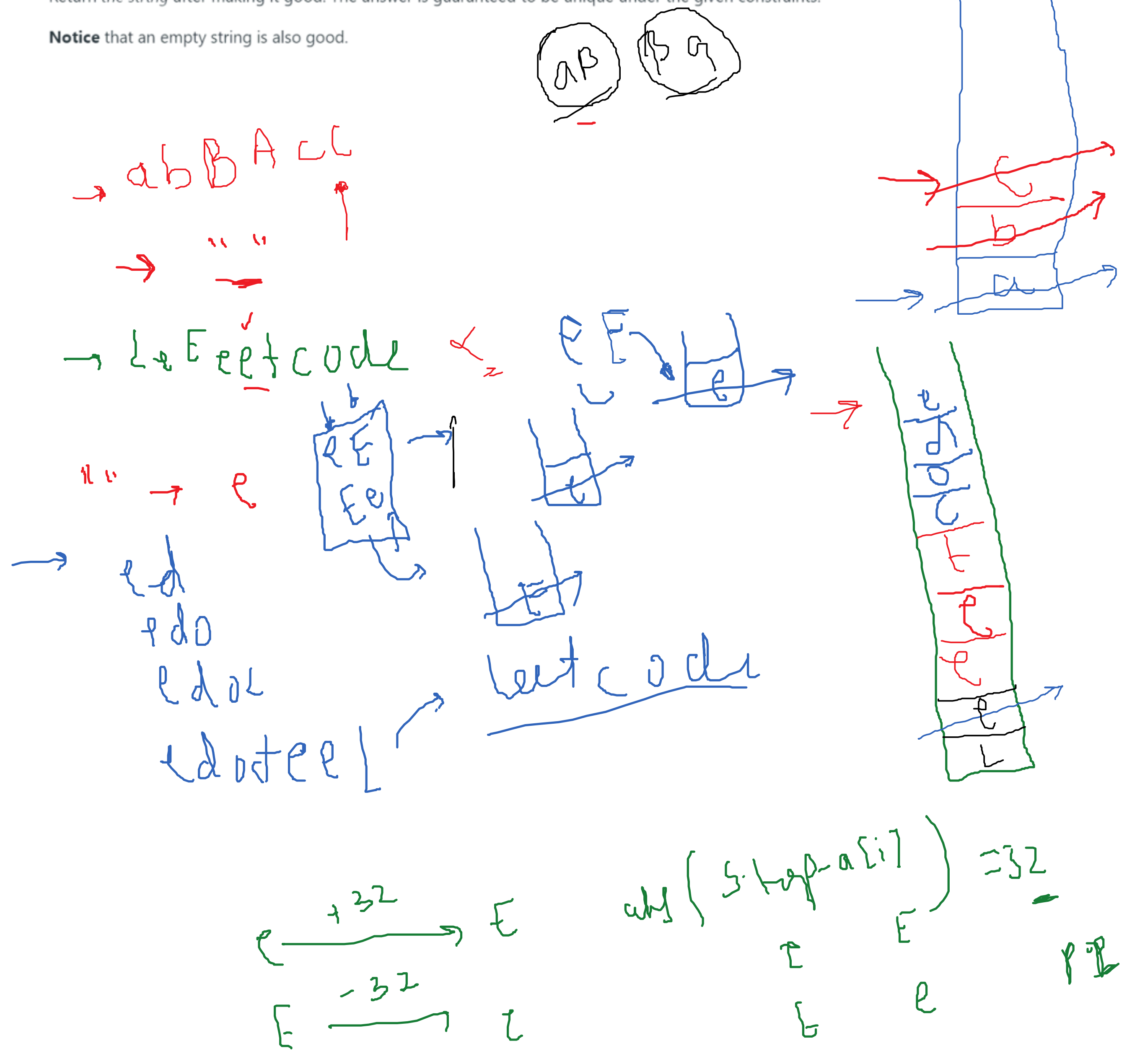
A good string is a string which doesn't have **two adjacent characters** `s[i]` and `s[i + 1]` where:

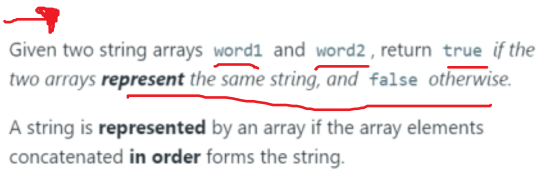
- `0 <= i <= s.length - 2`
- `s[i]` is a lower-case letter and `s[i + 1]` is the same letter but in upper-case or **vice-versa**.

To make the string good, you can choose **two adjacent** characters that make the string bad and remove them. You can keep doing this until the string becomes good.

Return *the string* after making it good. The answer is guaranteed to be unique under the given constraints.

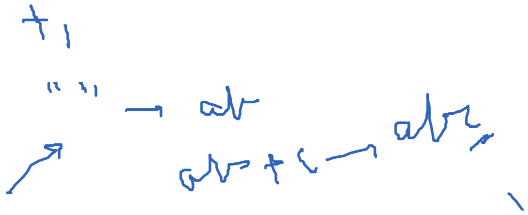
Notice that an empty string is also good.

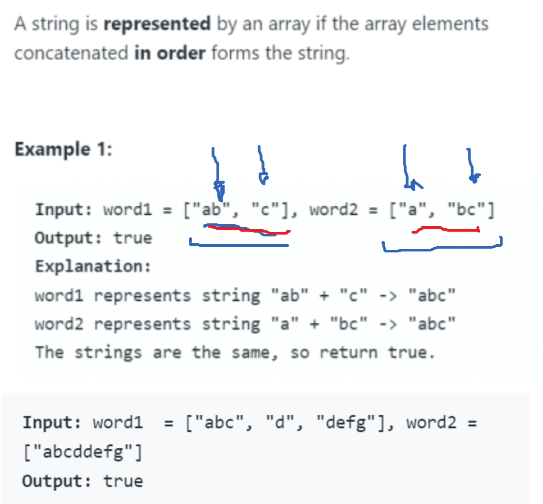




Given two string arrays `word1` and `word2`, return `true` *if the two arrays **represent** the same string, and* `false` *otherwise.*

A string is **represented** by an array if the array elements concatenated **in order** forms the string.



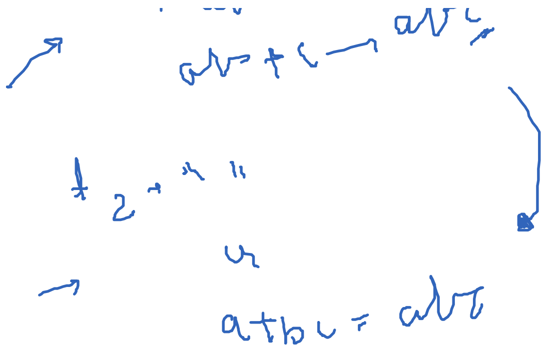


A string is **represented** by an array if the array elements concatenated **in order** forms the string.

Example 1:

```
Input: word1 = ["ab", "c"], word2 = ["a", "bc"]
Output: true
Explanation:
word1 represents string "ab" + "c" -> "abc"
word2 represents string "a" + "bc" -> "abc"
The strings are the same, so return true.
```

```
Input: word1  = ["abc", "d", "defg"], word2 =
["abcddefg"]
Output: true
```

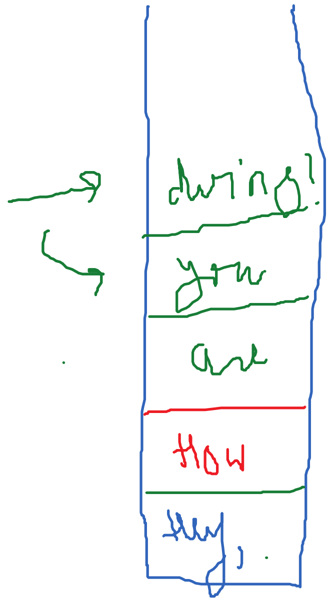


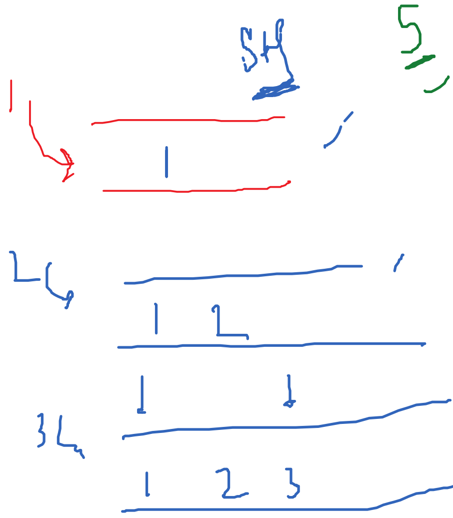
Reverse a string

ip - "Hey, How are you doing?"

op - "Doing? You are how hey,"

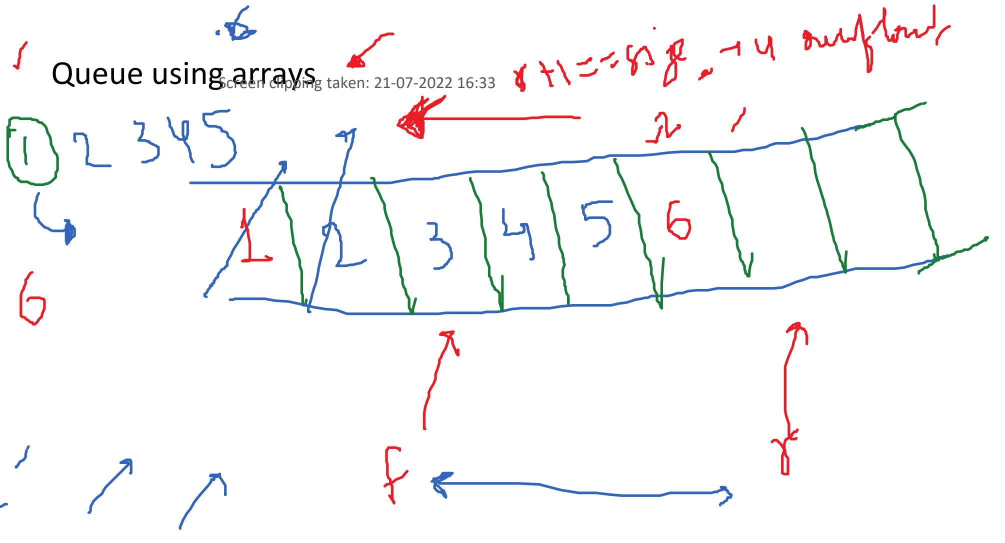
doing? _ you _ are _ how _ hey,

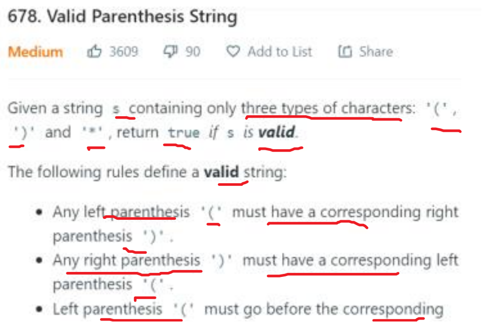




Queue using arrays

Screen clipping taken: 21-07-2022 16:33





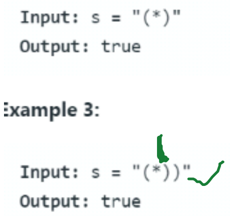
678. Valid Parenthesis String

Medium 3609 90 Add to List Share

Given a string `s` containing only three types of characters: `'('`, `')'` and `'*'`, return `true` *if* `s` *is **valid***.

The following rules define a **valid** string:

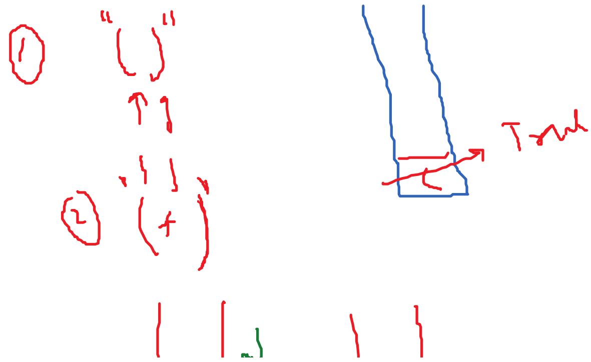
- Any left parenthesis `'('` must have a corresponding right parenthesis `')'`.
- Any right parenthesis `')'` must have a corresponding left parenthesis `'('`.
- Left parenthesis `'('` must go before the corresponding

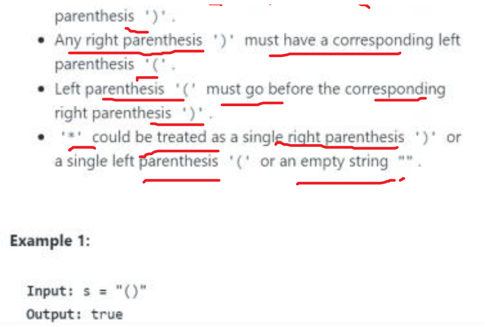


```
Input: s = "(*)"
Output: true
```

Example 3:

```
Input: s = "(*))"
Output: true
```



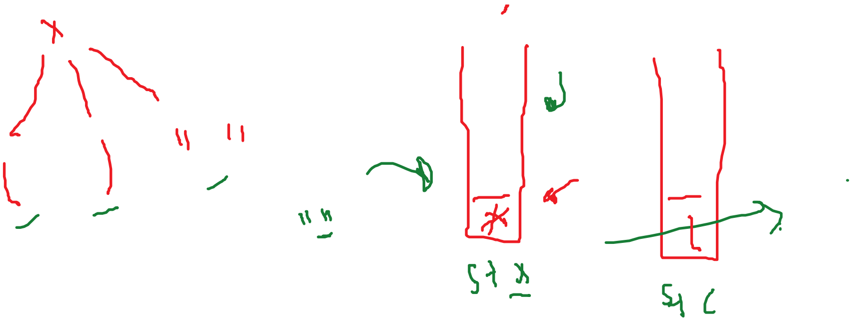


parenthesis ')' .

- Any right parenthesis ')' must have a corresponding left parenthesis '(' .
- Left parenthesis '(' must go before the corresponding right parenthesis ')' .
- '*' could be treated as a single right parenthesis ')' or a single left parenthesis '(' or an empty string "" .

Example 1:

```
Input: s = "()"
Output: true
```



Stack and queue 5

22 July 2022 14:02

Screen clipping taken: 22-07-2022 14:03

Largest Area Historgram

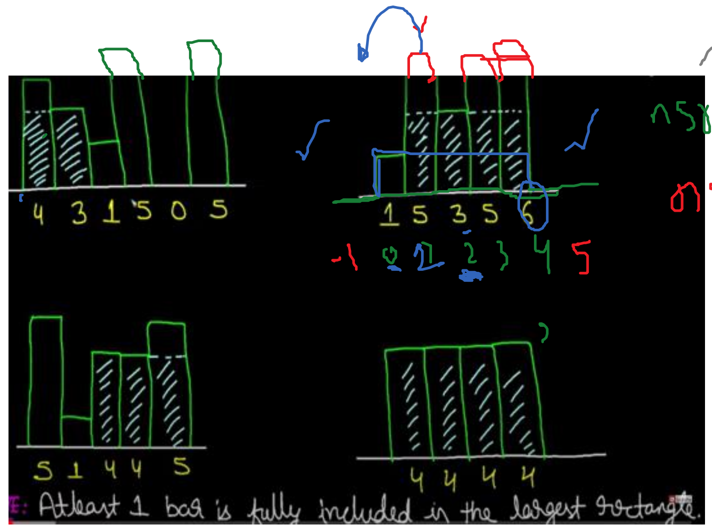
5+X-X

1 5 3 5 6

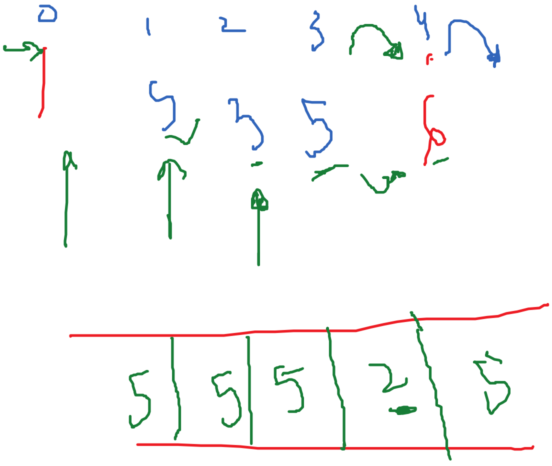
nsr = 5 2 5 5 5

nsl -1 0 0 2 3

6 1 4 2 1



Atleast 1 bar is fully included in the largest rectangle.



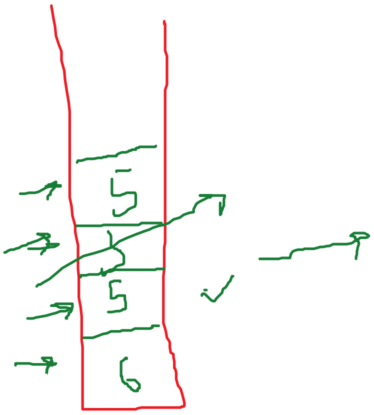
5 > 6 ✓

3 > 5 X

5 3 ✓

5 1 X

1 > 5 X



Postfix , prefix and infix

ab- -ab a-b

middle

1 → [ ], { }, { } L-R

2 → ^ R-L

3 → *, / L-R

4 → -, +

→ (4x2+3)

Prefix

oper, operands

Postfix L-R

→ (4 * 2 + 3)

Prefix

oper, operands

(*42)+3

+*423

Postfix

(42*)+3

→ 42*3+ ✓

→ 5 - 6/3

(-5/63)

5 - 63/

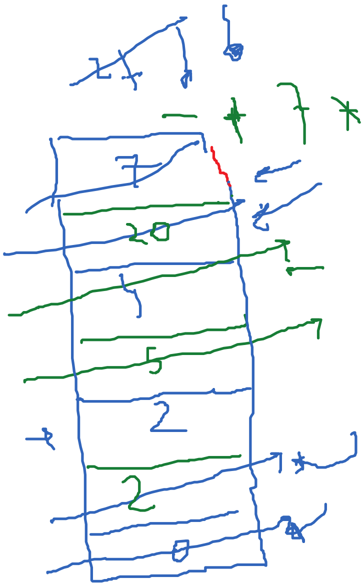
563/- ✓

0-9)

- + 7 * 4 5 + 2 0

Infix: ?

25



op1	2	4	7	27
	+	*	+	-
op2	0	5	20	2

2-5

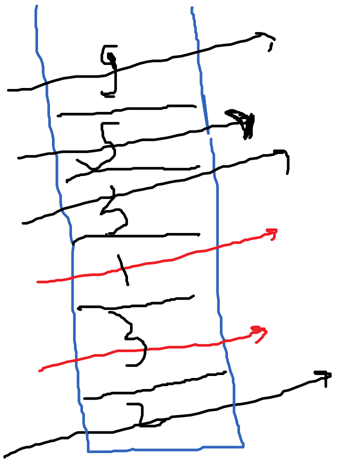
Postfix evalutaion

"2 3 1 * + 9"

1

op1 :	1	3	9
op2 :	3	2	5

4



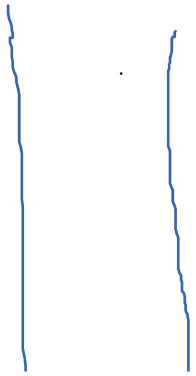
Infix to postfix

(a - b/c) * (a/k - l) =>

op → print

(→ push stk

... we get '(' ... pop print



(→ push STK

) → pop untill we got (pop

operator - if top prec > op. prec pop
with (prec

[ ]

^

*/

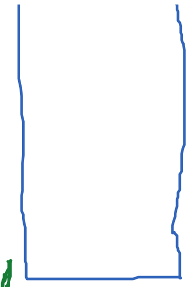
+-

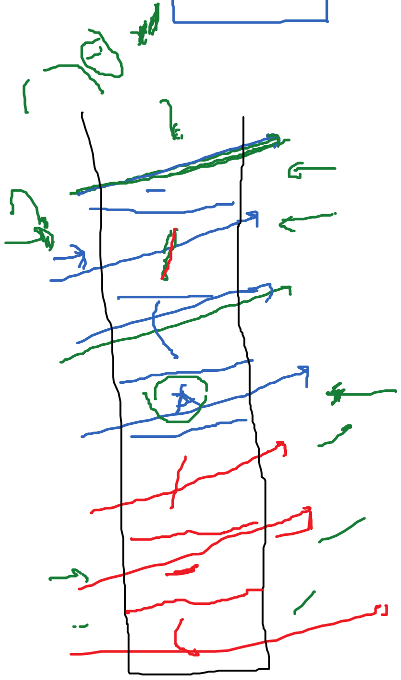
$(a - b/c) * (a/k - l) \Rightarrow$

/> = stop

≠ < = |

"abc/-ak/l-*

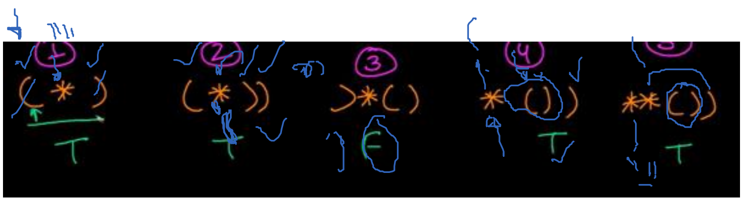


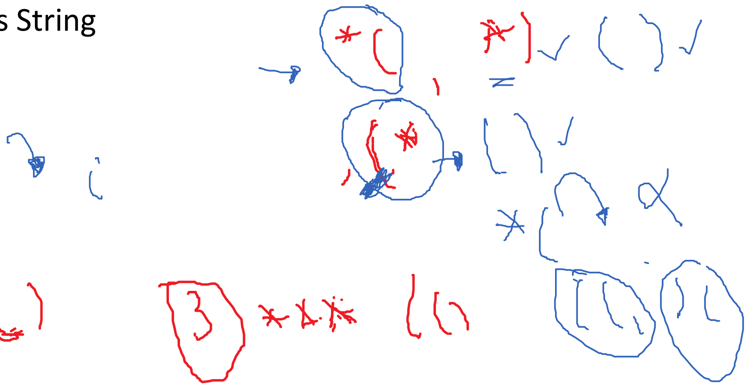


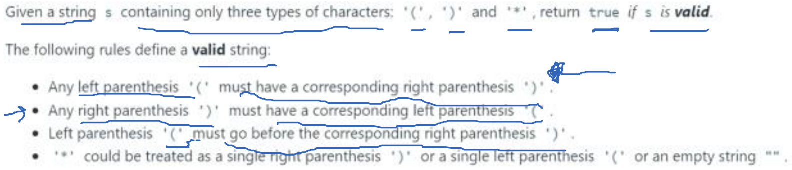
Stack and queue 6

22 July 2022 23:54

Valid Parenthesis String



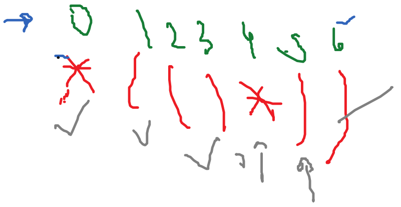


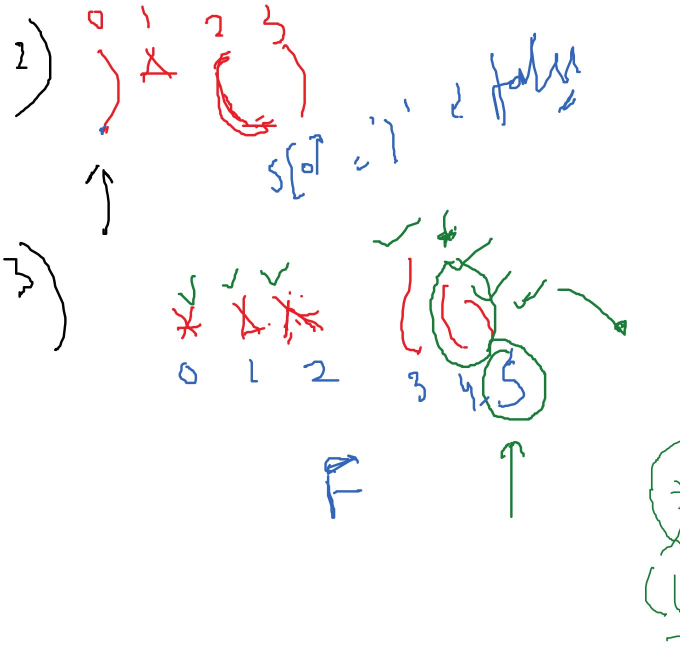


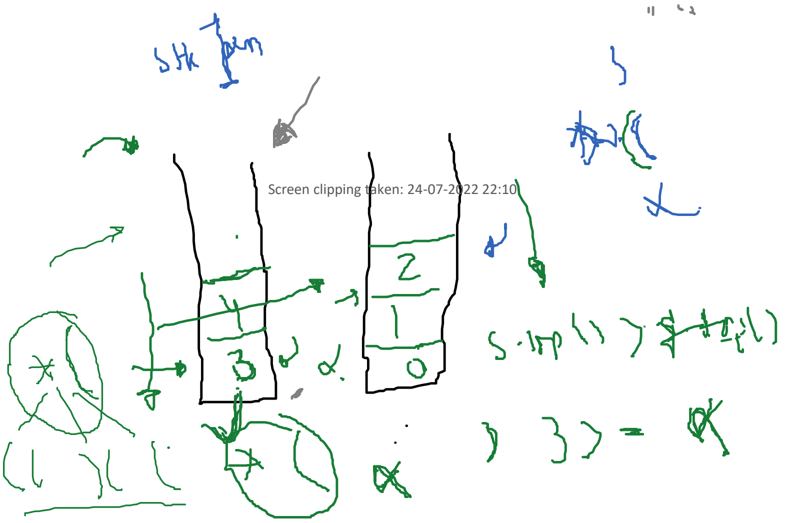
Given a string s containing only three types of characters: '(', ')' and '*', return true if s is ***valid***.

The following rules define a **valid** string:

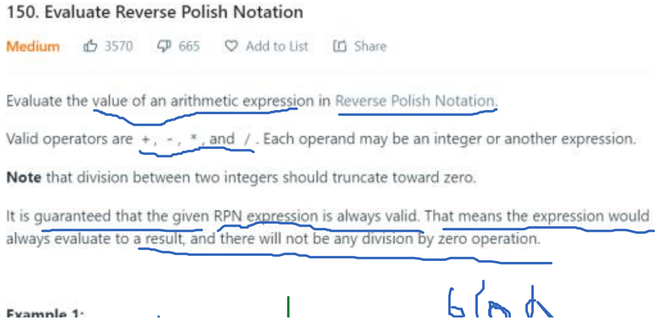
- Any left parenthesis '(' must have a corresponding right parenthesis ')'.
- Any right parenthesis ')' must have a corresponding left parenthesis '('.
- Left parenthesis '(' must go before the corresponding right parenthesis ')'.
- '*' could be treated as a single right parenthesis ')' or a single left parenthesis '(' or an empty string "".







Screen clipping taken: 24-07-2022 22:10



150. Evaluate Reverse Polish Notation

Medium 3570 665 Add to List Share

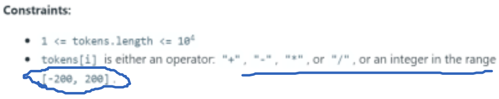
Evaluate the value of an arithmetic expression in Reverse Polish Notation.

Valid operators are +, -, *, and /. Each operand may be an integer or another expression.

Note that division between two integers should truncate toward zero.

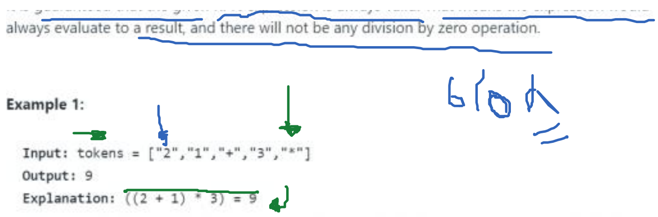
It is guaranteed that the given RPN expression is always valid. That means the expression would always evaluate to a result, and there will not be any division by zero operation.

Example 1:



Constraints:

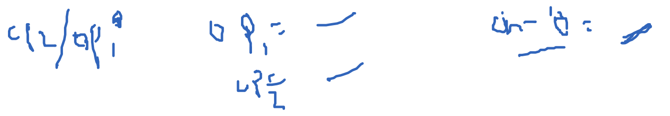
- $1 <= tokens.length <= 10^4$
- tokens[i] is either an operator: "+", "-", "*", or "/", or an integer in the range [-200, 200].

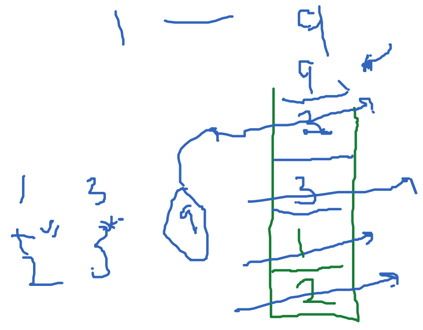


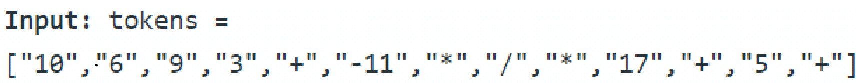
always evaluate to a result, and there will not be any division by zero operation.

Example 1:

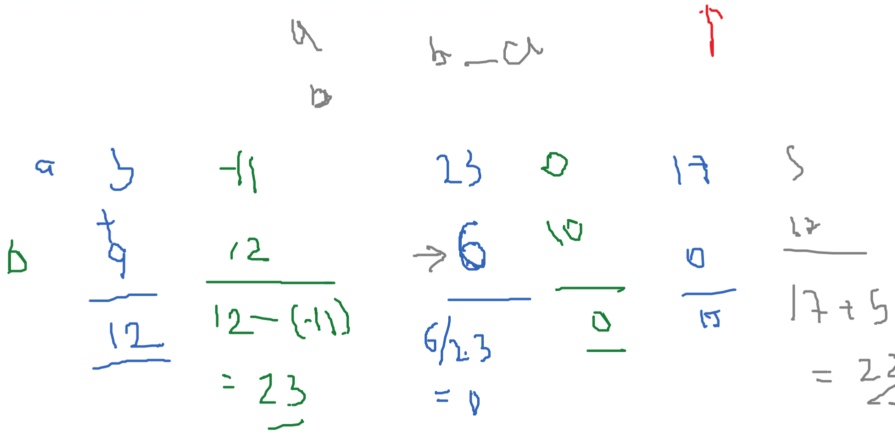
```
Input: tokens = ["2","1","+","3","*"]
Output: 9
Explanation: ((2 + 1) * 3) = 9
```

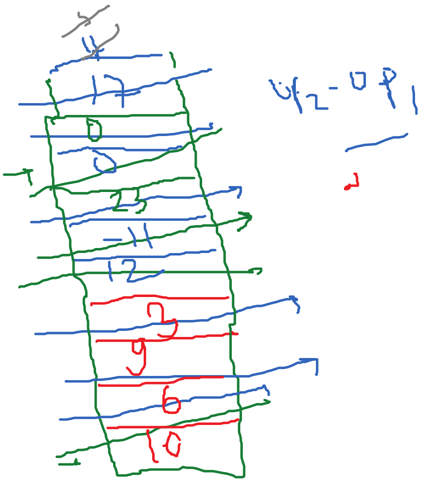


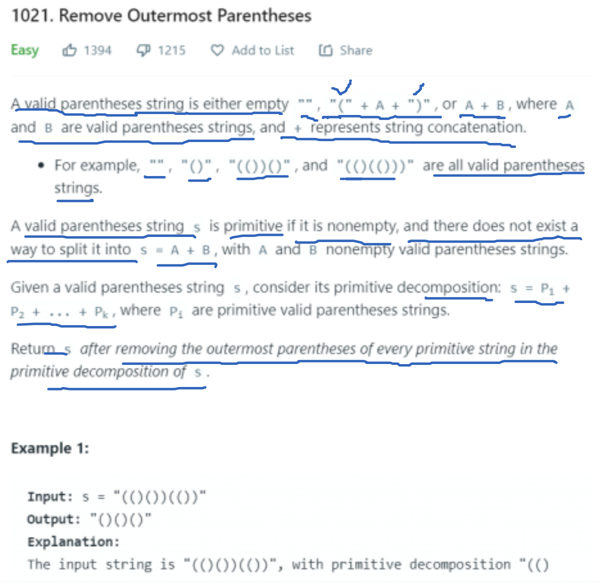




```
Input: tokens =
["10","6","9","3","+","-11","*","/","*","17","+","5","+"]
```







1021. Remove Outermost Parentheses

Easy 1394 1215 Add to List Share

A valid parentheses string is either empty `""`, `"(" + A + ")"`, or `A + B`, where `A` and `B` are valid parentheses strings, and `+` represents string concatenation.

- For example, `""`, `"()"`, `"(())()"`, and `"(()(()))"` are all valid parentheses strings.

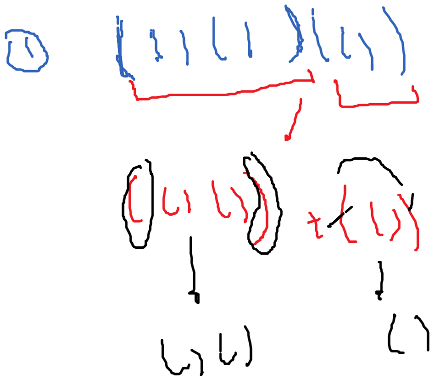
A valid parentheses string `s` is primitive if it is nonempty, and there does not exist a way to split it into `s = A + B`, with `A` and `B` nonempty valid parentheses strings.

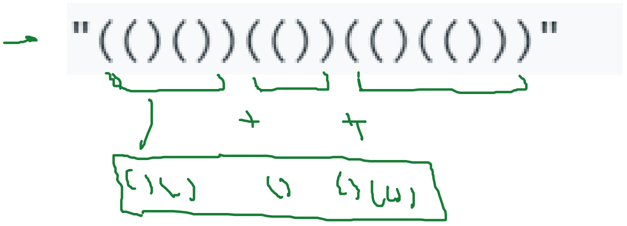
Given a valid parentheses string `s`, consider its primitive decomposition: $s = P_1 + P_2 + \dots + P_k$, where P_i are primitive valid parentheses strings.

Return `s` *after removing the outermost parentheses of every primitive string in the primitive decomposition of* `s`.

Example 1:

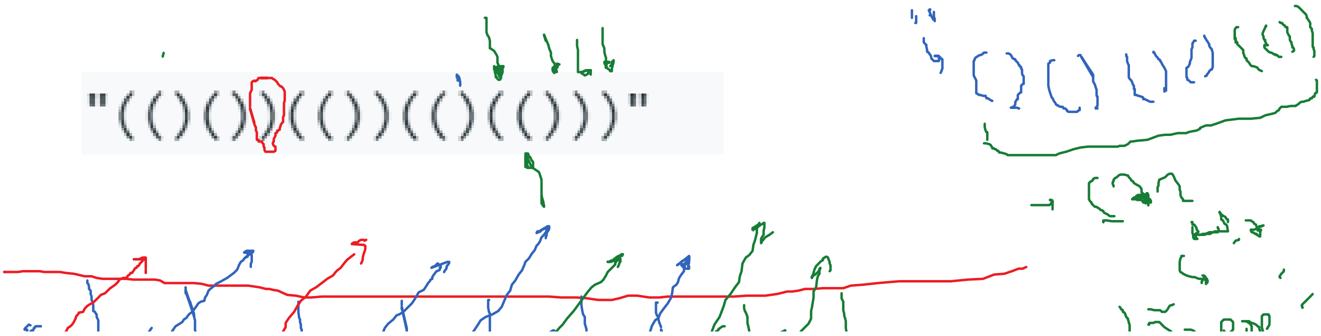
```
Input: s = "(()())(())"
Output: "()()()"
Explanation:
The input string is "(()())(())", with primitive decomposition "(()
```



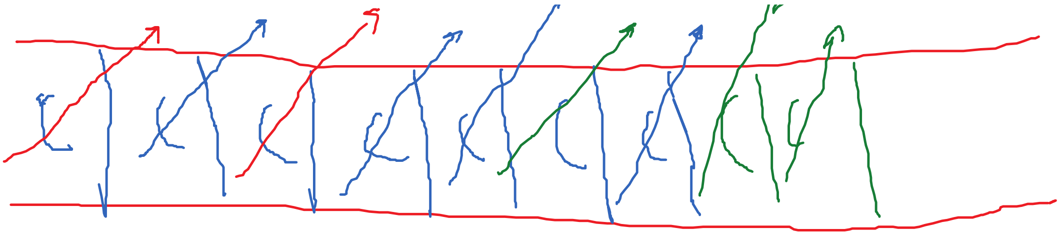


"(()())(())(()(()))"

i<n



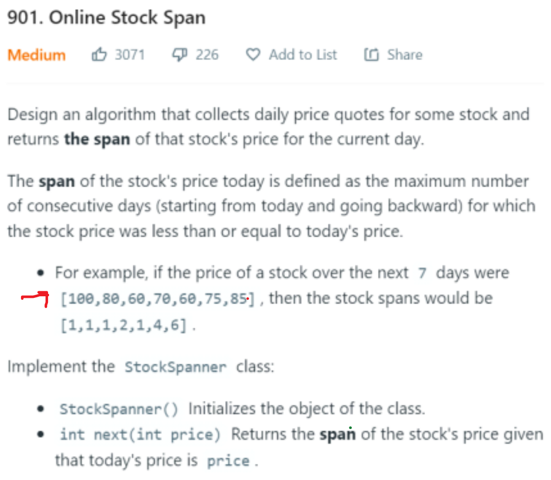
"(()())(())(()(()))"



and t=

Stack and queue 7

24 July 2022   22:51



901. Online Stock Span

Medium   3071   226   Add to List   Share

Design an algorithm that collects daily price quotes for some stock and returns **the span** of that stock's price for the current day.

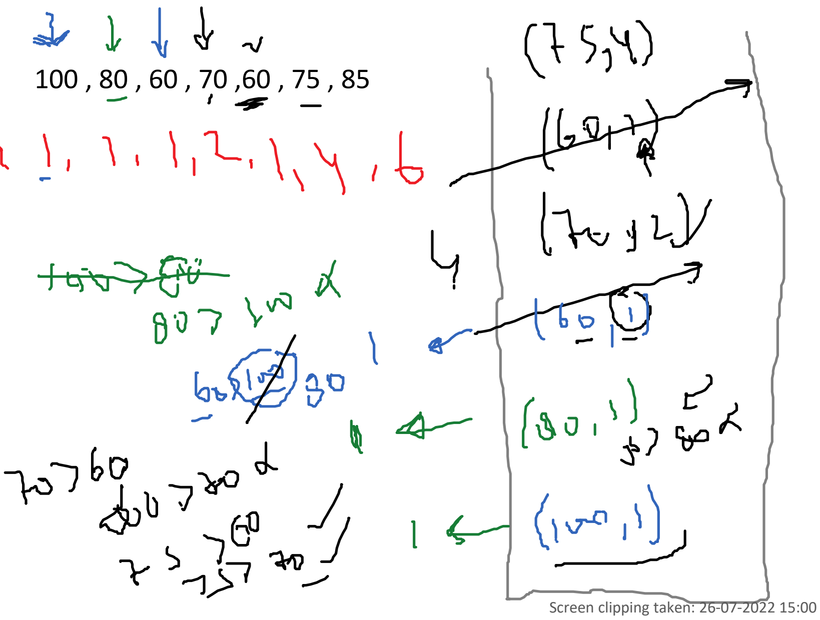
The **span** of the stock's price today is defined as the maximum number of consecutive days (starting from today and going backward) for which the stock price was less than or equal to today's price.

- For example, if the price of a stock over the next `7` days were `[100,80,60,70,60,75,85]`, then the stock spans would be `[1,1,1,2,1,4,6]`.

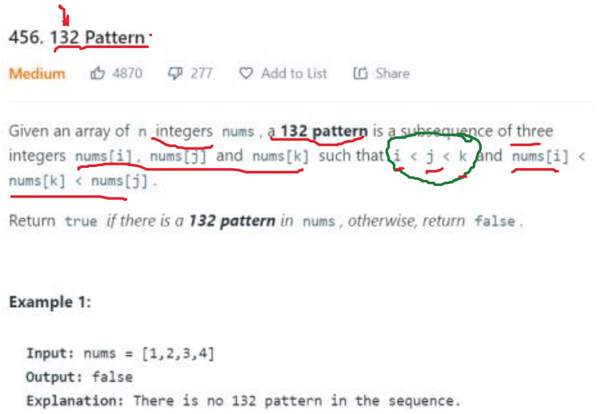
Implement the `StockSpanner` class:

- `StockSpanner()` Initializes the object of the class.
- `int next(int price)` Returns the **span** of the stock's price given that today's price is `price`.

1+ 1+2=4



Screen clipping taken: 26-07-2022 15:00



456. 132 Pattern

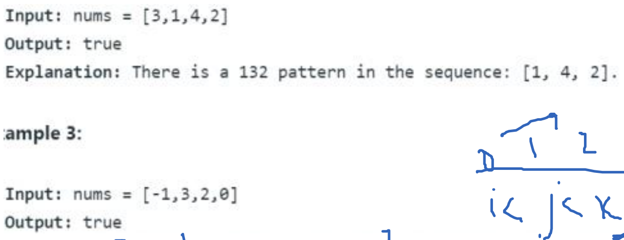
Medium   4870   277   Add to List   Share

Given an array of `n` integers `nums`, a **132 pattern** is a subsequence of three integers `nums[i]`, `nums[j]` and `nums[k]` such that `i < j < k` and `nums[i] < nums[k] < nums[j]`.

Return `true` *if there is a* ***132 pattern*** *in* `nums`*, otherwise, return* `false`.

Example 1:

```
Input: nums = [1,2,3,4]
Output: false
Explanation: There is no 132 pattern in the sequence.
```



```
Input: nums = [3,1,4,2]
Output: true
Explanation: There is a 132 pattern in the sequence: [1, 4, 2].
```

ample 3:

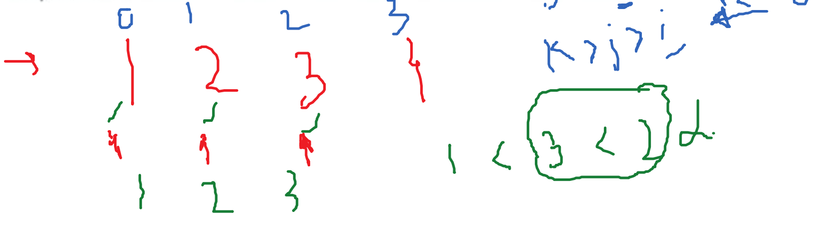
```
Input: nums = [-1,3,2,0]
Output: true
```

0 1 2

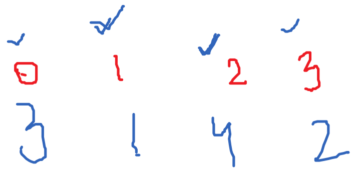
i < j < k

k > j > i

R to L



123 _ 234
124
134
brute force



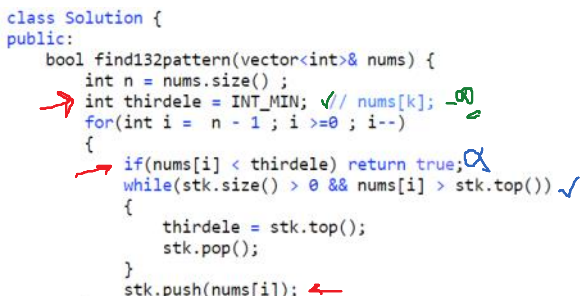
$n[i] < n[k] < n[j]$

$n[k] < n[j]$

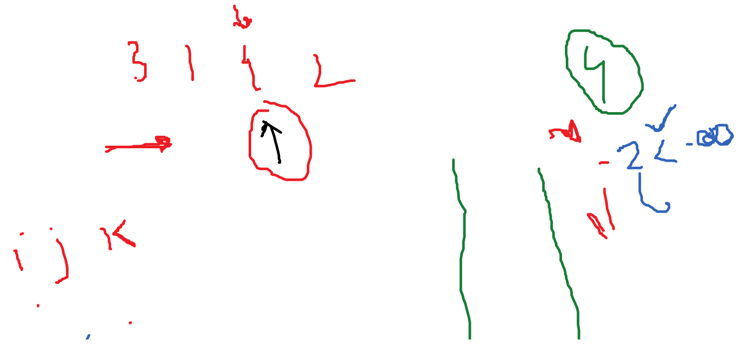
$n[k] < n[j]$

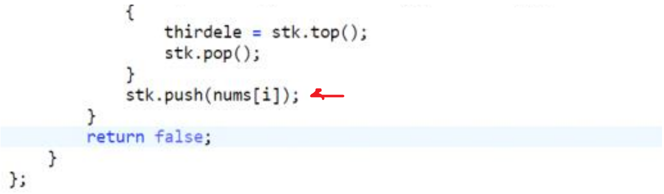
2 < 4 ✓

2 > 1 ✓



```
class Solution {
public:
    bool find132pattern(vector<int>& nums) {
        int n = nums.size() ;
        int thirdele = INT_MIN; // nums[k];
        for(int i =  n - 1 ; i >=0 ; i--)
        {
            if(nums[i] < thirdele) return true;
            while(stk.size() > 0 && nums[i] > stk.top())
            {
                thirdele = stk.top();
                stk.pop();
            }
            stk.push(nums[i]);
```

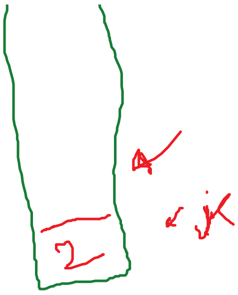


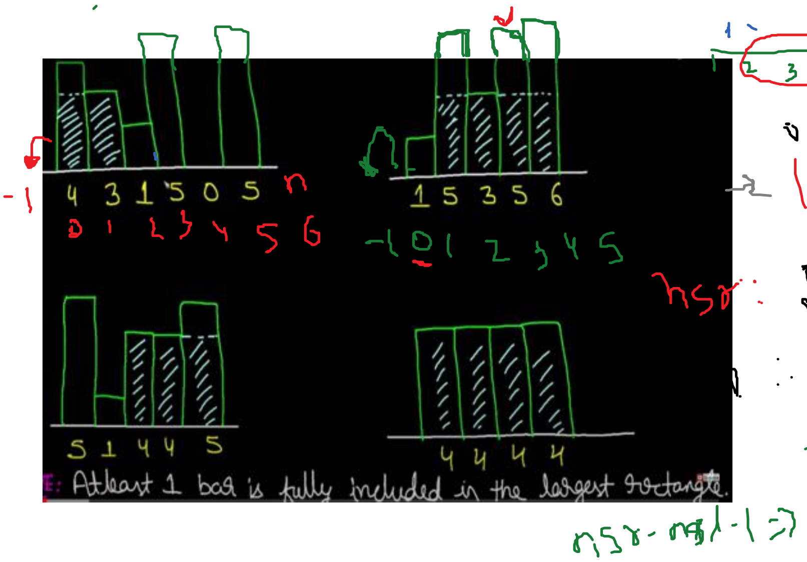


```
            {
                thirdele = stk.top();
                stk.pop();
            }
            stk.push(nums[i]);
        }
        return false;
    }
};
```

k<j

k>i





Atleast 1 box is fully included in the largest rectangle.

1 2 3 4 5 6

0 1 2 3 4

1 5 3 5 6

nsr: 5 2 5 5 5

nsl: -1 0 0 2 3

5 1 4 2 1

nsr - nsl - 1 =>

→ 1 5 3 5 6

0 1 2 3 4 5

nsr: 5 5 5 2 5

→ 5 2 5 5 5

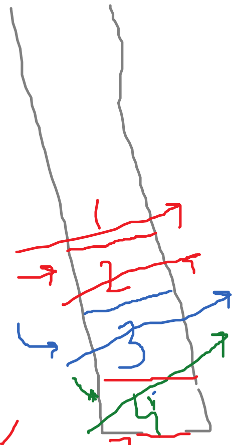
a[4] 6 >= 5

a[3] 5 >= 3

a[2] 3 >= 5 ✗

a[1] - 5 >= 1 ✓

a[2] : 3 >= 1



nsl:

0 1 2 3 4

→ 1 5 3 5 6

nsl: -1 -1 1 1

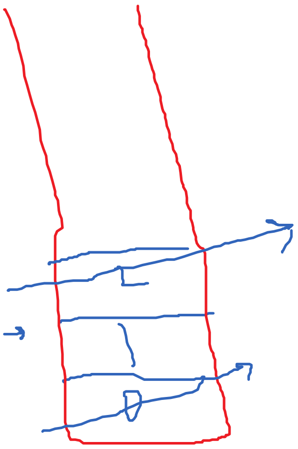
a[0] 5

1 >= 5 ✗

5 >= 3

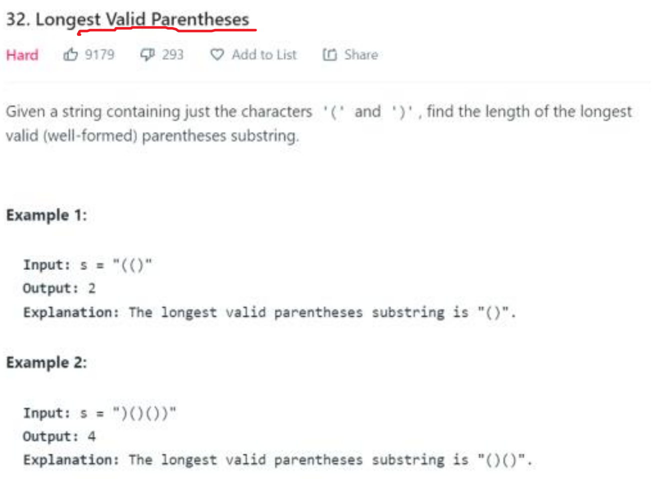
3 >= 5 ✗

5 >= 5 ✓



Stack and queue 8

27 July 2022 14:14



32. Longest Valid Parentheses

Hard 9179 293 Add to List Share

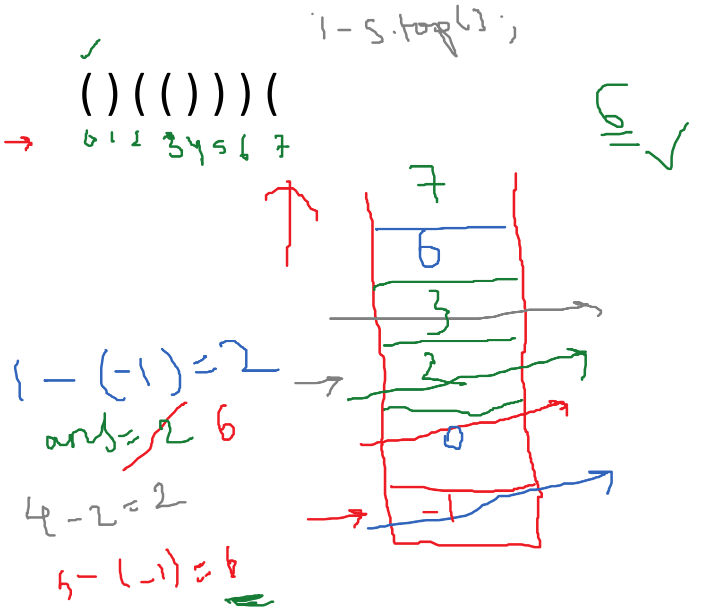
Given a string containing just the characters `'('` and `')'`, find the length of the longest valid (well-formed) parentheses substring.

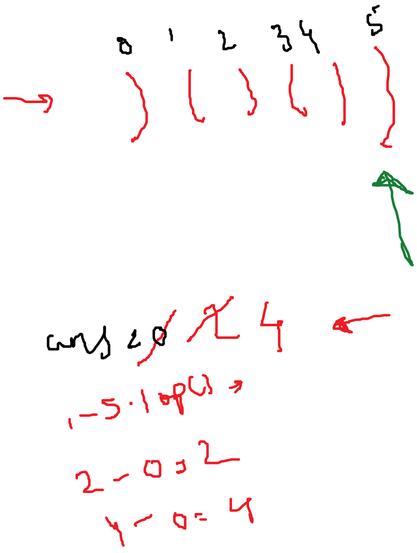
Example 1:

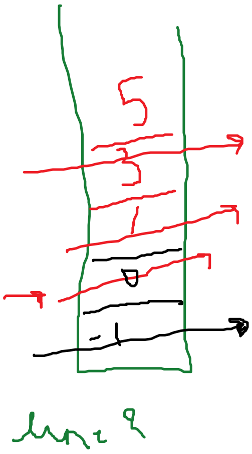
```
Input: s = "(()"
Output: 2
Explanation: The longest valid parentheses substring is "()".
```

Example 2:

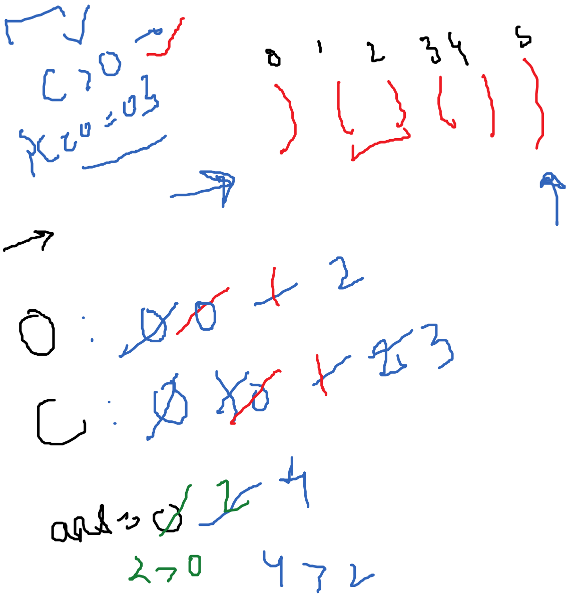
```
Input: s = ")()())"
Output: 4
Explanation: The longest valid parentheses substring is "()()".
```

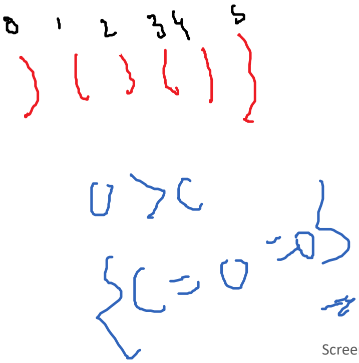




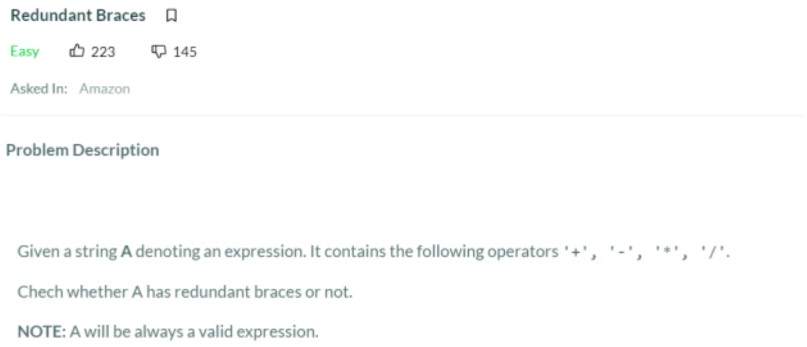


Opening == closing





Screen clipping taken: 27-07-2022 15:23



Redundant Braces

Easy 223 145

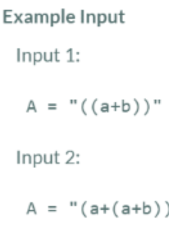
Asked In: Amazon

Problem Description

Given a string **A** denoting an expression. It contains the following operators '+', '-', '*', '/'.

Chech whether A has redundant braces or not.

NOTE: A will be always a valid expression.



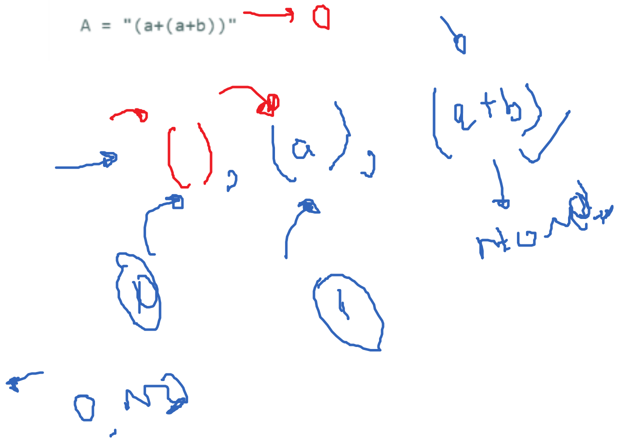
Example Input

Input 1:

```
A = "((a+b))"
```

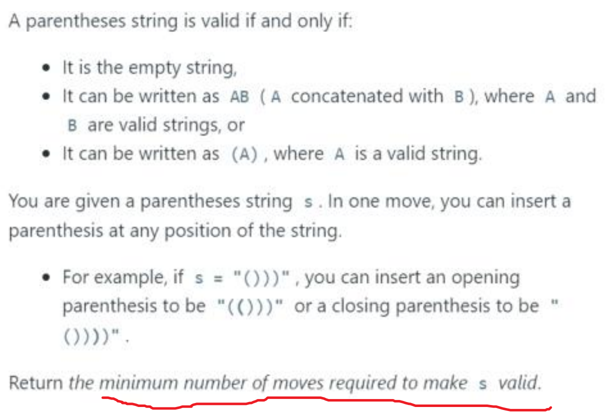
Input 2:

```
A = "(a+(a+b))"
```



"((a+b)+(e+(c*d)))/(a+c)"

Screen clipping taken: 27-07-2022 15:47



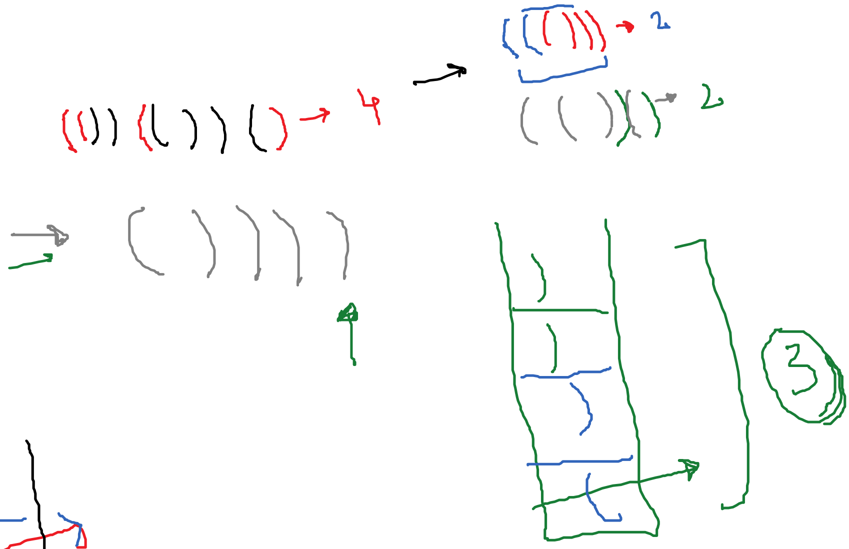
A parentheses string is valid if and only if:

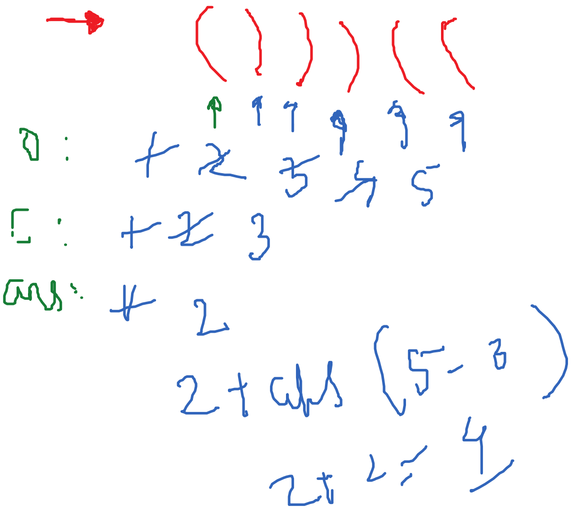
- It is the empty string,
- It can be written as `AB` (`A` concatenated with `B`), where `A` and `B` are valid strings, or
- It can be written as `(A)`, where `A` is a valid string.

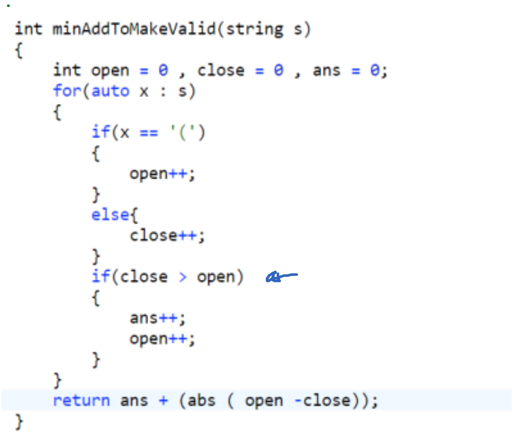
You are given a parentheses string `s`. In one move, you can insert a parenthesis at any position of the string.

- For example, if `s = "()))"`, you can insert an opening parenthesis to be `"(()))"` or a closing parenthesis to be `"())))"`.

Return *the minimum number of moves required to make* `s` *valid.*



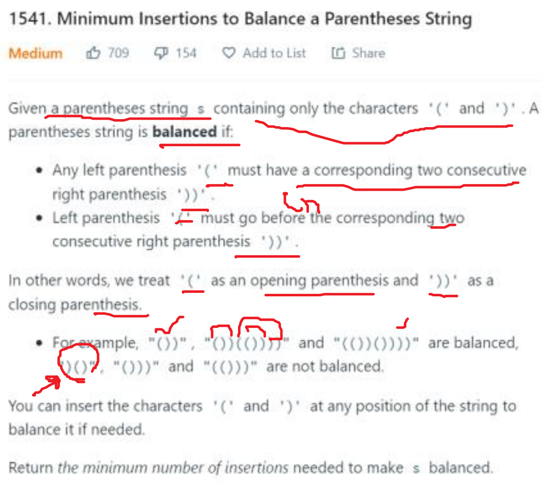




```
int minAddToMakeValid(string s)
{
    int open = 0 , close = 0 , ans = 0;
    for(auto x : s)
    {
        if(x == '(')
        {
            open++;
        }
        else{
            close++;
        }
        if(close > open)
        {
            ans++;
            open++;
        }
    }
    return ans + (abs ( open -close));
}
```

Stack And queue 9

30 July 2022 14:38



1541. Minimum Insertions to Balance a Parentheses String

Medium 👍 709 👎 154 ♡ Add to List ⇗ Share

Given a parentheses string `s` containing only the characters `'('` and `')'`. A parentheses string is **balanced** if:

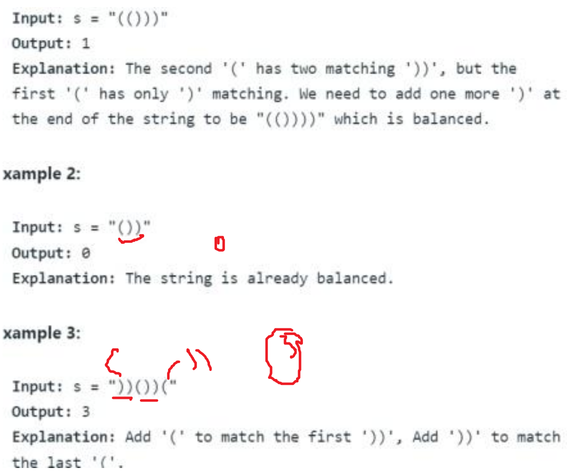
- Any left parenthesis `'('` must have a corresponding two consecutive right parenthesis `'))'`.
- Left parenthesis `'('` must go before the corresponding two consecutive right parenthesis `'))'`.

In other words, we treat `'('` as an opening parenthesis and `'))'` as a closing parenthesis.

- For example, `"())"`, `"())(())))"` and `"(())())))"` are balanced, `")()"`, `"()))"` and `"(()))"` are not balanced.

You can insert the characters `'('` and `')'` at any position of the string to balance it if needed.

Return *the minimum number of insertions* needed to make `s` balanced.



```
Input: s = "(()))"
Output: 1
Explanation: The second '(' has two matching '))', but the
first '(' has only ')' matching. We need to add one more ')' at
the end of the string to be "(())))" which is balanced.
```

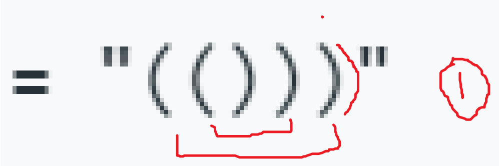
xample 2:

```
Input: s = "())"
Output: 0
Explanation: The string is already balanced.
```

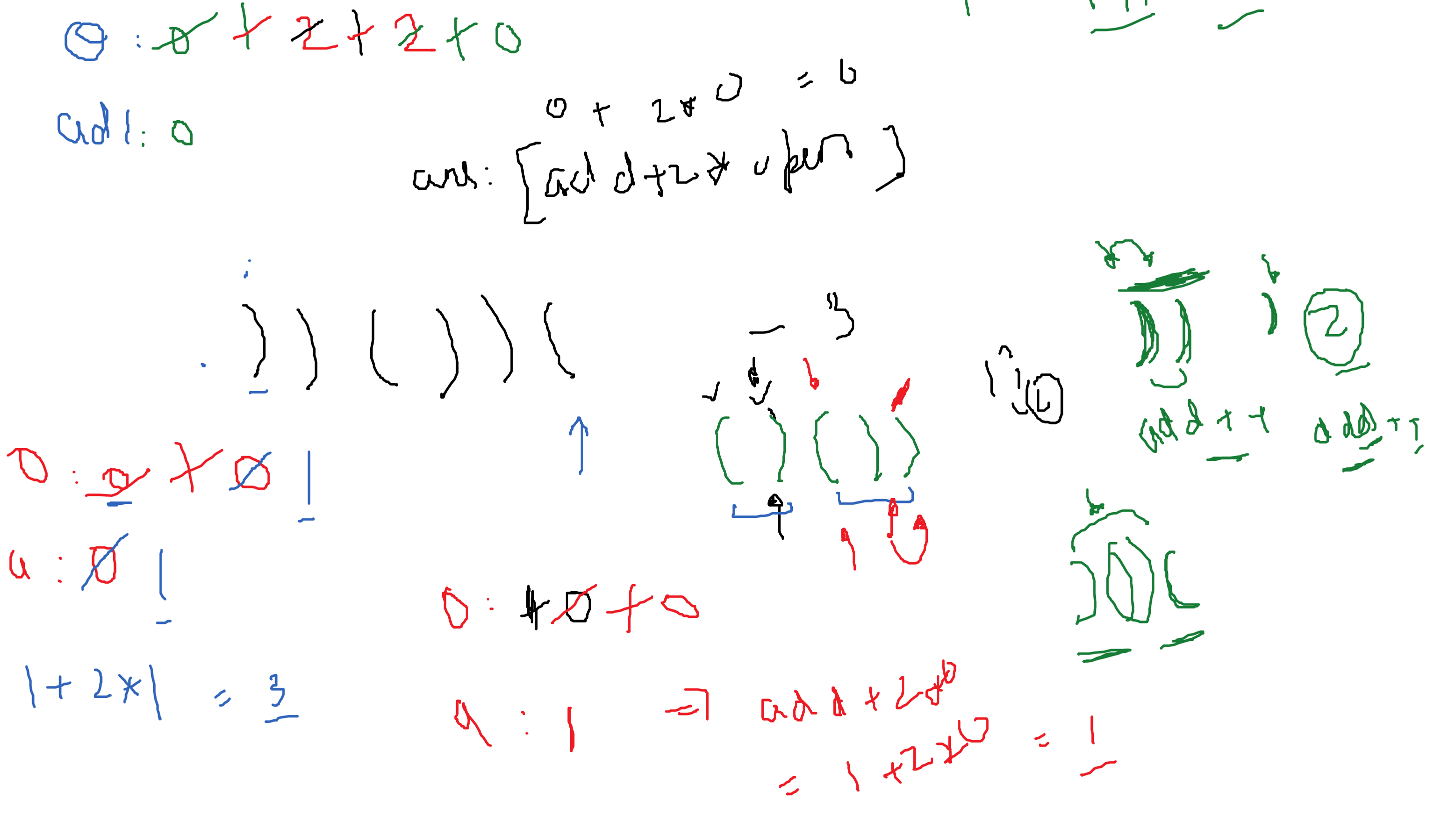
xample 3:

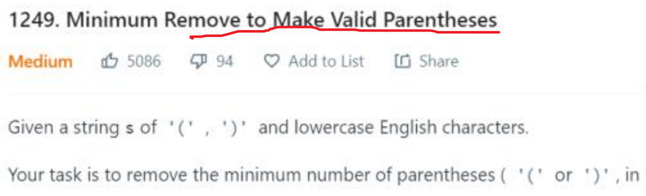
```
Input: s = "))())("
Output: 3
Explanation: Add '(' to match the first '))', Add '))' to match
the last '('.
```

s[i+1] != ')'
add++
i++
i++
i+=2
i++



= "(()))"



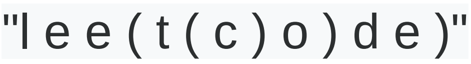


1249. Minimum Remove to Make Valid Parentheses

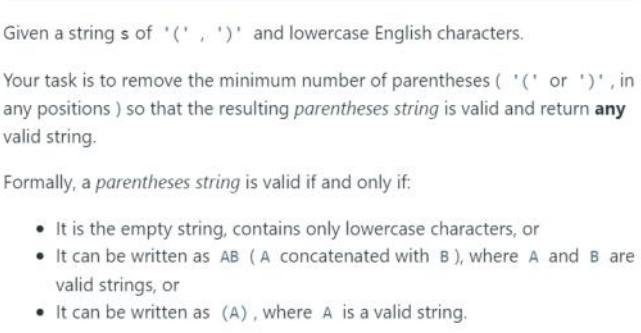
Medium 👍 5086 👎 94 ♡ Add to List ⇗ Share

Given a string s of `'('` , `')'` and lowercase English characters.

Your task is to remove the minimum number of parentheses (`'('` or `')'`, in



"lee(t(c)o)de)"



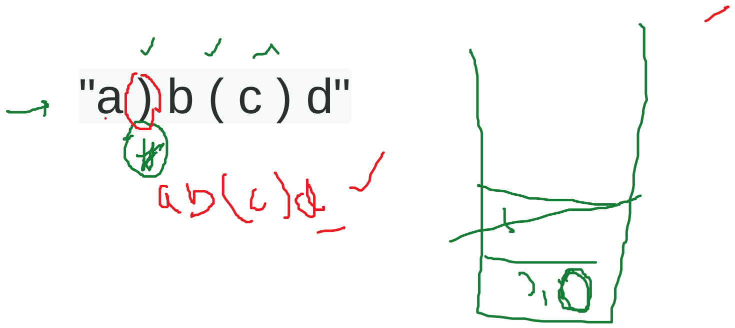
Given a string s of '(' , ')' and lowercase English characters.

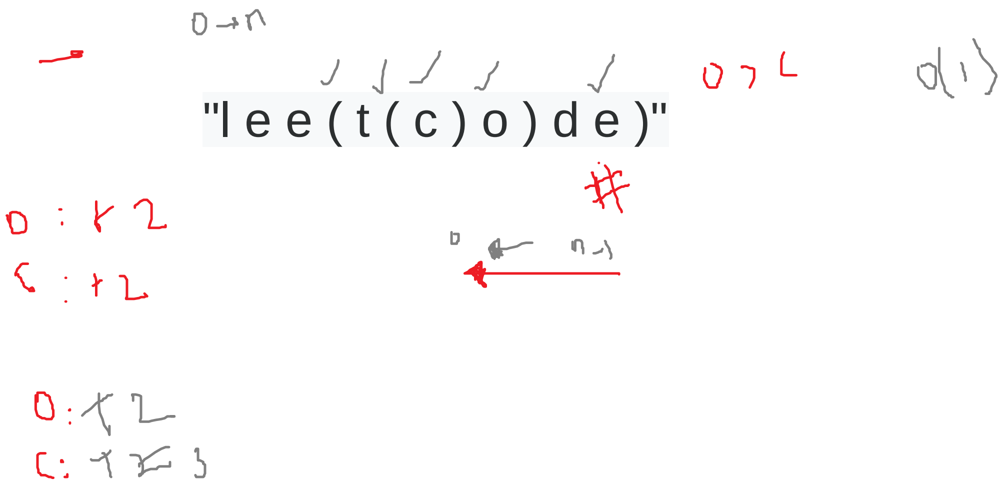
Your task is to remove the minimum number of parentheses ('(' or ')' , in any positions) so that the resulting *parentheses string* is valid and return **any** valid string.

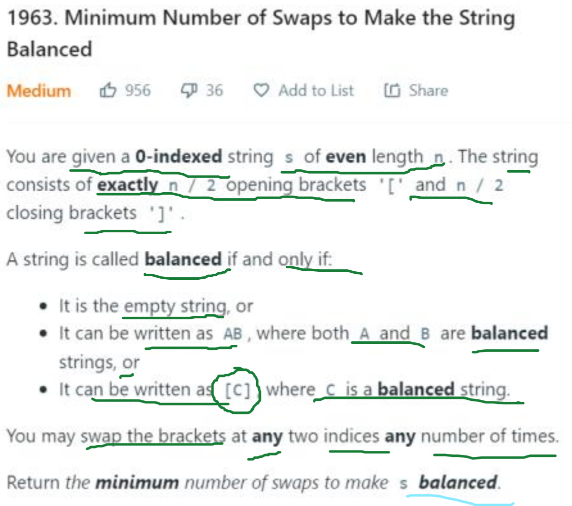
Formally, a *parentheses string* is valid if and only if:

- It is the empty string, contains only lowercase characters, or
- It can be written as AB (A concatenated with B), where A and B are valid strings, or
- It can be written as (A), where A is a valid string.

"l e e (t (c) o) d e)"







1963. Minimum Number of Swaps to Make the String Balanced

Medium 956 36 Add to List Share

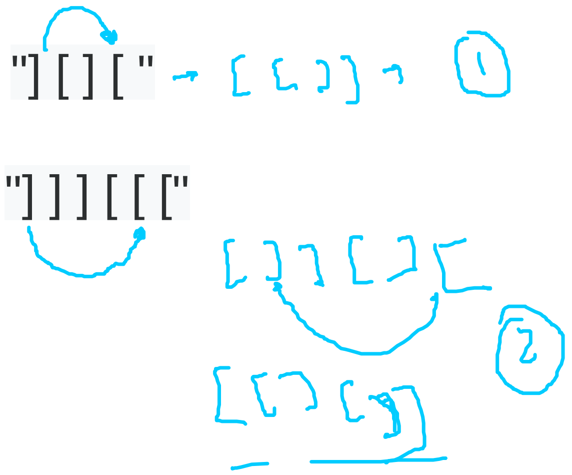
You are given a **0-indexed** string s of **even** length n . The string consists of **exactly** n / 2 opening brackets '[' and n / 2 closing brackets ']' .

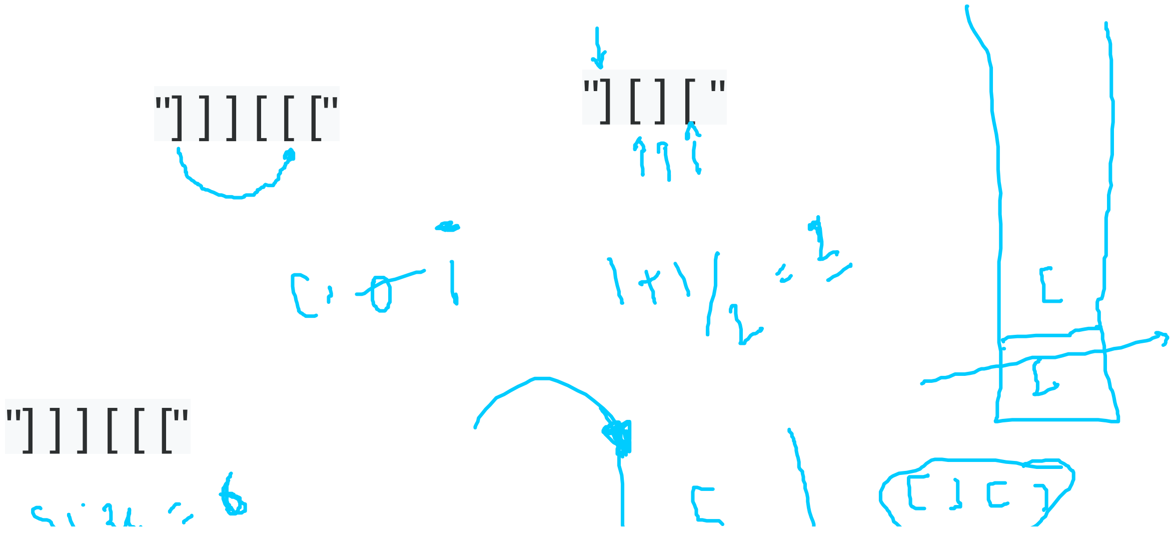
A string is called **balanced** if and only if:

- It is the empty string, or
- It can be written as AB , where both A and B are **balanced** strings, or
- It can be written as [C] , where C is a **balanced** string.

You may swap the brackets at **any** two indices **any** number of times.

Return *the* ***minimum*** *number of swaps to make* s ***balanced***.



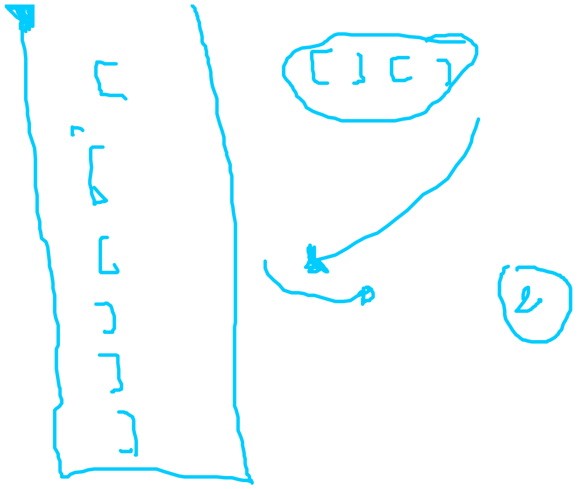


]]] [[[

size = 6

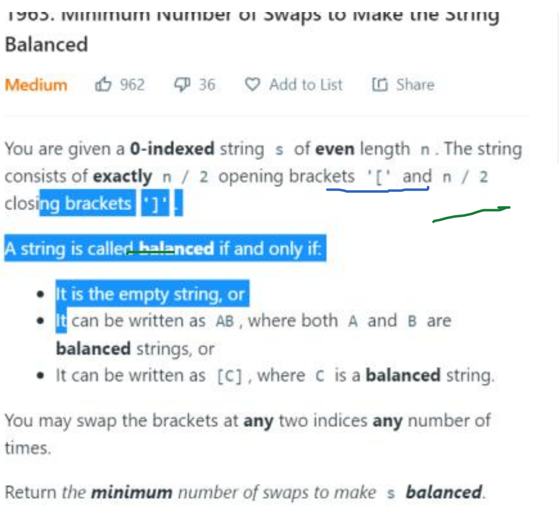
6 % 2 == 1 &&

6/2 =



Stack and queue 10

01 August 2022 14:48



1963. Minimum Number of Swaps to Make the String Balanced

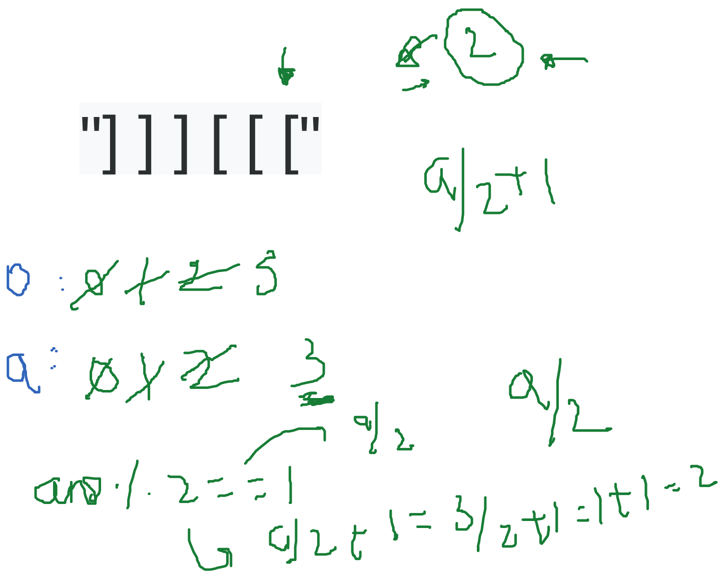
You are given a **0-indexed** string `s` of **even** length `n`. The string consists of **exactly** `n / 2` opening brackets `'['` and `n / 2` closing brackets `']'`.

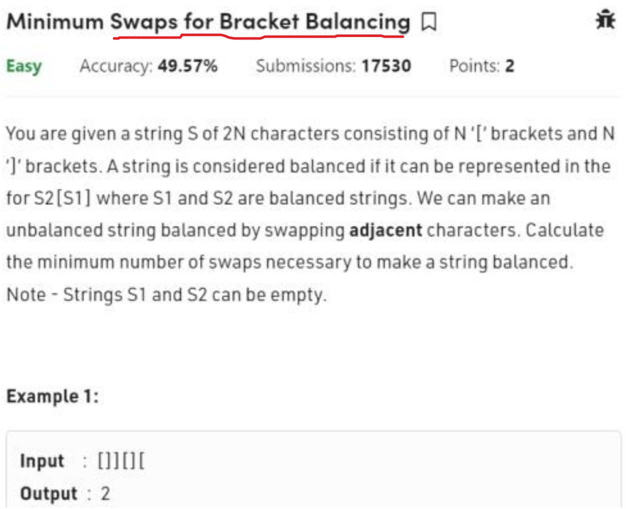
A string is called **balanced** if and only if:

- It is the empty string, or
- It can be written as `AB`, where both `A` and `B` are **balanced** strings, or
- It can be written as `[C]`, where `C` is a **balanced** string.

You may swap the brackets at **any** two indices **any** number of times.

Return *the **minimum** number of swaps to make* `s` ***balanced***.





Minimum Swaps for Bracket Balancing

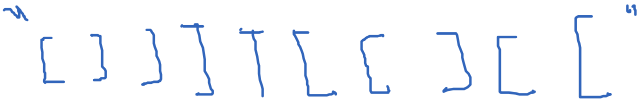
You are given a string S of 2N characters consisting of N '[' brackets and N ']' brackets. A string is considered balanced if it can be represented in the for S2[S1] where S1 and S2 are balanced strings. We can make an unbalanced string balanced by swapping **adjacent** characters. Calculate the minimum number of swaps necessary to make a string balanced.

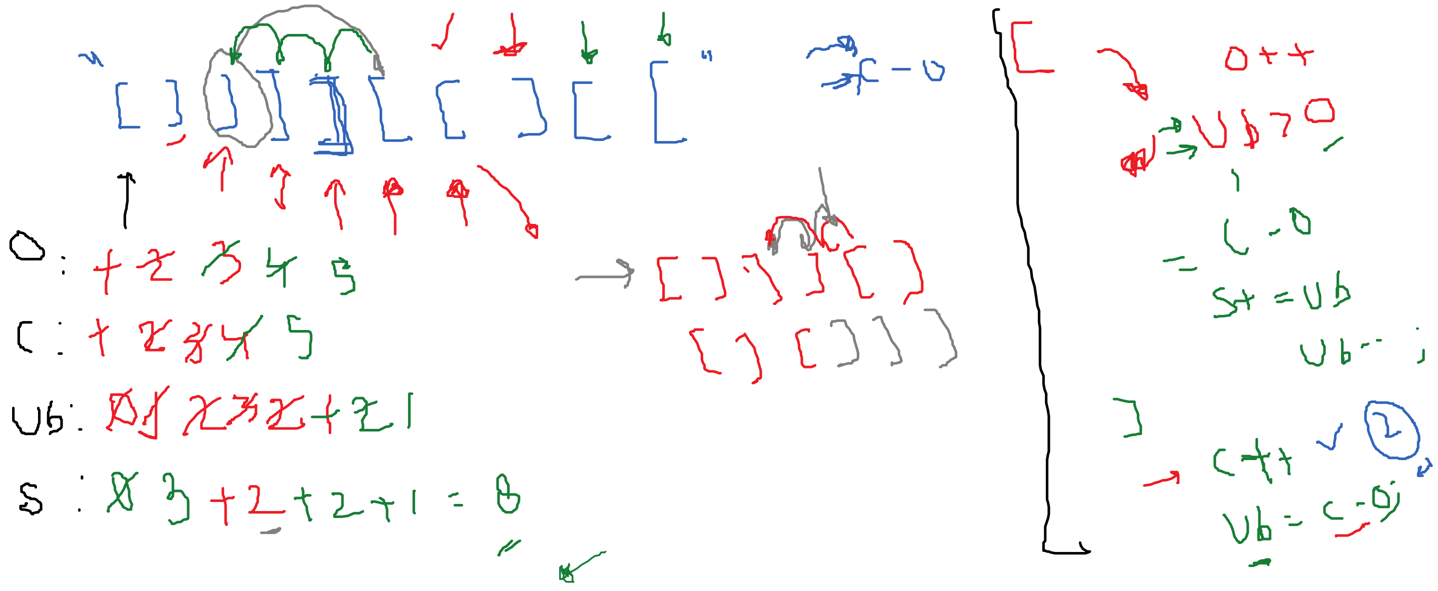
Note - Strings S1 and S2 can be empty.

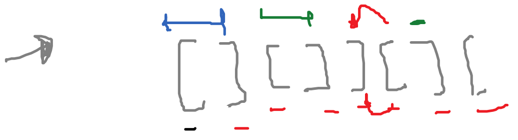
Example 1:

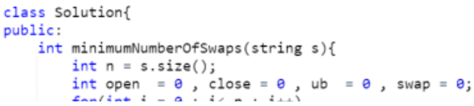
Input : [ ]] [ ] [

Output : 2

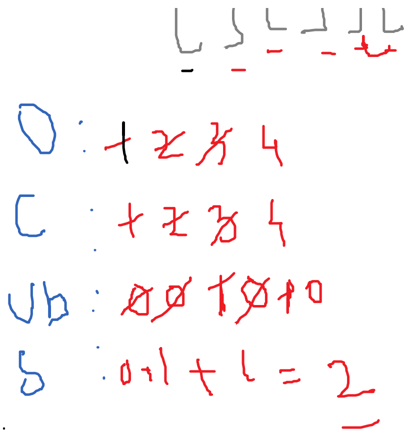




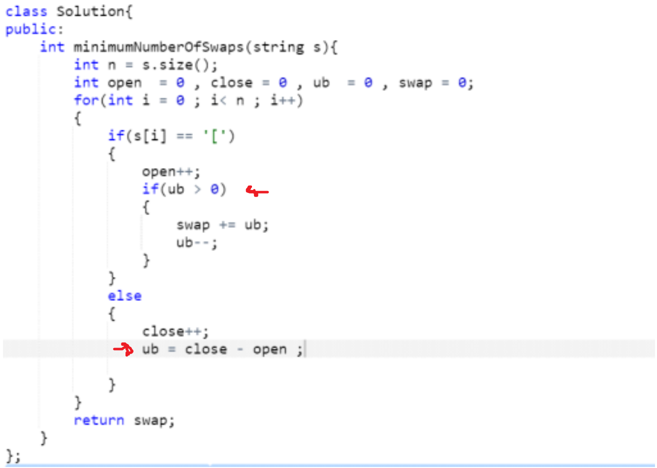




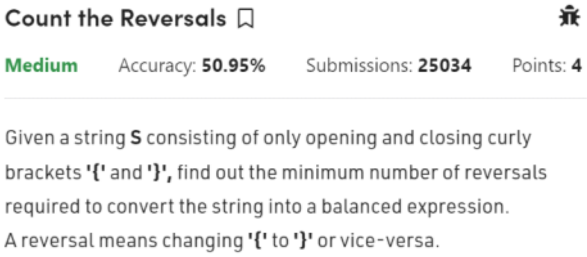
```
class Solution{
public:
    int minimumNumberOfSwaps(string s){
        int n = s.size();
        int open  = 0 , close = 0 , ub  = 0 , swap = 0;
        for(int i = 0 ; i<n ; i++)
```



Screen clipping taken: 01-08-2022 16:09



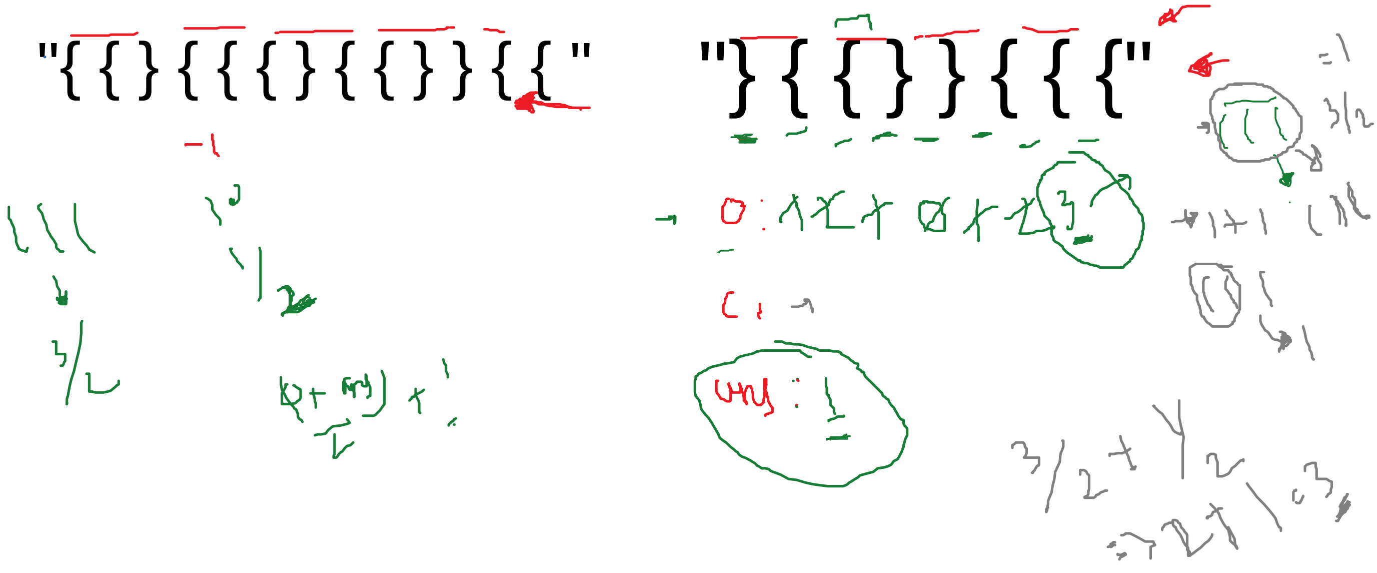
```
class Solution{
public:
    int minimumNumberOfSwaps(string s){
        int n = s.size();
        int open  = 0 , close = 0 , ub  = 0 , swap = 0;
        for(int i = 0 ; i< n ; i++)
        {
            if(s[i] == '[')
            {
                open++;
                if(ub > 0)
                {
                    swap += ub;
                    ub--;
                }
            }
            else
            {
                close++;
                ub = close - open ;
            }
        }
        return swap;
    }
};
```



Count the Reversals

Medium Accuracy: **50.95%** Submissions: **25034** Points: **4**

Given a string **S** consisting of only opening and closing curly brackets **'{'** and **'}',** find out the minimum number of reversals required to convert the string into a balanced expression.
A reversal means changing **'{'** to **'}'** or vice-versa.



Stack and queue 11

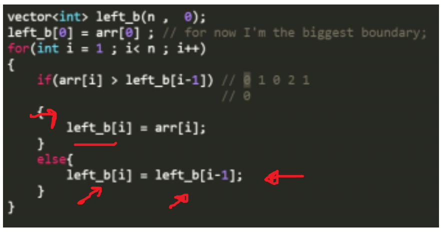
07 August 2022 14:17

42. Trapping Rain Water

Hard 20815 291 Add to List Share

lb = [0 | 1 | 1 | 2 | 2 | 2 | 2 ...

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.



```
vector<int> left_b(n ,  0);
left_b[0] = arr[0] ; // for now I'm the biggest boundary;
for(int i = 1 ; i< n ; i++)
{
    if(arr[i] > left_b[i-1]) // 0 1 0 2 1
                             // 0
    {
        left_b[i] = arr[i];
    }
    else{
        left_b[i] = left_b[i-1];
    }
}
```

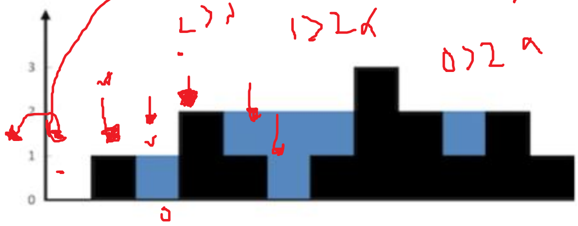
Example 1:

1>0

0>1 x

1>2 x

0>2 x



```
Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
```

2 3

r[n-1] = a[n-1]

[0 1 0 2 1 0 1 3 2 1 2 1]

left = 0 1 1 2 2 2 2 3 3 3 3 3

right = 1 2 2 2 3 3 3 3 3 3 3 3

Screen clipping taken: 07-08-2022 15:21

3 3 3 3 3 3 3 3 2 2 2 1

1-0 2-0

0-0 =0 + =1 + =2 + + + = and =

lb = 0, rb = 1

1 2

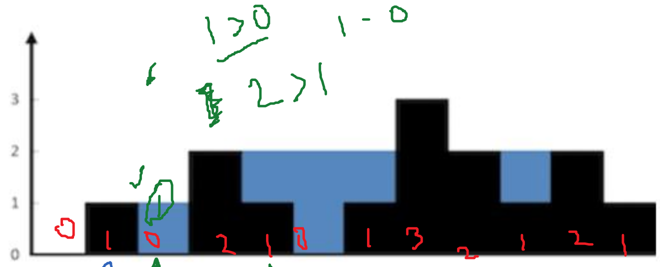
1>0 1-0

2>1

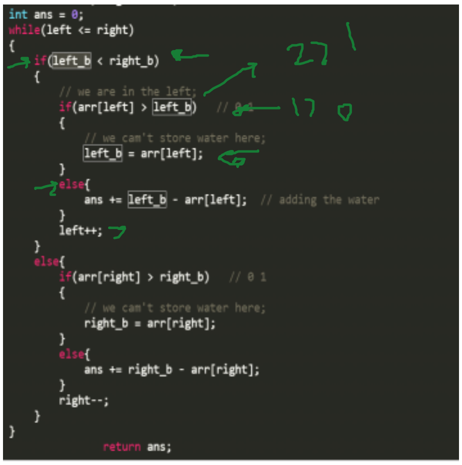
0 1

1<1

Example 1:



```
Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
```



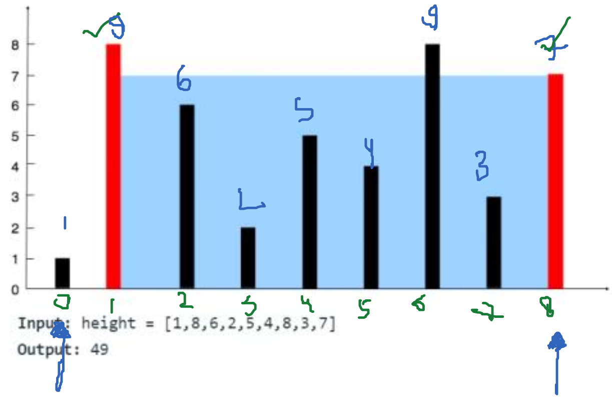
```
int ans = 0;
while(left <= right)
{
    if(left_b < right_b)
    {
        // we are in the left;
        if(arr[left] > left_b)   // 0 1
        {
            // we cam't store water here;
            left_b = arr[left];
        }
        else{
            ans += left_b - arr[left];  // adding the water
        }
        left++;
    }
    else{
        if(arr[right] > right_b)   // 0 1
        {
            // we cam't store water here;
            right_b = arr[right];
        }
        else{
            ans += right_b - arr[right];
        }
        right--;
    }
}
            return ans;
```

2 7 1

1 7 0

Container with Most Water

Example 1:



```
Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
```

min(1 , 8) => 1

$l = 0$

$r = n-1$

$l < r$

$ans = max(a[l], a[r]) \cdot (r - l)$

$a[l] < a[r]$

$l{+}{+}$

else

$r{+}{+}$