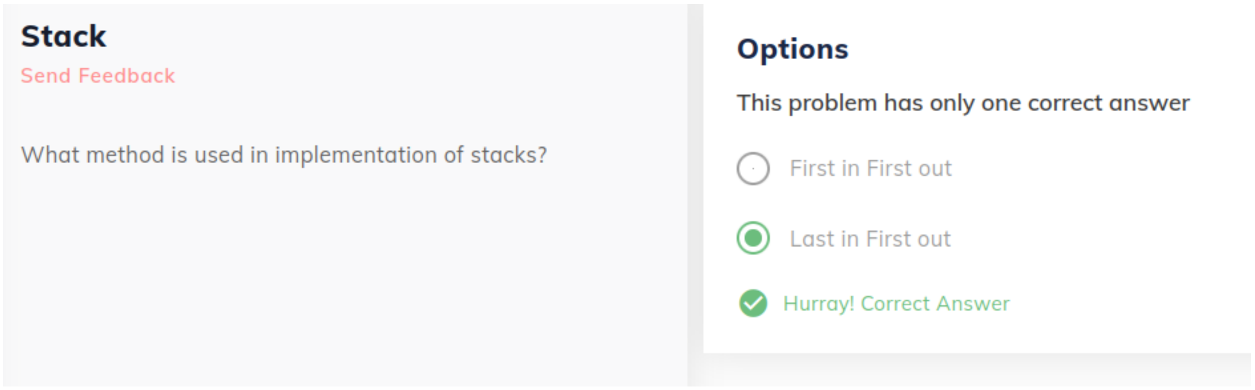
Stacks



Stack

Send Feedback

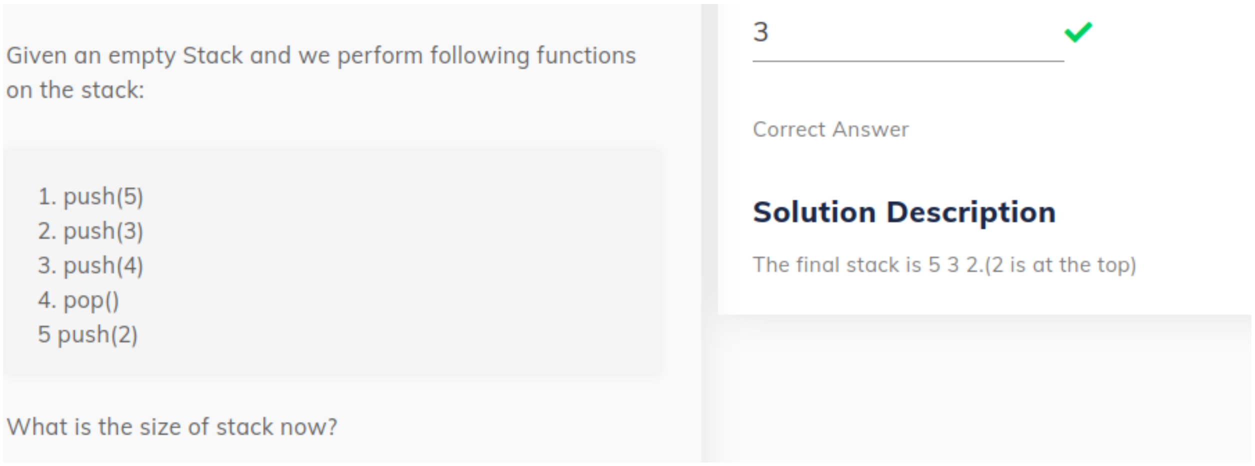
What method is used in implementation of stacks?

Options

This problem has only one correct answer

- First in First out
- [✓ Last in First out]

✓ Hurray! Correct Answer



Given an empty Stack and we perform following functions on the stack:

```
1. push(5)
2. push(3)
3. push(4)
4. pop()
5 push(2)
```

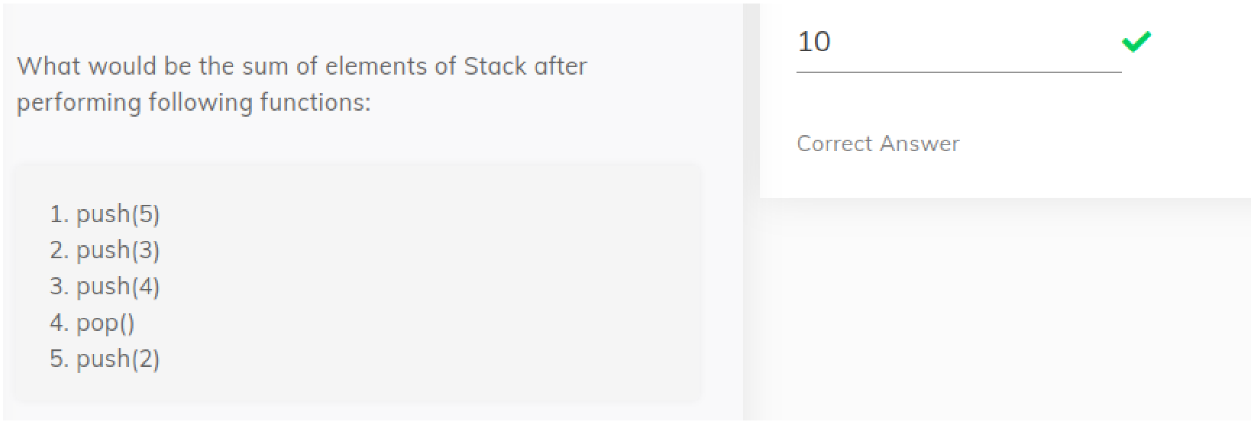
What is the size of stack now?

3 ✓

Correct Answer

Solution Description

The final stack is 5 3 2.(2 is at the top)

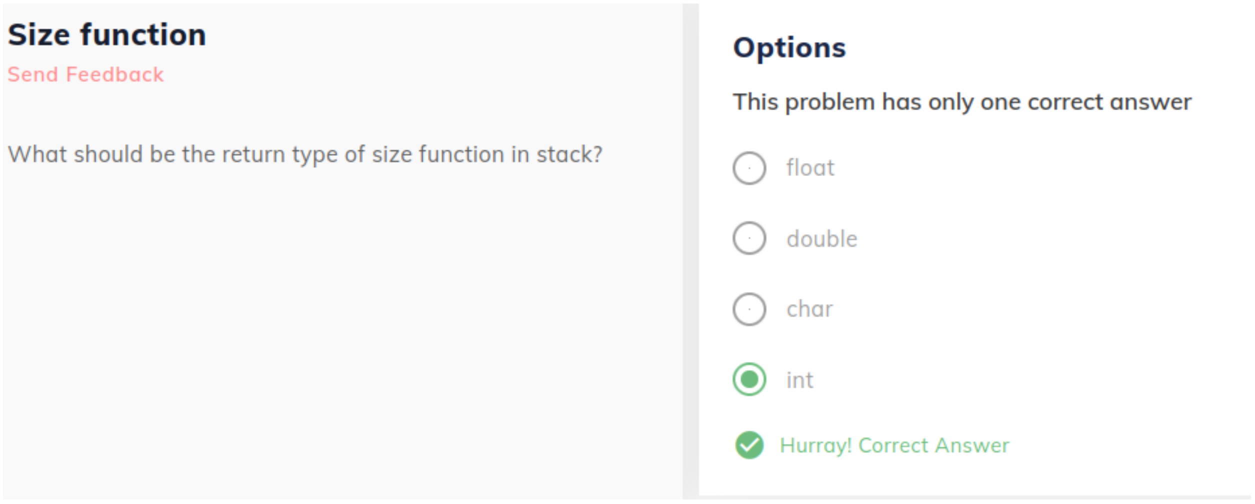


What would be the sum of elements of Stack after performing following functions:

```
1. push(5)
2. push(3)
3. push(4)
4. pop()
5. push(2)
```

10 ✓

Correct Answer



Size function

Send Feedback

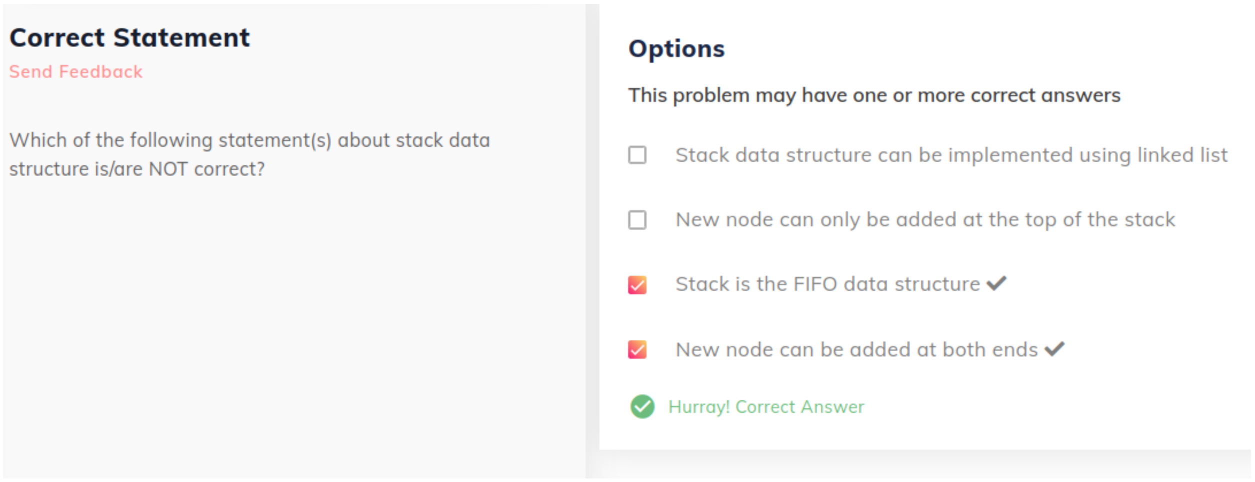
What should be the return type of size function in stack?

Options

This problem has only one correct answer

- float
- double
- char
- [✓ int]

✓ Hurray! Correct Answer



Correct Statement

Send Feedback

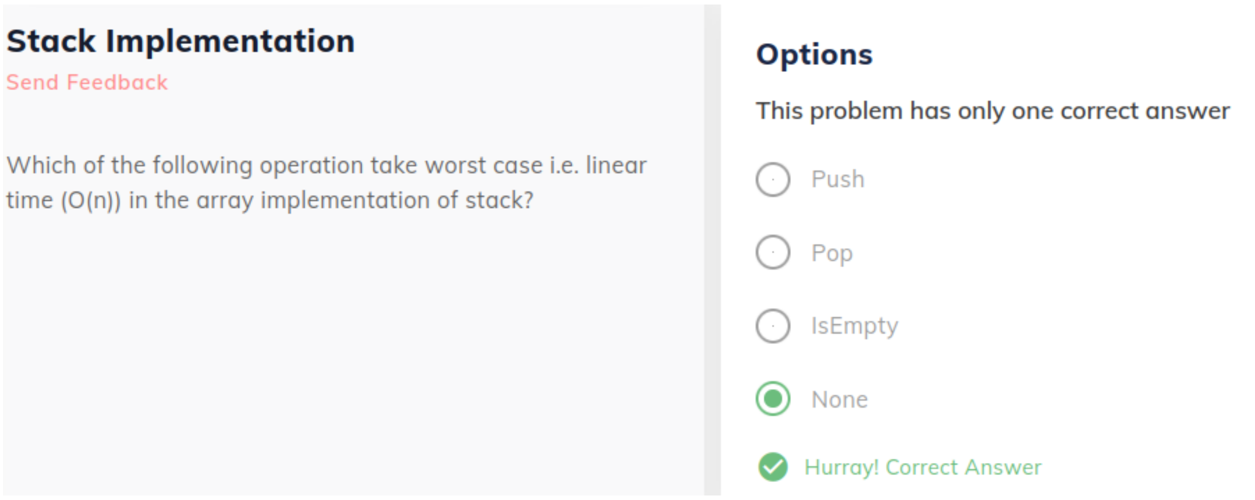
Which of the following statement(s) about stack data structure is/are NOT correct?

Options

This problem may have one or more correct answers

- ☐ Stack data structure can be implemented using linked list
- ☐ New node can only be added at the top of the stack
- [✓ Stack is the FIFO data structure] ✓
- [✓ New node can be added at both ends] ✓

✓ Hurray! Correct Answer



Stack Implementation

Send Feedback

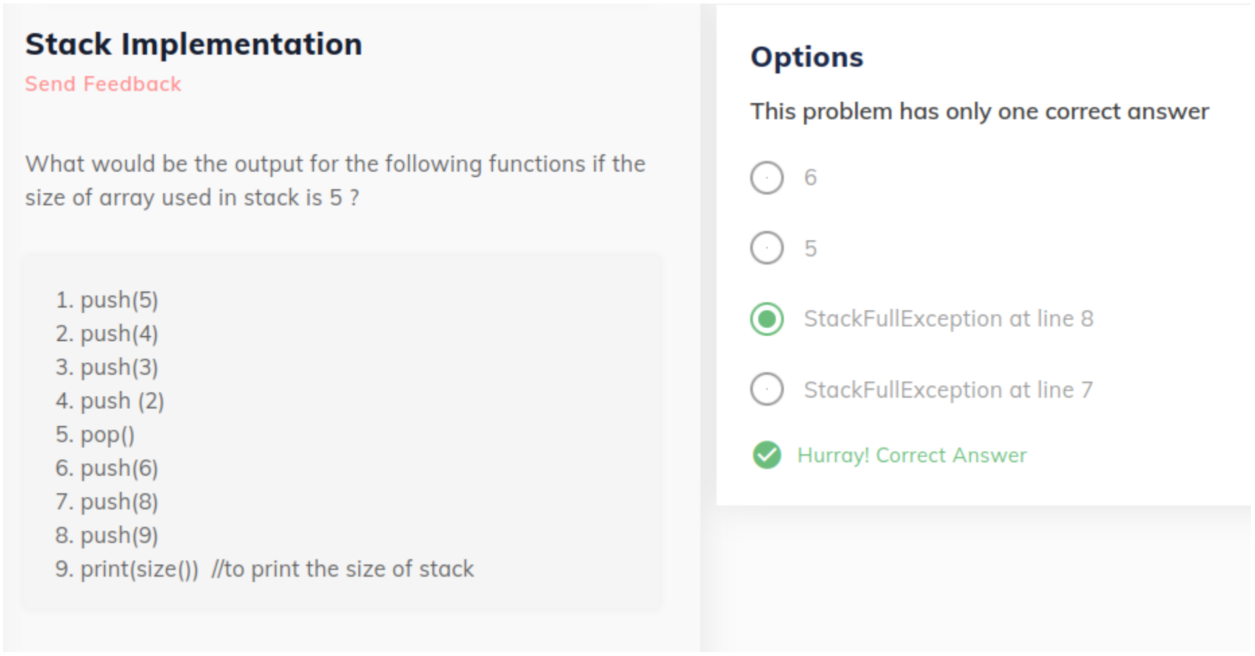
Which of the following operation take worst case i.e. linear time (O(n)) in the array implementation of stack?

Options

This problem has only one correct answer

- Push
- Pop
- IsEmpty
- [✓ None]

✓ Hurray! Correct Answer



Stack Implementation

Send Feedback

What would be the output for the following functions if the size of array used in stack is 5 ?

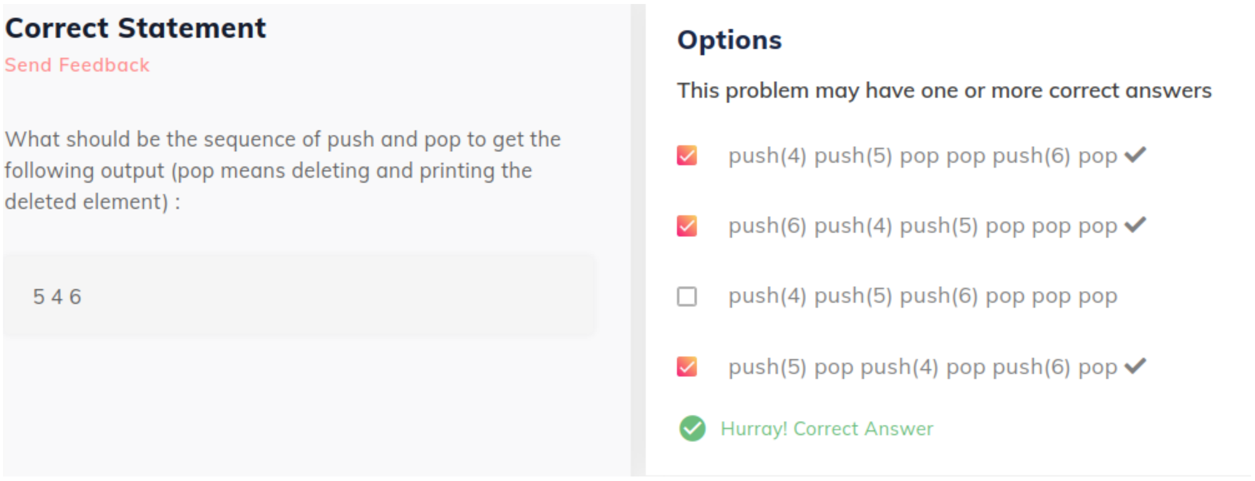
```
1. push(5)
2. push(4)
3. push(3)
4. push (2)
5. pop()
6. push(6)
7. push(8)
8. push(9)
9. print(size())  //to print the size of stack
```

Options

This problem has only one correct answer

- ○ 6
- ○ 5
- [✓ StackFullException at line 8]
- ○ StackFullException at line 7

✓ Hurray! Correct Answer



Correct Statement

Send Feedback

What should be the sequence of push and pop to get the following output (pop means deleting and printing the deleted element) :

```
5 4 6
```

Options

This problem may have one or more correct answers

- [✓ push(4) push(5) pop pop push(6) pop] ✔
- [✓ push(6) push(4) push(5) pop pop pop] ✔
- ☐ push(4) push(5) push(6) pop pop pop
- [✓ push(5) pop push(4) pop push(6) pop] ✔

✓ Hurray! Correct Answer

Predict the output

Send Feedback

What would be the output for the following functions (pop and size means printing also).

Note: No StackFullException occurs.

```
push(1)
push(2)
push(3)
size()
pop
push(8)
pop
pop
pop
```

Options

This problem has only one correct answer

- 3 8 3 2 1
- 2 3 8 2 1
- [✓ 3 3 8 2 1]
- 4 8 3 2 1

Hurray! Correct Answer

Double Stack

Send Feedback

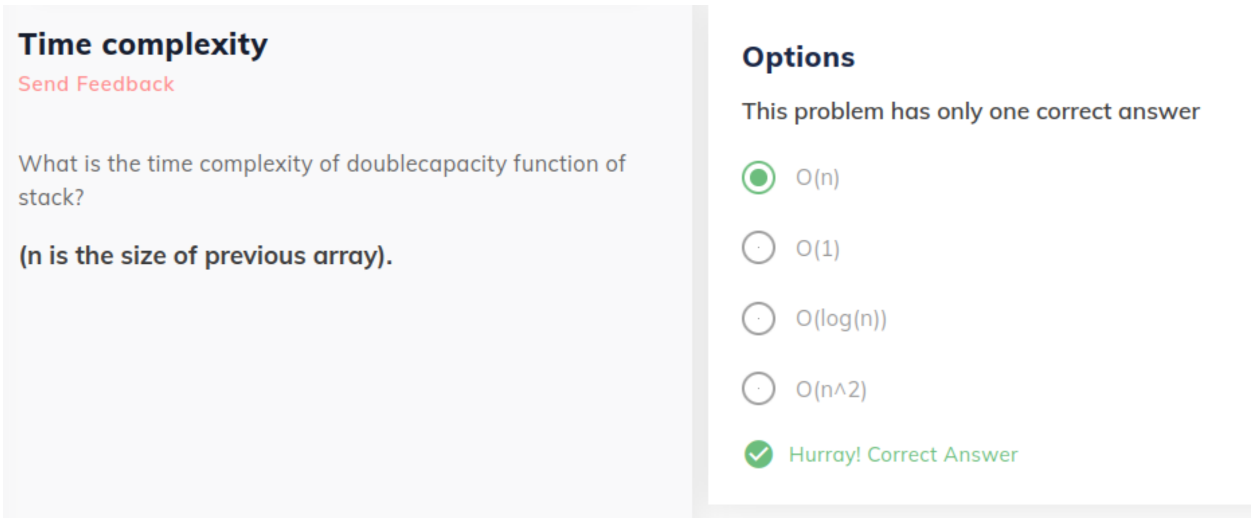
If initially the size of array of stack is 3,and we need to input 34 elements. How many times the doublecapacity function would be called?

Options

This problem has only one correct answer

- 3
- [✓ 4]
- 5
- 12
- 11

Hurray! Correct Answer



Time complexity

Send Feedback

What is the time complexity of doublecapacity function of stack?

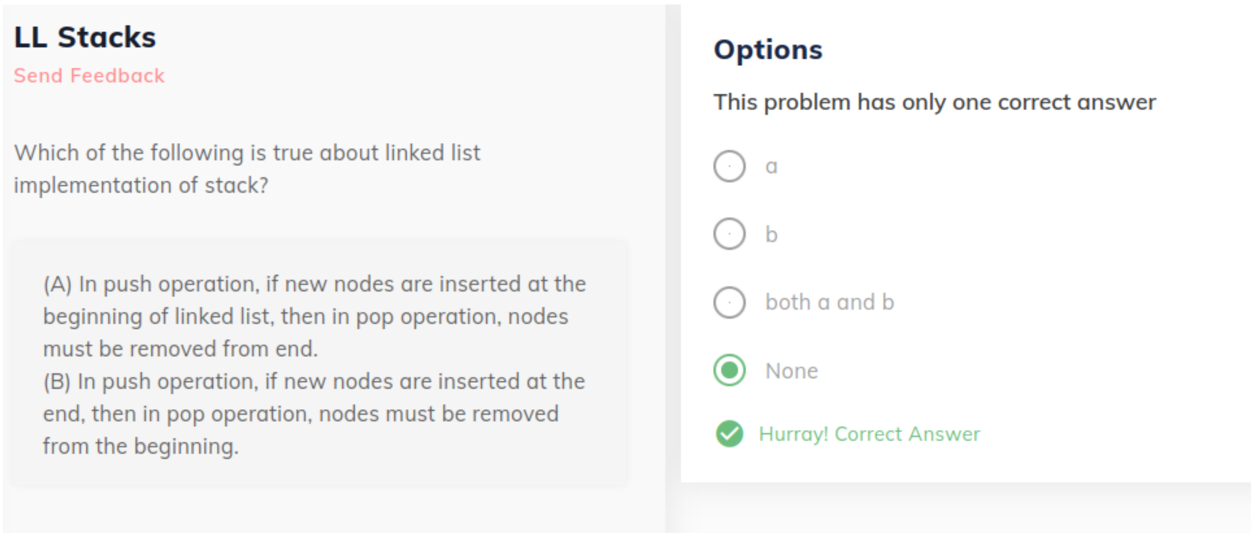
(n is the size of previous array).

Options

This problem has only one correct answer

- [✓ O(n)]
- O(1)
- O(log(n))
- O(n^2)

✓ Hurray! Correct Answer



LL Stacks

Send Feedback

Which of the following is true about linked list implementation of stack?

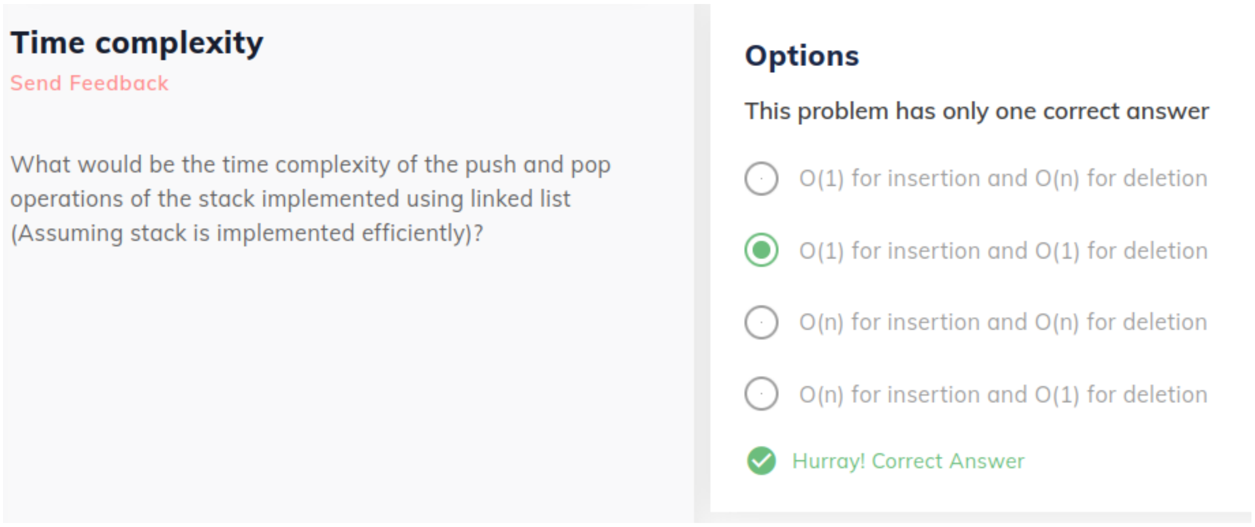
(A) In push operation, if new nodes are inserted at the beginning of linked list, then in pop operation, nodes must be removed from end.
(B) In push operation, if new nodes are inserted at the end, then in pop operation, nodes must be removed from the beginning.

Options

This problem has only one correct answer

- a
- b
- both a and b
- [✓ None]

✓ Hurray! Correct Answer



Time complexity

Send Feedback

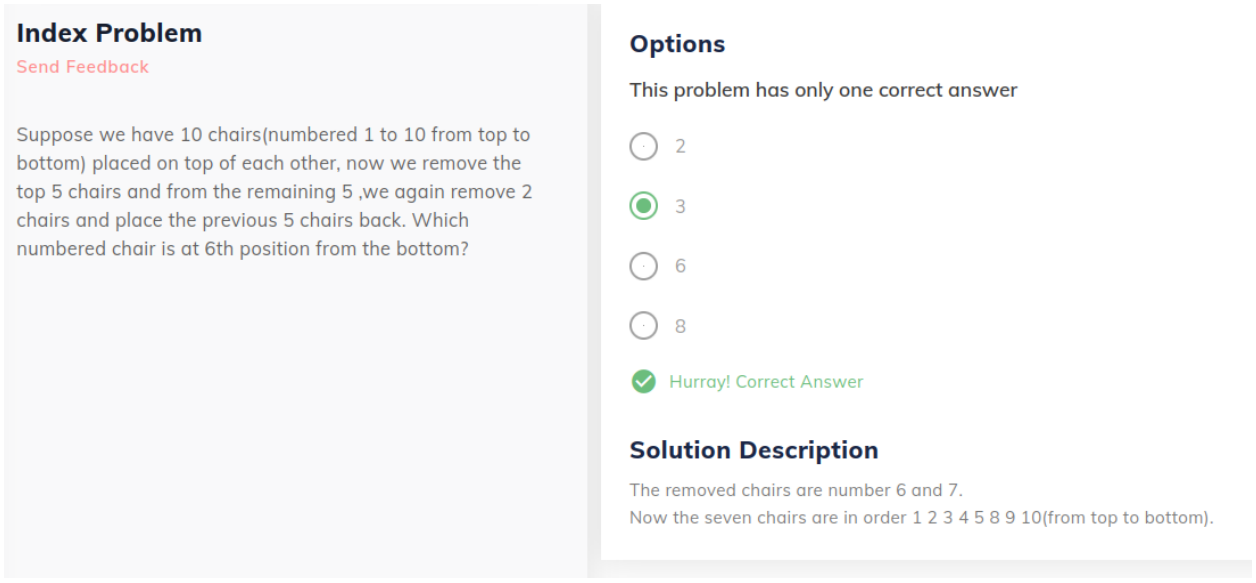
What would be the time complexity of the push and pop operations of the stack implemented using linked list (Assuming stack is implemented efficiently)?

Options

This problem has only one correct answer

- O(1) for insertion and O(n) for deletion
- [✓ O(1) for insertion and O(1) for deletion]
- O(n) for insertion and O(n) for deletion
- O(n) for insertion and O(1) for deletion

✓ Hurray! Correct Answer



Index Problem

Send Feedback

Suppose we have 10 chairs(numbered 1 to 10 from top to bottom) placed on top of each other, now we remove the top 5 chairs and from the remaining 5 ,we again remove 2 chairs and place the previous 5 chairs back. Which numbered chair is at 6th position from the bottom?

Options

This problem has only one correct answer

- 2
- [✓ 3]
- 6
- 8

✓ Hurray! Correct Answer

Solution Description

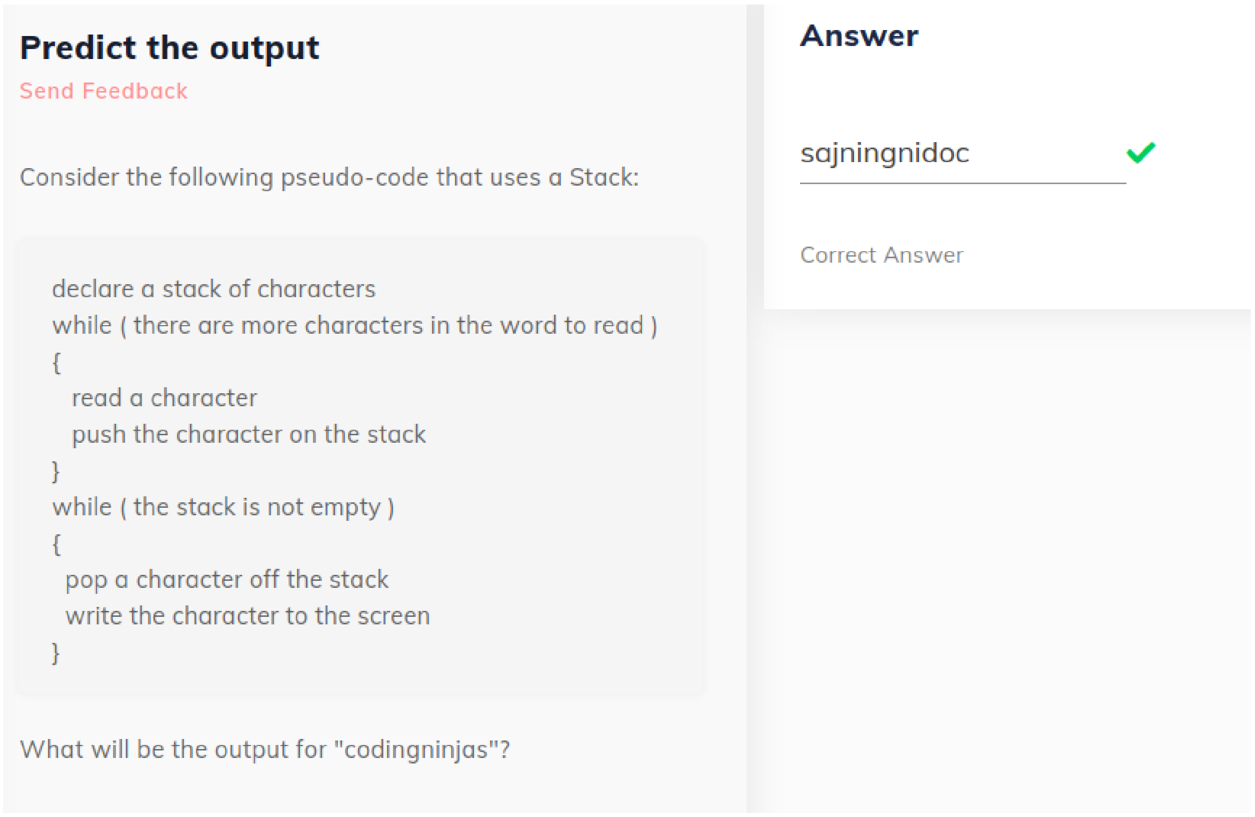
The removed chairs are number 6 and 7.
Now the seven chairs are in order 1 2 3 4 5 8 9 10(from top to bottom).

```
Stack Using LL
Send Feedback
Implement a Stack Data Structure specifically to store integer data using
a Singly Linked List.
The data members should be private.
You need to implement the following public functions :
1. Constructor:
It initialises the data members as required.
2. push(data) :
This function should take one argument of type integer. It pushes the
element into the stack and returns nothing.
3. pop() :
It pops the element from the top of the stack and in turn, returns the
element being popped or deleted. In case the stack is empty, it returns
-1.
4. top :
It returns the element being kept at the top of the stack. In case the
stack is empty, it returns -1.
5. size() :
It returns the size of the stack at any given instance of time.
6. isEmpty() :
It returns a boolean value indicating whether the stack is empty or not.
Operations Performed on the Stack:

Query-1(Denoted by an integer 1): Pushes an integer data to the stack.

Query-2(Denoted by an integer 2): Pops the data kept at the top of the
stack and returns it to the caller.

Query-3(Denoted by an integer 3): Fetches and returns the data being kept
at the top of the stack but doesn't 'remove it, unlike the pop function.

Query-4(Denoted by an integer 4): Returns the current size of the stack.

Query-5(Denoted by an integer 5): Returns a boolean value denoting whether
the stack is empty or not.
```

```
Input Format:

The first line contains an integer 'q' which denotes the number of queries
to be run against each operation in the stack.
Then the test cases follow.

Every 'q' lines represent an operation that needs to be performed.

For the push operation, the input line will contain two integers separated
by a single space, representing the type of the operation in integer and
the integer data being pushed into the stack.

For the rest of the operations on the stack, the input line will contain
only one integer value, representing the query being performed on the
stack.

Output Format:

For Query-1, you do not need to return anything.
For Query-2, prints the data being popped from the stack.
For Query-3, prints the data kept on the top of the stack.
For Query-4, prints the current size of the stack.
For Query-5, prints 'true' or 'false'(without quotes).

Output for every query will be printed in a separate line.

Note:

You are not required to print anything explicitly. It has already been
taken care of. Just implement the function.

Constraints:

1 <= q <= 10^5
1 <= x <= 5
-2^31 <= data <= 2^31 - 1 and data != -1

Where 'q' is the total number of queries being performed on the stack, 'x'
is the range for every query and data represents the integer pushed into
the stack.

Time Limit: 1 second

Sample Input 1:

6
1 13
1 47
4
5
2
3
```

```
Sample Output 1:

2
false
47
13

Sample Input 2:

4
5
2
1 10
5

Sample Output 2:

true
-1
false

Explanation of Sample Input 2:

There are 4 queries in total.
The first one is Query-5: It tells whether the stack is empty or not.
Since the stack is empty at this point, the output is  'true'.

The second one is Query-2: It pops the data from the stack. Since at this
point in time, no data exist in the stack hence, it prints -1.

The third one is Query-1: It pushes the specified data 10 into the stack
and since the function doesn't 'return anything, nothing is printed.

The fourth one is Query-5: It tells whether the stack is empty at this
point or not. Since the stack has one element and hence it is not empty,
false is printed.

public class Stack {

   //Define the data members
   private Node head;
   private int size;

   public Stack() {
       head = null;
       size = 0;
   }

   /*----------------- Public Functions of Stack -----------------*/

   public int getSize() {
       return size;
   }
```

```java
    public boolean isEmpty() {
        return size==0;
    }

    public void push(int element) {
        Node newNode = new Node(element);
        newNode.next = head;
        head = newNode;
        size++;
    }

    public int pop() {
        if(head==null)
            return -1;
        int temp = head.data;
        head = head.next;
        size--;
        return temp;
    }

    public int top() {
        if(head==null)
            return -1;
        else return head.data;
    }
}
```



Predict the output

Send Feedback

Consider the following pseudo-code that uses a Stack:

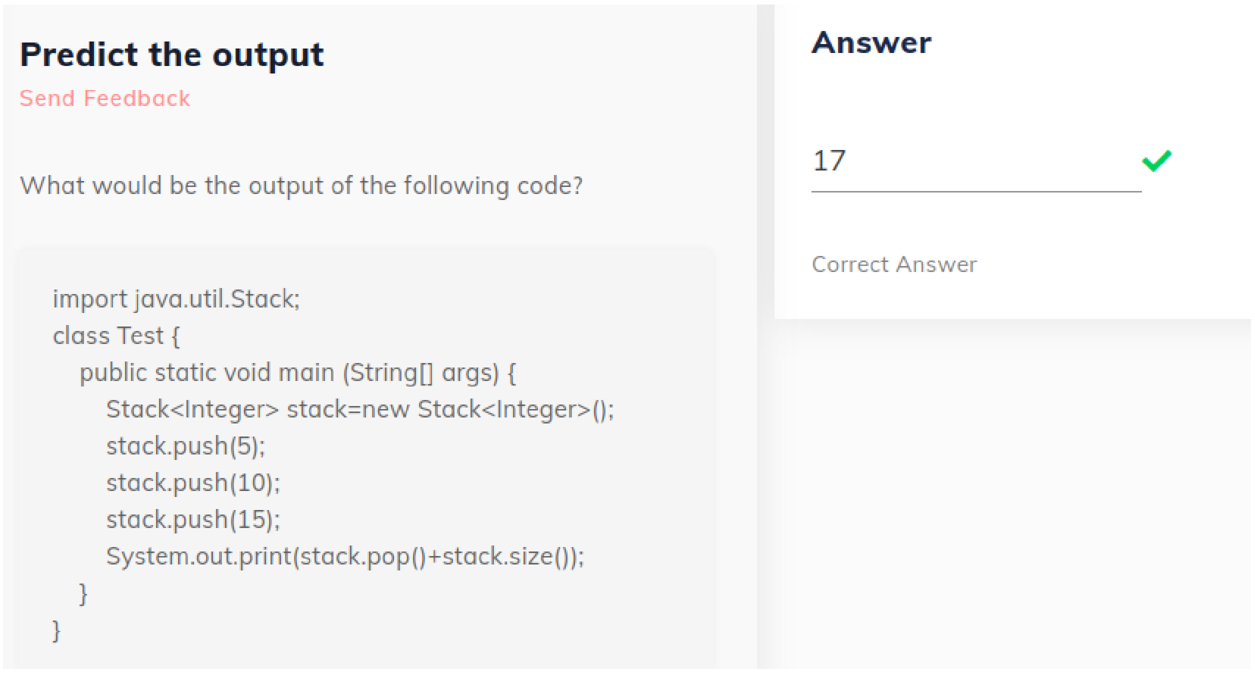
```
declare a stack of characters
while ( there are more characters in the word to read )
{
   read a character
   push the character on the stack
}
while ( the stack is not empty )
{
  pop a character off the stack
  write the character to the screen
}
```

What will be the output for "codingninjas"?

Answer

sajningnidoc ✓

Correct Answer



Predict the output

Send Feedback

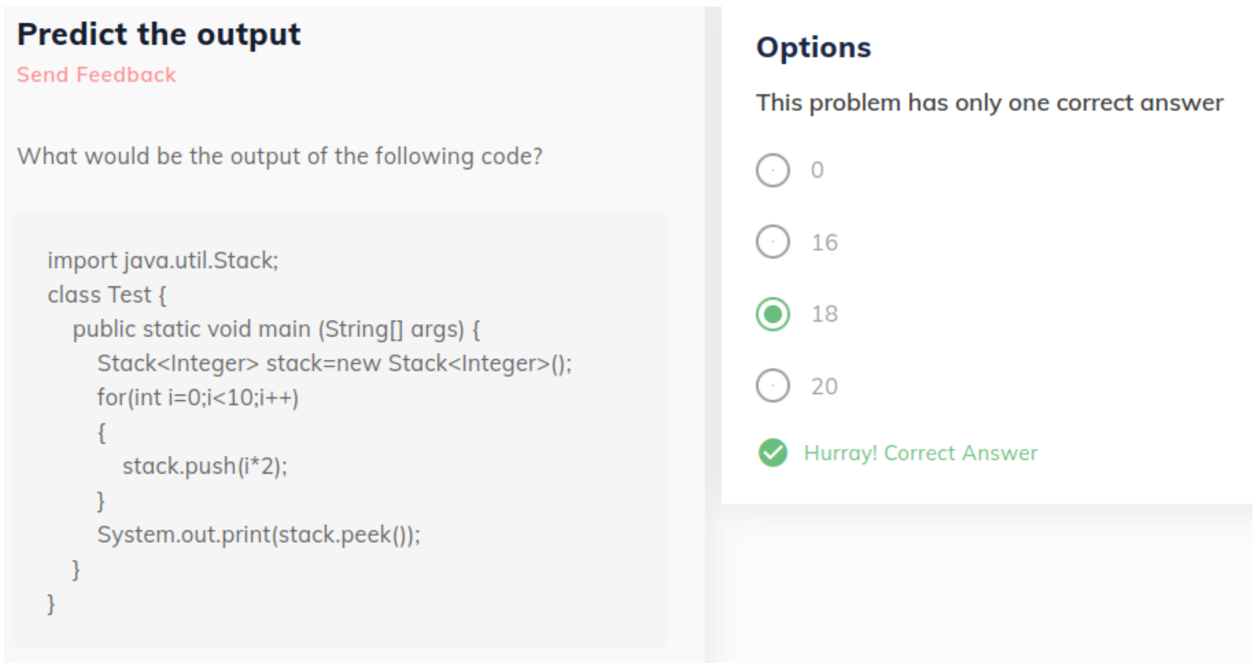
What would be the output of the following code?

```
import java.util.Stack;
class Test {
    public static void main (String[] args) {
        Stack<Integer> stack=new Stack<Integer>();
        stack.push(5);
        stack.push(10);
        stack.push(15);
        System.out.print(stack.pop()+stack.size());
    }
}
```

Answer

17 ✓

Correct Answer



Predict the output

Send Feedback

What would be the output of the following code?

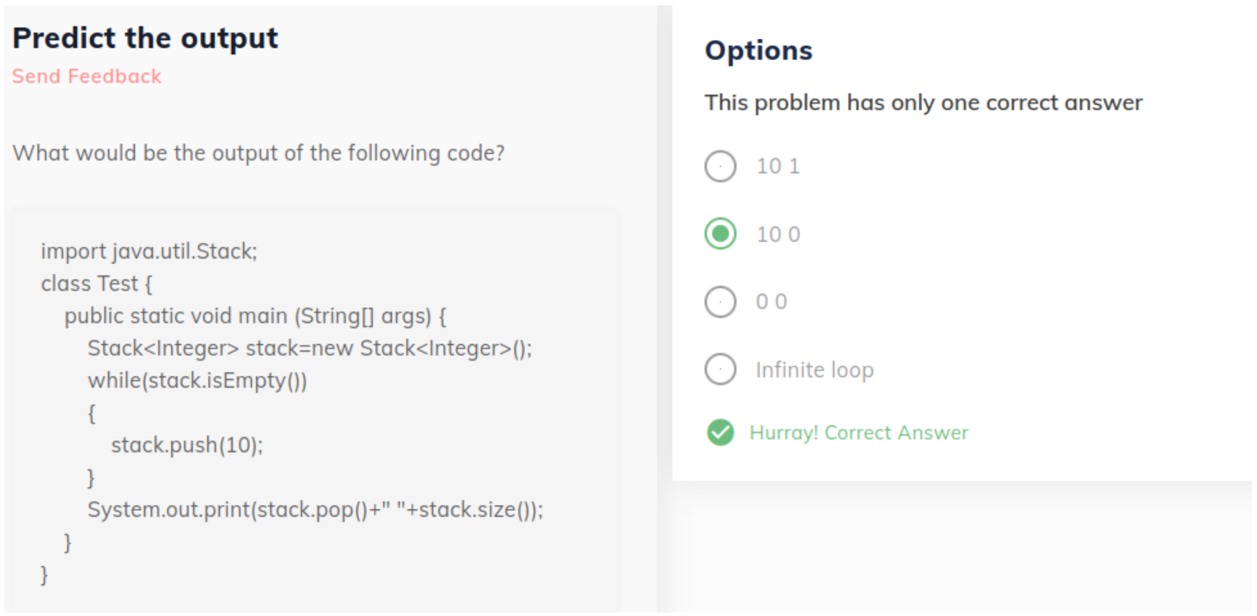
```
import java.util.Stack;
class Test {
    public static void main (String[] args) {
        Stack<Integer> stack=new Stack<Integer>();
        for(int i=0;i<10;i++)
        {
            stack.push(i*2);
        }
        System.out.print(stack.peek());
    }
}
```

Options

This problem has only one correct answer

- 0
- 16
- [✓ 18]
- 20

✓ Hurray! Correct Answer



Predict the output

Send Feedback

What would be the output of the following code?

```
import java.util.Stack;
class Test {
    public static void main (String[] args) {
        Stack<Integer> stack=new Stack<Integer>();
        while(stack.isEmpty())
        {
            stack.push(10);
        }
        System.out.print(stack.pop()+" "+stack.size());
    }
}
```

Options

This problem has only one correct answer

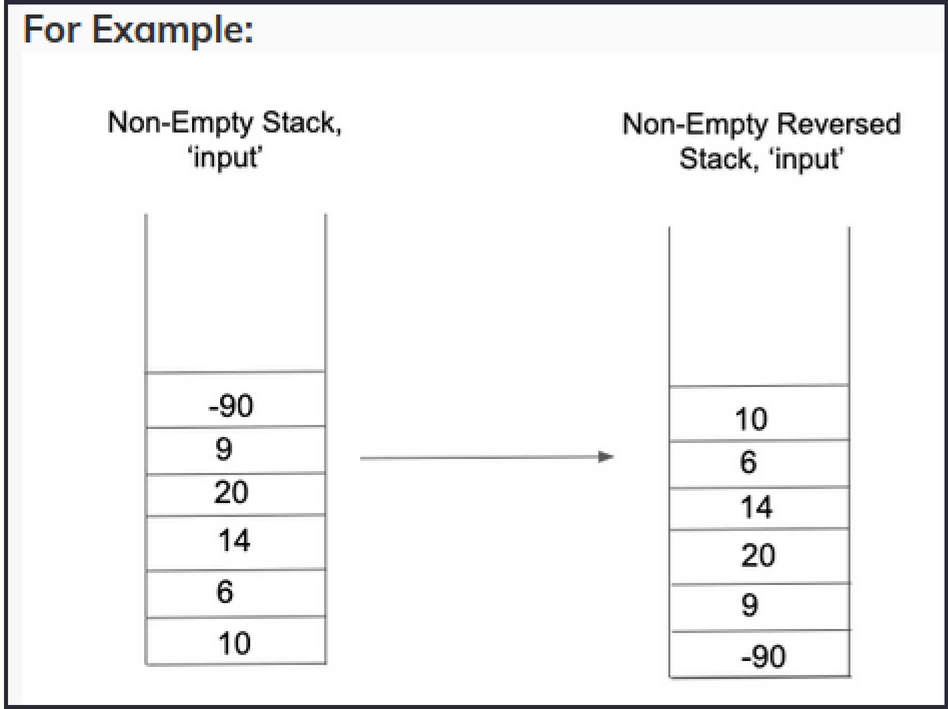
- 10 1
- [✓ 10 0]
- 0 0
- Infinite loop

✓ Hurray! Correct Answer

```
Brackets Balanced
Send Feedback
For a given a string expression containing only round brackets or
parentheses, check if they are balanced or not. Brackets are said to be
balanced if the bracket which opens last, closes first.
Example:

Expression: (()())
Since all the opening brackets have their corresponding closing brackets,
we say it is balanced and hence the output will be, 'true'.
```

```
You need to return a boolean value indicating whether the expression is
balanced or not.
Note:

The input expression will not contain spaces in between.

Input Format:

The first and the only line of input contains a string expression without
any spaces in between.

Output Format:

The only line of output prints 'true' or 'false'.

Note:

You don't have to print anything explicitly. It has been taken care of.
Just implement the function.

Constraints:

1 <= N <= 10^7
Where N is the length of the expression.

Time Limit: 1sec

Sample Input 1 :

(()()())

Sample Output 1 :

true

Sample Input 2 :

()()(()
Sample Output 2 :

false

Explanation to Sample Input 2:

The initial two pairs of brackets are balanced. But when you see, the
opening bracket at the fourth index doesn't have its corresponding closing
bracket which makes it imbalanced and in turn, making the whole expression
imbalanced. Hence the output prints 'false'.

import java.util.Stack;
public class Solution {

   public static boolean isBalanced(String expression) {
       //Your code goes here
       Stack<Character> node = new Stack<>();
       char[] arr = new char[expression.length()];
       for(int i=0; i<expression.length(); i++){
           arr[i] = expression.charAt(i);
       }
       for(int i=0; i<expression.length(); i++){
           if(arr[i]=='('){
               node.push('(');
           }
           else if(arr[i]==')'){
               if(node.isEmpty())
                   return false;
               else if(node.peek()=='(')
                   node.pop();
               else
                   return false;
           }
       }
       if(node.isEmpty())
           return true;
       return false;
   }
}
```

Reverse Stack
Send Feedback
You have been given two stacks that can store integers as the data. Out of the two given stacks, one is populated and the other one is empty. You are required to write a function that reverses the populated stack using the one which is empty.
For Example:



The first line of input contains an integer N, denoting the total number of elements in the stack.

The second line of input contains N integers separated by a single space, representing the order in which the elements are pushed into the stack.

Output Format:

The only line of output prints the order in which the stack elements are popped, all of them separated by a single space.

Note:

```
You are not required to print the expected output explicitly, it has
already been taken care of. Just make the changes in the input stack
itself.

Constraints:

1 <= N <= 10^3
-2^31 <= data <= 2^31 - 1

Time Limit: 1sec

Sample Input 1:

6
1 2 3 4 5 10

Note:

Here, 10 is at the top of the stack.

Sample Output 1:

1 2 3 4 5 10

Note:

Here, 1 is at the top of the stack.

Sample Input 2:

5
2 8 15 1 10

Sample Output 2:

2 8 15 1 10

import java.util.Stack;

public class Solution {
   public static void reverseStack(Stack<Integer> input, Stack<Integer>
extra) {
       //Your code goes here
       if(input.isEmpty())
           return;
       int temp = input.pop();
       reverseStack(input, extra);
       while(!input.isEmpty()){
           extra.push(input.pop());
       }
       input.push(temp);
       while(!extra.isEmpty()){
           input.push(extra.pop());
       }
   }
}
```

```
Check redundant brackets
Send Feedback
For a given expression in the form of a string, find if there exist any
redundant brackets or not. It is given that the expression contains only
rounded brackets or parenthesis and the input expression will always be
balanced.
A pair of the bracket is said to be redundant when a sub-expression is
surrounded by unnecessary or needless brackets.
Example:

Expression: (a+b)+c
Since there are no needless brackets, hence, the output must be 'false'.

Expression: ((a+b))
The expression can be reduced to (a+b). Hence the expression has redundant
brackets and the output will be 'true'.

Note:

You will not get a partial score for this problem. You will get marks only
if all the test cases are passed.
```

```
Input Format :

The first and the only line of input contains a string expression, without
any spaces in between.

Output Format :

The first and the only line of output will print either 'true' or
'false'(without the quotes) denoting whether the input expression contains
redundant brackets or not.

Note:

You are not required to print the expected result. It has already been
taken care of.

Constraints:

0 <= N <= 10^6
Where N is the length of the expression.

Time Limit: 1 second

Sample Input 1:

a+(b)+c

Sample Output 1:

true

Explanation:

The expression can be reduced to a+b+c. Hence, the brackets are redundant.

Sample Input 2:

(a+b)

Sample Output 2:
```

```
false

import java.util.Stack;

public class Solution {

   public static boolean checkRedundantBrackets(String expression){
        Stack<Character> stk = new Stack<>();

        for(int i=0; i<expression.length(); ++i){
            if(expression.charAt(i)!=')'){
                stk.push(expression.charAt(i));
            }
            else{
                int count = 0;
                while(!stk.isEmpty() && stk.peek()!='('){
                    if(stk.peek()=='+' || stk.peek()=='-' ||
stk.peek()=='/' || stk.peek()=='*')
                        count++;
                    stk.pop();
                }
                if(stk.isEmpty()){
                    return true;
                }
                if(count>0){
                    stk.pop();
                    continue;
                }
                else{
                    return true;
                }
            }
        }
        return false;
   }
}
```

Stock Span
Send Feedback

Afzal has been working with an organization called 'Money Traders' for the past few years. The organization is into the money trading business. His manager assigned him a task. For a given array/list of stock's prices for N days, find the stock's span for each day.
The span of the stock's price today is defined as the maximum number of consecutive days(starting from today and going backwards) for which the price of the stock was less than today's price.
For example, if the price of a stock over a period of 7 days are [100, 80, 60, 70, 60, 75, 85], then the stock spans will be [1, 1, 1, 2, 1, 4, 6].
Explanation:

On the sixth day when the price of the stock was 75, the span came out to be 4, because the last 4 prices(including the current price of 75) were less than the current or the sixth day's price.

Similarly, we can deduce the remaining results.

Afzal has to return an array/list of spans corresponding to each day's stock's price. Help him to achieve the task.
Input Format:

The first line of input contains an integer N, denoting the total number of days.

The second line of input contains the stock prices of each day. A single space will separate them.

Output Format:

The only line of output will print the span for each day's stock price. A single space will separate them.

Note:

You are not required to print the expected output explicitly. It has already been taken care of.

Constraints:

0 <= N <= 10^7

```
1 <= X <= 10^9
Where X denotes the stock's 'price for a day.

Time Limit: 1 second

Sample Input 1:

4
10 10 10 10

Sample Output 1:

1 1 1 1

Sample Input 2:

8
60 70 80 100 90 75 80 120

Sample Output 2:

1 2 3 4 1 1 2 8

import java.util.Stack;

public class Solution {

   public static int[] stockSpan(int[] price) {
       Stack<Integer> st = new Stack<>();
       st.push(0);
       int[] S = new int[price.length];
       S[0] = 1;
       for(int i = 1; i < price.length; i++){
           while((!st.empty()) && (price[st.peek()] < price[i])){
               st.pop();
           }
           S[i] = (st.empty()) ? (i + 1) : (i - st.peek());
           st.push(i);
       }
       return S;
```

```
    }
}
```

```
Minimum bracket Reversal
Send Feedback
For a given expression in the form of a string, find the minimum number of
brackets that can be reversed in order to make the expression balanced.
The expression will only contain curly brackets.
If the expression can't be balanced, return -1.
Example:

Expression: {{{{
If we reverse the second and the fourth opening brackets, the whole
expression will get balanced. Since we have to reverse two brackets to
make the expression balanced, the expected output will be 2.

Expression: {{{
In this example, even if we reverse the last opening bracket, we would be
left with the first opening bracket and hence will not be able to make the
expression balanced and the output will be -1.

Input Format :

The first and the only line of input contains a string expression, without
any spaces in between.

Output Format :

The only line of output will print the number of reversals required to
balance the whole expression. Prints -1, otherwise.

Note:

You don't have to print anything. It has already been taken care of.

Constraints:

0 <= N <= 10^6
```

```
Where N is the length of the expression.

Time Limit: 1sec

Sample Input 1:

{{{

Sample Output 1:

-1

Sample Input 2:

{{{{}}

Sample Output 2:

1

public class Solution {

   public static int countBracketReversals(String input) {
       //Your code goes here
       int countTotal=0, countClose=0;
       for(int i=0; i<input.length(); i++){
           countTotal++;
           if(input.charAt(i)=='}')
               countClose++;
       }
       int openLen = countTotal-countClose;
       if(openLen%2==0)
           return openLen/2;
       return -1;
   }
}
```