Remove Outermost Parentheses

We have given valid Parentheses - It means a bracket must have opening and Closing Parentheses.

Eg1- (()())() , True

Eg2, (()()) , False

↑

This bracket is not Closed

So, question says it have valid parentheses String S, and it will be consider to decomposition

$$(P_1+P_2)(P_3+P_4)(P_5)$$

Output => P_1+P_2 P_3+P_4 P_5 { Need to remove outer bracket }

How we can approach this problem -

Whenever we come across, opening bracket '(' = +1 , counter will inc by 1.

In closing bracket, Counter will decrease by 1.

')' = -1

Initially counter will be 0 {Counter = 0}

When I say "Counter = 0", that means there can be two posibilities -

1. I haven't started traversing across the string yet.
2. Second, the total no of opening parentheses is equal to total no of closing parentheses.

eg. (())

'(' = 2

')' = 2

2 - 2 = 0, both cancels each other.

Let's understand by dry run -

String - (()()) (())

Output - ()() ()

Itc^n 1 - (()())

↑

Initially, my pointer is pointing at first char

Counter = ∅ 1 , As this is a opening bracket, so counter will Increase by 1.

Output - , I will not add this bracket to my output.

Itⁿ 2:> String - (()())(())

Counter = ∅ ✗ 2

Output - (, will add this bracket to the Output

Because 2! = 0, counter is non zero

Itⁿ 3 ⇒ String - (()())(())

this is a closing bracket, so counter will decrease by 1.

Counter = ∅ ✗ ✗ 1 (∵ 1! = 0, This is non-zero, so, will add this to the Output)

Output - ( )

Itⁿ 4 ⇒ String - (()())(())

Counter = ∅ ✗ ✗ 1 2 (2! = 0)

Output - ( ) (

Itr 5 ⇒ String - (( ) ( )) ( )

Counter = 0 1 2 1 2 1 (1 != 0)

Output - ( ) ( )

Itr 5 ⇒ String - (( ) ( ))

Counter = 0 1 2 1 0

Since my counter become 0, I will not include this.

So, the reason is why I haven't took first bracket, whenever I completly handled one composition, counter become zero.

Summary - s = (( ) ( )) ( ) ⇒ ( ) ( ) ( ) (+1, -1)

Counter Before	Action (Add/Ignore)	Counter After
I-1, 0	C>0, False	Ignore	C=1
1	1>0, T	Add	C=2
2	1>0, T	Add	C=1
1	1>0, T	Add	C=2
2	2>0, T	Add	C=1

1 070 ,F Ignore C=0

Opening bracket mil raha to, firstly we are checking counter is greater than 0, if counter will be greater then 0, then it will append.

Then, counter will inc^n

But if will get closing bracket, then counter will decrease first,

And, then if counter will be greater then 0, then it will get appended.

```java
class Solution {
    public String removeOuterParentheses(String s) {
        StringBuilder ans = new StringBuilder();
        int counter = 0;

        for(char ch : s.toCharArray()){
            if(ch == '('){
                if(counter > 0){ // Ignore outermost '('
                    ans.append(ch);
                }
                counter++;
            }else{
                counter--;
                if(counter > 0){
                    ans.append(ch);
                }
            }
        }

        return ans.toString();

    }
}
```

TC - O(n)

SC = O(N), Ans may store N character in the worst case.

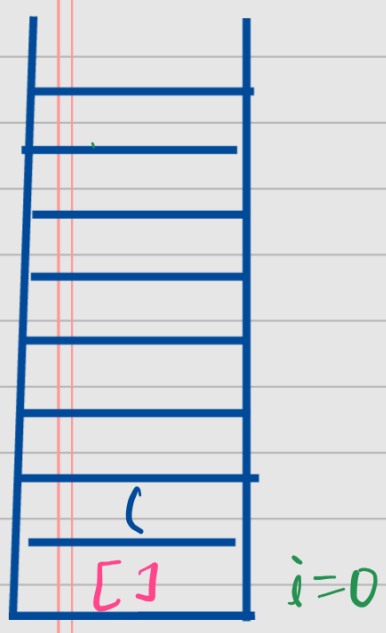
SC - O(1), Extra Space

Stack Approach

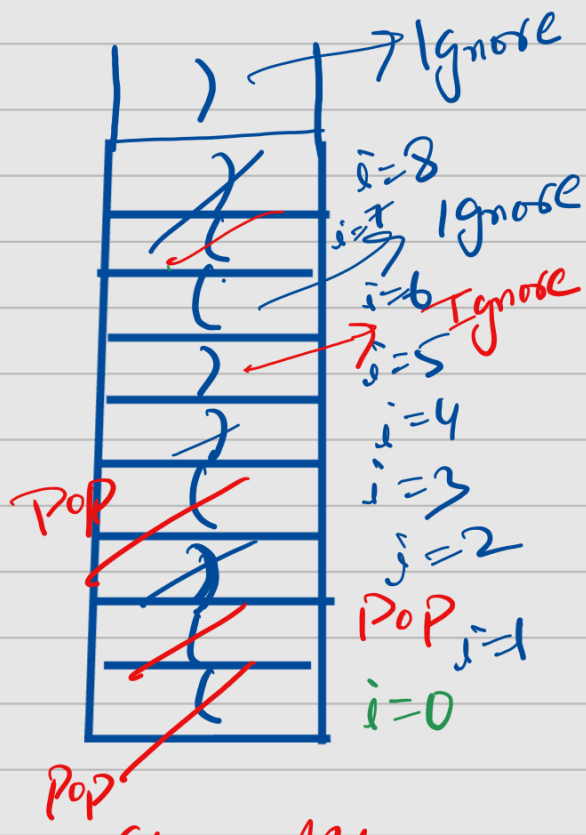
Ans (Store ⇒ SB)

eg ⇒ S = (( ) ( )) (( ))

i=0 i=1 i=2 i=3 i=4 i=5 i=6



Stack Before



Stack After

Ans

"()()()"
"()()("
"()()"
"()("
"()("
"()"
"("
"" (Stack is Empty)

```java
class Solution {
    public String removeOuterParentheses(String s) {

        StringBuilder ans = new StringBuilder();

        Stack<Character> st = new Stack<>();

        for(char ch : s.toCharArray()){
            if(ch == '('){
                if(!st.isEmpty()){   //If stack is not empty then append ch in ans otherwise only push in stack
                    ans.append(ch);
                }
                st.push(ch);
            }
        else{  // ch ==')'
        st.pop(); // If will get closing bracket then will pop() or delete the previous element in the stack because "()" will get a pair

        if(!st.isEmpty()){
                    ans.append(ch);
                }
        }
        }
        return ans.toString();
    }
}
```

TC = O(N)

SC = O(N)

Shreya