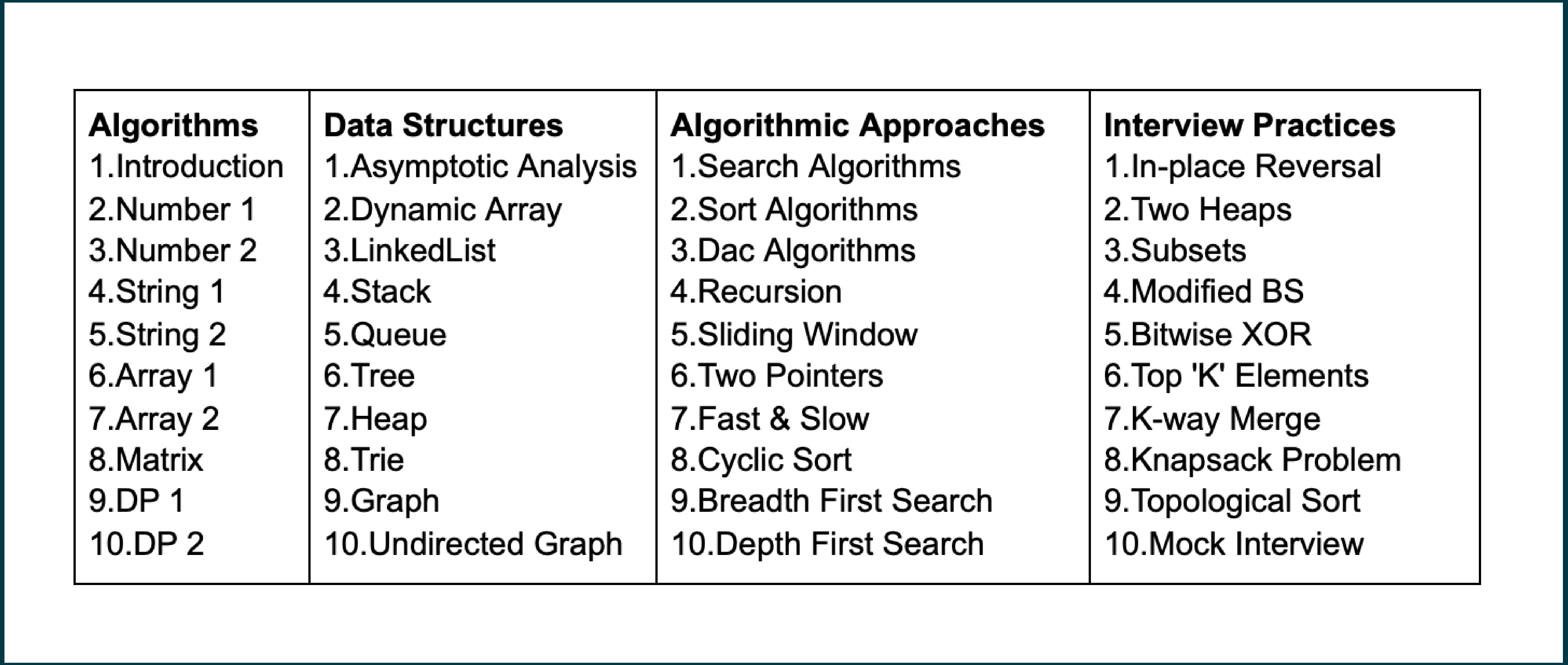
DSA - Algorithms

String 2

Course Planning

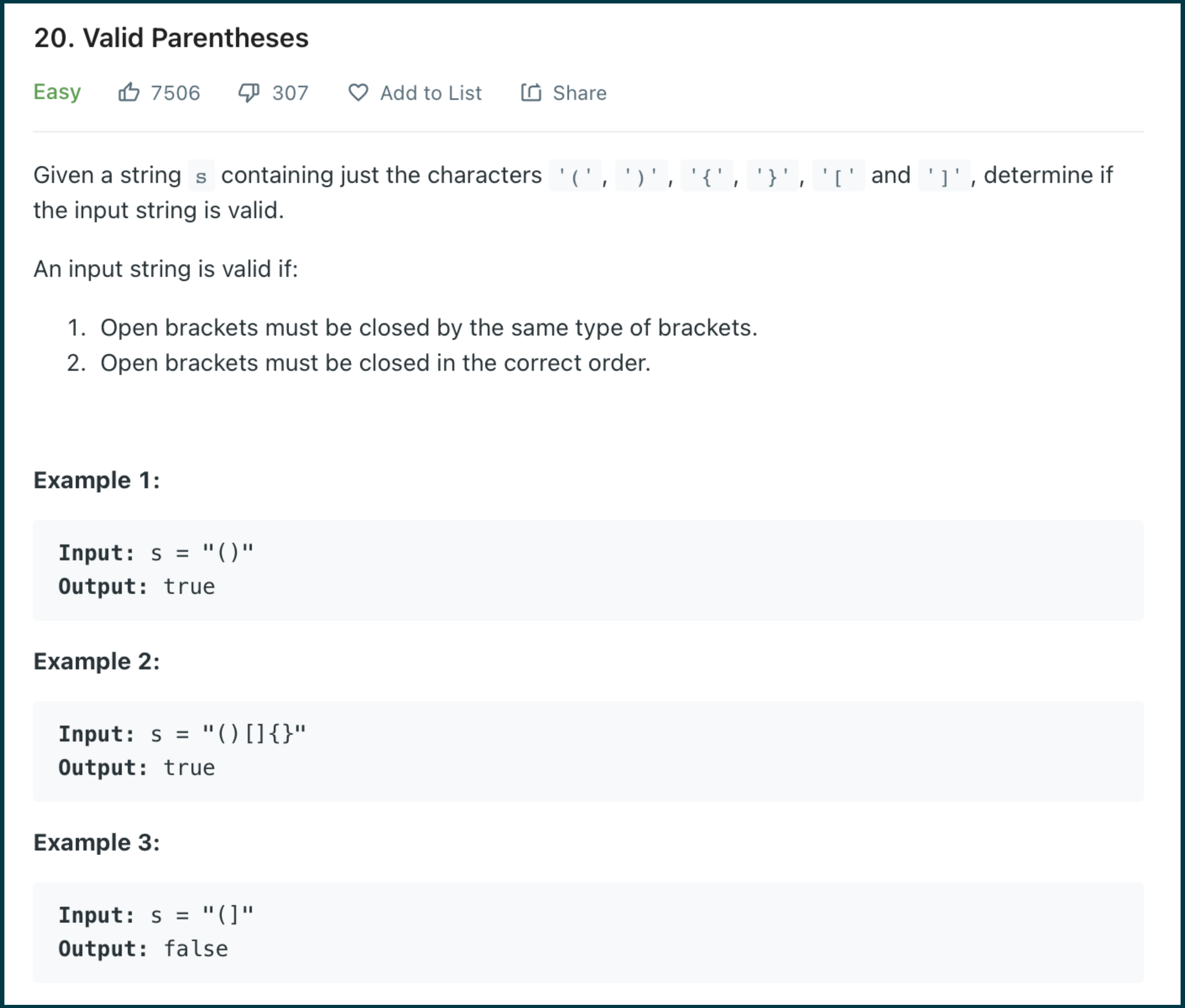


Algorithms	**Data Structures**	**Algorithmic Approaches**	**Interview Practices**
1.Introduction	1.Asymptotic Analysis	1.Search Algorithms	1.In-place Reversal
2.Number 1	2.Dynamic Array	2.Sort Algorithms	2.Two Heaps
3.Number 2	3.LinkedList	3.Dac Algorithms	3.Subsets
4.String 1	4.Stack	4.Recursion	4.Modified BS
5.String 2	5.Queue	5.Sliding Window	5.Bitwise XOR
6.Array 1	6.Tree	6.Two Pointers	6.Top 'K' Elements
7.Array 2	7.Heap	7.Fast & Slow	7.K-way Merge
8.Matrix	8.Trie	8.Cyclic Sort	8.Knapsack Problem
9.DP 1	9.Graph	9.Breadth First Search	9.Topological Sort
10.DP 2	10.Undirected Graph	10.Depth First Search	10.Mock Interview

https://kurbanovxurshid.medium.com/data-structures-and-algorithms-course-plan-1ab912ec1e17

Asked by Facebook

Explanation



20. Valid Parentheses

Easy 7506 307 Add to List Share

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.

Example 1:

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

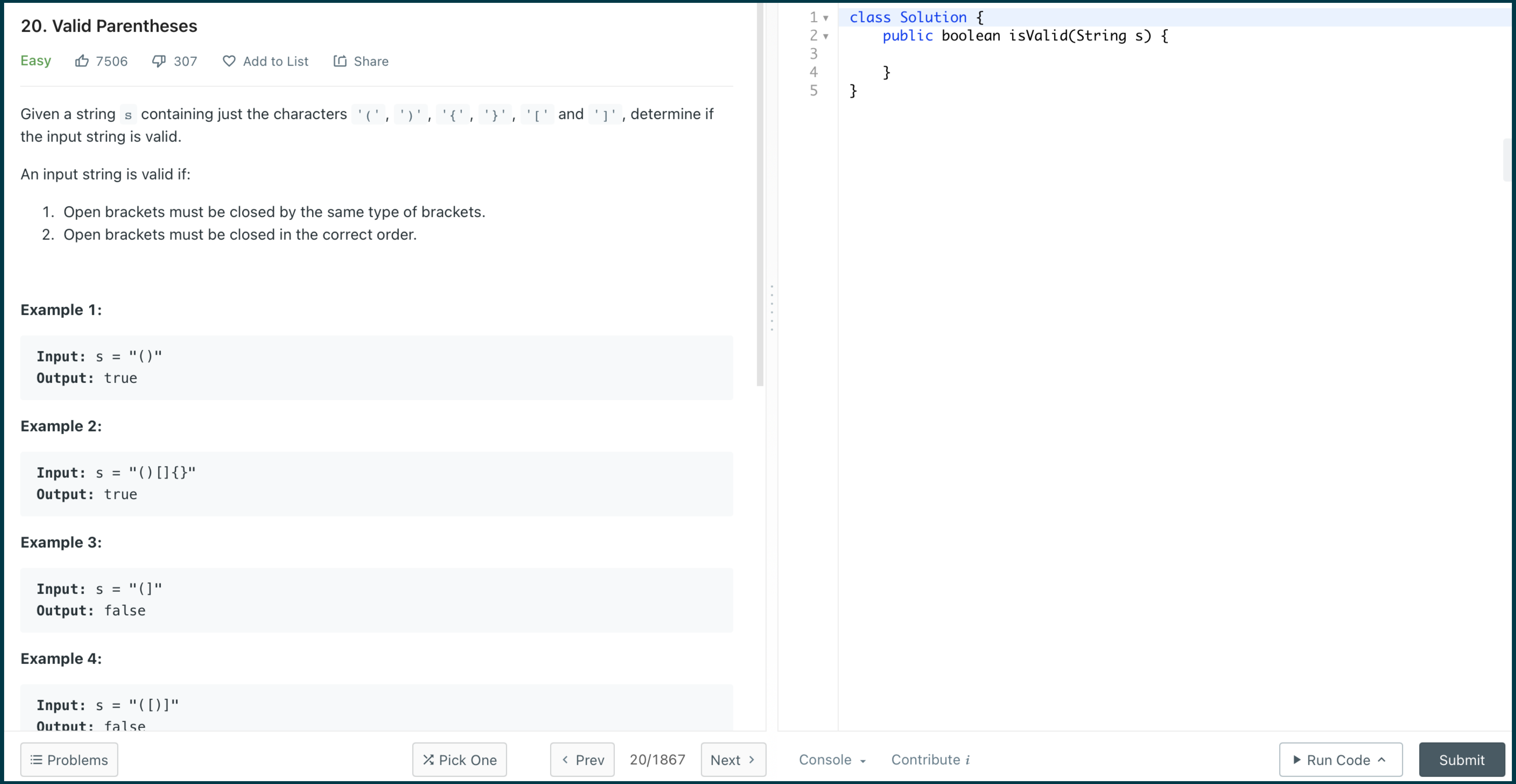
Example 2:

```
Input: s = "()[]{}"
Output: true
```

Example 3:

```
Input: s = "(]"
Output: false
```

Valid Parentheses



20. Valid Parentheses

Easy 7506 307 Add to List Share

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.

Example 1:

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

Example 2:

```
Input: s = "()[]{}"
Output: true
```

Example 3:

```
Input: s = "(]"
Output: false
```

Example 4:

```
Input: s = "([)]"
Output: false
```

```java
class Solution {
    public boolean isValid(String s) {

    }
}
```

https://leetcode.com/problems/valid-parentheses/

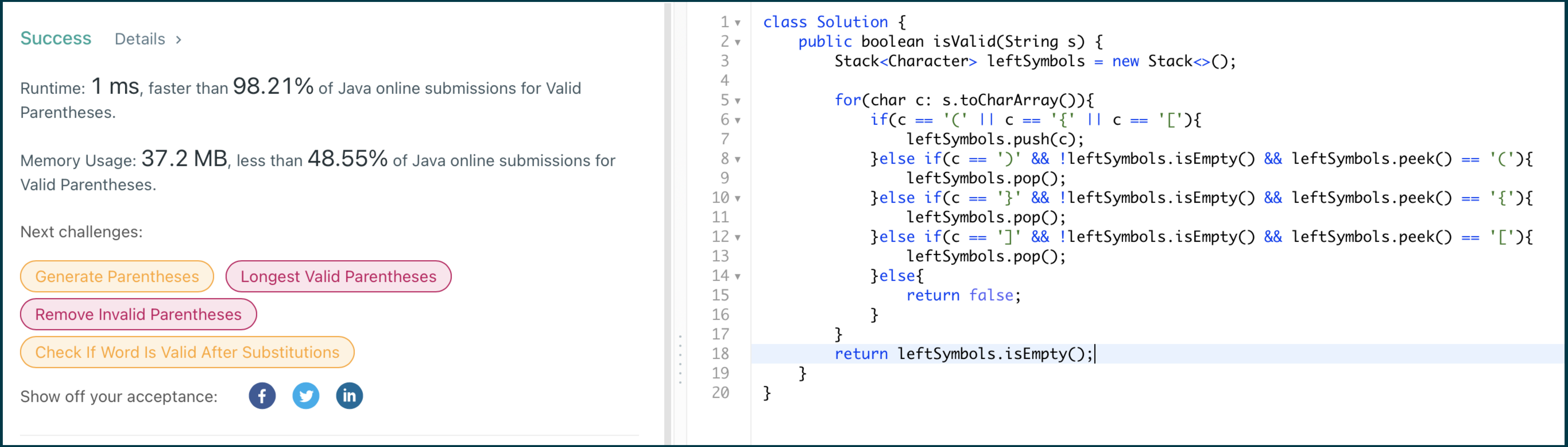
First Theory

Push open parenthesis in Stack
Pop close parenthesis if exists in the top of Stack

S = ( ) [ ] { }

S = ([ )]

First Solution



Success Details ›

Runtime: 1 ms, faster than 98.21% of Java online submissions for Valid Parentheses.

Memory Usage: 37.2 MB, less than 48.55% of Java online submissions for Valid Parentheses.

Next challenges:

Generate Parentheses
Longest Valid Parentheses
Remove Invalid Parentheses
Check If Word Is Valid After Substitutions

Show off your acceptance:

```java
class Solution {
    public boolean isValid(String s) {
        Stack<Character> leftSymbols = new Stack<>();

        for(char c: s.toCharArray()){
            if(c == '(' || c == '{' || c == '['){
                leftSymbols.push(c);
            }else if(c == ')' && !leftSymbols.isEmpty() && leftSymbols.peek() == '('){
                leftSymbols.pop();
            }else if(c == '}' && !leftSymbols.isEmpty() && leftSymbols.peek() == '{'){
                leftSymbols.pop();
            }else if(c == ']' && !leftSymbols.isEmpty() && leftSymbols.peek() == '['){
                leftSymbols.pop();
            }else{
                return false;
            }
        }
        return leftSymbols.isEmpty();
    }
}
```

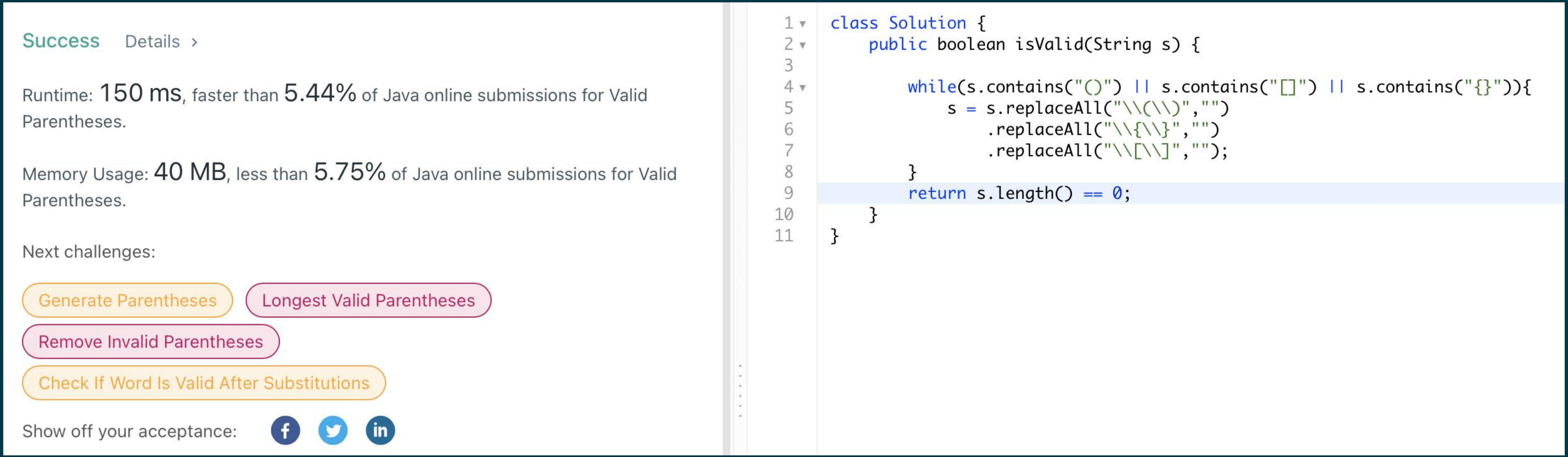
Second Theory

Removing occurrences of “()”, “[]”, and “{}” from the string using String.replaceAll

S = ( ) [ ] { }

S = ([ )]

Second Solution


Success Details ›

Runtime: 150 ms, faster than 5.44% of Java online submissions for Valid Parentheses.

Memory Usage: 40 MB, less than 5.75% of Java online submissions for Valid Parentheses.

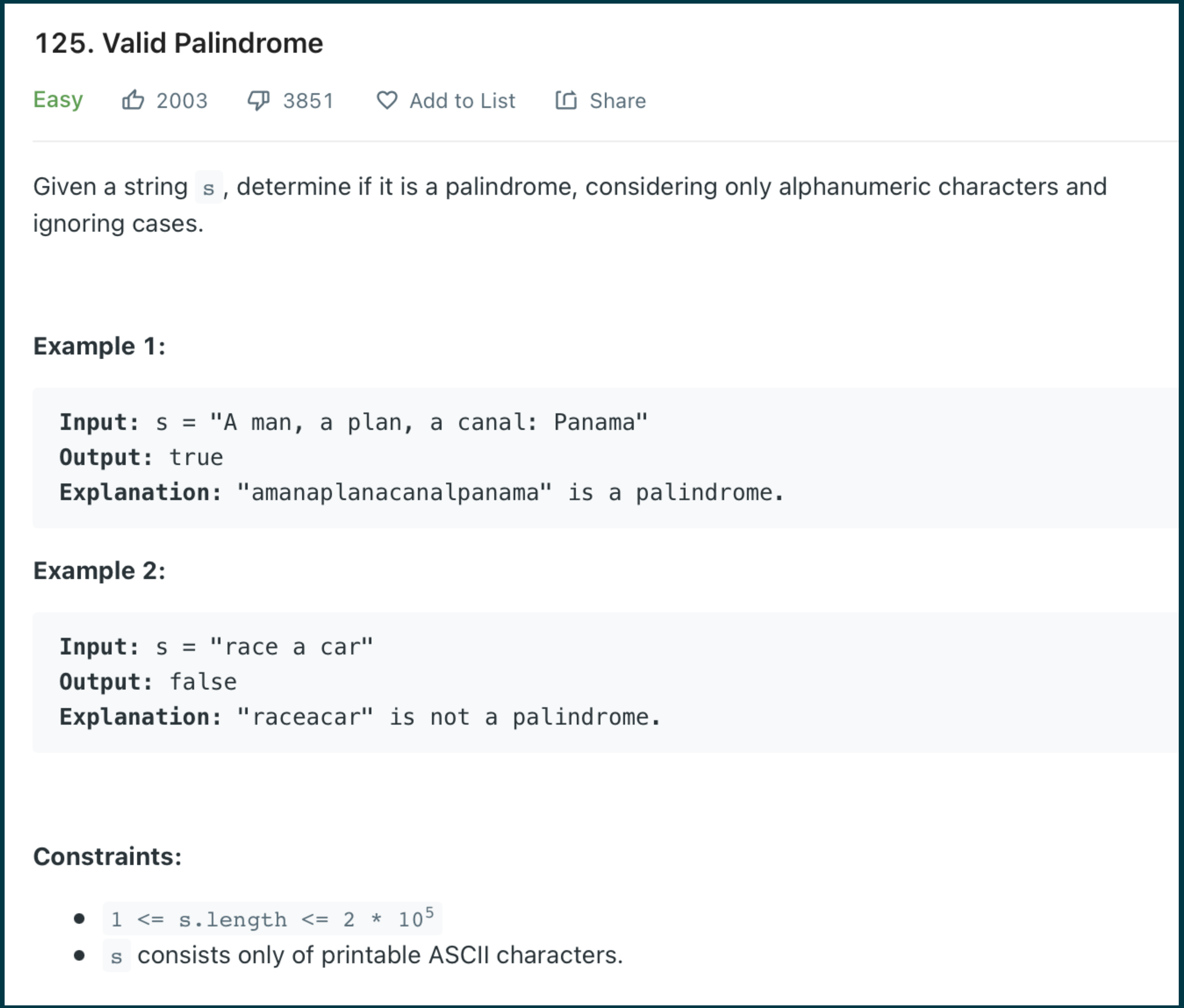
Next challenges:

Generate Parentheses
Longest Valid Parentheses
Remove Invalid Parentheses
Check If Word Is Valid After Substitutions
Show off your acceptance:

```java
class Solution {
    public boolean isValid(String s) {

        while(s.contains("()") || s.contains("[]") || s.contains("{}")){
            s = s.replaceAll("\\(\\)","")
                .replaceAll("\\{\\}","")
                .replaceAll("\\[\\]","");
        }
        return s.length() == 0;
    }
}
```

Task 1 – Valid Palindrome



125. Valid Palindrome

Easy 2003 3851 Add to List Share

Given a string `s`, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

Example 1:

```
Input: s = "A man, a plan, a canal: Panama"
Output: true
Explanation: "amanaplanacanalpanama" is a palindrome.
```

Example 2:

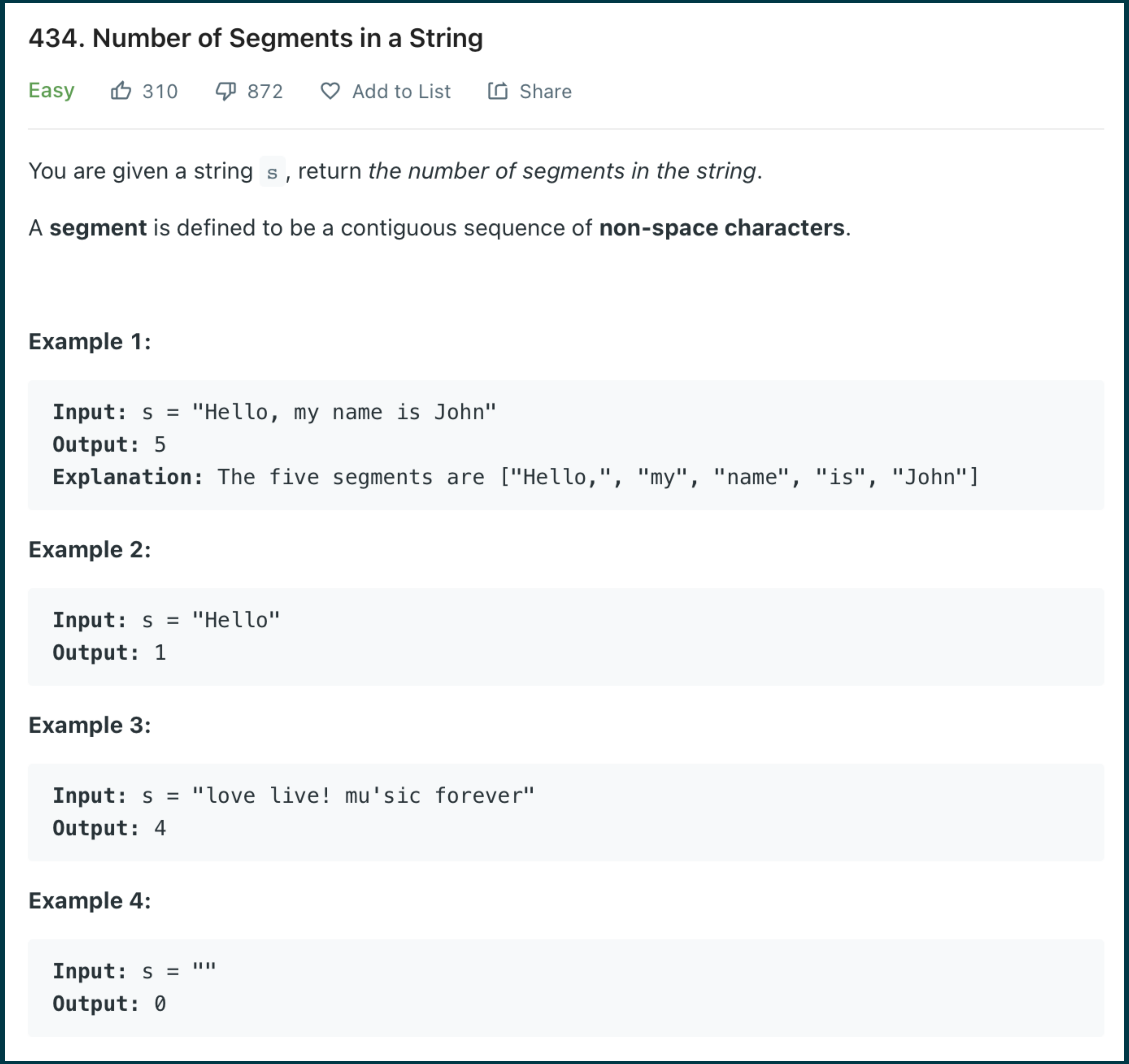
```
Input: s = "race a car"
Output: false
Explanation: "raceacar" is not a palindrome.
```

Constraints:

- `1 <= s.length <= 2 * ` 10^5
- `s` consists only of printable ASCII characters.

https://leetcode.com/problems/valid-palindrome/

Task 2 – Number of Segments in a String



434. Number of Segments in a String

Easy 310 872 Add to List Share

You are given a string `s`, return *the number of segments in the string*.

A **segment** is defined to be a contiguous sequence of **non-space characters**.

Example 1:

```
Input: s = "Hello, my name is John"
Output: 5
Explanation: The five segments are ["Hello,", "my", "name", "is", "John"]
```

Example 2:

```
Input: s = "Hello"
Output: 1
```

Example 3:

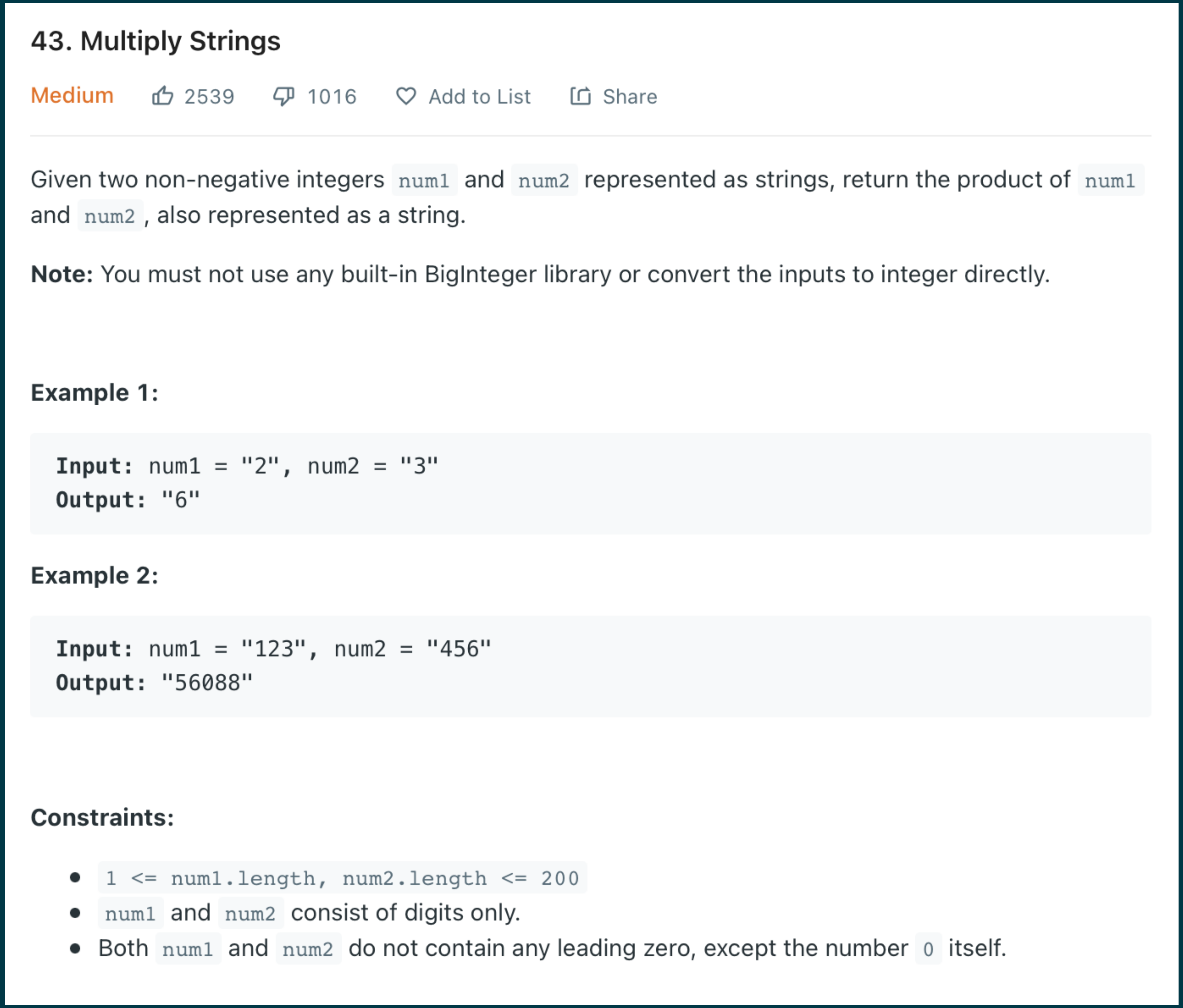
```
Input: s = "love live! mu'sic forever"
Output: 4
```

Example 4:

```
Input: s = ""
Output: 0
```

https://leetcode.com/problems/number-of-segments-in-a-string/

Task 3 – Multiply Strings



43. Multiply Strings

Medium 2539 1016 Add to List Share

Given two non-negative integers `num1` and `num2` represented as strings, return the product of `num1` and `num2`, also represented as a string.

Note: You must not use any built-in BigInteger library or convert the inputs to integer directly.

Example 1:

```
Input: num1 = "2", num2 = "3"
Output: "6"
```

Example 2:

```
Input: num1 = "123", num2 = "456"
Output: "56088"
```

Constraints:

- `1 <= num1.length, num2.length <= 200`
- `num1` and `num2` consist of digits only.
- Both `num1` and `num2` do not contain any leading zero, except the number `0` itself.

https://leetcode.com/problems/multiply-strings/