Codility_

## CodeCheck Report: trainingGJ2Z6B-3ER

Test Name:

Check out Codility training tasks

Summary Timeline

### Tasks summary

| Task | Time spent | Score |
|---|---|---|
| Brackets JavaScript ⚠ | 1 min | 100% |

### Total score

100%

### Tasks Details

Easy

1. **Brackets**

Determine whether a given string of parentheses (multiple types) is properly nested.

| Task Score | Correctness | Performance |
|---|---|---|
| 100% | 100% | 100% |

#### Task description

A string S consisting of N characters is considered to be *properly nested* if any of the following conditions is true:

- S is empty;
- S has the form "(U)" or "[U]" or "{U}" where U is a properly nested string;
- S has the form "VW" where V and W are properly nested strings.

For example, the string "{[()()]}" is properly nested but "([)()]" is not.

Write a function:

```
function solution(S);
```

that, given a string S consisting of N characters, returns 1 if S is properly nested and 0 otherwise.

For example, given S = "{[()()]}", the function should return 1 and given S = "([)()]", the function should return 0, as explained above.

Write an **efficient** algorithm for the following assumptions:

- N is an integer within the range [0..200,000];
- string S consists only of the following characters: "(", "{", "[", "]", "}" and/or ")".



#### Solution

| | |
|---|---|
| Programming language used: | JavaScript |
| Total time used: | 1 minutes |
| Effective time used: | 1 minutes |
| Notes: | *not defined yet* |

#### Task timeline

14:29:49 14:30:30

Code: 14:30:30 UTC, js, final, score: 100

show code in pop-up

```
1  // you can write to stdout for debugging purposes, e.g.
2  // console.log('this is a debug message');
3
4  function solution(S) {
5      // write your code in JavaScript (Node.js 8.9.4)
6      let arrayLength = S.length;
7      if(arrayLength === 0)
8        return 1;
9      if(arrayLength%2 === 1)
10       return 0;
11     const newArray = [];
12     newArray.push(S[0]);
13     let tempValue = "";
14     let mirrorValue = "";
15     for(let i = 1 ; i < arrayLength ; i++) {
16         tempValue = newArray.pop();
17         mirrorValue = "";
18         if(S[i]===")"||S[i]==="}"||S[i]==="]") {
19             if(tempValue === "(")
20                 mirrorValue = ")";
21             else if(tempValue === "{")
22                 mirrorValue = "}";
23             else if(tempValue === "[")
24                 mirrorValue = "]";
25         }
26         if(mirrorValue!==S[i]) {
27             newArray.push(tempValue);
28             newArray.push(S[i]);
29         }
30     }
31     let newArrayLength = newArray.length;
32     if(newArrayLength === 0 || newArray == "")
33         return 1;
34         else
35                 return 0;
36 }
```

#### Analysis summary

The solution obtained perfect score.

#### Analysis

Detected time complexity: **O(N)**

| expand all | Example tests | |
|---|---|---|
| ▶ | example1<br>example test 1 | ✔ OK |
| ▶ | example2<br>example test 2 | ✔ OK |
| expand all | **Correctness tests** | |
| ▶ | negative_match<br>invalid structures | ✔ OK |
| ▶ | empty<br>empty string | ✔ OK |
| ▶ | simple_grouped<br>simple grouped positive and negative test, length=22 | ✔ OK |
| expand all | **Performance tests** | |
| ▶ | large1<br>simple large positive test, 100K ('s followed by 100K )'s + )( | ✔ OK |
| ▶ | large2<br>simple large negative test, 10K+1 ('s followed by 10K )'s + )( + () | ✔ OK |
| ▶ | large_full_ternary_tree<br>tree of the form T=(TTT) and depth 11, length=177K+ | ✔ OK |
| ▶ | multiple_full_binary_trees<br>sequence of full trees of the form T=(TT), depths [1..10..1], with/without some brackets at the end, length=49K+ | ✔ OK |
| ▶ | broad_tree_with_deep_paths<br>string of the form [TTT...T] of 300 T's, each T being '{{{...}}}' nested 200-fold, length=120K+ | ✔ OK |