0001\_Easy\_ValidParentheses\_#20\_Breakdown

Problem:

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.
3. Every close bracket has a corresponding open bracket of the same type.

Examples:

**Example 1:**

**Input:** s = "()"

**Output:** true

**Example 2:**

**Input:** s = "()[]{}"

**Output:** true

**Example 3:**

**Input:** s = "(]"

**Output:** false

Observations:

* The only characters are the three types of brackets.

Questions:

* How much extra space is allowed?

What needs to be true for this problem to work:

* Each unique open parentheses must at some point have a matching closing parentheses, the most recent open parentheses must be closed first in order.

How would a person solve this problem:

* Look through the string, when I find a close parentheses, backtrack through the string to find the most recent open parentheses, if they match delete them, repeat until there are no more parentheses. If at some point there is a pair that should match that do not, return false.

Brute Force:

* Transform the string into a character array. Iterate through the string, for each character add it to a stack. If a character is a close parentheses, pop the previous character, if the characters are not reciprocals, return false, if they are keep iterating through the array. Return true if the size of the array is zero, because all of the characters are popped off.
* Time Complexity: O(n), the array must be fully iterated through, adding and popping to a stack are both O(1).
* Space Complexity: O(n), the maximum amount of space needed is the size of the array in a stack.

Optimize( BUD, bottleneck, unnecessary code, duplicate code):

* If the string size is odd, return false, there cannot be a close parentheses for each open parentheses.
* Instead of adding open parentheses to a stack, add the closing ones and pop them when a match is found.
* Time Complexity: O(n), the array must be fully iterated through.
* Space Complexity: O(n), the maximum amount of space needed is the size of the array in a stack.

Pseudocode:

* Check if the string length is even.
  + If not, return false.
* Create a new stack to hold characters.
* Transform the string to a character array.
* Iterate through the array.
* For each opening character, push a closing character onto the stack
  + If the stack is empty (no opening characters were found, or a closing character does not match, return false.
* Return if the stack is empty.