Input: string = "([])(){}(())()()"
A string made up of brackets and other optional characters

Output: true // it's balanced (has as many opening brackets of a certain type as it has closing brackets of that type and no bracket is unmatched)
// An opening bracket cannot match a closing bracket that comes before it, and similarily, a closing bracket cannot match a corresponding opening bracket that comes after it
// Brackets cannot overlap each other as in [(])
A boolean representing whether the string is balanced with regards to brackets

```
// O(n) time | O(n) space
function balancedBrackets(string) {
  const stack = [];

  const openingBrackets = '([{';
  const closingBrackets = ')]}';

  const brackets = {
    ')': '(',
    ']': '[',
    '}': '{',
  };

  for (const char of string) {
    if (openingBrackets.includes(char)) {
      stack.push(char);
    } else if (closingBrackets.includes(char)) {
      if (stack.length === 0) return false;
      if (stack[stack.length - 1] === brackets[char]) {
        stack.pop();
      } else {
        return false;
      }
    }
  }
  return stack.length === 0;
}
```

Time: O(n) (where n is the # of characters in the input string)
Space: O(n) as the input string could all be in the stack, at worst case (for ex when there are all open brackets)

Idea: Store all possible opening brackets in a string (or an array) and all closing brackets in another (string or array)
Create a hash map of closing brackets and their correspon. opening brackets. This is for constant time look up of what closing bracket equals what opening bracket
Create a stack to store all opening brackets

Loop through every char in the string. If the char is in openingBrackets add it to the string. If it is in closing brackets:
1. If stack is empty return false
2. Stack is not empty so check if opening bracket at top of stack is equal to the corresponding opening bracket of char (found via hash map)
   If it is, pop opening bracket off stack
   If it isn't return false
3. Check if the stack is empty at the end.
   If it is, return true (we popped off all values so we had corresponding closing brackets)
   If it isn't return false since we didn't have enough closing brackets.