BALANCED BRACKETS HACKERRANK

→ Which data structure to use?

A stack is used because it provides a simple and efficient way to track and match open and close brackets. Stack follows a LIFO order.

Here's why stack is used:

1- Matching Brackets:

Brackets must be matched in the correct order. An open bracket '{' must be matched with a close bracket '}' and in the same order they appeas. Using a stack allows to maintain the order.

In the stack based algorithm for checking balanced brackets, the "pop" operation is used to remove elements from the stack and push is used to add the element.

1- Initially, stack is empty.
2- '{' is encountered and pushed onto the stack. Stak contains '{' now.
3- '[' is encountered and pushed to the stack
Stack contains '{''[' now
4- '(' is encountered and pushed to the stack
Stack contains '{''['(' now

5- ')' is encountered and, so '(' is popped from the stack.
Stack contains {, [
6- ']' is encountered, so '[' is popped from the stack.
Stack contains {
7- '}' is encountered, so '{' is popped from the stack.
Stack is empty.

→ When an opening bracket is encontered it is pushed onto the top of stack.

→ When a closing bracket is encountered it is used to check if it matches the most recently added opening bracket. To do this, pop the most recent opening bracket and compared it with the closing bracket

2 Space and Time Efficiency.
A stack uses minimal memory and provides O(1) time complexity

ALGORITHM.

1. Create an empty stack to keep track of opening brackets.
2. Define a dictionary (bracket) to map opening and closing brackets.
3. Iterate through each characters in the input string.
4. For each 'char' in the string.
- If char is an opening bracket { , (or [push it to the stack.
- If char is not an opening bracket
  → If the stack is not empty pop the top element.
  → If bracket [top] is not equal to 'char' return 'NO' because the brackets are

not balanced.

- If the stack is empty return "NO" because there was a closing bracket without a corresponding opening bracket.

5. After processing all characters in the string check if stack is empty.
   - If empty return "Yes".
   - otherwise return "NO"

Yes indicates balanced brackets
No indicates not balanced.