Java\_Easy\_TwoPointers\_ValidPalindrome\_#125

Problem:

A phrase is a **palindrome** if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.

Given a string s, return true if it is a ***palindrome***, or false otherwise.

Questions:

* Is the string modifiable?
* How much space is allowed, am I allowed to copy the string?

What needs to be true for this problem to work:

* All letters need to be lower case.
* All non-alphanumeric characters need to be removed (keep only letters and numbers).
* The string reads the same forwards and backwards.

How would a person solve this:

* Make the string lowercase and remove non-alphanumeric characters.
* Compare the first and last character, telescope inwards until I reach the middle.
  + If at any point two compared characters do not match, the string is not a palindrome.
* Make a copy of the string, reverse it, compare it to the original, comparing from start to finish.
  + If at any point two compared characters do not match, the string is not a palindrome.

Examples:

* **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.
* **Example 3:**
* **Input:** s = " "
* **Output:** true
* **Explanation:** s is an empty string "" after removing non-alphanumeric characters.
* Since an empty string reads the same forward and backward, it is a palindrome.

Examples of Note:

* An empty string is true, and a one letter character of a string is true.
  + A simple check can be added at the beginning.

Brute force:

* Make the string lowercase and remove non-alphanumeric characters.
* Copy the string and reverse it.
* Compare the two strings from start to finish, if at any point two characters are different, return false, else return true
* Time Complexity: O(N)
  + Specifically O(3N), filter, reverse, compare.
* Space Complexity: O(N)
  + Specifically O(2N), for the reversed string.

Optimize, BUD (Bottlenecks, unnecessary work, duplicated work):

* To save on space, the beginning and end of the string can be checked without the need for duplication.
* To save on time, as each character is checked, it is excluded if it is non-alphanumeric, if it is a letter it is made lowercase.
* At any point of the comparison, the characters are not equal, false is returned.
* Time Complexity: O(N), the string is traversed over once.
  + Specifically O(N/2) as the front and back halves are iterated over simultaneously.
* Space Complexity: O(1), happens in place.

Pseudocode:

* Check if the string is less than or equal to 1
  + If it is return true.
* Create two pointers, one for the start at 0 and one for the end at the string’s length -1;
* While start is less than end
  + Create a character for left set equal to the start integer, and a character for right, set equal to the end integer.
  + check if the left character is non-alphanumeric
    - if it is increment start
  + check if the right character is non-alphanumeric
    - if it is decrement end
  + Make the characters left and right lowercase, compare
    - If they are not equal, return false
  + Increment start, decrement end.
* Return true.