is_palindrome

Here's an algorithm to check if a given string is a palindrome:

1. Remove all spaces, punctuation, and convert the string to lowercase. Let's call this modified string `cleanString`.
2. Initialize two pointers, `start` and `end`. `start` will point to the first character of `cleanString`, and `end` will point to the last character of `cleanString`.
3. While `start` is less than or equal to `end`, do the following:
   - If `cleanString [start]` is not equal to `cleanString [end]`, return `false` as it is not a palindrome.
   - Increment `start` by 1 and decrement `end` by 1.
4. If the loop completes without returning `false`, then the string is a palindrome. Return `true`.

```
function isPalindrome(string):

    cleanString =
removeSpacesPunctuationAndConvertToLowercase(string)

    start = 0

    end = length(cleanString) - 1



    while start <= end:

    if cleanString[start] != cleanString[end]:

    return False

    start = start + 1

    end = end - 1



    return True
```

****

Here's the breakdown of the time complexity:

1. Removing spaces, punctuation, and converting the string to lowercase: This operation takes O(n) time because we need to iterate through each character of the string once.
2. The while loop: In the worst case, the loop will iterate half of the string length, which is O(n/2). However, in the big O notation, we ignore constant factors and lower-order terms. So, we can simplify it to O(n).

Therefore, the overall time complexity of the algorithm is O(n).
The space complexity of the algorithm is O(1) because the space used does not depend on the input size. We are only using a few variables to store indices and perform comparisons, which require constant space regardless of the input size.

*****

Alternative solutions

Using String Reversal:

- Reverse the given string.
- Compare the reversed string with the original string.
- If they are equal, return `true`; otherwise, return `false`.

This solution has a time complexity of O(n) for reversing the string and an additional O(n) for string comparison. The overall time complexity is still O(n), but the space complexity will be O(n) since we need to store the reversed string.

```
// String Reversal

function isPalindrome(string) {

  const reversedString = string.split('').reverse().join('');

  return string === reversedString;

}
```