Problem 125: Valid Palindrome

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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

.

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.

Constraints:

1 <= s.length <= 2 * 10

5

s

consists only of printable ASCII characters.

Code Snippets

C++:

```
class Solution {
public:
bool isPalindrome(string s) {

}
};
```

Java:

```
class Solution {
public boolean isPalindrome(String s) {

}
}
```

Python3:

```
class Solution:
def isPalindrome(self, s: str) -> bool:
```

Python:

```
class Solution(object):
def isPalindrome(self, s):
"""
:type s: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} s
* @return {boolean}
*/
var isPalindrome = function(s) {

};
```

TypeScript:

```
function isPalindrome(s: string): boolean {

};
```

C#:

```
public class Solution {
public bool IsPalindrome(string s) {

}
}
```

C:

```
bool isPalindrome(char* s) {

}
```

Go:

```
func isPalindrome(s string) bool {
}
```

Kotlin:

```
class Solution {
fun isPalindrome(s: String): Boolean {

}
}
```

Swift:

```
class Solution {
func isPalindrome(_ s: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn is_palindrome(s: String) -> bool {

}
}
```

Ruby:

```
# @param {String} s
# @return {Boolean}
def is_palindrome(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return Boolean
*/
function isPalindrome($s) {

}
}
```

Dart:

```
class Solution {
bool isPalindrome(String s) {

}
}
```

Scala:

```
object Solution {
def isPalindrome(s: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec is_palindrome(s :: String.t) :: boolean
def is_palindrome(s) do

end
end
```

Erlang:

```
-spec is_palindrome(S :: unicode:unicode_binary()) -> boolean().
is_palindrome(S) ->
.
```

Racket:

```
(define/contract (is-palindrome s)
(-> string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Valid Palindrome
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
bool isPalindrome(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Valid Palindrome
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public boolean isPalindrome(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Valid Palindrome
Difficulty: Easy
Tags: array, string

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def isPalindrome(self, s: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def isPalindrome(self, s):
"""
:type s: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Valid Palindrome
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string} s
* @return {boolean}
*/
var isPalindrome = function(s) {
};
```

TypeScript Solution:

```
/**
* Problem: Valid Palindrome
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function isPalindrome(s: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Valid Palindrome
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool IsPalindrome(string s) {

}
}
```

C Solution:

```
/*
* Problem: Valid Palindrome
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

bool isPalindrome(char* s) {

}
```

Go Solution:

```
// Problem: Valid Palindrome
// Difficulty: Easy
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func isPalindrome(s string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun isPalindrome(s: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func isPalindrome(_ s: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Valid Palindrome
// Difficulty: Easy
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn is_palindrome(s: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {Boolean}
def is_palindrome(s)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return Boolean
*/
function isPalindrome($s) {

}
}
```

Dart Solution:

```
class Solution {
bool isPalindrome(String s) {


}
}
```

Scala Solution:

```
object Solution {
def isPalindrome(s: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec is_palindrome(s :: String.t) :: boolean
def is_palindrome(s) do

end
end
```

Erlang Solution:

```
-spec is_palindrome(S :: unicode:unicode_binary()) -> boolean().
is_palindrome(S) ->
.
```

Racket Solution:

```
(define/contract (is-palindrome s)
(-> string? boolean?)
)
```