

## Exercise - Palindromes (Again)

Just in case you have forgotten, palindromes are words or numbers that are spelled the same both forwards and backwards. There are a number of often seen and famous palindromes:

Ewe  
 Anna  
 Elle  
 Racecar  
 Able was I ere I saw elba!  
 A man, a plan, a canal, Panama!

Using only queues and stacks and a single call to `charAt` for each letter, develop an algorithm for determining whether a `String` is a palindrome or not. This means you do not get to call `charAt` for any letter in the string more than once and you cannot use extra lists or arrays in your algorithm.

Analyze the runtime of your algorithm. As we have already seen in class, there are multiple ways to implement queues and stacks. Think of how this would affect your runtime analysis.

### SOLUTION

Basic idea: store the string in both a queue and a stack at the same time!

When the items are pulled out of the queue and stack, you can compare them to ensure that they are the same. One goes forwards, the other backwards!

```
public static boolean isPalindrome(String aString) {
    Queue<Character> aQueue = new LinkedListQueue<Character>(); // O(1)
    Stack<Character> aStack = new ArrayStack<Character>(); // O(1)

    // get rid of whitespace, punctuation, and make upper or lower
    String normalizedString = normalize(aString); // O(n)

    for(int i = 0; i < normalizedString.length(); i++) { // n repetitions, O(1)
        char c = normalizedString.charAt(i); // O(1)
        aQueue.add(c); // O(1) (note the LinkedListQueue)
        aStack.push(c); // O(1)
    } // n*O(1) = O(n)

    while(!aQueue.isEmpty()){ // n repetitions, O(1) or O(n) time
        if (aQueue.remove() != aStack.pop()) // O(1)+O(1) = O(1)
            return false; // O(1)
    } // n*O(1) = O(n) for a good implementation, n*O(n) = O(n^2) for a bad one
    return true; // O(1)
} // O(n) when using a good ADT implementation, O(n^2) otherwise
```

If the above code used a linked list implementation of a stack, it would not affect the runtime. However, if the queue was a circular array implementation, it might cause the `add` and `remove` methods to run in  $O(n)$  time due to a bad implementation instead of  $O(1)$  time. However, this would not cause worse than  $O(n^2)$  behavior.