Palindrome check

REACTO Week 1: Introduction

Prompt

Given a string, create a function that returns a boolean corresponding to whether that string is a palindrome (spelled the same backwards and forwards). Our palindrome check should be case sensitive.

Examples

isPal('car') => false

isPal('racecar') => true

isPal('RaCecAr') => true

isPal('!? 100 ABCcba 001') => false

Edge cases

isPal('a') => true

isPal('') => true

(all solutions provided account for these edge cases)

Interviewer tips

- Don’t necessarily try to push the interviewee towards any one solution unless they’re struggling, in which case you should pick the solution you understand best and guide them that way.
- Notice that the recursive solution isn’t always the most optimized solution- recursive calls take up space on the call stack.
- If the interviewee tries to use lots built-in methods like split, join, etc., try to steer them away because not every language has those methods available. Two of our solutions use splice, which we’ll talk more about, but the ideal solution is the last one, with pointers.
- If the interviewee isn’t considering edge cases, prompt them to see if they have ideas about what they are and how to handle them

Solution - iterative

Time Complexity:

O(n)

Space Complexity:

O(1)

```
function isPalIterative(str){
  while(str.length > 1){
    let first = str[0].toLowerCase();
    let last = str[str.length - 1].toLowerCase();
    if(first != last) return false
    str = str.slice(1, str.length - 1);
  }
  return true
}
```

Approach - iterative

isPal(‘RaCecAr’)

Convert first and last char to lowercase, check if first and last are equal, if not return false, pass sliced string back into loop. Return true if the stopping condition is false (str.length < 1).

```
First iteration: isPal(‘RaCecAr’)

str: ‘RaCecAr’
first: ‘r’
Last: ‘r’
first === last; continue
```

Second iteration: isPal(‘RaCecAr’)

str: ‘aCecA’

first: ‘a’

Last: ‘a’

first === last; continue

```
Third iteration: isPal(‘RaCecAr’)

str: ‘Cec’

first: ‘c’

Last: ‘c’

first === last; continue
```

Fourth iteration (doesn’t enter loop): isPal(‘RaCecAr’)

str: ‘e’

str.length > 1 ? false; break

return true

Solution - iterative

Time Complexity:

O(n)

Space Complexity:

O(1)

```
function isPalIterative(str){
  while(str.length > 1){
    let first = str[0].toLowerCase();
    let last = str[str.length - 1].toLowerCase();
    if(first != last) return false
    str = str.slice(1, str.length - 1);
  }
  return true
}
```

Solution - Recursive

Time Complexity:

O(n)

Space Complexity:

O(n)

```
function isPalRecursive(str){
  if(str.length <= 1) {
    return true
  } else if (str[0].toLowerCase() !== str[str.length -1 ].toLowerCase()) {
    return false
  } else {
    str = str.slice(1, str.length - 1);
    return isPalRecursive(str)
  }
}
```

Approach - Recursive

isPal(‘RaCecAr’)

Check if the string length is <= 1 (this is our first base case), then check if the first and last chars when converted to lowercase are equal, and if not, return false. Otherwise, slice the string and recurse.

```
isPal(‘RaCecAr’)


str: ‘RaCecAr’                  Call stack
str.length <= 1 ? false         isPal(‘RaCecAr’)
first !== last ? false          isPal(‘aCecA’)
-> return isPal(‘aCecAr’)
```

```
isPal(‘aCecA’)

str: ‘aCecA’                    Call stack

str.length <= 1 ? false         isPal(‘RaCecAr’)

first !== last ? false          isPal(‘aCecA’)

-> return isPal(‘Cec’)          isPal(‘Cec’)
```

```
isPal(‘Cec’)

str: ‘Cec’                      Call stack
str.length <= 1 ? false         isPal(‘RaCecAr’)
first !== last ? false          isPal(‘aCecA’)
-> return isPal(‘e’)            isPal(‘Cec’)
                                isPal(‘e’)
```

```
isPal(‘e’)

str: ‘aCecA’                Call stack

str.length <= 1 ? true      isPal(‘RaCecAr’) pop off (return true)

-> return true              isPal(‘aCecA’) pop off

                            isPal(‘Cec’) pop off

                            isPal(‘e’) returns true, pop off
```

Solution - Recursive

Time Complexity:

O(n)

Space Complexity:

O(n)

```
function isPalRecursive(str){
  if(str.length <= 1) {
    return true
  } else if (str[0].toLowerCase() !== str[str.length -1 ].toLowerCase()) {
    return false
  } else {
    str = str.slice(1, str.length - 1);
    return isPalRecursive(str)
  }
}
```

Solution - pointers

Time Complexity:

O(n)

Space Complexity:

O(1)

```
function isPalPointers(str){
  let lowerCaseStr = str.toLowerCase();
  let leftIdx = 0;
  let rightIdx = lowerCaseStr.length - 1;
  while (leftIdx < rightIdx) {
    if (lowerCaseStr[leftIdx] !== lowerCaseStr[rightIdx]) return false;
    leftIdx++;
    rightIdx--;
  }
  return true;
}
```

Approach - Pointers

isPal('RaCecAr')

Set a left and right pointer and move them inward simultaneously along the string towards the center, checking at each step whether the string chars at left and right are equal, and if so, continuing, decrementing left, and incrementing right.

```
isPal('RaCecAr')

leftIdx: 0
rightIdx: 6
leftIdx < rightIdx ? true
str[leftIdx] === str[rightIdx] ? true
left -> 1; right -> 5
```

```
isPal(‘RaCecAr’)

leftIdx: 1

rightIdx: 5

leftIdx < rightIdx ? true

str[leftIdx] === str[rightIdx] ? true

Left -> 2; right -> 4
```

```
isPal(‘RaCecAr’)

leftIdx: 2

rightIdx: 4

leftIdx < rightIdx ? true

str[leftIdx] === str[rightIdx] ? true

Left -> 3; right -> 3
```

```
isPal(‘RaCecAr’)

leftIdx: 3

rightIdx: 3

leftIdx < rightIdx ? false; don’t enter while loop

-> return true
```

Solution - pointers

Time Complexity:

O(n)

Space Complexity:

O(1)

```
function isPalPointers(str){
  let lowerCaseStr = str.toLowerCase();
  let leftIdx = 0;
  let rightIdx = lowerCaseStr.length - 1;
  while (leftIdx < rightIdx) {
    if (lowerCaseStr[leftIdx] !== lowerCaseStr[rightIdx]) return false;
    leftIdx++;
    rightIdx--;
  }
  return true;
}
```

Links

Gist: https://gist.github.com/jackiefeit94/6675a0241cdc71aab67746101eeb46cc

AlgoExpert: https://www.algoexpert.io/questions/Palindrome%20Check

Medium article (more solutions):
https://medium.com/better-programming/algorithms-101-palindromes-8a06ea97af86