1. Split a String in Balanced Strings

// time complexity: O(N)

// space complexity: O(1)

class Solution {

public int balancedStringSplit(String s) {

int r = 0, ans = 0;

for(int i=0; i<s.length(); i++){

if(s.charAt(i) == 'R'){

r++;

}

else{

r--;

}

if(r == 0){

ans++;

}

}

return ans;

}

}

2. Valid Palindrome

// time complexity: O(N)

// space complexity: O(1)

class Solution:

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

low, high = 0, len(s) - 1;

while low < high:

while low < high and not s[low].isalnum():

low = low + 1

while low < high and not s[high].isalnum():

high = high - 1

if s[low].lower() != s[high].lower():

return False

low += 1

high -= 1

return True

3. Reverse Vowels in a String

// time complexity: O(N)

// space complexity: O(1)

class Solution {

boolean isVowel(char c) {

return c == 'a' || c == 'i' || c == 'e' || c == 'o' || c == 'u'

|| c == 'A' || c == 'I' || c == 'E' || c == 'O' || c == 'U';

}

// Function to swap characters at index x and y

void swap(char[] chars, int x, int y) {

char temp = chars[x];

chars[x] = chars[y];

chars[y] = temp;

}

public String reverseVowels(String s) {

int start = 0;

int end = s.length() - 1;

char[] sChar = s.toCharArray();

while(start < end){

// find the leftmost vowel

while(start < s.length() && !isVowel(sChar[start])){

start++;

}

// find the rightmost vowel

while(end >= 0 && !isVowel(sChar[end])){

end--;

}

if(start < end){

swap(sChar, start++, end--);

}

}

return new String(sChar);

}

}