Problem 541: Reverse String II

Problem Information

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

Problem Description

Given a string

s

and an integer

k

, reverse the first

k

characters for every

2k

characters counting from the start of the string.

If there are fewer than

k

characters left, reverse all of them. If there are less than

2k

but greater than or equal to

k

characters, then reverse the first

k

characters and leave the other as original.

Example 1:

Input:

s = "abcdefg", k = 2

Output:

"bacdfeg"

Example 2:

Input:

s = "abcd", k = 2

Output:

"bacd"

Constraints:

1 <= s.length <= 10

4

s

consists of only lowercase English letters.

1 <= k <= 10

4

Code Snippets

C++:

```
class Solution {
public:
string reverseStr(string s, int k) {

}
};
```

Java:

```
class Solution {
public String reverseStr(String s, int k) {

}
}
```

Python3:

```
class Solution:
def reverseStr(self, s: str, k: int) -> str:
```

Python:

```
class Solution(object):
def reverseStr(self, s, k):
"""
:type s: str
:type k: int
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {number} k
* @return {string}
*/
var reverseStr = function(s, k) {

};
```

TypeScript:

```
function reverseStr(s: string, k: number): string {

};
```

C#:

```
public class Solution {
public string ReverseStr(string s, int k) {

}
}
```

C:

```
char* reverseStr(char* s, int k) {

}
```

Go:

```
func reverseStr(s string, k int) string {

}
```

Kotlin:

```
class Solution {
fun reverseStr(s: String, k: Int): String {

}
}
```

Swift:

```
class Solution {
func reverseStr(_ s: String, _ k: Int) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn reverse_str(s: String, k: i32) -> String {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer} k
# @return {String}
def reverse_str(s, k)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return String
*/
function reverseStr($s, $k) {

}
}
```

Dart:

```
class Solution {
String reverseStr(String s, int k) {



}
}
```

Scala:

```
object Solution {
def reverseStr(s: String, k: Int): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec reverse_str(s :: String.t, k :: integer) :: String.t
def reverse_str(s, k) do

end
end
```

Erlang:

```
-spec reverse_str(S :: unicode:unicode_binary(), K :: integer()) ->
unicode:unicode_binary().
reverse_str(S, K) ->
.
```

Racket:

```
(define/contract (reverse-str s k)
(-> string? exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Reverse String II
* 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:
string reverseStr(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Reverse String II
* 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 String reverseStr(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Reverse String II
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 reverseStr(self, s: str, k: int) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def reverseStr(self, s, k):
"""
:type s: str
:type k: int
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Reverse String II
* 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
* @param {number} k
* @return {string}
*/
var reverseStr = function(s, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Reverse String II
* 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 reverseStr(s: string, k: number): string {

};
```

C# Solution:

```
/*
* Problem: Reverse String II
* 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 string ReverseStr(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: Reverse String II
* 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
```

```
*/

char* reverseStr(char* s, int k) {

}
```

Go Solution:

```
// Problem: Reverse String II
// 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 reverseStr(s string, k int) string {

}
```

Kotlin Solution:

```
class Solution {
fun reverseStr(s: String, k: Int): String {

}
}
```

Swift Solution:

```
class Solution {
func reverseStr(_ s: String, _ k: Int) -> String {

}
}
```

Rust Solution:

```
// Problem: Reverse String II
// 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 reverse_str(s: String, k: i32) -> String {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer} k
# @return {String}
def reverse_str(s, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return String
*/
function reverseStr($s, $k) {

}
}
```

Dart Solution:

```
class Solution {
String reverseStr(String s, int k) {

}
}
```

Scala Solution:

```
object Solution {
def reverseStr(s: String, k: Int): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec reverse_str(s :: String.t, k :: integer) :: String.t
def reverse_str(s, k) do

end
end
```

Erlang Solution:

```
-spec reverse_str(S :: unicode:unicode_binary(), K :: integer()) ->
unicode:unicode_binary().
reverse_str(S, K) ->
.
```

Racket Solution:

```
(define/contract (reverse-str s k)
(-> string? exact-integer? string?)
)
```