Problem 1528: Shuffle String

Problem Information

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

Problem Description

You are given a string

s

and an integer array

indices

of the

same length

. The string

s

will be shuffled such that the character at the

i

th

position moves to

indices[i]

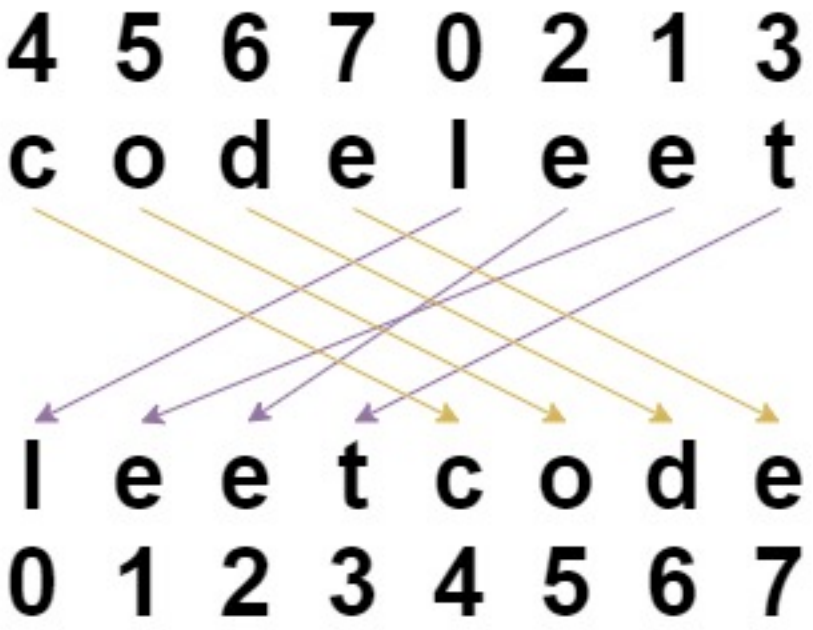
in the shuffled string.

Return

the shuffled string

.

Example 1:



Input:

s = "codeleet",

indices

= [4,5,6,7,0,2,1,3]

Output:

"leetcode"

Explanation:

As shown, "codeleet" becomes "leetcode" after shuffling.

Example 2:

Input:

s = "abc",

indices

= [0,1,2]

Output:

"abc"

Explanation:

After shuffling, each character remains in its position.

Constraints:

s.length == indices.length == n

1 <= n <= 100

s

consists of only lowercase English letters.

0 <= indices[i] < n

All values of

indices

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
string restoreString(string s, vector<int>& indices) {

}
};
```

Java:

```
class Solution {
public String restoreString(String s, int[] indices) {

}
}
```

Python3:

```
class Solution:
def restoreString(self, s: str, indices: List[int]) -> str:
```

Python:

```
class Solution(object):
def restoreString(self, s, indices):
"""
:type s: str
:type indices: List[int]
:rtype: str
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {number[]} indices
```

```
* @return {string}
*/
var restoreString = function(s, indices) {

};
```

TypeScript:

```
function restoreString(s: string, indices: number[]): string {

};
```

C#:

```
public class Solution {
public string RestoreString(string s, int[] indices) {

}
}
```

C:

```
char* restoreString(char* s, int* indices, int indicesSize) {

}
```

Go:

```
func restoreString(s string, indices []int) string {

}
```

Kotlin:

```
class Solution {
fun restoreString(s: String, indices: IntArray): String {

}
}
```

Swift:

```
class Solution {
func restoreString(_ s: String, _ indices: [Int]) -> String {

}
}
```

Rust:

```
impl Solution {
pub fn restore_string(s: String, indices: Vec<i32>) -> String {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer[]} indices
# @return {String}
def restore_string(s, indices)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer[] $indices
* @return String
*/
function restoreString($s, $indices) {

}
}
```

Dart:

```
class Solution {
String restoreString(String s, List<int> indices) {

}
```

```
}
```

Scala:

```
object Solution {
def restoreString(s: String, indices: Array[Int]): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec restore_string(s :: String.t, indices :: [integer]) :: String.t
def restore_string(s, indices) do

end
end
```

Erlang:

```
-spec restore_string(S :: unicode:unicode_binary(), Indices :: [integer()])
-> unicode:unicode_binary().
restore_string(S, Indices) ->
.
```

Racket:

```
(define/contract (restore-string s indices)
(-> string? (listof exact-integer?) string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shuffle String
* 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 restoreString(string s, vector<int>& indices) {

}
};
```

Java Solution:

```
/**
* Problem: Shuffle String
* 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 restoreString(String s, int[] indices) {

}
}
```

Python3 Solution:

```
"""
Problem: Shuffle String
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 restoreString(self, s: str, indices: List[int]) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def restoreString(self, s, indices):
"""
:type s: str
:type indices: List[int]
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Shuffle String
* 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[]} indices
* @return {string}
*/
var restoreString = function(s, indices) {

};
```

TypeScript Solution:

```
/**
* Problem: Shuffle String
* 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 restoreString(s: string, indices: number[]): string {

};
```

C# Solution:

```
/*
* Problem: Shuffle String
* 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 RestoreString(string s, int[] indices) {

}
}
```

C Solution:

```
/*
* Problem: Shuffle String
* 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* restoreString(char* s, int* indices, int indicesSize) {

}
```

Go Solution:

```
// Problem: Shuffle String
// 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 restoreString(s string, indices []int) string {

}
```

Kotlin Solution:

```
class Solution {
fun restoreString(s: String, indices: IntArray): String {

}
}
```

Swift Solution:

```
class Solution {
func restoreString(_ s: String, _ indices: [Int]) -> String {

}
}
```

Rust Solution:

```
// Problem: Shuffle String
// 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 restore_string(s: String, indices: Vec<i32>) -> String {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer[]} indices
# @return {String}
def restore_string(s, indices)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer[] $indices
* @return String
*/
function restoreString($s, $indices) {

}
}
```

Dart Solution:

```
class Solution {
String restoreString(String s, List<int> indices) {

}
}
```

Scala Solution:

```
object Solution {
def restoreString(s: String, indices: Array[Int]): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec restore_string(s :: String.t, indices :: [integer]) :: String.t
def restore_string(s, indices) do

end
end
```

Erlang Solution:

```
-spec restore_string(S :: unicode:unicode_binary(), Indices :: [integer()])
-> unicode:unicode_binary().
restore_string(S, Indices) ->
.
```

Racket Solution:

```
(define/contract (restore-string s indices)
(-> string? (listof exact-integer?) string?)
)
```