Problem 1374: Generate a String With Characters That Have Odd Counts

Problem Information

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

Problem Description

Given an integer

n

,

return a string with

n

characters such that each character in such string occurs

an odd number of times

.

The returned string must contain only lowercase English letters. If there are multiples valid strings, return

any

of them.

Example 1:

Input:

n = 4

Output:

"pppz"

Explanation:

"pppz" is a valid string since the character 'p' occurs three times and the character 'z' occurs once. Note that there are many other valid strings such as "ohhh" and "love".

Example 2:

Input:

n = 2

Output:

"xy"

Explanation:

"xy" is a valid string since the characters 'x' and 'y' occur once. Note that there are many other valid strings such as "ag" and "ur".

Example 3:

Input:

n = 7

Output:

"holasss"

Constraints:

1 <= n <= 500

Code Snippets

C++:

```
class Solution {
public:
string generateTheString(int n) {

}
};
```

Java:

```
class Solution {
public String generateTheString(int n) {

}
}
```

Python3:

```
class Solution:
def generateTheString(self, n: int) -> str:
```

Python:

```
class Solution(object):
def generateTheString(self, n):
"""
:type n: int
:rtype: str
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {string}
*/
```

```
var generateTheString = function(n) {

};
```

TypeScript:

```
function generateTheString(n: number): string {

};
```

C#:

```
public class Solution {
public string GenerateTheString(int n) {

}
}
```

C:

```
char* generateTheString(int n) {

}
```

Go:

```
func generateTheString(n int) string {

}
```

Kotlin:

```
class Solution {
fun generateTheString(n: Int): String {

}
}
```

Swift:

```
class Solution {
func generateTheString(_ n: Int) -> String {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn generate_the_string(n: i32) -> String {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {String}
def generate_the_string(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return String
*/
function generateTheString($n) {

}
}
```

Dart:

```
class Solution {
String generateTheString(int n) {

}
}
```

Scala:

```
object Solution {
def generateTheString(n: Int): String = {

}
}
```

Elixir:

```
defmodule Solution do
@spec generate_the_string(n :: integer) :: String.t
def generate_the_string(n) do

end
end
```

Erlang:

```
-spec generate_the_string(N :: integer()) -> unicode:unicode_binary().
generate_the_string(N) ->
.
```

Racket:

```
(define/contract (generate-the-string n)
(-> exact-integer? string?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Generate a String With Characters That Have Odd Counts
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class Solution {
public:
string generateTheString(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Generate a String With Characters That Have Odd Counts
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public String generateTheString(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Generate a String With Characters That Have Odd Counts
Difficulty: Easy
Tags: string

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def generateTheString(self, n: int) -> str:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def generateTheString(self, n):
"""
:type n: int
:rtype: str
"""
```

JavaScript Solution:

```
/**
* Problem: Generate a String With Characters That Have Odd Counts
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @return {string}
*/
var generateTheString = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Generate a String With Characters That Have Odd Counts
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function generateTheString(n: number): string {
```

```
};
```

C# Solution:

```
/*
* Problem: Generate a String With Characters That Have Odd Counts
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public string GenerateTheString(int n) {

}
}
```

C Solution:

```
/*
* Problem: Generate a String With Characters That Have Odd Counts
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

char* generateTheString(int n) {

}
```

Go Solution:

```
// Problem: Generate a String With Characters That Have Odd Counts
// Difficulty: Easy
```

```
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func generateTheString(n int) string {

}
```

Kotlin Solution:

```
class Solution {
fun generateTheString(n: Int): String {

}
}
```

Swift Solution:

```
class Solution {
func generateTheString(_ n: Int) -> String {

}
}
```

Rust Solution:

```
// Problem: Generate a String With Characters That Have Odd Counts
// Difficulty: Easy
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn generate_the_string(n: i32) -> String {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {String}
def generate_the_string(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return String
*/
function generateTheString($n) {

}
}
```

Dart Solution:

```
class Solution {
String generateTheString(int n) {

}
}
```

Scala Solution:

```
object Solution {
def generateTheString(n: Int): String = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec generate_the_string(n :: integer) :: String.t
def generate_the_string(n) do
```

```
end
end
```

Erlang Solution:

```
-spec generate_the_string(N :: integer()) -> unicode:unicode_binary().
generate_the_string(N) ->
.
```

Racket Solution:

```
(define/contract (generate-the-string n)
(-> exact-integer? string?)
)
```