Problem 2578: Split With Minimum Sum

Problem Information

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

Problem Description

Given a positive integer

num

, split it into two non-negative integers

num1

and

num2

such that:

The concatenation of

num1

and

num2

is a permutation of

num

.

In other words, the sum of the number of occurrences of each digit in

num1

and

num2

is equal to the number of occurrences of that digit in

num

.

num1

and

num2

can contain leading zeros.

Return

the

minimum

possible sum of

num1

and

num2

.

Notes:

It is guaranteed that

num

does not contain any leading zeros.

The order of occurrence of the digits in

num1

and

num2

may differ from the order of occurrence of

num

.

Example 1:

Input:

num = 4325

Output:

59

Explanation:

We can split 4325 so that

num1

is 24 and

num2

is 35, giving a sum of 59. We can prove that 59 is indeed the minimal possible sum.

Example 2:

Input:

num = 687

Output:

75

Explanation:

We can split 687 so that

num1

is 68 and

num2

is 7, which would give an optimal sum of 75.

Constraints:

10 <= num <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int splitNum(int num) {

}
};
```

Java:

```
class Solution {
public int splitNum(int num) {

}
}
```

Python3:

```
class Solution:
def splitNum(self, num: int) -> int:
```

Python:

```
class Solution(object):
def splitNum(self, num):
"""
:type num: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} num
* @return {number}
*/
var splitNum = function(num) {

};
```

TypeScript:

```
function splitNum(num: number): number {
```

```
};
```

C#:

```
public class Solution {
public int SplitNum(int num) {

}
}
```

C:

```
int splitNum(int num) {

}
```

Go:

```
func splitNum(num int) int {

}
```

Kotlin:

```
class Solution {
fun splitNum(num: Int): Int {

}
}
```

Swift:

```
class Solution {
func splitNum(_ num: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn split_num(num: i32) -> i32 {
```

```
}
}
```

Ruby:

```
# @param {Integer} num
# @return {Integer}
def split_num(num)

end
```

PHP:

```
class Solution {

/**
* @param Integer $num
* @return Integer
*/
function splitNum($num) {

}
}
```

Dart:

```
class Solution {
int splitNum(int num) {

}
}
```

Scala:

```
object Solution {
def splitNum(num: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec split_num(num :: integer) :: integer
def split_num(num) do

end
end
```

Erlang:

```
-spec split_num(Num :: integer()) -> integer().
split_num(Num) ->
.
```

Racket:

```
(define/contract (split-num num)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Split With Minimum Sum
* Difficulty: Easy
* Tags: greedy, math, sort
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int splitNum(int num) {

}
};
```

Java Solution:

```
/**
* Problem: Split With Minimum Sum
* Difficulty: Easy
* Tags: greedy, math, sort
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int splitNum(int num) {

}
}
```

Python3 Solution:

```
"""
Problem: Split With Minimum Sum
Difficulty: Easy
Tags: greedy, math, sort

Approach: Greedy algorithm with local optimal choices
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def splitNum(self, num: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def splitNum(self, num):
"""
:type num: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Split With Minimum Sum
* Difficulty: Easy
* Tags: greedy, math, sort
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} num
* @return {number}
*/
var splitNum = function(num) {

};
```

TypeScript Solution:

```
/**
* Problem: Split With Minimum Sum
* Difficulty: Easy
* Tags: greedy, math, sort
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function splitNum(num: number): number {

};
```

C# Solution:

```
/*
* Problem: Split With Minimum Sum
* Difficulty: Easy
* Tags: greedy, math, sort
*
```

```
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int SplitNum(int num) {

}
}
```

C Solution:

```
/*
* Problem: Split With Minimum Sum
* Difficulty: Easy
* Tags: greedy, math, sort
*
* Approach: Greedy algorithm with local optimal choices
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

int splitNum(int num) {

}
```

Go Solution:

```
// Problem: Split With Minimum Sum
// Difficulty: Easy
// Tags: greedy, math, sort
//
// Approach: Greedy algorithm with local optimal choices
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func splitNum(num int) int {

}
```

Kotlin Solution:

```
class Solution {
fun splitNum(num: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func splitNum(_ num: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Split With Minimum Sum
// Difficulty: Easy
// Tags: greedy, math, sort
//
// Approach: Greedy algorithm with local optimal choices
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn split_num(num: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} num
# @return {Integer}
def split_num(num)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $num
* @return Integer
*/
function splitNum($num) {

}
}
```

Dart Solution:

```
class Solution {
int splitNum(int num) {

}
}
```

Scala Solution:

```
object Solution {
def splitNum(num: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec split_num(num :: integer) :: integer
def split_num(num) do

end
end
```

Erlang Solution:

```
-spec split_num(Num :: integer()) -> integer().
split_num(Num) ->
.
```

Racket Solution:

```
(define/contract (split-num num)
(-> exact-integer? exact-integer?)
)
```