Problem 2160: Minimum Sum of Four Digit Number After Splitting Digits

Problem Information

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

Problem Description

You are given a

positive

integer

num

consisting of exactly four digits. Split

num

into two new integers

new1

and

new2

by using the

digits

found in

num

.

Leading zeros

are allowed in

new1

and

new2

, and

all

the digits found in

num

must be used.

For example, given

num = 2932

, you have the following digits: two

2

's, one

9

and one

3

. Some of the possible pairs

[new1, new2]

are

[22, 93]

,

[23, 92]

,

[223, 9]

and

[2, 329]

.

Return

the

minimum

possible sum of

new1

and

new2

.

Example 1:

Input:

num = 2932

Output:

52

Explanation:

Some possible pairs [new1, new2] are [29, 23], [223, 9], etc. The minimum sum can be obtained by the pair [29, 23]: 29 + 23 = 52.

Example 2:

Input:

num = 4009

Output:

13

Explanation:

Some possible pairs [new1, new2] are [0, 49], [490, 0], etc. The minimum sum can be obtained by the pair [4, 9]: 4 + 9 = 13.

Constraints:

1000 <= num <= 9999

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

```
};
```

C#:

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

}
}
```

C:

```
int minimumSum(int num) {

}
```

Go:

```
func minimumSum(num int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

```
}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Sum of Four Digit Number After Splitting Digits
* 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 minimumSum(int num) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Sum of Four Digit Number After Splitting Digits
* 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 minimumSum(int num) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Sum of Four Digit Number After Splitting Digits
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 minimumSum(self, num: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Sum of Four Digit Number After Splitting Digits
* 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 minimumSum = function(num) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Sum of Four Digit Number After Splitting Digits
* 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 minimumSum(num: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Sum of Four Digit Number After Splitting Digits
* 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 MinimumSum(int num) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Sum of Four Digit Number After Splitting Digits
* 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 minimumSum(int num) {

}
```

Go Solution:

```
// Problem: Minimum Sum of Four Digit Number After Splitting Digits
// 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 minimumSum(num int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Sum of Four Digit Number After Splitting Digits
// 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 minimum_sum(num: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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