Problem 3079: Find the Sum of Encrypted Integers

Problem Information

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

Problem Description

You are given an integer array

nums

containing

positive

integers. We define a function

encrypt

such that

encrypt(x)

replaces

every

digit in

x

with the

largest

digit in

x

. For example,

encrypt(523) = 555

and

encrypt(213) = 333

.

Return

the

sum

of encrypted elements

.

Example 1:

Input:

nums = [1,2,3]

Output:

6

Explanation:

The encrypted elements are

[1,2,3]

. The sum of encrypted elements is

1 + 2 + 3 == 6

.

Example 2:

Input:

nums = [10,21,31]

Output:

66

Explanation:

The encrypted elements are

[11,22,33]

. The sum of encrypted elements is

11 + 22 + 33 == 66

.

Constraints:

1 <= nums.length <= 50

1 <= nums[i] <= 1000

Code Snippets

C++:

```
class Solution {
public:
int sumOfEncryptedInt(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int sumOfEncryptedInt(int[] nums) {

}
}
```

Python3:

```
class Solution:
def sumOfEncryptedInt(self, nums: List[int]) -> int:
```

Python:

```
class Solution(object):
def sumOfEncryptedInt(self, nums):
"""
:type nums: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @return {number}
*/
var sumOfEncryptedInt = function(nums) {

};
```

TypeScript:

```
function sumOfEncryptedInt(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int SumOfEncryptedInt(int[] nums) {

}
}
```

C:

```
int sumOfEncryptedInt(int* nums, int numsSize) {

}
```

Go:

```
func sumOfEncryptedInt(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun sumOfEncryptedInt(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func sumOfEncryptedInt(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn sum_of_encrypted_int(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def sum_of_encrypted_int(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function sumOfEncryptedInt($nums) {

}
}
```

Dart:

```
class Solution {
int sumOfEncryptedInt(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def sumOfEncryptedInt(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec sum_of_encrypted_int(nums :: [integer]) :: integer
def sum_of_encrypted_int(nums) do

end
end
```

Erlang:

```
-spec sum_of_encrypted_int(Nums :: [integer()]) -> integer().
sum_of_encrypted_int(Nums) ->
.
```

Racket:

```
(define/contract (sum-of-encrypted-int nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Sum of Encrypted Integers
* Difficulty: Easy
* Tags: array, math
*
* 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:
int sumOfEncryptedInt(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Sum of Encrypted Integers
* Difficulty: Easy
* Tags: array, math
*
* 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 int sumOfEncryptedInt(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Sum of Encrypted Integers
Difficulty: Easy
Tags: array, math

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 sumOfEncryptedInt(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def sumOfEncryptedInt(self, nums):
"""
:type nums: List[int]
:rtype: int
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Sum of Encrypted Integers
* Difficulty: Easy
* Tags: array, math
*
* 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 {number[]} nums
* @return {number}
*/
var sumOfEncryptedInt = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Sum of Encrypted Integers
* Difficulty: Easy
* Tags: array, math
*
* 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 sumOfEncryptedInt(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Find the Sum of Encrypted Integers
* Difficulty: Easy
* Tags: array, math
*
* 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 int SumOfEncryptedInt(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Find the Sum of Encrypted Integers
* Difficulty: Easy
* Tags: array, math
*
* 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
*/

int sumOfEncryptedInt(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Find the Sum of Encrypted Integers
// Difficulty: Easy
// Tags: array, math
//
// 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 sumOfEncryptedInt(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun sumOfEncryptedInt(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func sumOfEncryptedInt(_ nums: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find the Sum of Encrypted Integers
// Difficulty: Easy
// Tags: array, math
//
// 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 sum_of_encrypted_int(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def sum_of_encrypted_int(nums)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function sumOfEncryptedInt($nums) {

}
}
```

Dart Solution:

```
class Solution {
int sumOfEncryptedInt(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def sumOfEncryptedInt(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec sum_of_encrypted_int(nums :: [integer]) :: integer
def sum_of_encrypted_int(nums) do

end
end
```

Erlang Solution:

```
-spec sum_of_encrypted_int(Nums :: [integer()]) -> integer().
sum_of_encrypted_int(Nums) ->
.
```

Racket Solution:

```
(define/contract (sum-of-encrypted-int nums)
(-> (listof exact-integer?) exact-integer?)
)
```