Problem 1748: Sum of Unique Elements

Problem Information

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

Problem Description

You are given an integer array

nums

. The unique elements of an array are the elements that appear

exactly once

in the array.

Return

the

sum

of all the unique elements of

nums

.

Example 1:

Input:

nums = [1,2,3,2]

Output:

4

Explanation:

The unique elements are [1,3], and the sum is 4.

Example 2:

Input:

nums = [1,1,1,1,1]

Output:

0

Explanation:

There are no unique elements, and the sum is 0.

Example 3:

Input:

nums = [1,2,3,4,5]

Output:

15

Explanation:

The unique elements are [1,2,3,4,5], and the sum is 15.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
```

```
 * @return {number}
 */
var sumOfUnique = function(nums) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Sum of Unique Elements
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

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

}
};
```

Java Solution:

```
/**
* Problem: Sum of Unique Elements
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int sumOfUnique(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of Unique Elements
Difficulty: Easy
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def sumOfUnique(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Unique Elements
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Unique Elements
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

function sumOfUnique(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Sum of Unique Elements
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int SumOfUnique(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Sum of Unique Elements
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int sumOfUnique(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Sum of Unique Elements
// Difficulty: Easy
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func sumOfUnique(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sum of Unique Elements
// Difficulty: Easy
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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