Problem 2172: Maximum AND Sum of Array

Problem Information

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

Problem Description

You are given an integer array

nums

of length

n

and an integer

numSlots

such that

2 * numSlots >= n

. There are

numSlots

slots numbered from

1

to

numSlots

.

You have to place all

n

integers into the slots such that each slot contains at

most

two numbers. The

AND sum

of a given placement is the sum of the

bitwise

AND

of every number with its respective slot number.

For example, the

AND sum

of placing the numbers

[1, 3]

into slot

1

and

[4, 6]

into slot

2

is equal to

(1 AND

1

) + (3 AND

1

) + (4 AND

2

) + (6 AND

2

) = 1 + 1 + 0 + 2 = 4

.

Return

the maximum possible

AND sum

of

nums

given

numSlots

slots.

Example 1:

Input:

nums = [1,2,3,4,5,6], numSlots = 3

Output:

9

Explanation:

One possible placement is [1, 4] into slot

1

, [2, 6] into slot

2

, and [3, 5] into slot

3

. This gives the maximum AND sum of (1 AND

1

) + (4 AND

1

) + (2 AND

2

) + (6 AND

2

) + (3 AND

3

) + (5 AND

3

) = 1 + 0 + 2 + 2 + 3 + 1 = 9.

Example 2:

Input:

nums = [1,3,10,4,7,1], numSlots = 9

Output:

24

Explanation:

One possible placement is [1, 1] into slot

1

, [3] into slot

3

, [4] into slot

4

, [7] into slot

7

, and [10] into slot

9

. This gives the maximum AND sum of (1 AND

1

) + (1 AND

1

) + (3 AND

3

) + (4 AND

4

) + (7 AND

7

) + (10 AND

9

) = 1 + 1 + 3 + 4 + 7 + 8 = 24. Note that slots 2, 5, 6, and 8 are empty which is permitted.

Constraints:

n == nums.length

1 <= numSlots <= 9

1 <= n <= 2 * numSlots

1 <= nums[i] <= 15

Code Snippets

C++:

```
class Solution {
public:
int maximumANDSum(vector<int>& nums, int numSlots) {

}
};
```

Java:

```
class Solution {
public int maximumANDSum(int[] nums, int numSlots) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximumANDSum(nums: number[], numSlots: number): number {

};
```

C#:

```
public class Solution {
public int MaximumANDSum(int[] nums, int numSlots) {

}
}
```

C:

```
int maximumANDSum(int* nums, int numsSize, int numSlots) {

}
```

Go:

```
func maximumANDSum(nums []int, numSlots int) int {

}
```

Kotlin:

```
class Solution {
fun maximumANDSum(nums: IntArray, numSlots: Int): Int {

}
}
```

Swift:

```
class Solution {
func maximumANDSum(_ nums: [Int], _ numSlots: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_and_sum(nums: Vec<i32>, num_slots: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} num_slots
# @return {Integer}
def maximum_and_sum(nums, num_slots)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $numSlots
* @return Integer
*/
function maximumANDSum($nums, $numSlots) {

}
}
```

Dart:

```
class Solution {
int maximumANDSum(List<int> nums, int numSlots) {
}
}
```

Scala:

```
object Solution {
def maximumANDSum(nums: Array[Int], numSlots: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_and_sum(nums :: [integer], num_slots :: integer) :: integer
def maximum_and_sum(nums, num_slots) do

end
end
```

Erlang:

```
-spec maximum_and_sum(Nums :: [integer()], NumSlots :: integer()) ->
integer().
maximum_and_sum(Nums, NumSlots) ->
.
```

Racket:

```
(define/contract (maximum-and-sum nums numSlots)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum AND Sum of Array
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int maximumANDSum(vector<int>& nums, int numSlots) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum AND Sum of Array
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int maximumANDSum(int[] nums, int numSlots) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum AND Sum of Array
Difficulty: Hard
Tags: array, dp

Approach: Use two pointers or sliding window technique
```

```

Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum AND Sum of Array
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Maximum AND Sum of Array
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maximumANDSum(nums: number[], numSlots: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum AND Sum of Array
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MaximumANDSum(int[] nums, int numSlots) {

}
}
```

C Solution:

```
/*
* Problem: Maximum AND Sum of Array
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

int maximumANDSum(int* nums, int numsSize, int numSlots) {

}
```

Go Solution:

```
// Problem: Maximum AND Sum of Array
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maximumANDSum(nums []int, numSlots int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumANDSum(nums: IntArray, numSlots: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func maximumANDSum(_ nums: [Int], _ numSlots: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum AND Sum of Array
// Difficulty: Hard
// Tags: array, dp
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn maximum_and_sum(nums: Vec<i32>, num_slots: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} num_slots
# @return {Integer}
def maximum_and_sum(nums, num_slots)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $numSlots
* @return Integer
*/
function maximumANDSum($nums, $numSlots) {

}
}
```

Dart Solution:

```
class Solution {
int maximumANDSum(List<int> nums, int numSlots) {

}
}
```

Scala Solution:

```
object Solution {
def maximumANDSum(nums: Array[Int], numSlots: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_and_sum(nums :: [integer], num_slots :: integer) :: integer
def maximum_and_sum(nums, num_slots) do

end
end
```

Erlang Solution:

```
-spec maximum_and_sum(Nums :: [integer()], NumSlots :: integer()) ->
integer().
maximum_and_sum(Nums, NumSlots) ->
.
```

Racket Solution:

```
(define/contract (maximum-and-sum nums numSlots)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```