Problem 2862: Maximum Element-Sum of a Complete Subset of Indices

Problem Information

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

Problem Description

You are given a

1

-indexed

array

nums

. Your task is to select a

complete subset

from

nums

where every pair of selected indices multiplied is a

perfect square,

. i. e. if you select

a

i

and

a

j

,

i * j

must be a perfect square.

Return the

sum

of the complete subset with the

maximum sum

.

Example 1:

Input:

nums = [8,7,3,5,7,2,4,9]

Output:

16

Explanation:

We select elements at indices 2 and 8 and

2 * 8

is a perfect square.

Example 2:

Input:

nums = [8,10,3,8,1,13,7,9,4]

Output:

20

Explanation:

We select elements at indices 1, 4, and 9.

1 * 4

,

1 * 9

,

4 * 9

are perfect squares.

Constraints:

1 <= n == nums.length <= 10

4

1 <= nums[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long maximumSum(List<Integer> nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long MaximumSum(IList<int> nums) {

}
}
```

C:

```
long long maximumSum(int* nums, int numsSize) {

}
```

Go:

```
func maximumSum(nums []int) int64 {

}
```

Kotlin:

```
class Solution {
fun maximumSum(nums: List<Int>): Long {

}
}
```

Swift:

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

}
```

```
}
```

Rust:

```
impl Solution {
pub fn maximum_sum(nums: Vec<i32>) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

```
object Solution {
def maximumSum(nums: List[Int]): Long = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Element-Sum of a Complete Subset of Indices
* Difficulty: Hard
* 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:
long long maximumSum(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Element-Sum of a Complete Subset of Indices
* Difficulty: Hard
* 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 long maximumSum(List<Integer> nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Element-Sum of a Complete Subset of Indices
Difficulty: Hard
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 maximumSum(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Element-Sum of a Complete Subset of Indices
* Difficulty: Hard
* 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 maximumSum = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Element-Sum of a Complete Subset of Indices
* Difficulty: Hard
* 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 maximumSum(nums: number[]): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Maximum Element-Sum of a Complete Subset of Indices
* Difficulty: Hard
* 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 long MaximumSum(IList<int> nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Element-Sum of a Complete Subset of Indices
* Difficulty: Hard
* 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
*/

long long maximumSum(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximum Element-Sum of a Complete Subset of Indices
// Difficulty: Hard
```

```
// 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 maximumSum(nums []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maximumSum(nums: List<Int>): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Element-Sum of a Complete Subset of Indices
// Difficulty: Hard
// 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 maximum_sum(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

```
object Solution {
def maximumSum(nums: List[Int]): Long = {

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

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