Problem 1799: Maximize Score After N Operations

Problem Information

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

Problem Description

You are given

nums

, an array of positive integers of size

2 * n

. You must perform

n

operations on this array.

In the

i

th

operation

(1-indexed)

, you will:

Choose two elements,

x

and

y

.

Receive a score of

i * gcd(x, y)

.

Remove

x

and

y

from

nums

.

Return

the maximum score you can receive after performing

n

operations.

The function

gcd(x, y)

is the greatest common divisor of

x

and

y

.

Example 1:

Input:

nums = [1,2]

Output:

1

Explanation:

The optimal choice of operations is: (1 * gcd(1, 2)) = 1

Example 2:

Input:

nums = [3,4,6,8]

Output:

11

Explanation:

The optimal choice of operations is: (1 * gcd(3, 6)) + (2 * gcd(4, 8)) = 3 + 8 = 11

Example 3:

Input:

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

Output:

14

Explanation:

The optimal choice of operations is: (1 * gcd(1, 5)) + (2 * gcd(2, 4)) + (3 * gcd(3, 6)) = 1 + 4 + 9 = 14

Constraints:

1 <= n <= 7

nums.length == 2 * n

1 <= nums[i] <= 10

6

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-score nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize Score After N Operations
* Difficulty: Hard
* Tags: array, dp, math
*
* 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 maxScore(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize Score After N Operations
* Difficulty: Hard
* Tags: array, dp, math
*
* 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 maxScore(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize Score After N Operations
Difficulty: Hard
Tags: array, dp, math

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximize Score After N Operations
* Difficulty: Hard
* Tags: array, dp, math
*
* 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
* @return {number}
*/
var maxScore = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize Score After N Operations
* Difficulty: Hard
* Tags: array, dp, math
*
* 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 maxScore(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximize Score After N Operations
* Difficulty: Hard
* Tags: array, dp, math
*
* 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 MaxScore(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximize Score After N Operations
* Difficulty: Hard
* Tags: array, dp, math
*
* 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 maxScore(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximize Score After N Operations
// Difficulty: Hard
// Tags: array, dp, math
//
// 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 maxScore(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximize Score After N Operations
// Difficulty: Hard
// Tags: array, dp, math
//
// 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 max_score(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-score nums)
(-> (listof exact-integer?) exact-integer?)
)
```