Problem 2941: Maximum GCD-Sum of a Subarray

Problem Information

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

Problem Description

You are given an array of integers

nums

and an integer

k

.

The

gcd-sum

of an array

a

is calculated as follows:

Let

s

be the sum of all the elements of

a

.

Let

g

be the

greatest common divisor

of all the elements of

a

.

The gcd-sum of

a

is equal to

s * g

.

Return

the

maximum gcd-sum

of a subarray of

nums

with at least

k

elements.

Example 1:

Input:

nums = [2,1,4,4,4,2], k = 2

Output:

48

Explanation:

We take the subarray [4,4,4], the gcd-sum of this array is 4 * (4 + 4 + 4) = 48. It can be shown that we can not select any other subarray with a gcd-sum greater than 48.

Example 2:

Input:

nums = [7,3,9,4], k = 1

Output:

81

Explanation:

We take the subarray [9], the gcd-sum of this array is 9 * 9 = 81. It can be shown that we can not select any other subarray with a gcd-sum greater than 81.

Constraints:

n == nums.length

1 <= n <= 10

5

1 <= nums[i] <= 10

6

1 <= k <= n

Code Snippets

C++:

```
class Solution {
public:
long long maxGcdSum(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public long maxGcdSum(int[] nums, int k) {

}
}
```

Python3:

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

Python:

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

```
:type k: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function maxGcdSum(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public long MaxGcdSum(int[] nums, int k) {

}
}
```

C:

```
long long maxGcdSum(int* nums, int numsSize, int k) {

}
```

Go:

```
func maxGcdSum(nums []int, k int) int64 {

}
```

Kotlin:

```
class Solution {
fun maxGcdSum(nums: IntArray, k: Int): Long {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def max_gcd_sum(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function maxGcdSum($nums, $k) {

}
```

```
}
```

Dart:

```
class Solution {
int maxGcdSum(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def maxGcdSum(nums: Array[Int], k: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_gcd_sum(nums :: [integer], k :: integer) :: integer
def max_gcd_sum(nums, k) do

end
end
```

Erlang:

```
-spec max_gcd_sum(Nums :: [integer()], K :: integer()) -> integer().
max_gcd_sum(Nums, K) ->
.
```

Racket:

```
(define/contract (max-gcd-sum nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum GCD-Sum of a Subarray
* Difficulty: Hard
* Tags: array, math, search
*
* 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 maxGcdSum(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum GCD-Sum of a Subarray
* Difficulty: Hard
* Tags: array, math, search
*
* 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 maxGcdSum(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum GCD-Sum of a Subarray
Difficulty: Hard
Tags: array, math, search
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum GCD-Sum of a Subarray
* Difficulty: Hard
* Tags: array, math, search
*
* 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
* @param {number} k
* @return {number}
*/
var maxGcdSum = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum GCD-Sum of a Subarray
* Difficulty: Hard
* Tags: array, math, search
*
* 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 maxGcdSum(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum GCD-Sum of a Subarray
* Difficulty: Hard
* Tags: array, math, search
*
* 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 MaxGcdSum(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum GCD-Sum of a Subarray
* Difficulty: Hard
* Tags: array, math, search
*
* 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 maxGcdSum(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum GCD-Sum of a Subarray
// Difficulty: Hard
// Tags: array, math, search
//
// 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 maxGcdSum(nums []int, k int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maxGcdSum(nums: IntArray, k: Int): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum GCD-Sum of a Subarray
// Difficulty: Hard
// Tags: array, math, search
//
// 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 max_gcd_sum(nums: Vec<i32>, k: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def max_gcd_sum(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function maxGcdSum($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maxGcdSum(List<int> nums, int k) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def maxGcdSum(nums: Array[Int], k: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_gcd_sum(nums :: [integer], k :: integer) :: integer
def max_gcd_sum(nums, k) do

end
end
```

Erlang Solution:

```
-spec max_gcd_sum(Nums :: [integer()], K :: integer()) -> integer().
max_gcd_sum(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (max-gcd-sum nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```