Problem 985: Sum of Even Numbers After Queries

Problem Information

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

Problem Description

You are given an integer array

nums

and an array

queries

where

queries[i] = [val

i

, index

i

]

.

For each query

i

, first, apply

nums[index

i

] = nums[index

i

] + val

i

, then print the sum of the even values of

nums

.

Return

an integer array

answer

where

answer[i]

is the answer to the

i

th

query

.

Example 1:

Input:

nums = [1,2,3,4], queries = [[1,0],[-3,1],[-4,0],[2,3]]

Output:

[8,6,2,4]

Explanation:

At the beginning, the array is [1,2,3,4]. After adding 1 to nums[0], the array is [2,2,3,4], and the sum of even values is 2 + 2 + 4 = 8. After adding -3 to nums[1], the array is [2,-1,3,4], and the sum of even values is 2 + 4 = 6. After adding -4 to nums[0], the array is [-2,-1,3,4], and the sum of even values is -2 + 4 = 2. After adding 2 to nums[3], the array is [-2,-1,3,6], and the sum of even values is -2 + 6 = 4.

Example 2:

Input:

nums = [1], queries = [[4,0]]

Output:

[0]

Constraints:

1 <= nums.length <= 10

4

-10

4

<= nums[i] <= 10

4

1 <= queries.length <= 10

4

-10

4

<= val

i

<= 10

4

0 <= index

i

< nums.length

Code Snippets

C++:

```
class Solution {
public:
vector<int> sumEvenAfterQueries(vector<int>& nums, vector<vector<int>>&
queries) {

}
};
```

Java:

```
class Solution {
public int[] sumEvenAfterQueries(int[] nums, int[][] queries) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function sumEvenAfterQueries(nums: number[], queries: number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] SumEvenAfterQueries(int[] nums, int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* sumEvenAfterQueries(int* nums, int numsSize, int** queries, int
queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go:

```
func sumEvenAfterQueries(nums []int, queries [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun sumEvenAfterQueries(nums: IntArray, queries: Array<IntArray>): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

```
}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[][]} queries
# @return {Integer[]}
def sum_even_after_queries(nums, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[][] $queries
* @return Integer[]
*/
function sumEvenAfterQueries($nums, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> sumEvenAfterQueries(List<int> nums, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def sumEvenAfterQueries(nums: Array[Int], queries: Array[Array[Int]]):
Array[Int] = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec sum_even_after_queries(nums :: [integer], queries :: [[integer]]) ::
[integer]
def sum_even_after_queries(nums, queries) do

end
end
```

Erlang:

```
-spec sum_even_after_queries(Nums :: [integer()], Queries :: [[integer()]])
-> [integer()].
sum_even_after_queries(Nums, Queries) ->
.
```

Racket:

```
(define/contract (sum-even-after-queries nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of Even Numbers After Queries
* Difficulty: Medium
* Tags: array
*
* 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:
vector<int> sumEvenAfterQueries(vector<int>& nums, vector<vector<int>>&
queries) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Even Numbers After Queries
* Difficulty: Medium
* Tags: array
*
* 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 int[] sumEvenAfterQueries(int[] nums, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of Even Numbers After Queries
Difficulty: Medium
Tags: array

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 sumEvenAfterQueries(self, nums: List[int], queries: List[List[int]]) ->
List[int]:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Even Numbers After Queries
* Difficulty: Medium
* Tags: array
*
* 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[][]} queries
* @return {number[]}
*/
var sumEvenAfterQueries = function(nums, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Even Numbers After Queries
* Difficulty: Medium
* Tags: array
*
```

```
* 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 sumEvenAfterQueries(nums: number[], queries: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Sum of Even Numbers After Queries
* Difficulty: Medium
* Tags: array
*
* 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 int[] SumEvenAfterQueries(int[] nums, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Sum of Even Numbers After Queries
* Difficulty: Medium
* Tags: array
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
```

```
*/
int* sumEvenAfterQueries(int* nums, int numsSize, int** queries, int
queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Sum of Even Numbers After Queries
// Difficulty: Medium
// Tags: array
//
// 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 sumEvenAfterQueries(nums []int, queries [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun sumEvenAfterQueries(nums: IntArray, queries: Array<IntArray>): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sum of Even Numbers After Queries
// Difficulty: Medium
// Tags: array
```

```
//
// 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 sum_even_after_queries(nums: Vec<i32>, queries: Vec<Vec<i32>>) ->
Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[][]} queries
# @return {Integer[]}
def sum_even_after_queries(nums, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[][] $queries
* @return Integer[]
*/
function sumEvenAfterQueries($nums, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> sumEvenAfterQueries(List<int> nums, List<List<int>> queries) {

}
```

```
}
```

Scala Solution:

```
object Solution {
def sumEvenAfterQueries(nums: Array[Int], queries: Array[Array[Int]]):
Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec sum_even_after_queries(nums :: [integer], queries :: [[integer]]) ::
[integer]
def sum_even_after_queries(nums, queries) do

end
end
```

Erlang Solution:

```
-spec sum_even_after_queries(Nums :: [integer()], Queries :: [[integer()]])
-> [integer()].
sum_even_after_queries(Nums, Queries) ->
.
```

Racket Solution:

```
(define/contract (sum-even-after-queries nums queries)
(-> (listof exact-integer?) (listof (listof exact-integer?)) (listof
exact-integer?))
)
```