Problem 2404: Most Frequent Even Element

Problem Information

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

Problem Description

Given an integer array

nums

, return

the most frequent even element

.

If there is a tie, return the

smallest

one. If there is no such element, return

-1

.

Example 1:

Input:

nums = [0,1,2,2,4,4,1]

Output:

2

Explanation:

The even elements are 0, 2, and 4. Of these, 2 and 4 appear the most. We return the smallest one, which is 2.

Example 2:

Input:

nums = [4,4,4,9,2,4]

Output:

4

Explanation:

4 is the even element appears the most.

Example 3:

Input:

nums = [29,47,21,41,13,37,25,7]

Output:

-1

Explanation:

There is no even element.

Constraints:

1 <= nums.length <= 2000

0 <= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (most-frequent-even nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Most Frequent Even Element
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

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

}
};
```

Java Solution:

```
/**
* Problem: Most Frequent Even Element
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int mostFrequentEven(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Most Frequent Even Element
Difficulty: Easy
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Most Frequent Even Element
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Most Frequent Even Element
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

function mostFrequentEven(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Most Frequent Even Element
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int MostFrequentEven(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Most Frequent Even Element
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int mostFrequentEven(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Most Frequent Even Element
// Difficulty: Easy
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func mostFrequentEven(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Most Frequent Even Element
// Difficulty: Easy
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (most-frequent-even nums)
(-> (listof exact-integer?) exact-integer?)
)
```