Problem 747: Largest Number At Least Twice of Others

Problem Information

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

Problem Description

You are given an integer array

nums

where the largest integer is

unique

.

Determine whether the largest element in the array is

at least twice

as much as every other number in the array. If it is, return

the

index

of the largest element, or return

-1

otherwise

.

Example 1:

Input:

nums = [3,6,1,0]

Output:

1

Explanation:

6 is the largest integer. For every other number in the array x, 6 is at least twice as big as x. The index of value 6 is 1, so we return 1.

Example 2:

Input:

nums = [1,2,3,4]

Output:

-1

Explanation:

4 is less than twice the value of 3, so we return -1.

Constraints:

2 <= nums.length <= 50

0 <= nums[i] <= 100

The largest element in

nums

is unique.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
```

```
* @return {number}
*/
var dominantIndex = function(nums) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (dominant-index nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Largest Number At Least Twice of Others
* Difficulty: Easy
* Tags: array, sort
*
* 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 dominantIndex(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Largest Number At Least Twice of Others
* Difficulty: Easy
* Tags: array, sort
*
* 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 dominantIndex(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Largest Number At Least Twice of Others
Difficulty: Easy
Tags: array, sort

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

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Largest Number At Least Twice of Others
* Difficulty: Easy
* Tags: array, sort
*
* 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 dominantIndex = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Largest Number At Least Twice of Others
* Difficulty: Easy
* Tags: array, sort
*
* 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 dominantIndex(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Largest Number At Least Twice of Others
* Difficulty: Easy
* Tags: array, sort
*
* 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 DominantIndex(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Largest Number At Least Twice of Others
* Difficulty: Easy
* Tags: array, sort
*
* 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
*/

int dominantIndex(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Largest Number At Least Twice of Others
// Difficulty: Easy
// Tags: array, sort
//
// 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 dominantIndex(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Largest Number At Least Twice of Others
// Difficulty: Easy
// Tags: array, sort
//
// 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 dominant_index(nums: Vec<i32>) -> i32 {

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (dominant-index nums)
(-> (listof exact-integer?) exact-integer?)
)
```