Problem 2733: Neither Minimum nor Maximum

Problem Information

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

Problem Description

Given an integer array

nums

containing

distinct

positive

integers, find and return

any

number from the array that is neither the

minimum

nor the

maximum

value in the array, or

-1

if there is no such number.

Return

the selected integer.

Example 1:

Input:

nums = [3,2,1,4]

Output:

2

Explanation:

In this example, the minimum value is 1 and the maximum value is 4. Therefore, either 2 or 3 can be valid answers.

Example 2:

Input:

nums = [1,2]

Output:

-1

Explanation:

Since there is no number in nums that is neither the maximum nor the minimum, we cannot select a number that satisfies the given condition. Therefore, there is no answer.

Example 3:

Input:

nums = [2,1,3]

Output:

2

Explanation:

Since 2 is neither the maximum nor the minimum value in nums, it is the only valid answer.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 100

All values in

nums

are distinct

Code Snippets

C++:

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

}
};
```

Java:

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

}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (find-non-min-or-max nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Neither Minimum nor Maximum
* 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 findNonMinOrMax(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Neither Minimum nor Maximum
* 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 findNonMinOrMax(int[] nums) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Neither Minimum nor Maximum
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 findNonMinOrMax(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Neither Minimum nor Maximum
* 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 findNonMinOrMax = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Neither Minimum nor Maximum
* 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 findNonMinOrMax(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Neither Minimum nor Maximum
* 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 FindNonMinOrMax(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Neither Minimum nor Maximum
* 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 findNonMinOrMax(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Neither Minimum nor Maximum
// 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 findNonMinOrMax(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Neither Minimum nor Maximum
// 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 find_non_min_or_max(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-non-min-or-max nums)
(-> (listof exact-integer?) exact-integer?)
)
```