Problem 1509: Minimum Difference Between Largest and Smallest Value in Three Moves

Problem Information

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

Problem Description

You are given an integer array

nums

.

In one move, you can choose one element of

nums

and change it to

any value

.

Return

the minimum difference between the largest and smallest value of

nums

after performing at most three moves

.

Example 1:

Input:

nums = [5,3,2,4]

Output:

0

Explanation:

We can make at most 3 moves. In the first move, change 2 to 3. nums becomes [5,3,3,4]. In the second move, change 4 to 3. nums becomes [5,3,3,3]. In the third move, change 5 to 3. nums becomes [3,3,3,3]. After performing 3 moves, the difference between the minimum and maximum is 3 - 3 = 0.

Example 2:

Input:

nums = [1,5,0,10,14]

Output:

1

Explanation:

We can make at most 3 moves. In the first move, change 5 to 0. nums becomes [1,0,0,10,14]. In the second move, change 10 to 0. nums becomes [1,0,0,0,14]. In the third move, change 14 to 1. nums becomes [1,0,0,0,1]. After performing 3 moves, the difference between the minimum and maximum is 1 - 0 = 1. It can be shown that there is no way to make the difference 0 in 3 moves.

Example 3:

Input:

nums = [3,100,20]

Output:

0

Explanation:

We can make at most 3 moves. In the first move, change 100 to 7. nums becomes [3,7,20]. In the second move, change 20 to 7. nums becomes [3,7,7]. In the third move, change 3 to 7. nums becomes [7,7,7]. After performing 3 moves, the difference between the minimum and maximum is 7 - 7 = 0.

Constraints:

1 <= nums.length <= 10

5

-10

9

<= nums[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-difference nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Difference Between Largest and Smallest Value in Three
Moves
* Difficulty: Medium
* Tags: array, greedy, 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 minDifference(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Difference Between Largest and Smallest Value in Three
Moves
* Difficulty: Medium
* Tags: array, greedy, 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 minDifference(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Difference Between Largest and Smallest Value in Three Moves
Difficulty: Medium
Tags: array, greedy, 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 minDifference(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Difference Between Largest and Smallest Value in Three
Moves
* Difficulty: Medium
* Tags: array, greedy, 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 minDifference = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Difference Between Largest and Smallest Value in Three
Moves
* Difficulty: Medium
* Tags: array, greedy, 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 minDifference(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Difference Between Largest and Smallest Value in Three
Moves
* Difficulty: Medium
* Tags: array, greedy, 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 MinDifference(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Difference Between Largest and Smallest Value in Three
Moves
* Difficulty: Medium
* Tags: array, greedy, 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 minDifference(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Difference Between Largest and Smallest Value in Three
Moves
// Difficulty: Medium
// Tags: array, greedy, 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 minDifference(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Difference Between Largest and Smallest Value in Three
Moves
// Difficulty: Medium
// Tags: array, greedy, 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 min_difference(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-difference nums)
(-> (listof exact-integer?) exact-integer?)
)
```