Problem 164: Maximum Gap

Problem Information

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

Problem Description

Given an integer array

nums

, return

the maximum difference between two successive elements in its sorted form

. If the array contains less than two elements, return

0

.

You must write an algorithm that runs in linear time and uses linear extra space.

Example 1:

Input:

nums = [3,6,9,1]

Output:

3

Explanation:

The sorted form of the array is [1,3,6,9], either (3,6) or (6,9) has the maximum difference 3.

Example 2:

Input:

nums = [10]

Output:

0

Explanation:

The array contains less than 2 elements, therefore return 0.

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

```
}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (maximum-gap nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Gap
* Difficulty: Medium
* 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 maximumGap(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Gap
* Difficulty: Medium
* 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 maximumGap(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Gap
Difficulty: Medium
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 maximumGap(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Gap
* Difficulty: Medium
```

```
* 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 maximumGap = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Gap
* Difficulty: Medium
* 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 maximumGap(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Gap
* Difficulty: Medium
* 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 MaximumGap(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Gap
* Difficulty: Medium
* 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 maximumGap(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximum Gap
// Difficulty: Medium
// 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 maximumGap(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Gap
// Difficulty: Medium
// 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 maximum_gap(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (maximum-gap nums)
(-> (listof exact-integer?) exact-integer?)
)
```