Problem 2016: Maximum Difference Between Increasing Elements

Problem Information

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

Problem Description

Given a

0-indexed

integer array

nums

of size

n

, find the

maximum difference

between

nums[i]

and

nums[j]

(i.e.,

nums[j] - nums[i]

), such that

0 <= i < j < n

and

nums[i] < nums[j]

.

Return

the

maximum difference

.

If no such

i

and

j

exists, return

-1

.

Example 1:

Input:

nums = [7,

1

,

5

,4]

Output:

4

Explanation:

The maximum difference occurs with i = 1 and j = 2, nums[j] - nums[i] = 5 - 1 = 4. Note that with i = 1 and j = 0, the difference nums[j] - nums[i] = 7 - 1 = 6, but i > j, so it is not valid.

Example 2:

Input:

nums = [9,4,3,2]

Output:

-1

Explanation:

There is no i and j such that i < j and nums[i] < nums[j].

Example 3:

Input:

nums = [

1

,5,2,

10

]

Output:

9

Explanation:

The maximum difference occurs with i = 0 and j = 3, nums[j] - nums[i] = 10 - 1 = 9.

Constraints:

n == nums.length

2 <= n <= 1000

1 <= nums[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Maximum Difference Between Increasing Elements
* Difficulty: Easy
* Tags: array
*
* 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 maximumDifference(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Difference Between Increasing Elements
* Difficulty: Easy
* Tags: array
*
* 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 maximumDifference(int[] nums) {
}
}
```

Python3 Solution:

```
"""
Problem: Maximum Difference Between Increasing Elements
Difficulty: Easy
Tags: array

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Difference Between Increasing Elements
* Difficulty: Easy
* Tags: array
*
* 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 maximumDifference = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Difference Between Increasing Elements
* Difficulty: Easy
* Tags: array
*
* 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 maximumDifference(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Difference Between Increasing Elements
* Difficulty: Easy
* Tags: array
*
* 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 MaximumDifference(int[] nums) {

}
}
```

C Solution:

```c
/*
* Problem: Maximum Difference Between Increasing Elements
* Difficulty: Easy
* Tags: array
*
* 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 maximumDifference(int* nums, int numsSize) {

}
```

Go Solution:

```go
// Problem: Maximum Difference Between Increasing Elements
// Difficulty: Easy
// Tags: array
//
// 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 maximumDifference(nums []int) int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun maximumDifference(nums: IntArray): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Difference Between Increasing Elements
// Difficulty: Easy
// Tags: array
//
// 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_difference(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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