Problem 3357: Minimize the Maximum Adjacent Element Difference

Problem Information

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

Problem Description

You are given an array of integers

nums

. Some values in

nums

are

missing

and are denoted by -1.

You must choose a pair of

positive

integers

(x, y)

exactly once

and replace each

missing element with either x or y. You need to minimize the maximum absolute difference between adjacent elements of nums after replacements. Return the minimum possible difference.

Example 1:

Input:

nums = [1,2,-1,10,8]

Output:

4

Explanation:

By choosing the pair as

(6, 7)

, nums can be changed to

[1, 2, 6, 10, 8]

.

The absolute differences between adjacent elements are:

|1 - 2| == 1

|2 - 6| == 4

|6 - 10| == 4

|10 - 8| == 2

Example 2:

Input:

nums = [-1,-1,-1]

Output:

0

Explanation:

By choosing the pair as

(4, 4)

, nums can be changed to

[4, 4, 4]

.

Example 3:

Input:

nums = [-1,10,-1,8]

Output:

1

Explanation:

By choosing the pair as

(11, 9)

, nums can be changed to

[11, 10, 9, 8]

.

Constraints:

2 <= nums.length <= 10

5

nums[i]

is either -1 or in the range

[1, 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: Minimize the Maximum Adjacent Element Difference
* Difficulty: Hard
* Tags: array, greedy, search
*
* 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: Minimize the Maximum Adjacent Element Difference
* Difficulty: Hard
* Tags: array, greedy, search
*
* 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: Minimize the Maximum Adjacent Element Difference
Difficulty: Hard
Tags: array, greedy, search

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: Minimize the Maximum Adjacent Element Difference
* Difficulty: Hard
* Tags: array, greedy, search
*
* 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: Minimize the Maximum Adjacent Element Difference
* Difficulty: Hard
* Tags: array, greedy, search
*
* 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: Minimize the Maximum Adjacent Element Difference
* Difficulty: Hard
* Tags: array, greedy, search
*
* 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: Minimize the Maximum Adjacent Element Difference
* Difficulty: Hard
* Tags: array, greedy, search
*
* 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: Minimize the Maximum Adjacent Element Difference
// Difficulty: Hard
// Tags: array, greedy, search
//
// 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: Minimize the Maximum Adjacent Element Difference
// Difficulty: Hard
// Tags: array, greedy, search
//
// 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?)
)
```