Problem 2817: Minimum Absolute Difference Between Elements With Constraint

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

and an integer

x

.

Find the

minimum absolute difference

between two elements in the array that are at least

x

indices apart.

In other words, find two indices

i

and

j

such that

abs(i - j) >= x

and

abs(nums[i] - nums[j])

is minimized.

Return

an integer denoting the

minimum

absolute difference between two elements that are at least

x

indices apart

.

Example 1:

Input:

nums = [4,3,2,4], x = 2

Output:

0

Explanation:

We can select nums[0] = 4 and nums[3] = 4. They are at least 2 indices apart, and their absolute difference is the minimum, 0. It can be shown that 0 is the optimal answer.

Example 2:

Input:

nums = [5,3,2,10,15], x = 1

Output:

1

Explanation:

We can select nums[1] = 3 and nums[2] = 2. They are at least 1 index apart, and their absolute difference is the minimum, 1. It can be shown that 1 is the optimal answer.

Example 3:

Input:

nums = [1,2,3,4], x = 3

Output:

3

Explanation:

We can select nums[0] = 1 and nums[3] = 4. They are at least 3 indices apart, and their absolute difference is the minimum, 3. It can be shown that 3 is the optimal answer.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

0 <= x < nums.length

Code Snippets

C++:

```
class Solution {
public:
int minAbsoluteDifference(vector<int>& nums, int x) {

}
};
```

Java:

```
class Solution {
public int minAbsoluteDifference(List<Integer> nums, int x) {

}
}
```

Python3:

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

Python:

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

```
:type x: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function minAbsoluteDifference(nums: number[], x: number): number {

};
```

C#:

```
public class Solution {
public int MinAbsoluteDifference(IList<int> nums, int x) {

}
}
```

C:

```
int minAbsoluteDifference(int* nums, int numsSize, int x) {

}
```

Go:

```
func minAbsoluteDifference(nums []int, x int) int {

}
```

Kotlin:

```
class Solution {
fun minAbsoluteDifference(nums: List<Int>, x: Int): Int {

}
}
```

Swift:

```
class Solution {
func minAbsoluteDifference(_ nums: [Int], _ x: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_absolute_difference(nums: Vec<i32>, x: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} x
# @return {Integer}
def min_absolute_difference(nums, x)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $x
* @return Integer
*/
function minAbsoluteDifference($nums, $x) {

}
```

```
}
```

Dart:

```
class Solution {
int minAbsoluteDifference(List<int> nums, int x) {

}
}
```

Scala:

```
object Solution {
def minAbsoluteDifference(nums: List[Int], x: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_absolute_difference(nums :: [integer], x :: integer) :: integer
def min_absolute_difference(nums, x) do

end
end
```

Erlang:

```
-spec min_absolute_difference(Nums :: [integer()], X :: integer()) ->
integer().
min_absolute_difference(Nums, X) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Absolute Difference Between Elements With Constraint
* Difficulty: Medium
* Tags: array, 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 minAbsoluteDifference(vector<int>& nums, int x) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Absolute Difference Between Elements With Constraint
* Difficulty: Medium
* Tags: array, 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 minAbsoluteDifference(List<Integer> nums, int x) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Absolute Difference Between Elements With Constraint
```

```
Difficulty: Medium
Tags: array, 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 minAbsoluteDifference(self, nums: List[int], x: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Absolute Difference Between Elements With Constraint
* Difficulty: Medium
* Tags: array, 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
* @param {number} x
* @return {number}
*/
var minAbsoluteDifference = function(nums, x) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Minimum Absolute Difference Between Elements With Constraint
* Difficulty: Medium
* Tags: array, 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 minAbsoluteDifference(nums: number[], x: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Absolute Difference Between Elements With Constraint
* Difficulty: Medium
* Tags: array, 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 MinAbsoluteDifference(IList<int> nums, int x) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Absolute Difference Between Elements With Constraint
```

```
* Difficulty: Medium
* Tags: array, 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 minAbsoluteDifference(int* nums, int numsSize, int x) {

}
```

Go Solution:

```
// Problem: Minimum Absolute Difference Between Elements With Constraint
// Difficulty: Medium
// Tags: array, 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 minAbsoluteDifference(nums []int, x int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minAbsoluteDifference(nums: List<Int>, x: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minAbsoluteDifference(_ nums: [Int], _ x: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Absolute Difference Between Elements With Constraint
// Difficulty: Medium
// Tags: array, 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_absolute_difference(nums: Vec<i32>, x: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} x
# @return {Integer}
def min_absolute_difference(nums, x)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $x
* @return Integer
*/
function minAbsoluteDifference($nums, $x) {

}
}
```

Dart Solution:

```
class Solution {
int minAbsoluteDifference(List<int> nums, int x) {

}
}
```

Scala Solution:

```
object Solution {
def minAbsoluteDifference(nums: List[Int], x: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_absolute_difference(nums :: [integer], x :: integer) :: integer
def min_absolute_difference(nums, x) do

end
end
```

Erlang Solution:

```
-spec min_absolute_difference(Nums :: [integer()], X :: integer()) ->
integer().
min_absolute_difference(Nums, X) ->
.
```

Racket Solution:

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