Problem 1848: Minimum Distance to the Target Element

Problem Information

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

Problem Description

Given an integer array

nums

(0-indexed)

and two integers

target

and

start

, find an index

i

such that

nums[i] == target

and

abs(i - start)

is

minimized

. Note that

abs(x)

is the absolute value of

x

.

Return

abs(i - start)

.

It is

guaranteed

that

target

exists in

nums

.

Example 1:

Input:

nums = [1,2,3,4,5], target = 5, start = 3

Output:

1

Explanation:

nums[4] = 5 is the only value equal to target, so the answer is abs(4 - 3) = 1.

Example 2:

Input:

nums = [1], target = 1, start = 0

Output:

0

Explanation:

nums[0] = 1 is the only value equal to target, so the answer is abs(0 - 0) = 0.

Example 3:

Input:

nums = [1,1,1,1,1,1,1,1,1,1], target = 1, start = 0

Output:

0

Explanation:

Every value of nums is 1, but nums[0] minimizes abs(i - start), which is abs(0 - 0) = 0.

Constraints:

1 <= nums.length <= 1000

1 <= nums[i] <= 10

4

0 <= start < nums.length

target

is in

nums

.

Code Snippets

C++:

```
class Solution {
public:
int getMinDistance(vector<int>& nums, int target, int start) {

}
};
```

Java:

```
class Solution {
public int getMinDistance(int[] nums, int target, int start) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} target
* @param {number} start
* @return {number}
*/
var getMinDistance = function(nums, target, start) {

};
```

TypeScript:

```
function getMinDistance(nums: number[], target: number, start: number):
number {

};
```

C#:

```
public class Solution {
public int GetMinDistance(int[] nums, int target, int start) {

}
}
```

C:

```
int getMinDistance(int* nums, int numsSize, int target, int start) {

}
```

Go:

```
func getMinDistance(nums []int, target int, start int) int {

}
```

Kotlin:

```
class Solution {
fun getMinDistance(nums: IntArray, target: Int, start: Int): Int {

}
}
```

Swift:

```
class Solution {
func getMinDistance(_ nums: [Int], _ target: Int, _ start: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn get_min_distance(nums: Vec<i32>, target: i32, start: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} target
# @param {Integer} start
# @return {Integer}
def get_min_distance(nums, target, start)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $target
* @param Integer $start
* @return Integer
*/
function getMinDistance($nums, $target, $start) {

}
}
```

Dart:

```
class Solution {
int getMinDistance(List<int> nums, int target, int start) {

}
}
```

Scala:

```
object Solution {
def getMinDistance(nums: Array[Int], target: Int, start: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec get_min_distance(nums :: [integer], target :: integer, start ::
integer) :: integer
def get_min_distance(nums, target, start) do

end
end
```

Erlang:

```
-spec get_min_distance(Nums :: [integer()], Target :: integer(), Start ::
integer()) -> integer().
```

```
get_min_distance(Nums, Target, Start) ->
.
```

Racket:

```
(define/contract (get-min-distance nums target start)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Distance to the Target Element
* 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 getMinDistance(vector<int>& nums, int target, int start) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Distance to the Target Element
* 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 getMinDistance(int[] nums, int target, int start) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Distance to the Target Element
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 getMinDistance(self, nums: List[int], target: int, start: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Distance to the Target Element
* 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
* @param {number} target
* @param {number} start
* @return {number}
*/
var getMinDistance = function(nums, target, start) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Distance to the Target Element
* 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 getMinDistance(nums: number[], target: number, start: number):
number {

};
```

C# Solution:

```
/*
* Problem: Minimum Distance to the Target Element
* 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 GetMinDistance(int[] nums, int target, int start) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Distance to the Target Element
* 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 getMinDistance(int* nums, int numsSize, int target, int start) {

}
```

Go Solution:

```
// Problem: Minimum Distance to the Target Element
// 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 getMinDistance(nums []int, target int, start int) int {

}
```

Kotlin Solution:

```
class Solution {
fun getMinDistance(nums: IntArray, target: Int, start: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func getMinDistance(_ nums: [Int], _ target: Int, _ start: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Distance to the Target Element
// 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 get_min_distance(nums: Vec<i32>, target: i32, start: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} target
# @param {Integer} start
# @return {Integer}
def get_min_distance(nums, target, start)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $target
* @param Integer $start
* @return Integer
*/
function getMinDistance($nums, $target, $start) {

}
}
```

Dart Solution:

```
class Solution {
int getMinDistance(List<int> nums, int target, int start) {

}
}
```

Scala Solution:

```
object Solution {
def getMinDistance(nums: Array[Int], target: Int, start: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec get_min_distance(nums :: [integer], target :: integer, start ::
integer) :: integer
def get_min_distance(nums, target, start) do

end
end
```

Erlang Solution:

```
-spec get_min_distance(Nums :: [integer()], Target :: integer(), Start ::
integer()) -> integer().
get_min_distance(Nums, Target, Start) ->
.
```

Racket Solution:

```
(define/contract (get-min-distance nums target start)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```