Problem 2059: Minimum Operations to Convert Number

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

containing

distinct

numbers, an integer

start

, and an integer

goal

. There is an integer

x

that is initially set to

start

, and you want to perform operations on

x

such that it is converted to

goal

. You can perform the following operation repeatedly on the number

x

:

If

0 <= x <= 1000

, then for any index

i

in the array (

0 <= i < nums.length

), you can set

x

to any of the following:

x + nums[i]

x - nums[i]

x ^ nums[i]

(bitwise-XOR)

Note that you can use each

nums[i]

any number of times in any order. Operations that set

x

to be out of the range

0 <= x <= 1000

are valid, but no more operations can be done afterward.

Return

the

minimum

number of operations needed to convert

x = start

into

goal

, and

-1

if it is not possible

.

Example 1:

Input:

nums = [2,4,12], start = 2, goal = 12

Output:

2

Explanation:

We can go from 2 → 14 → 12 with the following 2 operations. - 2 + 12 = 14 - 14 - 2 = 12

Example 2:

Input:

nums = [3,5,7], start = 0, goal = -4

Output:

2

Explanation:

We can go from 0 → 3 → -4 with the following 2 operations. - 0 + 3 = 3 - 3 - 7 = -4 Note that the last operation sets x out of the range 0 <= x <= 1000, which is valid.

Example 3:

Input:

nums = [2,8,16], start = 0, goal = 1

Output:

-1

Explanation:

There is no way to convert 0 into 1.

Constraints:

1 <= nums.length <= 1000

-10

9

<= nums[i], goal <= 10

9

0 <= start <= 1000

start != goal

All the integers in

nums

are distinct.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int minimumOperations(int[] nums, int start, int goal) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minimumOperations(nums: number[], start: number, goal: number):
number {

};
```

C#:

```
public class Solution {
public int MinimumOperations(int[] nums, int start, int goal) {

}
}
```

C:

```
int minimumOperations(int* nums, int numsSize, int start, int goal) {

}
```

Go:

```
func minimumOperations(nums []int, start int, goal int) int {

}
```

Kotlin:

```
class Solution {
fun minimumOperations(nums: IntArray, start: Int, goal: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int minimumOperations(List<int> nums, int start, int goal) {

}
}
```

Scala:

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

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_operations(nums :: [integer], start :: integer, goal ::
```

```
integer) :: integer
def minimum_operations(nums, start, goal) do

end
end
```

Erlang:

```
-spec minimum_operations(Nums :: [integer()], Start :: integer(), Goal ::
integer()) -> integer().
minimum_operations(Nums, Start, Goal) ->
.
```

Racket:

```
(define/contract (minimum-operations nums start goal)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Convert Number
* 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 minimumOperations(vector<int>& nums, int start, int goal) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Convert Number
* 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 minimumOperations(int[] nums, int start, int goal) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Convert Number
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 minimumOperations(self, nums: List[int], start: int, goal: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Convert Number
* 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} start
* @param {number} goal
* @return {number}
*/
var minimumOperations = function(nums, start, goal) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Convert Number
* 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 minimumOperations(nums: number[], start: number, goal: number):
number {

};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Convert Number
* 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 MinimumOperations(int[] nums, int start, int goal) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Convert Number
* 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 minimumOperations(int* nums, int numsSize, int start, int goal) {

}
```

Go Solution:

```
// Problem: Minimum Operations to Convert Number
// 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 minimumOperations(nums []int, start int, goal int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumOperations(nums: IntArray, start: Int, goal: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Convert Number
// 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 minimum_operations(nums: Vec<i32>, start: i32, goal: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} start
```

```
# @param {Integer} goal
# @return {Integer}
def minimum_operations(nums, start, goal)

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int minimumOperations(List<int> nums, int start, int goal) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_operations(nums :: [integer], start :: integer, goal ::
```

```
integer) :: integer
def minimum_operations(nums, start, goal) do

end
end
```

Erlang Solution:

```
-spec minimum_operations(Nums :: [integer()], Start :: integer(), Goal ::
integer()) -> integer().
minimum_operations(Nums, Start, Goal) ->
.
```

Racket Solution:

```
(define/contract (minimum-operations nums start goal)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```