Problem 2442: Count Number of Distinct Integers After Reverse Operations

Problem Information

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

Problem Description

You are given an array

nums

consisting of

positive

integers.

You have to take each integer in the array,

reverse its digits

, and add it to the end of the array. You should apply this operation to the original integers in

nums

.

Return

the number of

distinct

integers in the final array

.

Example 1:

Input:

nums = [1,13,10,12,31]

Output:

6

Explanation:

After including the reverse of each number, the resulting array is [1,13,10,12,31,

1,31,1,21,13

]. The reversed integers that were added to the end of the array are underlined. Note that for the integer 10, after reversing it, it becomes 01 which is just 1. The number of distinct integers in this array is 6 (The numbers 1, 10, 12, 13, 21, and 31).

Example 2:

Input:

nums = [2,2,2]

Output:

1

Explanation:

After including the reverse of each number, the resulting array is [2,2,2,

2,2,2

]. The number of distinct integers in this array is 1 (The number 2).

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

6

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def countDistinctIntegers(self, nums):
```

```
"""
:type nums: List[int]
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (count-distinct-integers nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Number of Distinct Integers After Reverse Operations
* Difficulty: Medium
* Tags: array, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Count Number of Distinct Integers After Reverse Operations
* Difficulty: Medium
* Tags: array, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int countDistinctIntegers(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Number of Distinct Integers After Reverse Operations
Difficulty: Medium
Tags: array, math, hash
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def countDistinctIntegers(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Number of Distinct Integers After Reverse Operations
* Difficulty: Medium
* Tags: array, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Count Number of Distinct Integers After Reverse Operations
* Difficulty: Medium
* Tags: array, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function countDistinctIntegers(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Number of Distinct Integers After Reverse Operations
* Difficulty: Medium
* Tags: array, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int CountDistinctIntegers(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Count Number of Distinct Integers After Reverse Operations
* Difficulty: Medium
* Tags: array, math, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

int countDistinctIntegers(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Count Number of Distinct Integers After Reverse Operations
// Difficulty: Medium
// Tags: array, math, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func countDistinctIntegers(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Number of Distinct Integers After Reverse Operations
// Difficulty: Medium
// Tags: array, math, hash
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (count-distinct-integers nums)
(-> (listof exact-integer?) exact-integer?)
)
```