Problem 3397: Maximum Number of Distinct Elements After Operations

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

k

.

You are allowed to perform the following

operation

on each element of the array

at most

once

:

Add an integer in the range

[-k, k]

to the element.

Return the

maximum

possible number of

distinct

elements in

nums

after performing the

operations

.

Example 1:

Input:

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

Output:

6

Explanation:

nums

changes to

[-1, 0, 1, 2, 3, 4]

after performing operations on the first four elements.

Example 2:

Input:

nums = [4,4,4,4], k = 1

Output:

3

Explanation:

By adding -1 to

nums[0]

and 1 to

nums[1]

,

nums

changes to

[3, 5, 4, 4]

.

Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

0 <= k <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int maxDistinctElements(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public int maxDistinctElements(int[] nums, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxDistinctElements(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int MaxDistinctElements(int[] nums, int k) {

}
}
```

C:

```
int maxDistinctElements(int* nums, int numsSize, int k) {

}
```

Go:

```
func maxDistinctElements(nums []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun maxDistinctElements(nums: IntArray, k: Int): Int {

}
```

```
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def max_distinct_elements(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function maxDistinctElements($nums, $k) {

}
}
```

Dart:

```
class Solution {
int maxDistinctElements(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def maxDistinctElements(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_distinct_elements(nums :: [integer], k :: integer) :: integer
def max_distinct_elements(nums, k) do

end
end
```

Erlang:

```
-spec max_distinct_elements(Nums :: [integer()], K :: integer()) ->
integer().
max_distinct_elements(Nums, K) ->
.
```

Racket:

```
(define/contract (max-distinct-elements nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Distinct Elements After Operations
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 maxDistinctElements(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Distinct Elements After Operations
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 maxDistinctElements(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Distinct Elements After Operations
Difficulty: Medium
Tags: array, greedy, sort
```

```
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 maxDistinctElements(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Number of Distinct Elements After Operations
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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} k
* @return {number}
*/
var maxDistinctElements = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Distinct Elements After Operations
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 maxDistinctElements(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Distinct Elements After Operations
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 MaxDistinctElements(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Distinct Elements After Operations
* Difficulty: Medium
* Tags: array, greedy, sort
*
* 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 maxDistinctElements(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum Number of Distinct Elements After Operations
// Difficulty: Medium
// Tags: array, greedy, sort
//
// 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 maxDistinctElements(nums []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxDistinctElements(nums: IntArray, k: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Distinct Elements After Operations
// Difficulty: Medium
```

```
// Tags: array, greedy, sort
//
// 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 max_distinct_elements(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def max_distinct_elements(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function maxDistinctElements($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
int maxDistinctElements(List<int> nums, int k) {

}
```

```
}
```

Scala Solution:

```
object Solution {
def maxDistinctElements(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_distinct_elements(nums :: [integer], k :: integer) :: integer
def max_distinct_elements(nums, k) do

end
end
```

Erlang Solution:

```
-spec max_distinct_elements(Nums :: [integer()], K :: integer()) ->
integer().
max_distinct_elements(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (max-distinct-elements nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```