

Problem 3346: Maximum Frequency of an Element After Performing Operations I

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given an integer array

`nums`

and two integers

`k`

and

`numOperations`

.

You must perform an

operation

`numOperations`

times on

`nums`

, where in each operation you:

Select an index

i

that was

not

selected in any previous operations.

Add an integer in the range

$[-k, k]$

to

$\text{nums}[i]$

.

Return the

maximum

possible

frequency

of any element in

nums

after performing the

operations

.

Example 1:

Input:

nums = [1,4,5], k = 1, numOperations = 2

Output:

2

Explanation:

We can achieve a maximum frequency of two by:

Adding 0 to

nums[1]

.

nums

becomes

[1, 4, 5]

.

Adding -1 to

nums[2]

.

nums

becomes

[1, 4, 4]

.

Example 2:

Input:

nums = [5,11,20,20], k = 5, numOperations = 1

Output:

2

Explanation:

We can achieve a maximum frequency of two by:

Adding 0 to

nums[1]

.

Constraints:

$1 \leq \text{nums.length} \leq 10$

5

$1 \leq \text{nums}[i] \leq 10$

5

$0 \leq k \leq 10$

5

$0 \leq \text{numOperations} \leq \text{nums.length}$

Code Snippets

C++:

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

    }
};
```

Java:

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

    }
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
 * @param {number[]} nums
 * @param {number} k
 * @param {number} numOperations
 * @return {number}
```

```
*/  
var maxFrequency = function(nums, k, numOperations) {  
  
};
```

TypeScript:

```
function maxFrequency(nums: number[], k: number, numOperations: number):  
number {  
  
};
```

C#:

```
public class Solution {  
    public int MaxFrequency(int[] nums, int k, int numOperations) {  
  
    }  
}
```

C:

```
int maxFrequency(int* nums, int numsSize, int k, int numOperations) {  
  
}
```

Go:

```
func maxFrequency(nums []int, k int, numOperations int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun maxFrequency(nums: IntArray, k: Int, numOperations: Int): Int {  
  
    }  
}
```

Swift:

```

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

    }
}

```

Rust:

```

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

    }
}

```

Ruby:

```

# @param {Integer[]} nums
# @param {Integer} k
# @param {Integer} num_operations
# @return {Integer}
def max_frequency(nums, k, num_operations)

end

```

PHP:

```

class Solution {

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

    }

}

```

Dart:

```

class Solution {
    int maxFrequency(List<int> nums, int k, int numOperations) {

```

```
}  
}
```

Scala:

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

Elixir:

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

Erlang:

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

Racket:

```
(define/contract (max-frequency nums k numOperations)  
  (-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)  
  )
```

Solutions

C++ Solution:


```

/*
 * Problem: Maximum Frequency of an Element After Performing Operations I
 * Difficulty: Medium
 * Tags: array, sort, 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 maxFrequency(vector<int>& nums, int k, int numOperations) {

    }
};

```

Java Solution:

```

/**
 * Problem: Maximum Frequency of an Element After Performing Operations I
 * Difficulty: Medium
 * Tags: array, sort, 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 maxFrequency(int[] nums, int k, int numOperations) {

    }
}

```

Python3 Solution:

```

"""
Problem: Maximum Frequency of an Element After Performing Operations I
Difficulty: Medium
Tags: array, sort, 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 maxFrequency(self, nums: List[int], k: int, numOperations: int) -> int:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

```

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

```

JavaScript Solution:

```

/**
 * Problem: Maximum Frequency of an Element After Performing Operations I
 * Difficulty: Medium
 * Tags: array, sort, 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} k
 * @param {number} numOperations
 * @return {number}
 */
var maxFrequency = function(nums, k, numOperations) {

```

```
};
```

TypeScript Solution:

```
/**
 * Problem: Maximum Frequency of an Element After Performing Operations I
 * Difficulty: Medium
 * Tags: array, sort, 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 maxFrequency(nums: number[], k: number, numOperations: number):
number {

};
```

C# Solution:

```
/*
 * Problem: Maximum Frequency of an Element After Performing Operations I
 * Difficulty: Medium
 * Tags: array, sort, 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 MaxFrequency(int[] nums, int k, int numOperations) {

    }
}
```

C Solution:

```
/*
 * Problem: Maximum Frequency of an Element After Performing Operations I
```

```

* Difficulty: Medium
* Tags: array, sort, 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 maxFrequency(int* nums, int numsSize, int k, int numOperations) {

}

```

Go Solution:

```

// Problem: Maximum Frequency of an Element After Performing Operations I
// Difficulty: Medium
// Tags: array, sort, 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 maxFrequency(nums []int, k int, numOperations int) int {

}

```

Kotlin Solution:

```

class Solution {
    fun maxFrequency(nums: IntArray, k: Int, numOperations: Int): Int {

    }
}

```

Swift Solution:

```

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

    }
}

```

Rust Solution:

```
// Problem: Maximum Frequency of an Element After Performing Operations I
// Difficulty: Medium
// Tags: array, sort, 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 max_frequency(nums: Vec<i32>, k: i32, num_operations: i32) -> i32 {

    }
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

    }

}
```

Dart Solution:

```
class Solution {  
  int maxFrequency(List<int> nums, int k, int numOperations) {  
  
  }  
}
```

Scala Solution:

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

Elixir Solution:

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

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-frequency nums k numOperations)  
  (-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)  
  )
```