

Problem 2439: Minimize Maximum of Array

Problem Information

Difficulty: **Medium**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a

0-indexed

array

nums

comprising of

n

non-negative integers.

In one operation, you must:

Choose an integer

i

such that

$1 \leq i < n$

and

nums[i] > 0

.

Decrease

nums[i]

by 1.

Increase

nums[i - 1]

by 1.

Return

the

minimum

possible value of the

maximum

integer of

nums

after performing

any

number of operations

.

Example 1:

Input:

nums = [3,7,1,6]

Output:

5

Explanation:

One set of optimal operations is as follows: 1. Choose $i = 1$, and nums becomes [4,6,1,6]. 2. Choose $i = 3$, and nums becomes [4,6,2,5]. 3. Choose $i = 1$, and nums becomes [5,5,2,5]. The maximum integer of nums is 5. It can be shown that the maximum number cannot be less than 5. Therefore, we return 5.

Example 2:

Input:

nums = [10,1]

Output:

10

Explanation:

It is optimal to leave nums as is, and since 10 is the maximum value, we return 10.

Constraints:

$n == \text{nums.length}$

$2 \leq n \leq 10$

5

$0 \leq \text{nums}[i] \leq 10$

Code Snippets

C++:

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

    }
};
```

Java:

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

    }
}
```

Python3:

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

Python:

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

JavaScript:

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

```
var minimizeArrayValue = function(nums) {  
  
};
```

TypeScript:

```
function minimizeArrayValue(nums: number[]): number {  
  
};
```

C#:

```
public class Solution {  
    public int MinimizeArrayValue(int[] nums) {  
  
    }  
}
```

C:

```
int minimizeArrayValue(int* nums, int numsSize) {  
  
}
```

Go:

```
func minimizeArrayValue(nums []int) int {  
  
}
```

Kotlin:

```
class Solution {  
    fun minimizeArrayValue(nums: IntArray): Int {  
  
    }  
}
```

Swift:

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

```
}  
}
```

Rust:

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

Ruby:

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

PHP:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums  
     * @return Integer  
     */  
    function minimizeArrayValue($nums) {  
  
    }  
}
```

Dart:

```
class Solution {  
    int minimizeArrayValue(List<int> nums) {  
  
    }  
}
```

Scala:

```

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

  }
}

```

Elixir:

```

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

  end
end

```

Erlang:

```

-spec minimize_array_value(Nums :: [integer()]) -> integer().
minimize_array_value(Nums) ->
.

```

Racket:

```

(define/contract (minimize-array-value nums)
  (-> (listof exact-integer?) exact-integer?)
  )

```

Solutions

C++ Solution:

```

/*
 * Problem: Minimize Maximum of Array
 * Difficulty: Medium
 * Tags: array, dp, greedy, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

```

```

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

    }
};

```

Java Solution:

```

/**
 * Problem: Minimize Maximum of Array
 * Difficulty: Medium
 * Tags: array, dp, greedy, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

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

}
}

```

Python3 Solution:

```

"""
Problem: Minimize Maximum of Array
Difficulty: Medium
Tags: array, dp, greedy, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

```

Python Solution:

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

JavaScript Solution:

```
/**
 * Problem: Minimize Maximum of Array
 * Difficulty: Medium
 * Tags: array, dp, greedy, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

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

};
```

TypeScript Solution:

```
/**
 * Problem: Minimize Maximum of Array
 * Difficulty: Medium
 * Tags: array, dp, greedy, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

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

```
};
```

C# Solution:

```
/*
 * Problem: Minimize Maximum of Array
 * Difficulty: Medium
 * Tags: array, dp, greedy, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

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

    }
}
```

C Solution:

```
/*
 * Problem: Minimize Maximum of Array
 * Difficulty: Medium
 * Tags: array, dp, greedy, search
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

int minimizeArrayValue(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimize Maximum of Array
// Difficulty: Medium
```

```
// Tags: array, dp, greedy, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func minimizeArrayValue(nums []int) int {

}
```

Kotlin Solution:

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

    }
}
```

Swift Solution:

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

    }
}
```

Rust Solution:

```
// Problem: Minimize Maximum of Array
// Difficulty: Medium
// Tags: array, dp, greedy, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

    }
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

    }

}
```

Dart Solution:

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

  }
}
```

Scala Solution:

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

  }
}
```

Elixir Solution:

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

```
end  
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (minimize-array-value nums)  
  (-> (listof exact-integer?) exact-integer?)  
  )
```