Problem 1760: Minimum Limit of Balls in a Bag

Problem Information

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

Problem Description

You are given an integer array

nums

where the

i

th

bag contains

nums[i]

balls. You are also given an integer

maxOperations

.

You can perform the following operation at most

maxOperations

times:

Take any bag of balls and divide it into two new bags with a

positive

number of balls.

For example, a bag of

5

balls can become two new bags of

1

and

4

balls, or two new bags of

2

and

3

balls.

Your penalty is the

maximum

number of balls in a bag. You want to

minimize

your penalty after the operations.

Return

the minimum possible penalty after performing the operations

.

Example 1:

Input:

nums = [9], maxOperations = 2

Output:

3

Explanation:

- Divide the bag with 9 balls into two bags of sizes 6 and 3. [

9

] -> [6,3]. - Divide the bag with 6 balls into two bags of sizes 3 and 3. [

6

,3] -> [3,3,3]. The bag with the most number of balls has 3 balls, so your penalty is 3 and you should return 3.

Example 2:

Input:

nums = [2,4,8,2], maxOperations = 4

Output:

2

Explanation:

- Divide the bag with 8 balls into two bags of sizes 4 and 4. [2,4,

8

,2] -> [2,4,4,4,2]. - Divide the bag with 4 balls into two bags of sizes 2 and 2. [2,

4

,4,4,2] -> [2,2,2,4,4,2]. - Divide the bag with 4 balls into two bags of sizes 2 and 2. [2,2,2,

4

,4,2] -> [2,2,2,2,2,4,2]. - Divide the bag with 4 balls into two bags of sizes 2 and 2. [2,2,2,2,2,

4

,2] -> [2,2,2,2,2,2,2,2]. The bag with the most number of balls has 2 balls, so your penalty is 2, and you should return 2.

Constraints:

1 <= nums.length <= 10

5

1 <= maxOperations, nums[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int minimumSize(vector<int>& nums, int maxOperations) {

}
};
```

Java:

```
class Solution {
public int minimumSize(int[] nums, int maxOperations) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minimumSize(nums: number[], maxOperations: number): number {

};
```

C#:

```
public class Solution {
public int MinimumSize(int[] nums, int maxOperations) {

}
}
```

C:

```
int minimumSize(int* nums, int numsSize, int maxOperations) {

}
```

Go:

```
func minimumSize(nums []int, maxOperations int) int {

}
```

Kotlin:

```
class Solution {
fun minimumSize(nums: IntArray, maxOperations: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} max_operations
# @return {Integer}
def minimum_size(nums, max_operations)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $maxOperations
* @return Integer
*/
function minimumSize($nums, $maxOperations) {

}
}
```

Dart:

```
class Solution {
int minimumSize(List<int> nums, int maxOperations) {

}
}
```

Scala:

```
object Solution {
def minimumSize(nums: Array[Int], maxOperations: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_size(nums :: [integer], max_operations :: integer) :: integer
```

```
def minimum_size(nums, max_operations) do

end
end
```

Erlang:

```
-spec minimum_size(Nums :: [integer()], MaxOperations :: integer()) ->
integer().
minimum_size(Nums, MaxOperations) ->
.
```

Racket:

```
(define/contract (minimum-size nums maxOperations)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Limit of Balls in a Bag
* 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 minimumSize(vector<int>& nums, int maxOperations) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Limit of Balls in a Bag
* 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 minimumSize(int[] nums, int maxOperations) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Limit of Balls in a Bag
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 minimumSize(self, nums: List[int], maxOperations: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Limit of Balls in a Bag
* 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} maxOperations
* @return {number}
*/
var minimumSize = function(nums, maxOperations) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Limit of Balls in a Bag
* 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 minimumSize(nums: number[], maxOperations: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Limit of Balls in a Bag
* 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 MinimumSize(int[] nums, int maxOperations) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Limit of Balls in a Bag
* 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 minimumSize(int* nums, int numsSize, int maxOperations) {

}
```

Go Solution:

```
// Problem: Minimum Limit of Balls in a Bag
// 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 minimumSize(nums []int, maxOperations int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumSize(nums: IntArray, maxOperations: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Limit of Balls in a Bag
// 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_size(nums: Vec<i32>, max_operations: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} max_operations
# @return {Integer}
def minimum_size(nums, max_operations)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $maxOperations
* @return Integer
*/
function minimumSize($nums, $maxOperations) {

}
}
```

Dart Solution:

```
class Solution {
int minimumSize(List<int> nums, int maxOperations) {

}
}
```

Scala Solution:

```
object Solution {
def minimumSize(nums: Array[Int], maxOperations: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_size(nums :: [integer], max_operations :: integer) :: integer
def minimum_size(nums, max_operations) do

end
end
```

Erlang Solution:

```
-spec minimum_size(Nums :: [integer()], MaxOperations :: integer()) ->
integer().
minimum_size(Nums, MaxOperations) ->
.
```

Racket Solution:

```
(define/contract (minimum-size nums maxOperations)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```