

# Problem 2790: Maximum Number of Groups With Increasing Length

## Problem Information

Difficulty: **Hard**

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

You are given a

0-indexed

array

usageLimits

of length

n

.

Your task is to create

groups

using numbers from

0

to

n - 1

, ensuring that each number,

i

, is used no more than

usageLimits[i]

times in total

across all groups

. You must also satisfy the following conditions:

Each group must consist of

distinct

numbers, meaning that no duplicate numbers are allowed within a single group.

Each group (except the first one) must have a length

strictly greater

than the previous group.

Return

an integer denoting the

maximum

number of groups you can create while satisfying these conditions.

Example 1:

Input:

usageLimits

= [1,2,5]

Output:

3

Explanation:

In this example, we can use 0 at most once, 1 at most twice, and 2 at most five times. One way of creating the maximum number of groups while satisfying the conditions is: Group 1 contains the number [2]. Group 2 contains the numbers [1,2]. Group 3 contains the numbers [0,1,2]. It can be shown that the maximum number of groups is 3. So, the output is 3.

Example 2:

Input:

usageLimits

= [2,1,2]

Output:

2

Explanation:

In this example, we can use 0 at most twice, 1 at most once, and 2 at most twice. One way of creating the maximum number of groups while satisfying the conditions is: Group 1 contains the number [0]. Group 2 contains the numbers [1,2]. It can be shown that the maximum number of groups is 2. So, the output is 2.

Example 3:

Input:

usageLimits

= [1,1]

Output:

1

Explanation:

In this example, we can use both 0 and 1 at most once. One way of creating the maximum number of groups while satisfying the conditions is: Group 1 contains the number [0]. It can be shown that the maximum number of groups is 1. So, the output is 1.

Constraints:

1 <= usageLimits.length <= 10

5

1 <= usageLimits[i] <= 10

9

## Code Snippets

**C++:**

```
class Solution {
public:
    int maxIncreasingGroups(vector<int>& usageLimits) {

    }
};
```

**Java:**

```
class Solution {
    public int maxIncreasingGroups(List<Integer> usageLimits) {

    }
}
```

```
}
```

### Python3:

```
class Solution:
    def maxIncreasingGroups(self, usageLimits: List[int]) -> int:
```

### Python:

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

### JavaScript:

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

};
```

### TypeScript:

```
function maxIncreasingGroups(usageLimits: number[]): number {

};
```

### C#:

```
public class Solution {
    public int MaxIncreasingGroups(IList<int> usageLimits) {

    }
}
```

### C:

```
int maxIncreasingGroups(int* usageLimits, int usageLimitsSize) {  
  
}
```

### Go:

```
func maxIncreasingGroups(usageLimits []int) int {  
  
}
```

### Kotlin:

```
class Solution {  
    fun maxIncreasingGroups(usageLimits: List<Int>): Int {  
  
    }  
}
```

### Swift:

```
class Solution {  
    func maxIncreasingGroups(_ usageLimits: [Int]) -> Int {  
  
    }  
}
```

### Rust:

```
impl Solution {  
    pub fn max_increasing_groups(usage_limits: Vec<i32>) -> i32 {  
  
    }  
}
```

### Ruby:

```
# @param {Integer[]} usage_limits  
# @return {Integer}  
def max_increasing_groups(usage_limits)  
  
end
```

### PHP:

```

class Solution {

  /**
   * @param Integer[] $usageLimits
   * @return Integer
   */
  function maxIncreasingGroups($usageLimits) {

  }

}

```

### Dart:

```

class Solution {
  int maxIncreasingGroups(List<int> usageLimits) {

  }

}

```

### Scala:

```

object Solution {
  def maxIncreasingGroups(usageLimits: List[Int]): Int = {

  }

}

```

### Elixir:

```

defmodule Solution do
  @spec max_increasing_groups(usage_limits :: [integer]) :: integer
  def max_increasing_groups(usage_limits) do

  end

end

```

### Erlang:

```

-spec max_increasing_groups(UsageLimits :: [integer()]) -> integer().
max_increasing_groups(UsageLimits) ->
.

```

### Racket:

```
(define/contract (max-increasing-groups usageLimits)
  (-> (listof exact-integer?) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
 * Problem: Maximum Number of Groups With Increasing Length
 * Difficulty: Hard
 * Tags: array, greedy, math, 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 maxIncreasingGroups(vector<int>& usageLimits) {

    }
};
```

### Java Solution:

```
/**
 * Problem: Maximum Number of Groups With Increasing Length
 * Difficulty: Hard
 * Tags: array, greedy, math, 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 maxIncreasingGroups(List<Integer> usageLimits) {

    }
}
```



```
}
```

### Python3 Solution:

```
"""
Problem: Maximum Number of Groups With Increasing Length
Difficulty: Hard
Tags: array, greedy, math, 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 maxIncreasingGroups(self, usageLimits: List[int]) -> int:
        # TODO: Implement optimized solution
        pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
 * Problem: Maximum Number of Groups With Increasing Length
 * Difficulty: Hard
 * Tags: array, greedy, math, 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[]} usageLimits
* @return {number}
*/
var maxIncreasingGroups = function(usageLimits) {

};

```

### TypeScript Solution:

```

/**
 * Problem: Maximum Number of Groups With Increasing Length
 * Difficulty: Hard
 * Tags: array, greedy, math, 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 maxIncreasingGroups(usageLimits: number[]): number {

};

```

### C# Solution:

```

/*
 * Problem: Maximum Number of Groups With Increasing Length
 * Difficulty: Hard
 * Tags: array, greedy, math, 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 MaxIncreasingGroups(IList<int> usageLimits) {

    }
}

```

### C Solution:

```
/*
 * Problem: Maximum Number of Groups With Increasing Length
 * Difficulty: Hard
 * Tags: array, greedy, math, 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 maxIncreasingGroups(int* usageLimits, int usageLimitsSize) {

}
```

### Go Solution:

```
// Problem: Maximum Number of Groups With Increasing Length
// Difficulty: Hard
// Tags: array, greedy, math, 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 maxIncreasingGroups(usageLimits []int) int {

}
```

### Kotlin Solution:

```
class Solution {
    fun maxIncreasingGroups(usageLimits: List<Int>): Int {

    }
}
```

### Swift Solution:

```
class Solution {
    func maxIncreasingGroups(_ usageLimits: [Int]) -> Int {
```

```
}  
}
```

### Rust Solution:

```
// Problem: Maximum Number of Groups With Increasing Length  
// Difficulty: Hard  
// Tags: array, greedy, math, 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_increasing_groups(usage_limits: Vec<i32>) -> i32 {  
  
    }  
}
```

### Ruby Solution:

```
# @param {Integer[]} usage_limits  
# @return {Integer}  
def max_increasing_groups(usage_limits)  
  
end
```

### PHP Solution:

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

### Dart Solution:

```
class Solution {  
  int maxIncreasingGroups(List<int> usageLimits) {  
  
  }  
}
```

### Scala Solution:

```
object Solution {  
  def maxIncreasingGroups(usageLimits: List[Int]): Int = {  
  
  }  
}
```

### Elixir Solution:

```
defmodule Solution do  
  @spec max_increasing_groups(usage_limits :: [integer]) :: integer  
  def max_increasing_groups(usage_limits) do  
  
  end  
end
```

### Erlang Solution:

```
-spec max_increasing_groups(UsageLimits :: [integer()]) -> integer().  
max_increasing_groups(UsageLimits) ->  
.
```

### Racket Solution:

```
(define/contract (max-increasing-groups usageLimits)  
  (-> (listof exact-integer?) exact-integer?)  
  )
```