Problem 1846: Maximum Element After Decreasing and Rearranging

Problem Information

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

Problem Description

You are given an array of positive integers

arr

. Perform some operations (possibly none) on

arr

so that it satisfies these conditions:

The value of the

first

element in

arr

must be

1

.

The absolute difference between any 2 adjacent elements must be less than or equal to

1

. In other words,

abs(arr[i] - arr[i - 1]) <= 1

for each

i

where

1 <= i < arr.length

(

0-indexed

).

abs(x)

is the absolute value of

x

.

There are 2 types of operations that you can perform any number of times:

Decrease

the value of any element of

arr

to a

smaller positive integer

.

Rearrange

the elements of

arr

to be in any order.

Return

the

maximum

possible value of an element in

arr

after performing the operations to satisfy the conditions

.

Example 1:

Input:

arr = [2,2,1,2,1]

Output:

2

Explanation:

We can satisfy the conditions by rearranging

arr

so it becomes

[1,2,2,2,1]

. The largest element in

arr

is 2.

Example 2:

Input:

arr = [100,1,1000]

Output:

3

Explanation:

One possible way to satisfy the conditions is by doing the following: 1. Rearrange

arr

so it becomes

[1,100,1000]

. 2. Decrease the value of the second element to 2. 3. Decrease the value of the third element to 3. Now

arr = [1,2,3]

, which

satisfies the conditions. The largest element in

arr is 3.

Example 3:

Input:

arr = [1,2,3,4,5]

Output:

5

Explanation:

The array already satisfies the conditions, and the largest element is 5.

Constraints:

1 <= arr.length <= 10

5

1 <= arr[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int maximumElementAfterDecrementingAndRearranging(vector<int>& arr) {

}
};
```

Java:

```
class Solution {
public int maximumElementAfterDecrementingAndRearranging(int[] arr) {

}
}
```

Python3:

```
class Solution:
def maximumElementAfterDecrementingAndRearranging(self, arr: List[int]) ->
int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximumElementAfterDecrementingAndRearranging(arr: number[]): number
{

};
```

C#:

```
public class Solution {
public int MaximumElementAfterDecrementingAndRearranging(int[] arr) {

}
}
```

C:

```
int maximumElementAfterDecrementingAndRearranging(int* arr, int arrSize) {

}
```

Go:

```
func maximumElementAfterDecrementingAndRearranging(arr []int) int {

}
```

Kotlin:

```
class Solution {
fun maximumElementAfterDecrementingAndRearranging(arr: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maximumElementAfterDecrementingAndRearranging(_ arr: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_element_after_decrementing_and_rearranging(arr: Vec<i32>) ->
i32 {

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @return {Integer}
def maximum_element_after_decrementing_and_rearranging(arr)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer
*/
function maximumElementAfterDecrementingAndRearranging($arr) {

}
}
```

Dart:

```
class Solution {
int maximumElementAfterDecrementingAndRearranging(List<int> arr) {

}
}
```

Scala:

```
object Solution {
def maximumElementAfterDecrementingAndRearranging(arr: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_element_after_decrementing_and_rearranging(arr :: [integer]) ::
integer
def maximum_element_after_decrementing_and_rearranging(arr) do

end
end
```

Erlang:

```
-spec maximum_element_after_decrementing_and_rearranging(Arr :: [integer()])
-> integer().
maximum_element_after_decrementing_and_rearranging(Arr) ->
.
```

Racket:

```
(define/contract (maximum-element-after-decrementing-and-rearranging arr)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Element After Decreasing and Rearranging
* 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 maximumElementAfterDecrementingAndRearranging(vector<int>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Element After Decreasing and Rearranging
* 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 maximumElementAfterDecrementingAndRearranging(int[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Element After Decreasing and Rearranging
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 maximumElementAfterDecrementingAndRearranging(self, arr: List[int]) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maximumElementAfterDecrementingAndRearranging(self, arr):
"""
:type arr: List[int]
```

```
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Element After Decreasing and Rearranging
* 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[]} arr
* @return {number}
*/
var maximumElementAfterDecrementingAndRearranging = function(arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Element After Decreasing and Rearranging
* 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 maximumElementAfterDecrementingAndRearranging(arr: number[]): number
{

};
```

C# Solution:

```
/*
* Problem: Maximum Element After Decreasing and Rearranging
* 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 MaximumElementAfterDecrementingAndRearranging(int[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Element After Decreasing and Rearranging
* 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 maximumElementAfterDecrementingAndRearranging(int* arr, int arrSize) {

}
```

Go Solution:

```
// Problem: Maximum Element After Decreasing and Rearranging
// 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 maximumElementAfterDecrementingAndRearranging(arr []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumElementAfterDecrementingAndRearranging(arr: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maximumElementAfterDecrementingAndRearranging(_ arr: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Element After Decreasing and Rearranging
// 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 maximum_element_after_decrementing_and_rearranging(arr: Vec<i32>) ->
i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} arr
# @return {Integer}
```

```
def maximum_element_after_decrementing_and_rearranging(arr)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer
*/
function maximumElementAfterDecrementingAndRearranging($arr) {

}
}
```

Dart Solution:

```
class Solution {
int maximumElementAfterDecrementingAndRearranging(List<int> arr) {

}
}
```

Scala Solution:

```
object Solution {
def maximumElementAfterDecrementingAndRearranging(arr: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_element_after_decrementing_and_rearranging(arr :: [integer]) ::
integer
def maximum_element_after_decrementing_and_rearranging(arr) do

end
```

```

end
```

Erlang Solution:

```
-spec maximum_element_after_decrementing_and_rearranging(Arr :: [integer()])
-> integer().
maximum_element_after_decrementing_and_rearranging(Arr) ->
.
```

Racket Solution:

```
(define/contract (maximum-element-after-decrementing-and-rearranging arr)
(-> (listof exact-integer?) exact-integer?)
)
```