Problem 1788: Maximize the Beauty of the Garden

Problem Information

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

Problem Description

There is a garden of

n

flowers, and each flower has an integer beauty value. The flowers are arranged in a line. You are given an integer array

flowers

of size

n

and each

flowers[i]

represents the beauty of the

i

th

flower.

A garden is

valid

if it meets these conditions:

The garden has at least two flowers.

The first and the last flower of the garden have the same beauty value.

As the appointed gardener, you have the ability to

remove

any (possibly none) flowers from the garden. You want to remove flowers in a way that makes the remaining garden

valid

. The beauty of the garden is the sum of the beauty of all the remaining flowers.

Return the maximum possible beauty of some

valid

garden after you have removed any (possibly none) flowers.

Example 1:

Input:

flowers = [1,2,3,1,2]

Output:

8

Explanation:

You can produce the valid garden [2,3,1,2] to have a total beauty of 2 + 3 + 1 + 2 = 8.

Example 2:

Input:

flowers = [100,1,1,-3,1]

Output:

3

Explanation:

You can produce the valid garden [1,1,1] to have a total beauty of 1 + 1 + 1 = 3.

Example 3:

Input:

flowers = [-1,-2,0,-1]

Output:

-2

Explanation:

You can produce the valid garden [-1,-1] to have a total beauty of -1 + -1 = -2.

Constraints:

2 <= flowers.length <= 10

5

-10

4

<= flowers[i] <= 10

4

It is possible to create a valid garden by removing some (possibly none) flowers.

Code Snippets

C++:

```
class Solution {
public:
int maximumBeauty(vector<int>& flowers) {

}
};
```

Java:

```
class Solution {
public int maximumBeauty(int[] flowers) {

}
}
```

Python3:

```
class Solution:
def maximumBeauty(self, flowers: List[int]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maximumBeauty(flowers: number[]): number {

};
```

C#:

```
public class Solution {
public int MaximumBeauty(int[] flowers) {

}
}
```

C:

```


int maximumBeauty(int* flowers, int flowersSize){

}
```

Go:

```
func maximumBeauty(flowers []int) int {

}
```

Kotlin:

```
class Solution {
fun maximumBeauty(flowers: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maximumBeauty(_ flowers: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_beauty(flowers: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} flowers
# @return {Integer}
def maximum_beauty(flowers)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $flowers
* @return Integer
*/
function maximumBeauty($flowers) {

}
}
```

Scala:

```
object Solution {
def maximumBeauty(flowers: Array[Int]): Int = {

}
```

```
}
```

Racket:

```
(define/contract (maximum-beauty flowers)
(-> (listof exact-integer?) exact-integer?)

)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize the Beauty of the Garden
* Difficulty: Hard
* Tags: array, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int maximumBeauty(vector<int>& flowers) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize the Beauty of the Garden
* Difficulty: Hard
* Tags: array, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

class Solution {
public int maximumBeauty(int[] flowers) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize the Beauty of the Garden
Difficulty: Hard
Tags: array, greedy, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximize the Beauty of the Garden
* Difficulty: Hard
* Tags: array, greedy, hash
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} flowers
* @return {number}
*/
var maximumBeauty = function(flowers) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize the Beauty of the Garden
* Difficulty: Hard
* Tags: array, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function maximumBeauty(flowers: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximize the Beauty of the Garden
* Difficulty: Hard
* Tags: array, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```
public class Solution {
public int MaximumBeauty(int[] flowers) {

}
}
```

C Solution:

```
/*
* Problem: Maximize the Beauty of the Garden
* Difficulty: Hard
* Tags: array, greedy, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/



int maximumBeauty(int* flowers, int flowersSize){

}
```

Go Solution:

```
// Problem: Maximize the Beauty of the Garden
// Difficulty: Hard
// Tags: array, greedy, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func maximumBeauty(flowers []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumBeauty(flowers: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maximumBeauty(_ flowers: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximize the Beauty of the Garden
// Difficulty: Hard
// Tags: array, greedy, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn maximum_beauty(flowers: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} flowers
# @return {Integer}
def maximum_beauty(flowers)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[] $flowers
* @return Integer
*/
function maximumBeauty($flowers) {

}
}
```

Scala Solution:

```
object Solution {
def maximumBeauty(flowers: Array[Int]): Int = {

}
}
```

Racket Solution:

```
(define/contract (maximum-beauty flowers)
(-> (listof exact-integer?) exact-integer?)

)
```