Problem 1482: Minimum Number of Days to Make m Bouquets

Problem Information

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

Problem Description

You are given an integer array

bloomDay

, an integer

m

and an integer

k

.

You want to make

m

bouquets. To make a bouquet, you need to use

k

adjacent flowers

from the garden.

The garden consists of

n

flowers, the

i

th

flower will bloom in the

bloomDay[i]

and then can be used in

exactly one

bouquet.

Return

the minimum number of days you need to wait to be able to make

m

bouquets from the garden

. If it is impossible to make m bouquets return

-1

.

Example 1:

Input:

bloomDay = [1,10,3,10,2], m = 3, k = 1

Output:

3

Explanation:

Let us see what happened in the first three days. x means flower bloomed and _ means flower did not bloom in the garden. We need 3 bouquets each should contain 1 flower. After day 1: [x, _, _, _, _] // we can only make one bouquet. After day 2: [x, _, _, _, x] // we can only make two bouquets. After day 3: [x, _, x, _, x] // we can make 3 bouquets. The answer is 3.

Example 2:

Input:

bloomDay = [1,10,3,10,2], m = 3, k = 2

Output:

-1

Explanation:

We need 3 bouquets each has 2 flowers, that means we need 6 flowers. We only have 5 flowers so it is impossible to get the needed bouquets and we return -1.

Example 3:

Input:

bloomDay = [7,7,7,7,12,7,7], m = 2, k = 3

Output:

12

Explanation:

We need 2 bouquets each should have 3 flowers. Here is the garden after the 7 and 12 days: After day 7: [x, x, x, x, _, x, x] We can make one bouquet of the first three flowers that bloomed. We cannot make another bouquet from the last three flowers that bloomed because they are not adjacent. After day 12: [x, x, x, x, x, x, x] It is obvious that we can make two bouquets in different ways.

Constraints:

bloomDay.length == n

1 <= n <= 10

5

1 <= bloomDay[i] <= 10

9

1 <= m <= 10

6

1 <= k <= n

Code Snippets

C++:

```
class Solution {
public:
int minDays(vector<int>& bloomDay, int m, int k) {

}
};
```

Java:

```
class Solution {
public int minDays(int[] bloomDay, int m, int k) {
}
}
```

Python3:

```
class Solution:
def minDays(self, bloomDay: List[int], m: int, k: int) -> int:
```

Python:

```
class Solution(object):
def minDays(self, bloomDay, m, k):
"""
:type bloomDay: List[int]
:type m: int
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} bloomDay
* @param {number} m
* @param {number} k
* @return {number}
*/
var minDays = function(bloomDay, m, k) {

};
```

TypeScript:

```
function minDays(bloomDay: number[], m: number, k: number): number {

};
```

C#:

```
public class Solution {
public int MinDays(int[] bloomDay, int m, int k) {
}
}
```

C:

```
int minDays(int* bloomDay, int bloomDaySize, int m, int k) {

}
```

Go:

```
func minDays(bloomDay []int, m int, k int) int {

}
```

Kotlin:

```
class Solution {
fun minDays(bloomDay: IntArray, m: Int, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func minDays(_ bloomDay: [Int], _ m: Int, _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_days(bloom_day: Vec<i32>, m: i32, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} bloom_day
# @param {Integer} m
# @param {Integer} k
# @return {Integer}
def min_days(bloom_day, m, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $bloomDay
* @param Integer $m
* @param Integer $k
* @return Integer
*/
function minDays($bloomDay, $m, $k) {

}
}
```

Dart:

```
class Solution {
int minDays(List<int> bloomDay, int m, int k) {

}
}
```

Scala:

```
object Solution {
def minDays(bloomDay: Array[Int], m: Int, k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_days(bloom_day :: [integer], m :: integer, k :: integer) :: integer
```

```
def min_days(bloom_day, m, k) do

end
end
```

Erlang:

```
-spec min_days(BloomDay :: [integer()], M :: integer(), K :: integer()) ->
integer().
min_days(BloomDay, M, K) ->
.
```

Racket:

```
(define/contract (min-days bloomDay m k)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Days to Make m Bouquets
* 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 minDays(vector<int>& bloomDay, int m, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Days to Make m Bouquets
* 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 minDays(int[] bloomDay, int m, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Days to Make m Bouquets
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 minDays(self, bloomDay: List[int], m: int, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minDays(self, bloomDay, m, k):
"""
:type bloomDay: List[int]
:type m: int
:type k: int
:rtype: int
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Number of Days to Make m Bouquets
* 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[]} bloomDay
* @param {number} m
* @param {number} k
* @return {number}
*/
var minDays = function(bloomDay, m, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Days to Make m Bouquets
* 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 minDays(bloomDay: number[], m: number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Days to Make m Bouquets
* 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 MinDays(int[] bloomDay, int m, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Days to Make m Bouquets
* 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 minDays(int* bloomDay, int bloomDaySize, int m, int k) {

}
```

Go Solution:

```
// Problem: Minimum Number of Days to Make m Bouquets
// 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 minDays(bloomDay []int, m int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minDays(bloomDay: IntArray, m: Int, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minDays(_ bloomDay: [Int], _ m: Int, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Days to Make m Bouquets
// 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 min_days(bloom_day: Vec<i32>, m: i32, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} bloom_day
# @param {Integer} m
# @param {Integer} k
# @return {Integer}
def min_days(bloom_day, m, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $bloomDay
* @param Integer $m
* @param Integer $k
* @return Integer
*/
function minDays($bloomDay, $m, $k) {

}
}
```

Dart Solution:

```
class Solution {
int minDays(List<int> bloomDay, int m, int k) {

}
}
```

Scala Solution:

```
object Solution {
def minDays(bloomDay: Array[Int], m: Int, k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_days(bloom_day :: [integer], m :: integer, k :: integer) :: integer
def min_days(bloom_day, m, k) do
```

```



end
end
```

Erlang Solution:

```
-spec min_days(BloomDay :: [integer()], M :: integer(), K :: integer()) ->
integer().
min_days(BloomDay, M, K) ->
.
```

Racket Solution:

```
(define/contract (min-days bloomDay m k)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```