Problem 2294: Partition Array Such That Maximum Difference Is K

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

k

. You may partition

nums

into one or more

subsequences

such that each element in

nums

appears in

exactly

one of the subsequences.

Return

the

minimum

number of subsequences needed such that the difference between the maximum and minimum values in each subsequence is

at most

k

.

A

subsequence

is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of the remaining elements.

Example 1:

Input:

nums = [3,6,1,2,5], k = 2

Output:

2

Explanation:

We can partition nums into the two subsequences [3,1,2] and [6,5]. The difference between the maximum and minimum value in the first subsequence is 3 - 1 = 2. The difference between the maximum and minimum value in the second subsequence is 6 - 5 = 1. Since two subsequences were created, we return 2. It can be shown that 2 is the minimum number of subsequences needed.

Example 2:

Input:

nums = [1,2,3], k = 1

Output:

2

Explanation:

We can partition nums into the two subsequences [1,2] and [3]. The difference between the maximum and minimum value in the first subsequence is 2 - 1 = 1. The difference between the maximum and minimum value in the second subsequence is 3 - 3 = 0. Since two subsequences were created, we return 2. Note that another optimal solution is to partition nums into the two subsequences [1] and [2,3].

Example 3:

Input:

nums = [2,2,4,5], k = 0

Output:

3

Explanation:

We can partition nums into the three subsequences [2,2], [4], and [5]. The difference between the maximum and minimum value in the first subsequences is 2 - 2 = 0. The difference between the maximum and minimum value in the second subsequences is 4 - 4 = 0. The difference between the maximum and minimum value in the third subsequences is 5 - 5 = 0. Since three subsequences were created, we return 3. It can be shown that 3 is the minimum number of subsequences needed.

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 10

5

0 <= k <= 10

5

Code Snippets

C++:

```
class Solution {
public:
int partitionArray(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public int partitionArray(int[] nums, int k) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def partitionArray(self, nums, k):
```

```
"""
:type nums: List[int]
:type k: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function partitionArray(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int PartitionArray(int[] nums, int k) {

}
}
```

C:

```
int partitionArray(int* nums, int numsSize, int k) {

}
```

Go:

```
func partitionArray(nums []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun partitionArray(nums: IntArray, k: Int): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def partition_array(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function partitionArray($nums, $k) {


}
}
```

Dart:

```
class Solution {
int partitionArray(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def partitionArray(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec partition_array(nums :: [integer], k :: integer) :: integer
def partition_array(nums, k) do

end
end
```

Erlang:

```
-spec partition_array(Nums :: [integer()], K :: integer()) -> integer().
partition_array(Nums, K) ->
.
```

Racket:

```
(define/contract (partition-array nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Partition Array Such That Maximum Difference Is K
* 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 partitionArray(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Partition Array Such That Maximum Difference Is K
* 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 partitionArray(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Partition Array Such That Maximum Difference Is K
```

```
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 partitionArray(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Partition Array Such That Maximum Difference Is K
* 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[]} nums
* @param {number} k
* @return {number}
*/
var partitionArray = function(nums, k) {
};
```

TypeScript Solution:

```
/**
* Problem: Partition Array Such That Maximum Difference Is K
* 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 partitionArray(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Partition Array Such That Maximum Difference Is K
* 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 PartitionArray(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Partition Array Such That Maximum Difference Is K
* 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 partitionArray(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Partition Array Such That Maximum Difference Is K
// 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 partitionArray(nums []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun partitionArray(nums: IntArray, k: Int): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Partition Array Such That Maximum Difference Is K
// 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 partition_array(nums: Vec<i32>, k: i32) -> i32 {


}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def partition_array(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function partitionArray($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
int partitionArray(List<int> nums, int k) {

}
}
```

Scala Solution:

```
object Solution {
def partitionArray(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec partition_array(nums :: [integer], k :: integer) :: integer
def partition_array(nums, k) do

end
end
```

Erlang Solution:

```
-spec partition_array(Nums :: [integer()], K :: integer()) -> integer().
partition_array(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (partition-array nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```