Problem 1681: Minimum Incompatibility

Problem Information

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

Problem Description

You are given an integer array

nums

and an integer

k

. You are asked to distribute this array into

k

subsets of

equal size

such that there are no two equal elements in the same subset.

A subset's

incompatibility

is the difference between the maximum and minimum elements in that array.

Return

the

minimum possible sum of incompatibilities

of the

k

subsets after distributing the array optimally, or return

-1

if it is not possible.

A subset is a group integers that appear in the array with no particular order.

Example 1:

Input:

nums = [1,2,1,4], k = 2

Output:

4

Explanation:

The optimal distribution of subsets is [1,2] and [1,4]. The incompatibility is (2-1) + (4-1) = 4. Note that [1,1] and [2,4] would result in a smaller sum, but the first subset contains 2 equal elements.

Example 2:

Input:

nums = [6,3,8,1,3,1,2,2], k = 4

Output:

6

Explanation:

The optimal distribution of subsets is [1,2], [2,3], [6,8], and [1,3]. The incompatibility is (2-1) + (3-2) + (8-6) + (3-1) = 6.

Example 3:

Input:

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

Output:

-1

Explanation:

It is impossible to distribute nums into 3 subsets where no two elements are equal in the same subset.

Constraints:

1 <= k <= nums.length <= 16

nums.length

is divisible by

k

1 <= nums[i] <= nums.length

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
```

```
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (minimum-incompatibility nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Incompatibility
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
```

```
int minimumIncompatibility(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Incompatibility
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Incompatibility
Difficulty: Hard
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Incompatibility
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Incompatibility
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

```
};
```

C# Solution:

```
/*
* Problem: Minimum Incompatibility
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```
/*
* Problem: Minimum Incompatibility
* Difficulty: Hard
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int minimumIncompatibility(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Minimum Incompatibility
// Difficulty: Hard
```

```
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func minimumIncompatibility(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Incompatibility
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (minimum-incompatibility nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```