Problem 3098: Find the Sum of Subsequence Powers

Problem Information

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

Problem Description

You are given an integer array

nums

of length

n

, and a

positive

integer

k

.

The

power

of a

subsequence

is defined as the

minimum

absolute difference between

any

two elements in the subsequence.

Return

the

sum

of

powers

of

all

subsequences of

nums

which have length

equal to

k

.

Since the answer may be large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

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

Output:

4

Explanation:

There are 4 subsequences in

nums

which have length 3:

[1,2,3]

,

[1,3,4]

,

[1,2,4]

, and

[2,3,4]

. The sum of powers is

|2 - 3| + |3 - 4| + |2 - 1| + |3 - 4| = 4

.

Example 2:

Input:

nums = [2,2], k = 2

Output:

0

Explanation:

The only subsequence in

nums

which has length 2 is

[2,2]

. The sum of powers is

|2 - 2| = 0

.

Example 3:

Input:

nums = [4,3,-1], k = 2

Output:

10

Explanation:

There are 3 subsequences in

nums

which have length 2:

[4,3]

,

[4,-1]

, and

[3,-1]

. The sum of powers is

|4 - 3| + |4 - (-1)| + |3 - (-1)| = 10

.

Constraints:

2 <= n == nums.length <= 50

-10

8

<= nums[i] <= 10

8

2 <= k <= n

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

```
}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (sum-of-powers nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Sum of Subsequence Powers
* Difficulty: Hard
```

```
* Tags: array, dp, sort
*
* 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 sumOfPowers(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Sum of Subsequence Powers
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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 sumOfPowers(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Sum of Subsequence Powers
Difficulty: Hard
Tags: array, dp, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find the Sum of Subsequence Powers
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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 sumOfPowers = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Sum of Subsequence Powers
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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 sumOfPowers(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Find the Sum of Subsequence Powers
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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 SumOfPowers(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Find the Sum of Subsequence Powers
* Difficulty: Hard
* Tags: array, dp, sort
*
* 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 sumOfPowers(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Find the Sum of Subsequence Powers
// Difficulty: Hard
// Tags: array, dp, sort
//
// 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 sumOfPowers(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find the Sum of Subsequence Powers
// Difficulty: Hard
// Tags: array, dp, sort
```

```
//
// 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 sum_of_powers(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (sum-of-powers nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```