Problem 3428: Maximum and Minimum Sums of at Most Size K Subsequences

Problem Information

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

Problem Description

You are given an integer array

nums

and a positive integer

k

. Return the sum of the

maximum

and

minimum

elements of all

subsequences

of

nums

with

at most

k

elements.

Since the answer may be very large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

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

Output:

24

Explanation:

The subsequences of

nums

with at most 2 elements are:

Subsequence

Minimum

Maximum

Sum

[1]

1

1

2

[2]

2

2

4

[3]

3

3

6

[1, 2]

1

2

3

[1, 3]

1

3

4

[2, 3]

2

3

5

Final Total

24

The output would be 24.

Example 2:

Input:

nums = [5,0,6], k = 1

Output:

2

2

Explanation:

For subsequences with exactly 1 element, the minimum and maximum values are the element itself. Therefore, the total is

5 + 5 + 0 + 0 + 6 + 6 = 22

.

Example 3:

Input:

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

Output:

12

Explanation:

The subsequences

[1, 1]

and

[1]

each appear 3 times. For all of them, the minimum and maximum are both 1. Thus, the total is 12.

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 10

9

1 <= k <= min(70, nums.length)

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
```

```
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-max-sums nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum and Minimum Sums of at Most Size K Subsequences
* Difficulty: Medium
* Tags: array, dp, math, 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 minMaxSums(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum and Minimum Sums of at Most Size K Subsequences
* Difficulty: Medium
* Tags: array, dp, math, 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 minMaxSums(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum and Minimum Sums of at Most Size K Subsequences
Difficulty: Medium
Tags: array, dp, math, 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 minMaxSums(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum and Minimum Sums of at Most Size K Subsequences
* Difficulty: Medium
* Tags: array, dp, math, 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 minMaxSums = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum and Minimum Sums of at Most Size K Subsequences
* Difficulty: Medium
* Tags: array, dp, math, 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 minMaxSums(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum and Minimum Sums of at Most Size K Subsequences
* Difficulty: Medium
* Tags: array, dp, math, 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 MinMaxSums(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Maximum and Minimum Sums of at Most Size K Subsequences
* Difficulty: Medium
* Tags: array, dp, math, 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 minMaxSums(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Maximum and Minimum Sums of at Most Size K Subsequences
// Difficulty: Medium
// Tags: array, dp, math, 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 minMaxSums(nums []int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum and Minimum Sums of at Most Size K Subsequences
// Difficulty: Medium
// Tags: array, dp, math, 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 min_max_sums(nums: Vec<i32>, k: i32) -> i32 {

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-max-sums nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```