Problem 1589: Maximum Sum Obtained of Any Permutation

Problem Information

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

Problem Description

We have an array of integers,

nums

, and an array of

requests

where

requests[i] = [start

i

, end

i

]

. The

i

th

request asks for the sum of

nums[start

i

] + nums[start

i

+ 1] + ... + nums[end

i

- 1] + nums[end

i

]

. Both

start

i

and

end

i

are

0-indexed

.

Return

the maximum total sum of all requests

among all permutations

of

nums

.

Since the answer may be too large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

nums = [1,2,3,4,5], requests = [[1,3],[0,1]]

Output:

19

Explanation:

One permutation of nums is [2,1,3,4,5] with the following result: requests[0] -> nums[1] + nums[2] + nums[3] = 1 + 3 + 4 = 8 requests[1] -> nums[0] + nums[1] = 2 + 1 = 3 Total sum: 8 + 3 = 11. A permutation with a higher total sum is [3,5,4,2,1] with the following result:

requests[0] -> nums[1] + nums[2] + nums[3] = 5 + 4 + 2 = 11 requests[1] -> nums[0] + nums[1] = 3 + 5 = 8 Total sum: 11 + 8 = 19, which is the best that you can do.

Example 2:

Input:

nums = [1,2,3,4,5,6], requests = [[0,1]]

Output:

11

Explanation:

A permutation with the max total sum is [6,5,4,3,2,1] with request sums [11].

Example 3:

Input:

nums = [1,2,3,4,5,10], requests = [[0,2],[1,3],[1,1]]

Output:

47

Explanation:

A permutation with the max total sum is [4,10,5,3,2,1] with request sums [19,18,10].

Constraints:

n == nums.length

1 <= n <= 10

5

0 <= nums[i] <= 10

5

1 <= requests.length <= 10

5

requests[i].length == 2

0 <= start

i

<= end

i

< n

Code Snippets

C++:

```
class Solution {
public:
int maxSumRangeQuery(vector<int>& nums, vector<vector<int>>& requests) {

}
};
```

Java:

```
class Solution {
public int maxSumRangeQuery(int[] nums, int[][] requests) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxSumRangeQuery(nums: number[], requests: number[][]): number {

};
```

C#:

```
public class Solution {
public int MaxSumRangeQuery(int[] nums, int[][] requests) {

}
}
```

C:

```
int maxSumRangeQuery(int* nums, int numsSize, int** requests, int
requestsSize, int* requestsColSize) {

}
```

Go:

```
func maxSumRangeQuery(nums []int, requests [][]int) int {

}
```

Kotlin:

```
class Solution {
fun maxSumRangeQuery(nums: IntArray, requests: Array<IntArray>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer[][]} requests
# @return {Integer}
def max_sum_range_query(nums, requests)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[][] $requests
* @return Integer
*/
function maxSumRangeQuery($nums, $requests) {

}
}
```

Dart:

```
class Solution {
int maxSumRangeQuery(List<int> nums, List<List<int>> requests) {

}
}
```

Scala:

```
object Solution {
def maxSumRangeQuery(nums: Array[Int], requests: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_sum_range_query(nums :: [integer], requests :: [[integer]]) ::
integer
def max_sum_range_query(nums, requests) do

end
end
```

Erlang:

```
-spec max_sum_range_query(Nums :: [integer()], Requests :: [[integer()]]) ->
integer().
max_sum_range_query(Nums, Requests) ->
.
```

Racket:

```
(define/contract (max-sum-range-query nums requests)
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Sum Obtained of Any Permutation
* 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 maxSumRangeQuery(vector<int>& nums, vector<vector<int>>& requests) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Sum Obtained of Any Permutation
* 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 maxSumRangeQuery(int[] nums, int[][] requests) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Sum Obtained of Any Permutation
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 maxSumRangeQuery(self, nums: List[int], requests: List[List[int]]) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Sum Obtained of Any Permutation
* 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[][]} requests
* @return {number}
*/
var maxSumRangeQuery = function(nums, requests) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Sum Obtained of Any Permutation
* 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 maxSumRangeQuery(nums: number[], requests: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Sum Obtained of Any Permutation
* 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 MaxSumRangeQuery(int[] nums, int[][] requests) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Sum Obtained of Any Permutation
* 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 maxSumRangeQuery(int* nums, int numsSize, int** requests, int
requestsSize, int* requestsColSize) {

}
```

Go Solution:

```
// Problem: Maximum Sum Obtained of Any Permutation
// 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 maxSumRangeQuery(nums []int, requests [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxSumRangeQuery(nums: IntArray, requests: Array<IntArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Sum Obtained of Any Permutation
// 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 max_sum_range_query(nums: Vec<i32>, requests: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer[][]} requests
# @return {Integer}
def max_sum_range_query(nums, requests)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer[][] $requests
* @return Integer
*/
function maxSumRangeQuery($nums, $requests) {

}
}
```

Dart Solution:

```
class Solution {
int maxSumRangeQuery(List<int> nums, List<List<int>> requests) {

}
}
```

Scala Solution:

```
object Solution {
def maxSumRangeQuery(nums: Array[Int], requests: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_sum_range_query(nums :: [integer], requests :: [[integer]]) ::
integer
def max_sum_range_query(nums, requests) do

end
end
```

Erlang Solution:

```
-spec max_sum_range_query(Nums :: [integer()], Requests :: [[integer()]]) ->
integer().
```

```

max_sum_range_query(Nums, Requests) ->
.
```

Racket Solution:

```
(define/contract (max-sum-range-query nums requests)
(-> (listof exact-integer?) (listof (listof exact-integer?)) exact-integer?)
)
```