Problem 2736: Maximum Sum Queries

Problem Information

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

Problem Description

You are given two

0-indexed

integer arrays

nums1

and

nums2

, each of length

n

, and a

1-indexed 2D array

queries

where

queries[i] = [x

i

, y

i

]

.

For the

i

th

query, find the

maximum value

of

nums1[j] + nums2[j]

among all indices

j

(0 <= j < n)

, where

nums1[j] >= x

i

and

nums2[j] >= y

i

, or

-1

if there is no

j

satisfying the constraints.

Return

an array

answer

where

answer[i]

is the answer to the

i

th

query.

Example 1:

Input:

nums1 = [4,3,1,2], nums2 = [2,4,9,5], queries = [[4,1],[1,3],[2,5]]

Output:

[6,10,7]

Explanation:

For the 1st query

x

i

= 4

and

y

i

= 1

, we can select index

j = 0

since

nums1[j] >= 4

and

nums2[j] >= 1

. The sum

nums1[j] + nums2[j]

is 6, and we can show that 6 is the maximum we can obtain.

For the 2nd query

x

i

= 1

and

y

i

= 3

, we can select index

j = 2

since

nums1[j] >= 1

and

nums2[j] >= 3

. The sum

nums1[j] + nums2[j]

is 10, and we can show that 10 is the maximum we can obtain.

For the 3rd query

x

i

= 2

and

y

i

= 5

, we can select index

j = 3

since

nums1[j] >= 2

and

nums2[j] >= 5

. The sum

nums1[j] + nums2[j]

is 7, and we can show that 7 is the maximum we can obtain.

Therefore, we return

[6,10,7]

.

Example 2:

Input:

nums1 = [3,2,5], nums2 = [2,3,4], queries = [[4,4],[3,2],[1,1]]

Output:

[9,9,9]

Explanation:

For this example, we can use index

j = 2

for all the queries since it satisfies the constraints for each query.

Example 3:

Input:

nums1 = [2,1], nums2 = [2,3], queries = [[3,3]]

Output:

[-1]

Explanation:

There is one query in this example with

x

i

= 3 and

y

i

= 3. For every index, j, either nums1[j] <

x

i

or nums2[j] <

y

i

. Hence, there is no solution.

Constraints:

nums1.length == nums2.length

n == nums1.length

1 <= n <= 10

5

1 <= nums1[i], nums2[i] <= 10

9

1 <= queries.length <= 10

5

queries[i].length == 2

x

i

== queries[i][1]

y

i

== queries[i][2]

1 <= x

i

, y

i

<= 10

9

Code Snippets

C++:

```
class Solution {
public:
vector<int> maximumSumQueries(vector<int>& nums1, vector<int>& nums2,
vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public int[] maximumSumQueries(int[] nums1, int[] nums2, int[][] queries) {

}
}
```

Python3:

```
class Solution:
def maximumSumQueries(self, nums1: List[int], nums2: List[int], queries:
List[List[int]]) -> List[int]:
```

Python:

```
class Solution(object):
def maximumSumQueries(self, nums1, nums2, queries):
"""
:type nums1: List[int]
:type nums2: List[int]
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} nums1
* @param {number[]} nums2
* @param {number[][]} queries
* @return {number[]}
*/
var maximumSumQueries = function(nums1, nums2, queries) {

};
```

TypeScript:

```
function maximumSumQueries(nums1: number[], nums2: number[], queries:
number[][]): number[] {

};
```

C#:

```
public class Solution {
public int[] MaximumSumQueries(int[] nums1, int[] nums2, int[][] queries) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* maximumSumQueries(int* nums1, int nums1Size, int* nums2, int nums2Size,
int** queries, int queriesSize, int* queriesColSize, int* returnSize) {

}
```

Go:

```
func maximumSumQueries(nums1 []int, nums2 []int, queries [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun maximumSumQueries(nums1: IntArray, nums2: IntArray, queries:
Array<IntArray>): IntArray {

}
}
```

Swift:

```
class Solution {
func maximumSumQueries(_ nums1: [Int], _ nums2: [Int], _ queries: [[Int]]) ->
[Int] {

}
}
```

Rust:

```
impl Solution {
pub fn maximum_sum_queries(nums1: Vec<i32>, nums2: Vec<i32>, queries:
Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @param {Integer[][]} queries
# @return {Integer[]}
def maximum_sum_queries(nums1, nums2, queries)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @param Integer[][] $queries
* @return Integer[]
*/
function maximumSumQueries($nums1, $nums2, $queries) {

}
}
```

Dart:

```
class Solution {
List<int> maximumSumQueries(List<int> nums1, List<int> nums2, List<List<int>>
queries) {

}
}
```

Scala:

```
object Solution {
def maximumSumQueries(nums1: Array[Int], nums2: Array[Int], queries:
Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec maximum_sum_queries(nums1 :: [integer], nums2 :: [integer], queries ::
[[integer]]) :: [integer]
def maximum_sum_queries(nums1, nums2, queries) do

end
end
```

Erlang:

```
-spec maximum_sum_queries(Nums1 :: [integer()], Nums2 :: [integer()], Queries
:: [[integer()]]) -> [integer()].
maximum_sum_queries(Nums1, Nums2, Queries) ->
.
```

Racket:

```
(define/contract (maximum-sum-queries nums1 nums2 queries)
(-> (listof exact-integer?) (listof exact-integer?) (listof (listof
exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Sum Queries
* Difficulty: Hard
* Tags: array, tree, sort, search, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
vector<int> maximumSumQueries(vector<int>& nums1, vector<int>& nums2,
vector<vector<int>>& queries) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Maximum Sum Queries
* Difficulty: Hard
* Tags: array, tree, sort, search, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int[] maximumSumQueries(int[] nums1, int[] nums2, int[][] queries) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Sum Queries
Difficulty: Hard
Tags: array, tree, sort, search, stack

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def maximumSumQueries(self, nums1: List[int], nums2: List[int], queries:
List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maximumSumQueries(self, nums1, nums2, queries):
"""
:type nums1: List[int]
:type nums2: List[int]
:type queries: List[List[int]]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Sum Queries
* Difficulty: Hard
* Tags: array, tree, sort, search, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {number[]} nums1
* @param {number[]} nums2
* @param {number[][]} queries
* @return {number[]}
*/
var maximumSumQueries = function(nums1, nums2, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Sum Queries
* Difficulty: Hard
* Tags: array, tree, sort, search, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

```
function maximumSumQueries(nums1: number[], nums2: number[], queries:
number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Maximum Sum Queries
* Difficulty: Hard
* Tags: array, tree, sort, search, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int[] MaximumSumQueries(int[] nums1, int[] nums2, int[][] queries) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Sum Queries
* Difficulty: Hard
* Tags: array, tree, sort, search, stack
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* maximumSumQueries(int* nums1, int nums1Size, int* nums2, int nums2Size,
int** queries, int queriesSize, int* queriesColSize, int* returnSize) {
```

```
}
```

Go Solution:

```
// Problem: Maximum Sum Queries
// Difficulty: Hard
// Tags: array, tree, sort, search, stack
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func maximumSumQueries(nums1 []int, nums2 []int, queries [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun maximumSumQueries(nums1: IntArray, nums2: IntArray, queries:
Array<IntArray>): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func maximumSumQueries(_ nums1: [Int], _ nums2: [Int], _ queries: [[Int]]) ->
[Int] {

}
}
```

Rust Solution:

```
// Problem: Maximum Sum Queries
// Difficulty: Hard
// Tags: array, tree, sort, search, stack
//
```

```
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn maximum_sum_queries(nums1: Vec<i32>, nums2: Vec<i32>, queries:
Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @param {Integer[][]} queries
# @return {Integer[]}
def maximum_sum_queries(nums1, nums2, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @param Integer[][] $queries
* @return Integer[]
*/
function maximumSumQueries($nums1, $nums2, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<int> maximumSumQueries(List<int> nums1, List<int> nums2, List<List<int>>
queries) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def maximumSumQueries(nums1: Array[Int], nums2: Array[Int], queries:
Array[Array[Int]]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec maximum_sum_queries(nums1 :: [integer], nums2 :: [integer], queries ::
[[integer]]) :: [integer]
def maximum_sum_queries(nums1, nums2, queries) do

end
end
```

Erlang Solution:

```
-spec maximum_sum_queries(Nums1 :: [integer()], Nums2 :: [integer()], Queries
:: [[integer()]]) -> [integer()].
maximum_sum_queries(Nums1, Nums2, Queries) ->
.
```

Racket Solution:

```
(define/contract (maximum-sum-queries nums1 nums2 queries)
(-> (listof exact-integer?) (listof exact-integer?) (listof (listof
exact-integer?)) (listof exact-integer?))
)
```