Problem 3478: Choose K Elements With Maximum Sum

Problem Information

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

Problem Description

You are given two integer arrays,

nums1

and

nums2

, both of length

n

, along with a positive integer

k

.

For each index

i

from

0

to

n - 1

, perform the following:

Find

all

indices

j

where

nums1[j]

is less than

nums1[i]

.

Choose

at most

k

values of

nums2[j]

at these indices to

maximize

the total sum.

Return an array

answer

of size

n

, where

answer[i]

represents the result for the corresponding index

i

.

Example 1:

Input:

nums1 = [4,2,1,5,3], nums2 = [10,20,30,40,50], k = 2

Output:

[80,30,0,80,50]

Explanation:

For

i = 0

: Select the 2 largest values from

nums2

at indices

[1, 2, 4]

where

nums1[j] < nums1[0]

, resulting in

50 + 30 = 80

.

For

i = 1

: Select the 2 largest values from

nums2

at index

[2]

where

nums1[j] < nums1[1]

, resulting in 30.

For

i = 2

: No indices satisfy

nums1[j] < nums1[2]

, resulting in 0.

For

i = 3

: Select the 2 largest values from

nums2

at indices

[0, 1, 2, 4]

where

nums1[j] < nums1[3]

, resulting in

50 + 30 = 80

.

For

i = 4

: Select the 2 largest values from

nums2

at indices

[1, 2]

where

nums1[j] < nums1[4]

, resulting in

30 + 20 = 50

.

Example 2:

Input:

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

Output:

[0,0,0,0]

Explanation:

Since all elements in

nums1

are equal, no indices satisfy the condition

nums1[j] < nums1[i]

for any

i

, resulting in 0 for all positions.

Constraints:

n == nums1.length == nums2.length

1 <= n <= 10

5

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

6

1 <= k <= n

Code Snippets

C++:

```
class Solution {
public:
vector<long long> findMaxSum(vector<int>& nums1, vector<int>& nums2, int k) {

}
};
```

Java:

```
class Solution {
public long[] findMaxSum(int[] nums1, int[] nums2, int k) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def findMaxSum(self, nums1, nums2, k):
"""
```

```
:type nums1: List[int]
:type nums2: List[int]
:type k: int
:rtype: List[int]
"""
```

JavaScript:

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

};
```

TypeScript:

```
function findMaxSum(nums1: number[], nums2: number[], k: number): number[] {

};
```

C#:

```
public class Solution {
public long[] FindMaxSum(int[] nums1, int[] nums2, int k) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* findMaxSum(int* nums1, int nums1Size, int* nums2, int nums2Size,
int k, int* returnSize) {

}
```

Go:

```
func findMaxSum(nums1 []int, nums2 []int, k int) []int64 {

}
```

Kotlin:

```
class Solution {
fun findMaxSum(nums1: IntArray, nums2: IntArray, k: Int): LongArray {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn find_max_sum(nums1: Vec<i32>, nums2: Vec<i32>, k: i32) -> Vec<i64> {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
List<int> findMaxSum(List<int> nums1, List<int> nums2, int k) {

}
}
```

Scala:

```
object Solution {
def findMaxSum(nums1: Array[Int], nums2: Array[Int], k: Int): Array[Long] = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec find_max_sum(Nums1 :: [integer()], Nums2 :: [integer()], K ::
integer()) -> [integer()].
```

```
find_max_sum(Nums1, Nums2, K) ->
.
```

Racket:

```
(define/contract (find-max-sum nums1 nums2 k)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer? (listof
exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Choose K Elements With Maximum Sum
* Difficulty: Medium
* Tags: array, sort, queue, heap
*
* 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:
vector<long long> findMaxSum(vector<int>& nums1, vector<int>& nums2, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Choose K Elements With Maximum Sum
* Difficulty: Medium
* Tags: array, sort, queue, heap
*
* 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 long[] findMaxSum(int[] nums1, int[] nums2, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Choose K Elements With Maximum Sum
Difficulty: Medium
Tags: array, sort, queue, heap

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 findMaxSum(self, nums1: List[int], nums2: List[int], k: int) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Choose K Elements With Maximum Sum
* Difficulty: Medium
* Tags: array, sort, queue, heap
*
* 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[]} nums1
* @param {number[]} nums2
* @param {number} k
* @return {number[]}
*/
var findMaxSum = function(nums1, nums2, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Choose K Elements With Maximum Sum
* Difficulty: Medium
* Tags: array, sort, queue, heap
*
* 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 findMaxSum(nums1: number[], nums2: number[], k: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Choose K Elements With Maximum Sum
* Difficulty: Medium
* Tags: array, sort, queue, heap
```

```
*
* 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 long[] FindMaxSum(int[] nums1, int[] nums2, int k) {

}
}
```

C Solution:

```
/*
* Problem: Choose K Elements With Maximum Sum
* Difficulty: Medium
* Tags: array, sort, queue, heap
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
long long* findMaxSum(int* nums1, int nums1Size, int* nums2, int nums2Size,
int k, int* returnSize) {

}
```

Go Solution:

```
// Problem: Choose K Elements With Maximum Sum
// Difficulty: Medium
// Tags: array, sort, queue, heap
//
// 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 findMaxSum(nums1 []int, nums2 []int, k int) []int64 {

}
```

Kotlin Solution:

```
class Solution {
fun findMaxSum(nums1: IntArray, nums2: IntArray, k: Int): LongArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Choose K Elements With Maximum Sum
// Difficulty: Medium
// Tags: array, sort, queue, heap
//
// 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 find_max_sum(nums1: Vec<i32>, nums2: Vec<i32>, k: i32) -> Vec<i64> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
List<int> findMaxSum(List<int> nums1, List<int> nums2, int k) {

}
}
```

Scala Solution:

```
object Solution {
def findMaxSum(nums1: Array[Int], nums2: Array[Int], k: Int): Array[Long] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_max_sum(nums1 :: [integer], nums2 :: [integer], k :: integer) ::
```

```
[integer]
def find_max_sum(nums1, nums2, k) do

end
end
```

Erlang Solution:

```
-spec find_max_sum(Nums1 :: [integer()], Nums2 :: [integer()], K ::
integer()) -> [integer()].
find_max_sum(Nums1, Nums2, K) ->
.
```

Racket Solution:

```
(define/contract (find-max-sum nums1 nums2 k)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer? (listof
exact-integer?))
)
```