Problem 373: Find K Pairs with Smallest Sums

Problem Information

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

Problem Description

You are given two integer arrays

nums1

and

nums2

sorted in

non-decreasing order

and an integer

k

.

Define a pair

(u, v)

which consists of one element from the first array and one element from the second array.

Return

the

k

pairs

(u

1

, v

1

), (u

2

, v

2

), ..., (u

k

, v

k

)

with the smallest sums

.

Example 1:

Input:

nums1 = [1,7,11], nums2 = [2,4,6], k = 3

Output:

[[1,2],[1,4],[1,6]]

Explanation:

The first 3 pairs are returned from the sequence:
[1,2],[1,4],[1,6],[7,2],[7,4],[11,2],[7,6],[11,4],[11,6]

Example 2:

Input:

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

Output:

[[1,1],[1,1]]

Explanation:

The first 2 pairs are returned from the sequence: [1,1],[1,1],[1,2],[2,1],[1,2],[2,2],[1,3],[1,3],[2,3]

Constraints:

1 <= nums1.length, nums2.length <= 10

5

-10

9

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

9

nums1

and

nums2

both are sorted in

non-decreasing order

.

1 <= k <= 10

4

k <= nums1.length * nums2.length

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<List<Integer>> kSmallestPairs(int[] nums1, int[] nums2, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public IList<IList<int>> KSmallestPairs(int[] nums1, int[] nums2, int k) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** kSmallestPairs(int* nums1, int nums1Size, int* nums2, int nums2Size,
int k, int* returnSize, int** returnColumnSizes) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun kSmallestPairs(nums1: IntArray, nums2: IntArray, k: Int): List<List<Int>>
{

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (k-smallest-pairs nums1 nums2 k)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer? (listof
(listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find K Pairs with Smallest Sums
* 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<vector<int>> kSmallestPairs(vector<int>& nums1, vector<int>& nums2,
int k) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Find K Pairs with Smallest Sums
* 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 List<List<Integer>> kSmallestPairs(int[] nums1, int[] nums2, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Find K Pairs with Smallest Sums
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 kSmallestPairs(self, nums1: List[int], nums2: List[int], k: int) ->
List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find K Pairs with Smallest Sums
* 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 kSmallestPairs = function(nums1, nums2, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Find K Pairs with Smallest Sums
* 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 kSmallestPairs(nums1: number[], nums2: number[], k: number):
number[][] {

};
```

C# Solution:

```
/*
* Problem: Find K Pairs with Smallest Sums
* 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 IList<IList<int>> KSmallestPairs(int[] nums1, int[] nums2, int k) {

}
}
```

C Solution:

```
/*
* Problem: Find K Pairs with Smallest Sums
* 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
```

```
*/
int** kSmallestPairs(int* nums1, int nums1Size, int* nums2, int nums2Size,
int k, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Find K Pairs with Smallest Sums
// 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 kSmallestPairs(nums1 []int, nums2 []int, k int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun kSmallestPairs(nums1: IntArray, nums2: IntArray, k: Int): List<List<Int>>
{

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find K Pairs with Smallest Sums
// 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 k_smallest_pairs(nums1: Vec<i32>, nums2: Vec<i32>, k: i32) ->
Vec<Vec<i32>> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (k-smallest-pairs nums1 nums2 k)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer? (listof
(listof exact-integer?)))
)
```