Problem 3572: Maximize Y■Sum by Picking a Triplet of Distinct X■Values

Problem Information

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

Problem Description

You are given two integer arrays

x

and

y

, each of length

n

. You must choose three

distinct

indices

i

,

j

, and

k

such that:

x[i] != x[j]

x[j] != x[k]

x[k] != x[i]

Your goal is to

maximize

the value of

y[i] + y[j] + y[k]

under these conditions. Return the

maximum

possible sum that can be obtained by choosing such a triplet of indices.

If no such triplet exists, return -1.

Example 1:

Input:

x = [1,2,1,3,2], y = [5,3,4,6,2]

Output:

14

Explanation:

Choose

i = 0

(

x[i] = 1

,

y[i] = 5

),

j = 1

(

x[j] = 2

,

y[j] = 3

),

k = 3

(

x[k] = 3

,

y[k] = 6

).

All three values chosen from

x

are distinct.

5 + 3 + 6 = 14

is the maximum we can obtain. Hence, the output is 14.

Example 2:

Input:

x = [1,2,1,2], y = [4,5,6,7]

Output:

-1

Explanation:

There are only two distinct values in

x

. Hence, the output is -1.

Constraints:

n == x.length == y.length

3 <= n <= 10

5

1 <= x[i], y[i] <= 10

6

Code Snippets

C++:

```
class Solution {
public:
int maxSumDistinctTriplet(vector<int>& x, vector<int>& y) {

}
};
```

Java:

```
class Solution {
public int maxSumDistinctTriplet(int[] x, int[] y) {

}
}
```

Python3:

```
class Solution:
def maxSumDistinctTriplet(self, x: List[int], y: List[int]) -> int:
```

Python:

```
class Solution(object):
def maxSumDistinctTriplet(self, x, y):
"""
:type x: List[int]
:type y: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} x
* @param {number[]} y
* @return {number}
*/
var maxSumDistinctTriplet = function(x, y) {
```

```
};
```

TypeScript:

```
function maxSumDistinctTriplet(x: number[], y: number[]): number {

};
```

C#:

```
public class Solution {
public int MaxSumDistinctTriplet(int[] x, int[] y) {

}
}
```

C:

```
int maxSumDistinctTriplet(int* x, int xSize, int* y, int ySize) {

}
```

Go:

```
func maxSumDistinctTriplet(x []int, y []int) int {

}
```

Kotlin:

```
class Solution {
fun maxSumDistinctTriplet(x: IntArray, y: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func maxSumDistinctTriplet(_ x: [Int], _ y: [Int]) -> Int {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn max_sum_distinct_triplet(x: Vec<i32>, y: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} x
# @param {Integer[]} y
# @return {Integer}
def max_sum_distinct_triplet(x, y)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $x
* @param Integer[] $y
* @return Integer
*/
function maxSumDistinctTriplet($x, $y) {

}
}
```

Dart:

```
class Solution {
int maxSumDistinctTriplet(List<int> x, List<int> y) {

}
}
```

Scala:

```
object Solution {
def maxSumDistinctTriplet(x: Array[Int], y: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_sum_distinct_triplet(x :: [integer], y :: [integer]) :: integer
def max_sum_distinct_triplet(x, y) do

end
end
```

Erlang:

```
-spec max_sum_distinct_triplet(X :: [integer()], Y :: [integer()]) ->
integer().
max_sum_distinct_triplet(X, Y) ->
.
```

Racket:

```
(define/contract (max-sum-distinct-triplet x y)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize Y■Sum by Picking a Triplet of Distinct X■Values
* Difficulty: Medium
* Tags: array, greedy, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

class Solution {
public:
int maxSumDistinctTriplet(vector<int>& x, vector<int>& y) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize Y■Sum by Picking a Triplet of Distinct X■Values
* Difficulty: Medium
* Tags: array, greedy, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int maxSumDistinctTriplet(int[] x, int[] y) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize Y■Sum by Picking a Triplet of Distinct X■Values
Difficulty: Medium
Tags: array, greedy, hash, sort, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def maxSumDistinctTriplet(self, x: List[int], y: List[int]) -> int:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxSumDistinctTriplet(self, x, y):
"""
:type x: List[int]
:type y: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximize Y■Sum by Picking a Triplet of Distinct X■Values
* Difficulty: Medium
* Tags: array, greedy, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} x
* @param {number[]} y
* @return {number}
*/
var maxSumDistinctTriplet = function(x, y) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize Y■Sum by Picking a Triplet of Distinct X■Values
* Difficulty: Medium
* Tags: array, greedy, hash, sort, queue, heap
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function maxSumDistinctTriplet(x: number[], y: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Maximize Y■Sum by Picking a Triplet of Distinct X■Values
* Difficulty: Medium
* Tags: array, greedy, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int MaxSumDistinctTriplet(int[] x, int[] y) {

}
}
```

C Solution:

```
/*
* Problem: Maximize Y■Sum by Picking a Triplet of Distinct X■Values
* Difficulty: Medium
* Tags: array, greedy, hash, sort, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int maxSumDistinctTriplet(int* x, int xSize, int* y, int ySize) {
```

```
}
```

Go Solution:

```
// Problem: Maximize Y■Sum by Picking a Triplet of Distinct X■Values
// Difficulty: Medium
// Tags: array, greedy, hash, sort, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func maxSumDistinctTriplet(x []int, y []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxSumDistinctTriplet(x: IntArray, y: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxSumDistinctTriplet(_ x: [Int], _ y: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximize Y■Sum by Picking a Triplet of Distinct X■Values
// Difficulty: Medium
// Tags: array, greedy, hash, sort, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map
```

```
impl Solution {
pub fn max_sum_distinct_triplet(x: Vec<i32>, y: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} x
# @param {Integer[]} y
# @return {Integer}
def max_sum_distinct_triplet(x, y)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $x
* @param Integer[] $y
* @return Integer
*/
function maxSumDistinctTriplet($x, $y) {

}
}
```

Dart Solution:

```
class Solution {
int maxSumDistinctTriplet(List<int> x, List<int> y) {

}
}
```

Scala Solution:

```
object Solution {
def maxSumDistinctTriplet(x: Array[Int], y: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_sum_distinct_triplet(x :: [integer], y :: [integer]) :: integer
def max_sum_distinct_triplet(x, y) do

end
end
```

Erlang Solution:

```
-spec max_sum_distinct_triplet(X :: [integer()], Y :: [integer()]) ->
integer().
max_sum_distinct_triplet(X, Y) ->
.
```

Racket Solution:

```
(define/contract (max-sum-distinct-triplet x y)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```