

# Problem 760: Find Anagram Mappings

## Problem Information

Difficulty: [Easy](#)

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

You are given two integer arrays

nums1

and

nums2

where

nums2

is

an anagram

of

nums1

. Both arrays may contain duplicates.

Return

an index mapping array

mapping

from

nums1

to

nums2

where

mapping[i] = j

means the

i

th

element in

nums1

appears in

nums2

at index

j

. If there are multiple answers, return

any of them

An array

a  
is  
an anagram  
of an array

b  
means  
b

is made by randomizing the order of the elements in

a  
.

Example 1:

Input:

nums1 = [12,28,46,32,50], nums2 = [50,12,32,46,28]

Output:

[1,4,3,2,0]

Explanation:

As mapping[0] = 1 because the 0

th

element of nums1 appears at nums2[1], and mapping[1] = 4 because the 1

st

element of nums1 appears at nums2[4], and so on.

Example 2:

Input:

nums1 = [84,46], nums2 = [84,46]

Output:

[0,1]

Constraints:

1 <= nums1.length <= 100

nums2.length == nums1.length

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

5

nums2

is an anagram of

nums1

.

## Code Snippets

C++:

```
class Solution {  
public:
```

```
vector<int> anagramMappings(vector<int>& nums1, vector<int>& nums2) {  
    }  
};
```

### Java:

```
class Solution {  
public int[] anagramMappings(int[] nums1, int[] nums2) {  
    }  
}
```

### Python3:

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

### Python:

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

### JavaScript:

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

### TypeScript:

```
function anagramMappings(nums1: number[], nums2: number[]): number[] {  
};
```

### C#:

```
public class Solution {  
    public int[] AnagramMappings(int[] nums1, int[] nums2) {  
        return new int[] {  
    };  
}
```

### C:

```
/**  
 * Note: The returned array must be malloced, assume caller calls free().  
 */  
int* anagramMappings(int* nums1, int nums1Size, int* nums2, int nums2Size,  
int* returnSize) {  
    *returnSize = nums1Size;  
    for (int i = 0; i < nums1Size; i++) {  
        nums1[i] = nums2[*(nums2 + (i * nums2Size))];  
    }  
}
```

### Go:

```
func anagramMappings(nums1 []int, nums2 []int) []int {  
    return nil  
}
```

### Kotlin:

```
class Solution {  
    fun anagramMappings(nums1: IntArray, nums2: IntArray): IntArray {  
        return IntArray(nums1.size) { i ->  
            nums1[i] = nums2[i]  
        }  
    }  
}
```

### Swift:

```
class Solution {  
    func anagramMappings(_ nums1: [Int], _ nums2: [Int]) -> [Int] {  
        var result: [Int] = []  
        for i in 0..            result.append(nums2[nums1[i]])  
        }  
        return result  
    }  
}
```

**Rust:**

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

**Ruby:**

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

**PHP:**

```
class Solution {  
  
    /**  
     * @param Integer[] $nums1  
     * @param Integer[] $nums2  
     * @return Integer[]  
     */  
    function anagramMappings($nums1, $nums2) {  
  
    }  
}
```

**Dart:**

```
class Solution {  
    List<int> anagramMappings(List<int> nums1, List<int> nums2) {  
  
    }  
}
```

**Scala:**

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

```
}
```

```
}
```

### Elixir:

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

### Erlang:

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

### Racket:

```
(define/contract (anagram-mappings numsl nums2)
  (-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```

## Solutions

### C++ Solution:

```
/*
 * Problem: Find Anagram Mappings
 * Difficulty: Easy
 * Tags: array, string, hash
 *
 * 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:
vector<int> anagramMappings(vector<int>& nums1, vector<int>& nums2) {

}
};

```

### Java Solution:

```

/**
 * Problem: Find Anagram Mappings
 * Difficulty: Easy
 * Tags: array, string, hash
 *
 * 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[] anagramMappings(int[] nums1, int[] nums2) {

}
}

```

### Python3 Solution:

```

"""
Problem: Find Anagram Mappings
Difficulty: Easy
Tags: array, string, hash

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

```

### Python Solution:

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

### JavaScript Solution:

```
/**
 * Problem: Find Anagram Mappings
 * Difficulty: Easy
 * Tags: array, string, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

var anagramMappings = function(nums1, nums2) {
```

### TypeScript Solution:

```
/**
 * Problem: Find Anagram Mappings
 * Difficulty: Easy
 * Tags: array, string, hash
 *
 * 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 anagramMappings(nums1: number[], nums2: number[]): number[] {  
};
```

### C# Solution:

```
/*  
 * Problem: Find Anagram Mappings  
 * Difficulty: Easy  
 * Tags: array, string, hash  
 *  
 * 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[] AnagramMappings(int[] nums1, int[] nums2) {  
        // Implementation  
    }  
}
```

### C Solution:

```
/*  
 * Problem: Find Anagram Mappings  
 * Difficulty: Easy  
 * Tags: array, string, hash  
 *  
 * Approach: Use two pointers or sliding window technique  
 * Time Complexity: O(n) or O(n log n)  
 * Space Complexity: O(n) for hash map  
 */  
  
/**  
 * Note: The returned array must be malloced, assume caller calls free().  
 */  
int* anagramMappings(int* nums1, int nums1Size, int* nums2, int nums2Size,  
int* returnSize) {  
    // Implementation  
}
```

```
}
```

### Go Solution:

```
// Problem: Find Anagram Mappings
// Difficulty: Easy
// Tags: array, string, hash
//
// 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 anagramMappings(nums1 []int, nums2 []int) []int {
}
```

### Kotlin Solution:

```
class Solution {
    fun anagramMappings(nums1: IntArray, nums2: IntArray): IntArray {
        return IntArray(nums1.size) { i ->
            val index = findIndex(nums1[i])
            if (index == -1) -1 else nums2[index]
        }
    }
}
```

### Swift Solution:

```
class Solution {
    func anagramMappings(_ nums1: [Int], _ nums2: [Int]) -> [Int] {
        return [Int](repeating: -1, count: nums1.count)
    }
}
```

### Rust Solution:

```
// Problem: Find Anagram Mappings
// Difficulty: Easy
// Tags: array, string, hash
//
// 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 anagram_mappings(nums1: Vec<i32>, nums2: Vec<i32>) -> Vec<i32> {  
        let mut map = std::collections::HashMap::new();  
        for (i, &n) in nums1.iter().enumerate() {  
            map.insert(n, i);  
        }  
        let mut result = Vec::new();  
        for n in &nums1 {  
            result.push(map[&n]);  
        }  
        result  
    }  
}
```

### Ruby Solution:

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

### PHP Solution:

```
class Solution {  
  
    /**  
     * @param Integer[] $nums1  
     * @param Integer[] $nums2  
     * @return Integer[]  
     */  
    function anagramMappings($nums1, $nums2) {  
  
    }  
}
```

### Dart Solution:

```
class Solution {  
    List<int> anagramMappings(List<int> nums1, List<int> nums2) {  
        List<int> result = List<int>.filled(nums1.length, 0);  
        Map<int, int> map = {};  
        for (int i = 0; i < nums1.length; i++) {  
            map[nums1[i]] = i;  
        }  
        for (int i = 0; i < nums1.length; i++) {  
            result[i] = map[nums2[i]]!;  
        }  
        return result;  
    }  
}
```

### Scala Solution:

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

### Elixir Solution:

```
defmodule Solution do  
  @spec anagram_mappings(list :: [integer], list :: [integer]) :: [integer]  
  def anagram_mappings(list1, list2) do  
  
  end  
end
```

### Erlang Solution:

```
-spec anagram_mappings(list :: [integer()], list :: [integer()]) ->  
[integer()].  
anagram_mappings(List1, List2) ->  
.
```

### Racket Solution:

```
(define/contract (anagram-mappings numsl numsr)  
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))  
)
```