Problem 2122: Recover the Original Array

Problem Information

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

Problem Description

Alice had a

0-indexed

array

arr

consisting of

n

positive

integers. She chose an arbitrary

positive integer

k

and created two new

0-indexed

integer arrays

lower

and

higher

in the following manner:

lower[i] = arr[i] - k

, for every index

i

where

0 <= i < n

higher[i] = arr[i] + k

, for every index

i

where

0 <= i < n

Unfortunately, Alice lost all three arrays. However, she remembers the integers that were present in the arrays

lower

and

higher

, but not the array each integer belonged to. Help Alice and recover the original array.

Given an array

nums

consisting of

2n

integers, where

exactly

n

of the integers were present in

lower

and the remaining in

higher

, return

the

original

array

arr

. In case the answer is not unique, return

any

valid array

.

Note:

The test cases are generated such that there exists

at least one

valid array

arr

.

Example 1:

Input:

nums = [2,10,6,4,8,12]

Output:

[3,7,11]

Explanation:

If arr = [3,7,11] and k = 1, we get lower = [2,6,10] and higher = [4,8,12]. Combining lower and higher gives us [2,6,10,4,8,12], which is a permutation of nums. Another valid possibility is that arr = [5,7,9] and k = 3. In that case, lower = [2,4,6] and higher = [8,10,12].

Example 2:

Input:

nums = [1,1,3,3]

Output:

[2,2]

Explanation:

If arr = [2,2] and k = 1, we get lower = [1,1] and higher = [3,3]. Combining lower and higher gives us [1,1,3,3], which is equal to nums. Note that arr cannot be [1,3] because in that case, the only possible way to obtain [1,1,3,3] is with k = 0. This is invalid since k must be positive.

Example 3:

Input:

nums = [5,435]

Output:

[220]

Explanation:

The only possible combination is arr = [220] and k = 215. Using them, we get lower = [5] and higher = [435].

Constraints:

2 * n == nums.length

1 <= n <= 1000

1 <= nums[i] <= 10

9

The test cases are generated such that there exists

at least one

valid array

arr

.

Code Snippets

C++:

```
class Solution {
public:
vector<int> recoverArray(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int[] recoverArray(int[] nums) {

}
}
```

Python3:

```
class Solution:
def recoverArray(self, nums: List[int]) -> List[int]:
```

Python:

```
class Solution(object):
def recoverArray(self, nums):
"""
:type nums: List[int]
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @return {number[]}
*/
var recoverArray = function(nums) {

};
```

TypeScript:

```
function recoverArray(nums: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] RecoverArray(int[] nums) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* recoverArray(int* nums, int numsSize, int* returnSize) {

}
```

Go:

```
func recoverArray(nums []int) []int {

}
```

Kotlin:

```
class Solution {
fun recoverArray(nums: IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func recoverArray(_ nums: [Int]) -> [Int] {

}
```

```
}
```

Rust:

```
impl Solution {
pub fn recover_array(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer[]}
def recover_array(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer[]
*/
function recoverArray($nums) {

}
}
```

Dart:

```
class Solution {
List<int> recoverArray(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def recoverArray(nums: Array[Int]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec recover_array(nums :: [integer]) :: [integer]
def recover_array(nums) do

end
end
```

Erlang:

```
-spec recover_array(Nums :: [integer()]) -> [integer()].
recover_array(Nums) ->
.
```

Racket:

```
(define/contract (recover-array nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Recover the Original Array
* Difficulty: Hard
* Tags: array, hash, sort
*
* 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> recoverArray(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Recover the Original Array
* Difficulty: Hard
* Tags: array, hash, sort
*
* 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[] recoverArray(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Recover the Original Array
Difficulty: Hard
Tags: array, hash, sort

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 recoverArray(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def recoverArray(self, nums):
"""
:type nums: List[int]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Recover the Original Array
* Difficulty: Hard
* Tags: array, hash, sort
*
* 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[]} nums
* @return {number[]}
*/
var recoverArray = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Recover the Original Array
* Difficulty: Hard
* Tags: array, hash, sort
*
* 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 recoverArray(nums: number[]): number[] {
```

```
};
```

C# Solution:

```
/*
* Problem: Recover the Original Array
* Difficulty: Hard
* Tags: array, hash, sort
*
* 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[] RecoverArray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Recover the Original Array
* Difficulty: Hard
* Tags: array, hash, sort
*
* 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* recoverArray(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Recover the Original Array
// Difficulty: Hard
// Tags: array, hash, sort
//
// 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 recoverArray(nums []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun recoverArray(nums: IntArray): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func recoverArray(_ nums: [Int]) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Recover the Original Array
// Difficulty: Hard
// Tags: array, hash, sort
//
// 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 recover_array(nums: Vec<i32>) -> Vec<i32> {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer[]}
def recover_array(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer[]
*/
function recoverArray($nums) {

}
}
```

Dart Solution:

```
class Solution {
List<int> recoverArray(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def recoverArray(nums: Array[Int]): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec recover_array(nums :: [integer]) :: [integer]
def recover_array(nums) do

end
end
```

Erlang Solution:

```
-spec recover_array(Nums :: [integer()]) -> [integer()].
recover_array(Nums) ->
.
```

Racket Solution:

```
(define/contract (recover-array nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```