# Problem 888: Fair Candy Swap

## Problem Information

**Difficulty:** Easy
**Acceptance Rate:** 0.00%
**Paid Only:** No

## Problem Description

Alice and Bob have a different total number of candies. You are given two integer arrays

aliceSizes

and

bobSizes

where

aliceSizes[i]

is the number of candies of the

i

th

box of candy that Alice has and

bobSizes[j]

is the number of candies of the

j

th

box of candy that Bob has.

Since they are friends, they would like to exchange one candy box each so that after the exchange, they both have the same total amount of candy. The total amount of candy a person has is the sum of the number of candies in each box they have.

Return a

n integer array

answer

where

answer[0]

is the number of candies in the box that Alice must exchange, and

answer[1]

is the number of candies in the box that Bob must exchange

. If there are multiple answers, you may

return any

one of them. It is guaranteed that at least one answer exists.

Example 1:

Input:

aliceSizes = [1,1], bobSizes = [2,2]

Output:

[1,2]

Example 2:

Input:

aliceSizes = [1,2], bobSizes = [2,3]

Output:

[1,2]

Example 3:

Input:

aliceSizes = [2], bobSizes = [1,3]

Output:

[2,3]

Constraints:

1 <= aliceSizes.length, bobSizes.length <= 10

4

1 <= aliceSizes[i], bobSizes[j] <= 10

5

Alice and Bob have a different total number of candies.

There will be at least one valid answer for the given input.

## Code Snippets

**C++:**

```
class Solution {
public:
vector<int> fairCandySwap(vector<int>& aliceSizes, vector<int>& bobSizes) {

}
};
```

**Java:**

```
class Solution {
public int[] fairCandySwap(int[] aliceSizes, int[] bobSizes) {

}
}
```

**Python3:**

```
class Solution:
def fairCandySwap(self, aliceSizes: List[int], bobSizes: List[int]) ->
List[int]:
```

**Python:**

```
class Solution(object):
def fairCandySwap(self, aliceSizes, bobSizes):
"""
:type aliceSizes: List[int]
:type bobSizes: List[int]
:rtype: List[int]
"""
```

**JavaScript:**

```
/**
* @param {number[]} aliceSizes
* @param {number[]} bobSizes
* @return {number[]}
*/
var fairCandySwap = function(aliceSizes, bobSizes) {

};
```

**TypeScript:**

```
function fairCandySwap(aliceSizes: number[], bobSizes: number[]): number[] {

};
```

**C#:**

```
public class Solution {
public int[] FairCandySwap(int[] aliceSizes, int[] bobSizes) {

}
}
```

**C:**

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* fairCandySwap(int* aliceSizes, int aliceSizesSize, int* bobSizes, int
bobSizesSize, int* returnSize) {

}
```

**Go:**

```
func fairCandySwap(aliceSizes []int, bobSizes []int) []int {

}
```

**Kotlin:**

```
class Solution {
fun fairCandySwap(aliceSizes: IntArray, bobSizes: IntArray): IntArray {

}
}
```

**Swift:**

```
class Solution {
func fairCandySwap(_ aliceSizes: [Int], _ bobSizes: [Int]) -> [Int] {

}
}
```

**Rust:**

```
impl Solution {
pub fn fair_candy_swap(alice_sizes: Vec<i32>, bob_sizes: Vec<i32>) ->
Vec<i32> {

}
}
```

**Ruby:**

```
# @param {Integer[]} alice_sizes
# @param {Integer[]} bob_sizes
# @return {Integer[]}
def fair_candy_swap(alice_sizes, bob_sizes)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[] $aliceSizes
* @param Integer[] $bobSizes
* @return Integer[]
*/
function fairCandySwap($aliceSizes, $bobSizes) {

}
}
```

**Dart:**

```
class Solution {
List<int> fairCandySwap(List<int> aliceSizes, List<int> bobSizes) {

}
}
```

**Scala:**

```
object Solution {
def fairCandySwap(aliceSizes: Array[Int], bobSizes: Array[Int]): Array[Int] =
{

}
}
```

**Elixir:**

```
defmodule Solution do
@spec fair_candy_swap(alice_sizes :: [integer], bob_sizes :: [integer]) ::
[integer]
def fair_candy_swap(alice_sizes, bob_sizes) do

end
end
```

**Erlang:**

```
-spec fair_candy_swap(AliceSizes :: [integer()], BobSizes :: [integer()]) ->
[integer()].
fair_candy_swap(AliceSizes, BobSizes) ->
.
```

**Racket:**

```
(define/contract (fair-candy-swap aliceSizes bobSizes)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Fair Candy Swap
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* 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> fairCandySwap(vector<int>& aliceSizes, vector<int>& bobSizes) {

}
};
```

### Java Solution:

```
/**
* Problem: Fair Candy Swap
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* 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[] fairCandySwap(int[] aliceSizes, int[] bobSizes) {

}
}
```

### Python3 Solution:

```
"""
Problem: Fair Candy Swap
Difficulty: Easy
Tags: array, hash, sort, search

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 fairCandySwap(self, aliceSizes: List[int], bobSizes: List[int]) ->
List[int]:
# TODO: Implement optimized solution
pass
```

**Python Solution:**

```
class Solution(object):
def fairCandySwap(self, aliceSizes, bobSizes):
"""
:type aliceSizes: List[int]
:type bobSizes: List[int]
:rtype: List[int]
"""
```

**JavaScript Solution:**

```
/**
* Problem: Fair Candy Swap
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* 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[]} aliceSizes
* @param {number[]} bobSizes
* @return {number[]}
*/
var fairCandySwap = function(aliceSizes, bobSizes) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Fair Candy Swap
* Difficulty: Easy
```

```
* Tags: array, hash, sort, search
*
* 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 fairCandySwap(aliceSizes: number[], bobSizes: number[]): number[] {

};
```

### C# Solution:

```
/*
* Problem: Fair Candy Swap
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* 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[] FairCandySwap(int[] aliceSizes, int[] bobSizes) {

}
}
```

### C Solution:

```
/*
* Problem: Fair Candy Swap
* Difficulty: Easy
* Tags: array, hash, sort, search
*
* 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* fairCandySwap(int* aliceSizes, int aliceSizesSize, int* bobSizes, int
bobSizesSize, int* returnSize) {

}
```

### Go Solution:

```
// Problem: Fair Candy Swap
// Difficulty: Easy
// Tags: array, hash, sort, search
//
// 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 fairCandySwap(aliceSizes []int, bobSizes []int) []int {

}
```

### Kotlin Solution:

```
class Solution {
fun fairCandySwap(aliceSizes: IntArray, bobSizes: IntArray): IntArray {

}
}
```

### Swift Solution:

```
class Solution {
func fairCandySwap(_ aliceSizes: [Int], _ bobSizes: [Int]) -> [Int] {

}
}
```

### Rust Solution:

```
// Problem: Fair Candy Swap
// Difficulty: Easy
// Tags: array, hash, sort, search
//
// 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 fair_candy_swap(alice_sizes: Vec<i32>, bob_sizes: Vec<i32>) ->
Vec<i32> {

}
}
```

### Ruby Solution:

```
# @param {Integer[]} alice_sizes
# @param {Integer[]} bob_sizes
# @return {Integer[]}
def fair_candy_swap(alice_sizes, bob_sizes)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[] $aliceSizes
* @param Integer[] $bobSizes
* @return Integer[]
*/
function fairCandySwap($aliceSizes, $bobSizes) {

}
}
```

### Dart Solution:

```
class Solution {
List<int> fairCandySwap(List<int> aliceSizes, List<int> bobSizes) {
```

```
}
}
```

### Scala Solution:

```
object Solution {
def fairCandySwap(aliceSizes: Array[Int], bobSizes: Array[Int]): Array[Int] =
{

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec fair_candy_swap(alice_sizes :: [integer], bob_sizes :: [integer]) ::
[integer]
def fair_candy_swap(alice_sizes, bob_sizes) do

end
end
```

### Erlang Solution:

```
-spec fair_candy_swap(AliceSizes :: [integer()], BobSizes :: [integer()]) ->
[integer()].
fair_candy_swap(AliceSizes, BobSizes) ->
.
```

### Racket Solution:

```
(define/contract (fair-candy-swap aliceSizes bobSizes)
(-> (listof exact-integer?) (listof exact-integer?) (listof exact-integer?))
)
```