Given an array of integers numbers, we'd like to find the closest pair of elements that add up to sum. Return the distance between the closest pair (absolute difference between the two indices). If there isn't a pair that adds up to sum, return -1.

**Example**

* For numbers = [1, 0, 2, 4, 3, 0] and sum = 5 the output should be findClosestPair(numbers, sum) = 2. 1 and 4 have a sum of 5, but 2 and 3 are closer.
* for numbers = [2, 3, 7] and sum = 8 the output should be findClosestPair(numbers, sum) = -1. There are no pairs that have a sum of 8.

**Input / Output**

* **[execution time limit] 3 seconds (cs)**
* **[input] array.integer numbers**

An array of integers.

*Guaranteed constraints:*  
2 ≤ numbers.length ≤ 3 · 105  
-1000 ≤ numbers[i] ≤ 1000

* **[input] integer sum**

An integer representing the sum of the pair we're looking for.

*Guaranteed constraints:*  
-2000 ≤ sum ≤ 2000

* **[output] integer**

An integer representing the difference between the indices of the closest pair of elements that add up to sum (or -1 if there isn't one).

**[C#] Syntax Tips**

// Prints help message to the console

// Returns a string

**string** **helloWorld**(**string** name) {

Console.Write("This prints to the console when you Run Tests");

**return** "Hello, " + name;

}

<https://codefights.com/challenge/4Q22wFQPwR7zq2FS4>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

public class GFG

{

public static int findClosestPair (int[] numbers, int sum)

{

int min\_dist = int.MaxValue;

if (numbers.Length < 2)

{

return -1;

}

bool hay = false;

Dictionary<int, int> hm = new Dictionary<int, int>();

//foreach (int value in numbers)

for (int i = 0; i < numbers.Length; i++)

{

int target = sum - numbers[i]; // if target number is not in set then add

if (!hm.ContainsKey(target))

{

//set.Add(value);

//hm.Add(numbers[i], i);

hm[numbers[i]] = i;

}

else

{

//System.out.printf("(%d, %d) %n", value, target);

//Console.WriteLine(numbers[i] + " " + target);

min\_dist = Math.Min(min\_dist, Math.Abs(i - hm[target]));

hay = true;

}

}

if (!hay) return -1;

return min\_dist;

}

static void Main(string[] args)

{

//int[] numbers = { 1, 0, 2, 4, 3, 0 };

//int sum = 5;

//int[] numbers = { 6, 5, -3, 2, -5, 1 };

//int sum = 1;

int[] numbers = { 2, 3, 7 };

int sum = 6;

Console.WriteLine(printPairsUsingSet(numbers, sum));

Console.ReadLine();

}

}