**QPREP8-Two Sum**

**Module Introduction**

Write a program to take an array of integers as input and output the indices of two numbers that add up to a specified sum.

#### Objective

Given an array of integers as input, output the indices of two numbers in the array which add up to a specified target.

Assume that each input would have exactly one solution and you cannot use the same element twice. If 2 different elements have the same value, they can be used.

#### Examples

**Example 1**

Input:

5 --> Number of elements in array

2 4 5 8 9 --> Array elements

7 --> Target sum value

Output:

0 2

Since 0 and 2 are the indices where the numbers 2 and 5 which add up to 7 are seen

***SOLUTION STEPS FROM NEXT PAGE:***

**Write down at least 3 examples in the following format. Kindly stick to the format.**

**Suggestion:**

EXAMPLE#1

INPUT:

4

4 1 9 5

9

OUTPUT:

0 3

EXAMPLE#2

INPUT:

9

11 22 33 44 55 66 77 88 99

110

OUTPUT:

3 5

EXAMPLE#3

INPUT:

3

4 8 19

12

OUTPUT:

0 1

**Detail your problem understanding here**

**Suggestion:**

A list of integers and a target integer value will be provided as input.

Go through the list of integers and find 2 numbers such that they would add up to the target value specified.

**Does this problem follow a known algorithmic pattern or standard application of a data structure? If there are multiple approaches, which one would you choose and why? Write down your chosen approach in 2-3 sentences like you would explain to a 10 year old.**

This problem can be solved with brute force comparing all pairs of numbers.

If extra space can be used, the problem can be converted in a search problem which reduces the complexity to O(n)

Storing (specified target - currentNumber) in a HashTable and doing lookups makes the solution much simpler.

4

4 1 9 5

9

In this eg, (9-4=5), (9-1=8), (9-9=0) etc gets stored in HashTable. Before storing, a lookup for the value is done, the presence of which means (4,5) can be returned as the pair.

**Write the pseudocode here in plain English**

Create a Hash Map.

Insert each number from input array into the Hash Map with the value as key and it’s index as value.

Iterate through each number in the element array {

Find the complementary number for the current number that will add up to the target

Complementary number = (target - current number)

Lookup the Hash Map for the Complementary number.

If lookup succeeds {

Return the index of the current number and the index of the complimentary number looked up from the Hash Map.

}

}

In the above solution, we have iterated through the numbers twice. Once to populate the Hash Map and again to find the complementary number.

This can be optimized by having just one iteration which will populate the Hash Map as we go along checking for the complementary number.

**Can you specify a few boundary or edge cases here?**

**Edge cases**

EXAMPLE#1

INPUT:

5

1 2 3 2 1

2

OUTPUT

0 4

EXAMPLE#2

INPUT:

5

0 2 0 3 1

0

OUTPUT

0 2

**Write the functions you would create here**

HashMap<Integer, Integer> createHashOfNumbers(int numbers[ ])

#### Summary

Starting with a brief explanation of the problem statement followed by pseudocode and then implementing the solution helps you approach the problem in a systematic way. This methodology helps with easy as well as hard problems.

**Time Complexity: O(n)**

Since we traverse the list of n numbers only once with the hash table based solution.

**Space Complexity: O(n)**

The space needed depends on the number of elements to be stored in the hash, which could be n in the worst case..

#### Concepts

Concepts covered in this Module

* Array
* Hash Table

Similar problems

* <https://leetcode.com/problems/3sum/>
* <https://leetcode.com/problems/subarray-sum-equals-k/>
* <https://leetcode.com/problems/two-sum-less-than-k/>

References

* <https://www.tutorialspoint.com/data_structures_algorithms/hash_data_structure.htm>
* <https://www.geeksforgeeks.org/hashing-set-1-introduction/>
* <https://www.interviewcake.com/concept/java/hash-map>
* <https://www.geeksforgeeks.org/hashing-data-structure/>

SOLUTION:

APPROACH 1:

import java.io.\*;

import java.util.\*;

public class TwoSum {

// Implement your solution by completing the below function

public int[] twoSum(int[] nums, int target) {

for (int i = 0; i < nums.length; i++) {

for (int j = i + 1; j < nums.length; j++) {

if (nums[j] == target - nums[i]) {

return new int[] { i, j };

}

}

}

throw new IllegalArgumentException("No two sum solution");

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int[] numbers = new int[scanner.nextInt()];

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

numbers[i] = scanner.nextInt();

int target = scanner.nextInt();

scanner.close();

int[] complements = new TwoSum().twoSum(numbers, target);

System.out.print(complements[0] + " " + complements[1]);

}

}

**Complexity Analysis:**

* **Time Complexity:**
* **Space Complexity:**

public int[] twoSum(int[] nums, int target) {

Map<Integer, Integer> map = new HashMap<>();

for (int i = 0; i < nums.length; i++) {

map.put(nums[i], i);

}

for (int i = 0; i < nums.length; i++) {

int complement = target - nums[i];

if (map.containsKey(complement) && map.get(complement) != i) {

return new int[] { i, map.get(complement) };

}

}

throw new IllegalArgumentException("No two sum solution");

}