Glad you found the solution efficient! This approach is known as the **two-pass hash table approach** (in this case, using a set instead of a hash map). Here’s a breakdown of the approach, what it's called, and when it can be used:

**1. Name of the Approach:**

This solution uses a **hashing technique**, commonly referred to as the **set-based approach for pair-sum problems** or the **hash map approach**. It is a popular method for solving problems that involve looking for pairs or tracking elements based on some condition.

**2. How It Works:**

* **Set Lookup**: The essence of this approach is to use a set (or a dictionary) for fast lookups. Sets allow checking whether an element exists in constant time O(1).
* **Single Pass**: You make a single pass over the list of numbers, computing the complement (target - num) and checking if it’s already been encountered. This gives you an efficient way to find pairs without needing nested loops, which would take O(n^2) time.

**3. Scenarios Where This Approach Can Be Used:**

**a. Two Sum Problem (similar to what we did here):**

* **Problem**: Given a list of integers, find all unique pairs that add up to a specific target sum.
* **Application**: The set-based approach works perfectly for this scenario because of the quick lookups for complements.

**b. Finding Duplicates:**

* **Problem**: Check if a list contains any duplicate values.
* **How to Use**: As you iterate through the list, use a set to track the elements you've seen. If an element is already in the set, you’ve found a duplicate.

A black rectangular object with a black border

Description automatically generated

**c. Finding Missing or Extra Elements:**

* **Problem**: Given two lists, find elements that are in one list but not the other.
* **How to Use**: Store one list's elements in a set, then iterate over the other list to see if elements are missing or extra compared to the first set.

A screenshot of a computer

Description automatically generated

**d. Checking for Subarrays with a Given Sum:**

* **Problem**: Given an array of integers, find if there exists a subarray with a sum equal to zero (or any specific target sum).
* **How to Use**: By keeping a cumulative sum of elements as you iterate, you can check if the cumulative sum has been seen before (using a set). If it has, there’s a subarray with the target sum.

A screenshot of a computer

Description automatically generated

**e. Finding Longest Consecutive Subsequence:**

* **Problem**: Given an unsorted array of integers, find the length of the longest consecutive elements sequence.
* **How to Use**: Store the elements in a set, and then iterate through the set to find the longest sequence by checking if each number has a consecutive predecessor.

A screenshot of a computer program

Description automatically generated

**4. Why Use This Approach?**

This set-based approach is highly effective for:

* **Reducing Time Complexity**: The alternative to this approach is usually a brute-force nested loop, which takes O(n^2) time. The set-based method reduces this to O(n).
* **Handling Unique Elements Efficiently**: Sets are inherently designed to store only unique elements, which makes them perfect for problems where duplicate elements are a concern.
* **Fast Lookups**: Checking if an element exists in a set takes constant time O(1) on average, making it ideal for problems where you need to repeatedly check for the presence of a value.

**Summary:**

* **Name**: Set-based or Hash-based lookup approach.
* **Key Use Cases**: Pair-sum problems, finding duplicates, subarray sums, finding missing elements, and longest consecutive sequences.
* **Benefits**: Fast O(1) lookups and efficient O(n) solutions compared to O(n^2) brute-force methods.