Two Sum Solution Guide

Give revision notes for this DSA problem. public int[] twoSum(int[] nums, int target) {
    Map<Integer, Integer> map = new HashMap<>();
    for(int i = 0; i < nums.length; i++) {
        int complement = target - nums[i];
        if(map.containsKey(complement)) {
            return new int[]{map.get(complement), i};
        }
        map.put(nums[i], i);
    }
    throw new IllegalArgumentException("No solution found");
}

Revision Notes: Two Sum Problem

Problem Statement:

Find two numbers in an array such that they add up to a specific target. Return their indices as an array.

Key Points:

1. Algorithm Overview

- Use a **HashMap** to store each number and its index as you traverse the array.
- For each number, calculate its **complement** (`target - nums[i]`).
- Check if the complement is already in the HashMap:
    - If yes, return the indices of the current number and its complement.
    - Otherwise, add the current number and its index to the HashMap.

2. Code Explanation

java

```java
public int[] twoSum(int[] nums, int target) {
    // Create a HashMap to store numbers and their indices
    Map<Integer, Integer> map = new HashMap<>();

    // Loop through the array
    for (int i = 0; i < nums.length; i++) {
        int complement = target - nums[i]; // Calculate the complement
        if (map.containsKey(complement)) { // Check if complement exists in the map
            return new int[]{map.get(complement), i}; // Return indices if found
        }
        map.put(nums[i], i); // Add the number and its index to the map
    }

    // If no solution is found, throw an exception
    throw new IllegalArgumentException("No solution found");
}
```

Complexity Analysis

- **Time Complexity**: $O(n)$
  - Each lookup or insert operation in a HashMap takes $O(1)$, and we iterate through the array once.
- **Space Complexity**: $O(n)$
  - The HashMap stores at most n elements (where n is the size of the input array).

Edge Cases to Consider

1. **Input Array Length**:
   - Ensure the input array has at least two elements.
2. **Duplicate Elements**:
   - The solution handles duplicates correctly as long as they contribute to the target.

3. **No Valid Pair**:
    - If no solution exists, the exception is thrown.
4. **Negative Numbers**:
    - Negative values are supported as part of the complement calculation.

Important Concepts to Revise

1. **HashMap**:
    - Key-value pair operations: `put`, `get`, and `containsKey`.
2. **Complement Calculation**:
    - Subtraction and its role in identifying pairs.
3. **Time and Space Complexity**:
    - Importance of HashMap for reducing complexity to $O(n)$ compared to a nested loop solution ($O(n^2)$).

Common Interview Questions

1. **Why use a HashMap instead of a nested loop?**
    - A HashMap enables constant-time lookups, making the algorithm more efficient.
2. **What happens if there are multiple valid pairs?**
    - This implementation returns the first pair found due to the sequential traversal of the array.
3. **Can this solution handle sorted arrays efficiently?**
    - While this solution works on any array, a two-pointer approach is more suitable for sorted arrays.

Practice Problems for Mastery

1. Find three numbers in an array that sum to a target (3-Sum problem).
2. Find all unique pairs in an array that sum to a target.
3. Find two numbers in a sorted array that sum to a target (use two-pointer technique).