Two Sum - Revision Notes

Problem Statement:

Given an array `nums` and an integer `target`, return **indices** of the two numbers such that they add up to `target`.

- **Input:** `nums = [2,7,11,15], target = 9`
- **Output:** `[0,1]` (Since `2 + 7 = 9`)
- **Must be solved in O(n) time complexity.**

Approach: Using HashMap

1️⃣ **Create a HashMap:** Store numbers as **keys** and their **indices** as values. 2️⃣ **Traverse the array:** For each `nums[i]`, compute `complement = target - nums[i]`. 3️⃣ **Check if complement exists in HashMap:** If yes, return `[index of complement, i]`. 4️⃣ **Otherwise, store `` in HashMap:** So we can find it later.

Final Correct Code:

```
import java.util.*;

class Solution {
    public int[] twoSum(int[] nums, int target) {
        HashMap<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);
        }

        return new int[]{}; // No solution found
```

```
    }
}
```

Your Mistakes & Fixes:

❌ **Mistake:** Used `containsValue()` instead of `containsKey()`. ✅ **Fix:** `containsKey(complement)` is correct because we store numbers as keys, not indices.

❌ **Mistake:** Used `j++` for index storage instead of directly storing `i`. ✅ **Fix:** Store `i` as value in the HashMap (`map.put(nums[i], i)`).

❌ **Mistake:** Returned `{0}` instead of `new int[]{}` when no solution was found. ✅ **Fix:** Use `return new int[]{};` for proper array return.

Key Takeaways:

- **HashMap is best for O(n) lookup time.**
- **Keys = numbers, Values = indices.**
- **Always check for ``.**

🚀 **You solved this problem! Keep going!** 🔥