**Contains Duplicate - Revision Notes**

## Problem Statement

- Given an integer array `nums`, return `true` if any value appears **at least twice** in the array, and `false` if all elements are unique.

## Your Approach (Using HashMap)

## Idea:

- We use a **HashMap** to store elements as keys and their indices as values.
- If an element is already present in the HashMap (`containsKey()`), it means we found a duplicate.
- Otherwise, we store the element in the HashMap.

## Code:

```
import java.util.*;

class Solution {
    public boolean containsDuplicate(int[] nums) {
        HashMap<Integer,Integer> big = new HashMap<>();
        boolean ans = false;

        for(int i = 0; i < nums.length; i++) {
            if(big.containsKey(nums[i])) {
                ans = true;
            } else {
                big.put(nums[i], i);
            }
        }
        return ans;
    }
}
```

## Time Complexity:

- **Best/Average Case:** `O(n)` (Each lookup and insertion in HashMap is `O(1)`, and we iterate through `n` elements)
- **Worst Case:** `O(n)` (If no duplicates exist, we traverse all `n` elements)

### Space Complexity:

- `O(n)` (In the worst case, we store all `n` elements in the HashMap)

---

## Optimized Approach (Using HashSet)

### Idea:

- Instead of using a HashMap, we can use a **HashSet**, which only stores unique elements.
- If we encounter a duplicate, we return `true` immediately.

### Optimized Code:

```
import java.util.*;

class Solution {
    public boolean containsDuplicate(int[] nums) {
        HashSet<Integer> set = new HashSet<>();

        for(int num : nums) {
            if(set.contains(num)) return true;
            set.add(num);
        }
        return false;
    }
}
```

### Time Complexity:

- **O(n)** (Each insertion and lookup in HashSet is `O(1)` on average)

### Space Complexity:

- **O(n)** (Stores at most `n` elements in the worst case)

---

## Mistakes & Learning Points

1. **Incorrectly using `containsValue()` in HashMap:**
    - Your initial approach checked for the complement using `.containsValue()`, but we should use `.containsKey()`.
2. **Using HashSet for better performance:**
    - Since we only need to check for duplicates (not store indices), **HashSet is more memory-efficient**.
3. **Early Return Optimization:**
    - We can return `true` **immediately** when a duplicate is found instead of continuing the loop.

---

## Final Takeaways

✅ **Best Approach:** Use `HashSet` for simplicity and efficiency.

✅ **Time Complexity:** `O(n)`, which is optimal for this problem.

✅ **Avoid storing unnecessary data** like indices when they are not needed.

---

**End of Notes 🚀**