**217. Contains Duplicate -------------------------------------------------------------------------------------**

<https://leetcode.com/problems/contains-duplicate/>

1. **Listen -------------------------------------------------------------------------------------------------------**

**Problem Statement:**

Given an integer array nums, return true if any value appears **at least twice** in the array, and return false if every element is distinct.

**Input:**

integer array nums

**Goal:**

Attempt to find if all elements in an array are unique

**Return:**

return true if any value appears **at least twice** in the array

return false if every element is distinct

**Constraints:**

* 1 <= nums.length <= 105
* -109 <= nums[i] <= 109

**Test Cases:**

* All elements are unique
* There is a single duplicate element
* There are multiple duplicate elements

1. **Examples ---------------------------------------------------------------------------------------------------**

**Example 1: -----------------------------------------------------------------------------------------------------**

**Input:** nums = [1,2,3,1]

**Output:** true

1. **Brute Force**

**Solution 1: ------------------------------------------------------------------------------------------------------**

**Time = O(N^2)**

**Space = O(1)**

**Intuition**

The naïve approach would be to, for every element, check every other element in the array to see if there is a duplicate of itself.

Therefore, for each element, there would be a scan over all N elements in the array. Since we do this N times, the runtime would be O(N^2) with no extra space used.

**Solution 2: ------------------------------------------------------------------------------------------------------**

**Time = O(NlogN)**

**Space = O(1)**

**Intuition**

We can sort the array and check neighboring elements.

The loop needs to start at index 1 just in case the size of the input array is == 1.

We check the current index against the previous index for equality.

If two elements are equal, return true.

If we have finished the loop, that means there are no duplicate elements, return false.

The O(NlogN) sorting takes priority over the O(N) bound of the loop traversal over N elements.

Therefore, runtime would be O(NlogN) with no extra space used.

1. **Optimize**

**Solution 3: Time = O(N), Space = O(N)**

We can walk through the array, adding elements to a set as we go. Recall that a set only allows unique values to be added to it.

We can have a check that says, “if we have seen this number before, then this is the duplicate element”. Then we can return true from the function.

If we go through the entire array and find that we never run into this problem, then we can return false, because duplicate elements were never found.

Walking through an array of N elements requires O(N) runtime.

There can be up to N elements in the set if there are no duplicates, therefore O(N) space.

1. **Implement**

**Solution 1:**

public boolean containsDuplicate(int[] nums) {

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

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

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

return true;

}

}

}

return false;

}

Notice how we start at i+1 for the inner loop.

We don’t need to start j at 0 because we have already tested all the previous elements < i against the current and future elements, therefore, doing it again would just be redundant.

**Solution 2:**

public boolean containsDuplicate(int[] nums) {

Array.sort(nums);

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

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

return true;

}

}

return false;

}

Notice how if nums.length == 1, the loop will never execute and we will go straight to returning false.

**Solution 3:**

public boolean containsDuplicate(int[] nums) {

HashSet<Integer> set = new HashSet<>();

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

if(set.contains(nums[i])) {

return true;

}

set.add(nums[i];

}

return false;

}

This is a classic example of trading time for space.

We significantly cut down on runtime by a factor of n, but we can use up to n space.