Problem: 219. Contains Duplicate II

Difficulty: Easy

Problem Statement:

Given an integer array `nums` and an integer `k`, return `true` if there are two distinct indices `i` and `j` in the array such that `nums[i] == nums[j]` and `abs(i - j) <= k`. Otherwise, return `false`.

Approach:

1. Use a `HashMap<Integer, Integer>` to store the last seen index of each element.
2. Iterate through the array:
    - If the element already exists in the map, check if the difference between the current index and the stored index is less than or equal to `k`.
    - If the condition is met, return `true`.
    - Otherwise, update the stored index of the element.
3. If no such pair is found, return `false`.

Code Implementation:

```
class Solution {
    public boolean containsNearbyDuplicate(int[] nums, int k) {
        HashMap<Integer, Integer> big = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            if (big.containsKey(nums[i])) {
                if (Math.abs(big.get(nums[i]) - i) <= k) {
                    return true;
                }
            }
            big.put(nums[i], i); // Update the last seen index of nums[i]
        }
        return false;
    }
}
```

Example Walkthrough:

Input: `nums = [1,0,1,1]`, `k = 1`

Execution:

- `1` is added at index `0`.
- `0` is added at index `1`.
- `1` is seen again at index `2`. The difference `2 - 0 = 2`, which is **not** ≤ `k`.
- `1` is seen again at index `3`. The difference `3 - 2 = 1`, which **is** ≤ `k`, so return `true`.

Output: `true`

Complexity Analysis:

- **Time Complexity:** `O(n)`, as we traverse the array once.
- **Space Complexity:** `O(n)`, as we store at most `n` elements in the hashmap.

Edge Cases Considered:

- `k` is larger than the array length.
- The array has only one element.
- The array has no duplicate elements.
- The same number appears multiple times but at distances greater than `k`.

Final Notes:

- This problem is a variation of the classic **duplicate detection** problem but with an additional constraint.
- The hashmap helps in keeping track of the most recent index of each number efficiently.
- The approach ensures optimal time complexity without redundant checks.