LeetCode 215. Kth Largest Element in an Array

1. Problem Title & Link

- **215. Kth Largest Element in an Array**
- https://leetcode.com/problems/kth-largest-element-in-an-array/

2. Problem Statement (Short Summary)

Given an integer array nums and an integer k, return the kth largest element in the array.

⚠️ Note: It's the **kth largest element**, not the kth distinct element.

3. Examples (Input → Output)

Input: nums = [3,2,1,5,6,4], k = 2

Output: 5

Input: nums = [3,2,3,1,2,4,5,5,6], k = 4

Output: 4

4. Constraints

- 1 <= k <= nums.length <= 10^5
- -10^4 <= nums[i] <= 10^4

5. Thought Process (Step by Step)

- **Brute Force**: Sort array → return nums[n-k]. (O(n log n))
- **Better**: Use **Heap** (priority queue).
  - Min-Heap of size k.
  - Keep only k largest elements → root is kth largest. (O(n log k))
- **Optimal**: Quickselect (similar to QuickSort partition). Avg O(n), worst O(n²).

For students → **Heap solution** is easier to understand, but we'll also mention sort and Quickselect.

6. Pseudocode (Heap Approach)

```
function findKthLargest(nums, k):
    create minHeap
    for num in nums:
        push num into minHeap
        if size of heap > k:
            pop smallest element
```

```
    return top of heap
```

7. Code Implementation

✅ **Python (Heap)**

```
import heapq

class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        minHeap = []
        for num in nums:
            heapq.heappush(minHeap, num)
            if len(minHeap) > k:
                heapq.heappop(minHeap)
        return minHeap[0]
```

✅ **Java (Heap)**

```
class Solution {
    public int findKthLargest(int[] nums, int k) {
                    PriorityQueue<Integer> minHeap = new
PriorityQueue<>();
        for (int num : nums) {
            minHeap.offer(num);
            if (minHeap.size() > k) {
                minHeap.poll();
            }
        }
        return minHeap.peek();
    }
}
```

8. Time & Space Complexity Analysis

- **Heap Approach**
  - Time: O(n log k)
  - Space: O(k)

- **Sorting Approach**
  - Time: O(n log n)
  - Space: O(1)
- **Quickselect**
  - Avg: O(n)
  - Worst: $O(n^2)$

9. Common Mistakes / Edge Cases

- Returning nums[k-1] after sorting ascending instead of nums[n-k].
- Misunderstanding “kth largest” vs “kth smallest”.
- Heap approach → popping more than needed.

10. Variations / Follow-Ups

- Find kth **smallest** element.
- Maintain running kth largest in a data stream (LeetCode 703).
- Handle duplicate values carefully.

11. Dry Run (Heap Approach)

👉 Input: nums = [3, 2, 1, 5, 6, 4], k = 2

Steps:

- Start with empty heap.

1. Push 3 → heap = [3]
2. Push 2 → heap = [2, 3]
3. Push 1 → heap = [1, 3, 2] → size > 2 → pop → heap = [2, 3]
4. Push 5 → heap = [2, 3, 5] → size > 2 → pop → heap = [3, 5]
5. Push 6 → heap = [3, 5, 6] → size > 2 → pop → heap = [5, 6]
6. Push 4 → heap = [4, 6, 5] → size > 2 → pop → heap = [5, 6]

✅ Final Heap = [5, 6]

✅ Top = 5 → kth largest element