**1. Remove Duplicates from a Sorted Array**

**Q1:** Why does the “remove duplicates” algorithm work only correctly on *sorted* arrays?  
**A:**  
The logic used compares **only adjacent elements** (arr[i] and arr[i+1]).  
In a sorted array, duplicates always occur **next to each other**, so we can remove them by skipping consecutive equal elements.  
If the array is unsorted, duplicates could be far apart and this method would miss them.  
**Example:**  
Sorted → {10, 20, 20, 30} → Works ✅  
Unsorted → {20, 10, 20, 30} → Fails ❌

**Q2:** Why do we use a **temporary array** instead of directly modifying the original?  
**A:**  
Because overwriting while iterating can cause loss of data or incorrect comparisons.  
The temp array stores unique elements first, then we copy them back to the original array or return it.  
It also avoids the complexity of shifting elements repeatedly.

**Q3:** What is the time complexity of removing duplicates from a sorted array?  
**A:**  
We only make **one pass** over the array → **O(n)**.  
Copying back also takes **O(n)**, so total time complexity is **O(n)**.  
Space complexity is **O(n)** for the temp array (can be optimized to O(1) using in-place logic).

**2. Merge Two Sorted Arrays**

**Q4:** Why does the merge process use three pointers (i, j, k)?  
**A:**  
i → tracks position in first array  
j → tracks position in second array  
k → tracks position in merged array  
We need all three because we are merging into a **new array** while simultaneously comparing two input arrays.

**Q5:** What happens if one array is exhausted before the other?  
**A:**  
We directly copy the **remaining elements** from the non-exhausted array into the merged array because they are already sorted and larger than the last inserted element.  
This is why we have two extra while loops after the main merging loop.

**Q6:** How is merging two sorted arrays different from sorting one unsorted array?  
**A:**

* Merging → **O(n + m)** (linear in total elements)
* Sorting → **O((n+m) log(n+m))** (slower for large arrays)  
  That’s why merge is faster when arrays are already sorted.

**3. Count Occurrences of an Element**

**Q7:** How do we count occurrences efficiently?  
**A:**  
We iterate once and use a counter that increments when the target element matches the current element.  
**Example:**  
Array = {1, 2, 2, 3, 2}, target = 2 → Count = 3.

**Q8:** How to optimize counting if the array is sorted?  
**A:**  
Instead of scanning all elements, we can use **binary search** to find:

* First occurrence index
* Last occurrence index  
  Count = (lastIndex - firstIndex + 1) → **O(log n)** time.

**Q9:** Why is it important to initialize count to 0?  
**A:**  
Because we’re accumulating results from scratch — any leftover value would lead to incorrect results.

**4. Check if Array is Palindrome**

**Q10:** Why do we use two pointers (start and end) instead of reversing the array?  
**A:**

* **Two-pointer method** → Saves space (O(1)) and compares elements directly.
* **Reversing method** → Needs extra memory or modifies the original array.  
  Two-pointer is faster and safer.

**Q11:** What happens if we start assuming the array is **not** a palindrome?  
**A:**  
We can initialize isPalindrome = false and then set it to true only if **all** comparisons pass.  
This forces a full check rather than early assumption.

**Q12:** What is the time complexity?  
**A:**  
We only check each element once (half the array) → **O(n/2) = O(n)**.

**5. Move All Zeros to End**

**Q13:** Why do we use an index pointer in this problem?  
**A:**  
The index tracks the **next position to place a non-zero element**.  
It avoids multiple swaps and keeps non-zero elements in order.

**Q14:** Why do we need the second while loop?  
**A:**  
After moving all non-zero elements forward, there may still be leftover spaces in the array.  
We fill these with zeros using the second loop.

**Q15:** How can we make this in-place without extra space?  
**A:**  
We already do it in-place here — we only move elements inside the same array without using an additional array.