**1. Remove Duplicates**

1. Write a program to remove duplicates from a **sorted** integer array without using any extra array.
2. Modify the above program to handle **unsorted** arrays.
3. Remove duplicates from a sorted array **in-place** and return the new length of the unique array.
4. Write a function to return a **Set** containing unique elements from an array.
5. Remove duplicates from a **string array** and print the result.
6. Remove duplicates from a sorted array but **keep only the first occurrence** of each element.
7. Remove duplicates and return the **frequency** of each unique element.
8. Remove duplicates from an array without using Collections or Arrays helper methods.
9. Remove duplicates from a sorted array and shift the elements to **fill empty spaces** with -1.
10. Remove duplicates from a list of **characters**.

**2. Merge Two Arrays**

1. Merge two **sorted** arrays into one sorted array without using Arrays.sort().
2. Merge two **unsorted** arrays and sort the final array.
3. Merge two arrays and remove duplicates from the result.
4. Merge two arrays and find the **median** of the merged array.
5. Merge two sorted arrays **in-place** (assume one array has extra space).
6. Merge two arrays and find **common elements**.
7. Merge two arrays and display the **maximum and minimum** values.
8. Merge two arrays by **alternating elements** from each.
9. Merge two sorted arrays into **descending order**.
10. Merge two arrays into a third array without using System.arraycopy().

**3. Count Occurrences of an Element**

1. Count how many times a given element occurs in an integer array.
2. Count occurrences of all elements and print in (element, count) format.
3. Count occurrences in a **string array**.
4. Find the element with the **maximum occurrences**.
5. Find the element with the **minimum occurrences**.
6. Count occurrences in an unsorted array without sorting.
7. Count occurrences using a **HashMap**.
8. Count occurrences and remove duplicates from the result.
9. Check if any element occurs **more than n/2 times**.
10. Count the frequency of **characters** in a string.

**4. Check if Array is Palindrome**

1. Check if a given integer array is a palindrome.
2. Modify the above to check if a **string array** is a palindrome.
3. Check if a number array is palindrome ignoring **zeros**.
4. Check if a sentence stored in a char array is palindrome **ignoring spaces**.
5. Check if an array is palindrome using **recursion**.
6. Find all **subarrays** of length 3 that are palindromes.
7. Check if an array is palindrome ignoring **case sensitivity** (for strings).
8. Check if an array is palindrome when reversed **in-place**.
9. Check if a linked list converted from an array is palindrome.
10. Check if an array becomes palindrome after **at most one change**.

**5. Move All Zeros to the End**

1. Move all zeros to the end while keeping the order of non-zero elements.
2. Move all zeros to the start while keeping the order of non-zero elements.
3. Move all zeros to the end without using an extra array.
4. Move all zeros to the end and replace them with -1.
5. Move all zeros to the end using only **one pass** through the array.
6. Count how many zeros were moved to the end.
7. Move all zeros to the end in a **string array**.
8. Move all even numbers to the end (zero included) keeping order of odds.
9. Move all zeros to the end in a **2D array** row-wise.
10. Move all zeros to the end in-place using **swap method**