**1. Remove Duplicates from an Array**

**Definition:**  
Eliminate repeated elements so that only unique elements remain.  
*(Usually easiest if the array is sorted first.)*

**Example:**  
Input → [10, 20, 20, 30, 30, 30, 40]  
Output → [10, 20, 30, 40]

**Java Code (for sorted arrays):**

import java.util.Arrays;

public class RemoveDuplicates {

public static void main(String[] args) {

int[] arr = {10, 20, 20, 30, 30, 30, 40};

int n = arr.length;

int[] temp = new int[n];

int j = 0;

for (int i = 0; i < n - 1; i++) {

if (arr[i] != arr[i + 1]) {

temp[j++] = arr[i];

}

}

temp[j++] = arr[n - 1];

// Copy unique elements back to original

int[] uniqueArr = Arrays.copyOf(temp, j);

System.out.println(Arrays.toString(uniqueArr));

}

}

| **i** | **arr[i]** | **arr[i+1]** | **arr[i] != arr[i+1]?** | **Action** | **temp after write** | **j** |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | 10 | 20 | true | temp[j]=10 | [10, \_, \_, \_, \_, \_, \_] | 1 |
| 1 | 20 | 20 | false | — | [10, \_, \_, \_, \_, \_, \_] | 1 |
| 2 | 20 | 30 | true | temp[j]=20 | [10, 20, \_, \_, \_, \_, \_] | 2 |
| 3 | 30 | 30 | false | — | [10, 20, \_, \_, \_, \_, \_] | 2 |
| 4 | 30 | 30 | false | — | [10, 20, \_, \_, \_, \_, \_] | 2 |
| 5 | 30 | 40 | true | temp[j]=30 | [10, 20, 30, \_, \_, \_, \_] | 3 |

**Practice Problem:**  
Remove duplicates from [1, 1, 2, 2, 3, 4, 4, 5].

**2. Merge Two Arrays**

**Definition:**  
Combine two arrays into one array.

**Example:**  
Array1 → [1, 3, 5]  
Array2 → [2, 4, 6]  
Merged → [1, 3, 5, 2, 4, 6]  
*(If both arrays are sorted, you can merge them into a sorted array.)*

**Java Code (simple merge):**

import java.util.Arrays;

public class MergeArrays {

public static void main(String[] args) {

int[] arr1 = {1, 3, 5};

int[] arr2 = {2, 4, 6};

int[] merged = new int[arr1.length + arr2.length];

int index = 0;

for (int num : arr1) {

merged[index++] = num;

}

for (int num : arr2) {

merged[index++] = num;

}

System.out.println(Arrays.toString(merged));

}

}

**Java Code (merge sorted arrays into one sorted array):**

import java.util.Arrays;

public class MergeSortedArrays {

public static void main(String[] args) {

int[] arr1 = {1, 3, 5,7};

int[] arr2 = {2, 4, 6};

int[] merged = new int[arr1.length + arr2.length];

int i = 0, j = 0, k = 0;

while (i < arr1.length && j < arr2.length) {

if (arr1[i] < arr2[j]) {

merged[k++] = arr1[i++];

} else {

merged[k++] = arr2[j++];

}

}

while (i < arr1.length) {

merged[k++] = arr1[i++];

}

while (j < arr2.length) {

merged[k++] = arr2[j++];

}

System.out.println(Arrays.toString(merged));

}

}

| **Step** | **i** | **j** | **Compare** | **Chosen value** | **merged[]** | **k after** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 1 < 2 | 1 from arr1 | [1] | 1 |
| 2 | 1 | 0 | 3 > 2 | 2 from arr2 | [1, 2] | 2 |
| 3 | 1 | 1 | 3 < 4 | 3 from arr1 | [1, 2, 3] | 3 |
| 4 | 2 | 1 | 5 > 4 | 4 from arr2 | [1, 2, 3, 4] | 4 |
| 5 | 2 | 2 | 5 < 6 | 5 from arr1 | [1, 2, 3, 4, 5] | 5 |

## Remaining Elements — Copy Leftover

while (i < arr1.length) {

merged[k++] = arr1[i++];

}

* Runs if there are any **leftover** elements in arr1 (in this case, none)

while (j < arr2.length) {

merged[k++] = arr2[j++];

}

* Runs if there are any **leftover** elements in arr2.
* Here, j = 2 → inserts 6 into merged.

**Practice Problem:**  
Merge {5, 10, 15} and {2, 4, 6} into a single **sorted** array.

**3. Count Occurrences of an Element**

**Definition:**  
Count how many times a given value appears in an array.

**Example:**  
Array → [10, 20, 20, 30, 20, 40]  
Element → 20 → Count = 3

**Java Code:**

public class CountOccurrences {

public static void main(String[] args) {

int[] arr = {10, 20, 20, 30, 20, 40};

int target = 20;

int count = 0;

for (int num : arr) {

if (num == target) {

count++;

}

}

System.out.println("Count of " + target + " = " + count);

}

}

**Practice:** Count how many times 5 appears in {5, 3, 5, 2, 5, 1}.

**4. Check if Array is Palindrome**

**Definition:**  
An array is a palindrome if it reads the same from start to end and end to start.

**Example:**  
[1, 2, 3, 2, 1] → Palindrome ✅  
[1, 2, 3, 4, 5] → Not palindrome ❌

**Java Code:**

public class PalindromeArray {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 2, 1};

boolean isPalindrome = true;

int start = 0, end = arr.length - 1;

while (start < end) {

if (arr[start] != arr[end]) {

isPalindrome = false;

break;

}

start++;

end--;

}

System.out.println(isPalindrome ? "Palindrome" : "Not Palindrome");

}

}

**Practice:** Check if {5, 10, 15, 10, 5} is a palindrome.

public class PalindromeArray {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 2, 1};

boolean isPalindrome = false; // Assume false initially

int start = 0, end = arr.length - 1;

// We will check first, then set true only if all match

while (start < end) {

if (arr[start] != arr[end]) {

isPalindrome = false; // Mismatch found, keep false

break;

}

start++;

end--;

// If we reached here without breaking, so far it's matching

if (start >= end) {

isPalindrome = true;

}

}

System.out.println(isPalindrome ? "Palindrome" : "Not Palindrome");

}

}

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

**Definition:**  
Shift all 0 elements to the end, keeping the order of non-zero elements.

**Example:**  
[0, 1, 0, 3, 12] → [1, 3, 12, 0, 0]

**Java Code:**

import java.util.Arrays;

public class MoveZeros {

public static void main(String[] args) {

int[] arr = {0, 1, 0, 3, 12};

int index = 0;

for (int num : arr) {

if (num != 0) {

arr[index++] = num;

}

}

while (index < arr.length) {

arr[index++] = 0;

}

System.out.println(Arrays.toString(arr));

}

}

| **num** | **index** | **arr after operation** |
| --- | --- | --- |
| 0 | 0 | {0, 1, 0, 3, 12} (skip zero) |
| 1 | 0→1 | {1, 1, 0, 3, 12} |
| 0 | 1 | skip |
| 3 | 1→2 | {1, 3, 0, 3, 12} |
| 12 | 2→3 | {1, 3, 12, 3, 12} |

Example:  
After non-zero placement → {1, 3, 12, 3, 12}, index = 3  
Fill → {1, 3, 12, 0, 0} ✅

**Practice:** Move zeros to the end in {1, 0, 2, 0, 3, 0, 4}.