1. Bubble Sort - Sort Student Marks

Problem Statement:
A school maintains student marks in an array. Implement **Bubble Sort** to sort the student marks in **ascending order**.

Hint:

- Traverse through the array multiple times.
- Compare adjacent elements and swap if needed.
- Repeat the process until no swaps are required.

```
import java.util.*;

public class BubbleSort {

    public static void bubbleSort(int arr[]) {
        for (int i = 0; i < arr.length - 1; i++) {
            boolean isSwapped = false;
            for (int j = 0; j < arr.length - 1 - i; j++) {
                if (arr[j+1] < arr[j]) {
                    int temp = arr[j+1];
                    arr[j+1] = arr[j];
                    arr[j] = temp;
                    isSwapped = true;
                }
            }

            if (!isSwapped) {
                break;
            }
        }
    }

    public static void printArray(int arr[]) {
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter size of array:");
        int n = sc.nextInt();

        int arr[] = new int[n];
        System.out.println("Enter elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        // int arr[] = { 3, 6, 7, 0, 2 }; // demo array for check
        System.out.println("Array before sorting:");
        printArray(arr);

        bubbleSort(arr);

        System.out.println("Array after sorting:");
        printArray(arr);

        sc.close();

    }
}
```

---

2. Insertion Sort - Sort Employee IDs

Problem Statement:
A company stores **employee IDs** in an unsorted array. Implement **Insertion Sort** to sort the employee IDs in **ascending order**.

Hint:

- Divide the array into sorted and unsorted parts.
- Pick an element from the unsorted part and insert it into its correct position in the sorted part.
- Repeat for all elements.

```
import java.util.*;

public class InsertionSort {

    public static void insertionSort(int arr[]) {
        for(int i=1;i<arr.length;i++){
            int key = arr[i];
            int j = i - 1;

            while (j >= 0 && arr[j] > key) {
                arr[j + 1] = arr[j];
                j--;
            }
            arr[j + 1] = key;
        }
    }

    public static void printArray(int arr[]) {
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter size of array:");
        int n = sc.nextInt();

        int arr[] = new int[n];
```

```
        System.out.println("Enter elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        // int arr[] = { 3, 2 }; // demo array for check
        System.out.println("Array before sorting:");
        printArray(arr);

        insertionSort(arr);

        System.out.println("Array after sorting:");
        printArray(arr);

        sc.close();

    }
}
```

---

3. Merge Sort - Sort an Array of Book Prices

Problem Statement:
A bookstore maintains a list of book prices in an array. Implement **Merge Sort** to sort the prices **in ascending order**.

Hint:

- **Divide** the array into two halves recursively.
- **Sort** both halves individually.
- **Merge** the sorted halves by comparing elements.

```
import java.util.*;

public class MergeSort {

    public static void mergeSort(int arr[], int left, int right) {
        if (left < right) {
            int mid = left + (right - left) / 2;
            mergeSort(arr, left, mid);
            mergeSort(arr, mid + 1, right);
            merge(arr, left, mid, right);
        }
    }

    public static void merge(int arr[], int left, int mid, int right) {
        int n1 = mid - left + 1;
        int n2 = right - mid;
        int leftArr[] = new int[n1];
        int rightArr[] = new int[n2];

        for (int i = 0; i < n1; i++)
            leftArr[i] = arr[left + i];
        for (int j = 0; j < n2; j++)
            rightArr[j] = arr[mid + 1 + j];

        int i = 0, j = 0, k = left;
        while (i < n1 && j < n2) {
            if (leftArr[i] <= rightArr[j])
                arr[k++] = leftArr[i++];
            else
                arr[k++] = rightArr[j++];
        }
```

```
        while (i < n1)
            arr[k++] = leftArr[i++];
        while (j < n2)
            arr[k++] = rightArr[j++];
    }

    public static void printArray(int arr[]) {
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter size of array:");
        int n = sc.nextInt();

        int arr[] = new int[n];
        System.out.println("Enter elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        // int arr[] = { 3, 2, 5, 1, 9, 5, 0, 9}; // demo array for check
        System.out.println("Array before sorting:");
        printArray(arr);

        mergeSort(arr, 0 ,arr.length-1);

        System.out.println("Array after sorting:");
        printArray(arr);

        sc.close();

    }
}
```

4. Quick Sort - Sort Product Prices

Problem Statement:
An e-commerce company wants to display product prices in **ascending order**. Implement **Quick Sort** to sort the product prices.

Hint:

- Pick a **pivot** element (first, last, or random).
- **Partition** the array such that elements smaller than the pivot are on the left and larger ones are on the right.
- Recursively apply Quick Sort on left and right partitions.

```
import java.util.*;

public class QuickSort {

    public static void quickSort(int arr[], int low, int high) {
        if (low < high) {
            int pi = partition(arr, low, high);
            quickSort(arr, low, pi - 1);
            quickSort(arr, pi + 1, high);
        }
    }

    public static int partition(int arr[], int low, int high) {
        int pivot = arr[high], i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;
        return i + 1;
    }

    public static void printArray(int arr[]) {
        for (int i = 0; i < arr.length; i++) {
```

```
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter size of array:");
        int n = sc.nextInt();

        int arr[] = new int[n];
        System.out.println("Enter elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        // int arr[] = { 3, 2, 6, 7, 0, 1, 9 }; // demo array for check
        System.out.println("Array before sorting:");
        printArray(arr);

        quickSort(arr, 0, arr.length-1);

        System.out.println("Array after sorting:");
        printArray(arr);

        sc.close();

    }

}
```

5. Selection Sort - Sort Exam Scores

Problem Statement:
A university needs to sort students' **exam scores** in ascending order. Implement **Selection Sort** to achieve this.

Hint:

- Find the **minimum element** in the array.
- Swap it with the first unsorted element.
- Repeat the process for the remaining elements.

```
import java.util.*;

public class SelectionSort {

    public static void selectionSort(int arr[]) {
        for (int i = 0; i < arr.length; i++) {
            int min = i;
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[j] < arr[min]) {
                    min = j;
                }
            }
            if (min != i) {
                int temp = arr[i];
                arr[i] = arr[min];
                arr[min] = temp;
            }
        }
    }

    public static void printArray(int arr[]) {
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter size of array:");
        int n = sc.nextInt();
```

```
        int arr[] = new int[n];
        System.out.println("Enter elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        // int arr[] = { 3, 2, 5, 1, 9, 5, 0, 9}; // demo array for check
        System.out.println("Array before sorting:");
        printArray(arr);

        selectionSort(arr);

        System.out.println("Array after sorting:");
        printArray(arr);

        sc.close();

    }

}
```

6. Heap Sort - Sort Job Applicants by Salary

Problem Statement:
A company receives job applications with different **expected salary demands**. Implement **Heap Sort** to sort these salary demands in **ascending order**.

Hint:

- Build a **Max Heap** from the array.
- Extract the largest element (root) and place it at the end.
- Reheapify the remaining elements and repeat until sorted.

```
import java.util.*;

public class HeapSort {

    public static void heapSort(int arr[]) {
        int n = arr.length;
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(arr, n, i);
        for (int i = n - 1; i > 0; i--) {
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;
            heapify(arr, i, 0);
        }
    }

    public static void heapify(int arr[], int n, int i) {
        int largest = i, left = 2 * i + 1, right = 2 * i + 2;
        if (left < n && arr[left] > arr[largest])
            largest = left;
        if (right < n && arr[right] > arr[largest])
            largest = right;
        if (largest != i) {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;
            heapify(arr, n, largest);
        }
    }

    public static void printArray(int arr[]) {
```

```
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter size of array:");
        int n = sc.nextInt();

        int arr[] = new int[n];
        System.out.println("Enter elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        // int arr[] = { 3, 2, 6, 7, 0, 1, 9 }; // demo array for check
        System.out.println("Array before sorting:");
        printArray(arr);

        heapSort(arr);

        System.out.println("Array after sorting:");
        printArray(arr);

        sc.close();

    }

}
```

7. Counting Sort - Sort Student Ages

Problem Statement:
A school collects students' **ages** (ranging from 10 to 18) and wants them sorted. Implement **Counting Sort** for this task.

Hint:

- Create a **count array** to store the frequency of each age.
- Compute cumulative frequencies to determine positions.
- Place elements in their correct positions in the output array.

```
import java.util.*;

public class CountingSort {

    public static void countingSort(int arr[]) {
        int n = arr.length;
        int max = 0;

        for (int i = 0; i < n; i++) {
            max = Math.max(max, arr[i]);
        }

        int[] freqArray = new int[max + 1];

        for (int i = 0; i < n; i++) {
            freqArray[arr[i]]++;
        }

        for (int i = 1; i <= max; i++) {
            freqArray[i] += freqArray[i - 1];
        }

        int[] ans = new int[n];

        for (int i = n - 1; i >= 0; i--) {
            ans[freqArray[arr[i]] - 1] = arr[i];
            freqArray[arr[i]]--;
        }

        for (int i = 0; i < n; i++) {
            arr[i] = ans[i];
```

```
        }
    }

    public static void printArray(int arr[]) {
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter size of array:");
        int n = sc.nextInt();

        int arr[] = new int[n];
        System.out.println("Enter elements:");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        // int arr[] = { 3, 2, 5, 1, 9, 5, 0, 9}; // demo array for check
        System.out.println("Array before sorting:");
        printArray(arr);

        countingSort(arr);

        System.out.println("Array after sorting:");
        printArray(arr);

        sc.close();
    }
}
```