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.

Sol:

```
public class BubbleSort {

    public static void bubbleSort(int[] marks) {

        int n = marks.length;

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

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

                if (marks[j] > marks[j + 1]) {

                    int temp = marks[j];

                    marks[j] = marks[j + 1];

                    marks[j + 1] = temp;

                }

            }

        }

    }

    public static void main(String[] args) {

        int[] marks = {85, 70, 90, 60, 75};
```

```
        bubbleSort(marks);

        for (int mark : marks) {

            System.out.print(mark + " ");

        }

    }

}
```

---

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.

Sol:

```
public class InsertionSort {

    public static void insertionSort(int[] ids) {

        int n = ids.length;

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

            int key = ids[i];

            int j = i - 1;

            while (j >= 0 && ids[j] > key) {

                ids[j + 1] = ids[j];
```

```
            j--;
        }
        ids[j + 1] = key;
    }
  }
  public static void main(String[] args) {
    int[] employeeIds = {105, 103, 108, 101, 104};
    insertionSort(employeeIds);
    for (int id : employeeIds) {
      System.out.print(id + " ");
    }
  }
}
```

---

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.

Sol:

```
public class MergeSort {
  public static void mergeSort(int[] prices, int left, int right) {
```

```
    if (left < right) {
        int mid = (left + right) / 2;
        mergeSort(prices, left, mid);
        mergeSort(prices, mid + 1, right);
        merge(prices, left, mid, right);
    }
}
private static void merge(int[] prices, int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
    int[] leftArray = new int[n1];
    int[] rightArray = new int[n2];
    System.arraycopy(prices, left, leftArray, 0, n1);
    System.arraycopy(prices, mid + 1, rightArray, 0, n2);
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (leftArray[i] <= rightArray[j]) {
            prices[k] = leftArray[i];
            i++;
        } else {
            prices[k] = rightArray[j];
            j++;
        }
        k++;
```

```
        }
        while (i < n1) {
            prices[k] = leftArray[i];
            i++;
            k++;
        }
        while (j < n2) {
            prices[k] = rightArray[j];
            j++;
            k++;
        }
    }
    public static void main(String[] args) {
        int[] bookPrices = {350, 200, 500, 150, 400};
        mergeSort(bookPrices, 0, bookPrices.length - 1);
        for (int price : bookPrices) {
            System.out.print(price + " ");
        }
    }
}
```

---

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.

Sol:

```
public class QuickSort {

    public static void quickSort(int[] prices, int low, int high) {

        if (low < high) {

            int pivotIndex = partition(prices, low, high);

            quickSort(prices, low, pivotIndex - 1);

            quickSort(prices, pivotIndex + 1, high);

        }

    }

    private static int partition(int[] prices, int low, int high) {

        int pivot = prices[high];

        int i = low - 1;

        for (int j = low; j < high; j++) {

            if (prices[j] <= pivot) {

                i++;

                swap(prices, i, j);

            }

        }
```

```
        swap(prices, i + 1, high);
        return i + 1;
    }
    private static void swap(int[] prices, int i, int j) {
        int temp = prices[i];
        prices[i] = prices[j];
        prices[j] = temp;
    }
    public static void main(String[] args) {
        int[] productPrices = {250, 450, 300, 100, 500};
        quickSort(productPrices, 0, productPrices.length - 1);
        for (int price : productPrices) {
            System.out.print(price + " ");
        }
    }
}
```

---

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.

Sol:

```
public class SelectionSort {
    public static void selectionSort(int[] scores) {
        for (int i = 0; i < scores.length - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < scores.length; j++) {
                if (scores[j] < scores[minIndex]) {
                    minIndex = j;
                }
            }
            swap(scores, i, minIndex);
        }
    }
    private static void swap(int[] scores, int i, int j) {
        int temp = scores[i];
        scores[i] = scores[j];
        scores[j] = temp;
    }
    public static void main(String[] args) {
        int[] examScores = {85, 90, 70, 95, 80};
        selectionSort(examScores);
        for (int score : examScores) {
            System.out.print(score + " ");
        }
```

```
    }
}
```

---

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.

Sol:

```
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);

        }

    }

    private static void heapify(int[] arr, int n, int i) {

        int largest = i, l = 2 * i + 1, r = 2 * i + 2;
```

```
        if (l < n && arr[l] > arr[largest]) largest = l;
        if (r < n && arr[r] > arr[largest]) largest = r;
        if (largest != i) {
            int swap = arr[i];
            arr[i] = arr[largest];
            arr[largest] = swap;
            heapify(arr, n, largest);
        }
    }
    public static void main(String[] args) {
        int[] salaries = {45000, 60000, 35000, 70000, 50000};
        heapSort(salaries);
        for (int s : salaries) System.out.print(s + " ");
    }
}
```

---

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.

Sol:

```
public class CountingSort {
    public static void countingSort(int[] ages) {
        int max = 18, min = 10;
        int range = max - min + 1;
        int[] count = new int[range];
        int[] output = new int[ages.length];
        for (int age : ages) count[age - min]++;
        for (int i = 1; i < range; i++) count[i] += count[i - 1];
        for (int i = ages.length - 1; i >= 0; i--) {
            output[count[ages[i] - min] - 1] = ages[i];
            count[ages[i] - min]--;
        }
        System.arraycopy(output, 0, ages, 0, ages.length);
    }
    public static void main(String[] args) {
        int[] ages = {12, 15, 10, 18, 12, 14, 17, 11};
        countingSort(ages);
        for (int age : ages) System.out.print(age + " ");
    }
}
```