**Basic Sorting Algorithms**

**1.Bubble Sort** : a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process continues until the list is sorted.

* **Time Complexity:** O(n²) in worst and average cases, O(n) in best case (when already sorted).

**2. Insertion Sort:** It builds the sorted array one item at a time, picking the next element and inserting it into its correct position within the already sorted portion.

* **Time Complexity:** O(n²) in worst and average cases, O(n) in best case.

**3. Selection Sort:** It works by repeatedly finding the smallest (or largest) element in the unsorted portion and swapping it with the first element of that portion.

* **Time Complexity:** O(n²) in all cases.

import java.util.Arrays;

public class SortingAlgorithms {

    public static void bubbleSort(int[] arr) {

        int n = arr.length;

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

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

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

                    int temp = arr[j];

                    arr[j] = arr[j + 1];

                    arr[j + 1] = temp; } } } }

public static void selectionSort(int[] arr) {

        int n = arr.length;

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

            int minIndex = i;

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

                if (arr[j] < arr[minIndex]) {

                    minIndex = j; } }

            int temp = arr[minIndex];

            arr[minIndex] = arr[i];

arr[i] = temp; } }

public static void insertionSort(int[] arr) {

        int n = arr.length;

        for (int i = 1; i < n; i++) {

            int key = arr[i];

            int j = i - 1;

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

                arr[j + 1] = arr[j];

                j = j - 1; }

            arr[j + 1] = key; } }

public static void printArray(int[] arr) {

        for (int num : arr) {

            System.out.print(num + " "); }

          System.out.println(); }

    public static void main(String[] args) {

        int[] arr1 = {27,12,3,8,11,18,1,15};

        int[] arr2 = {32,54,76,89,12,45,23};

        int[] arr3 = {34,6,12,4,9,23,31,12};

        System.out.println("Intial array");

        printArray(arr1);

        bubbleSort(arr1);

        System.out.println("Sorted Array using bubbleSort");

        printArray(arr1);

        System.out.println("");

System.out.println("Intial array");

        printArray(arr2);

        selectionSort(arr2);

        System.out.println("Sorted Array using SelectionSort");

        printArray(arr2);

        System.out.println("");

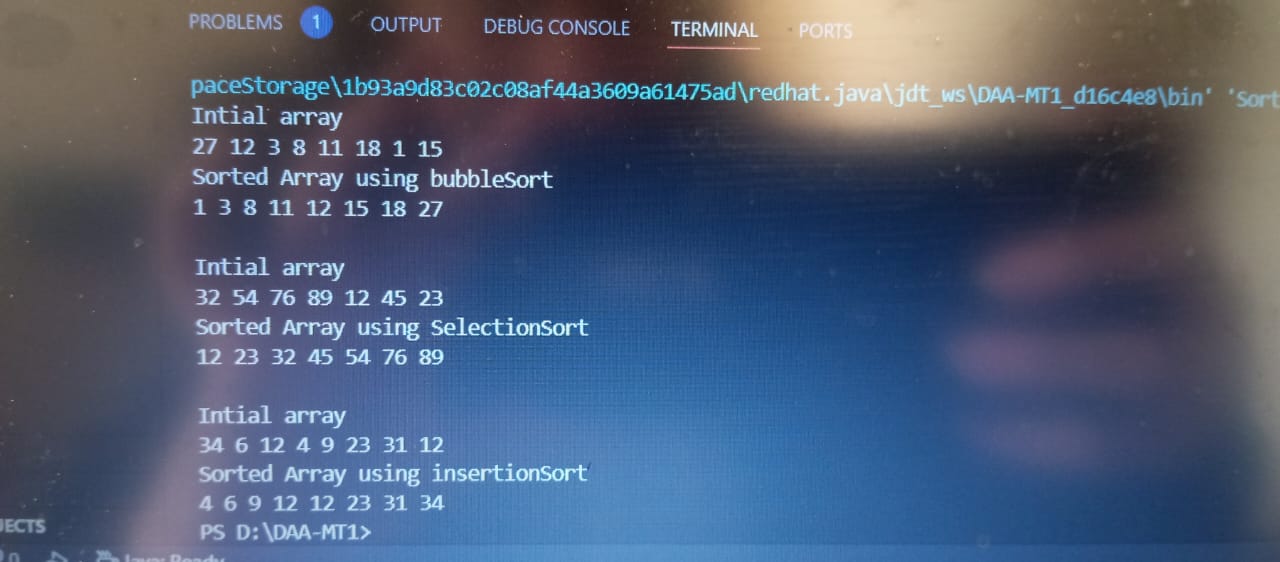
  System.out.println("Intial array");

        printArray(arr3);

        insertionSort(arr3);

        System.out.println("Sorted Array using insertionSort");

        printArray(arr3); } }



**Heap Implementation and operation**

import java.util.Scanner;

class MinHeap {

    private int[] heap;

    private int size;

    private static final int MAX\_SIZE = 100;

public MinHeap() {

        heap = new int[MAX\_SIZE];

        size = 0; }

      private int parent(int i) { return (i - 1) / 2; }

    private int leftChild(int i) { return 2 \* i + 1; }

    private int rightChild(int i) { return 2 \* i + 2; }

public void insert(int value) {

        if (size >= MAX\_SIZE) {

            System.out.println("Heap is full!");

            return;

        }

        heap[size] = value;

        heapifyUp(size);

        size++;

    }

private void heapifyUp(int index) {

        while (index > 0 && heap[index] < heap[parent(index)]) {

            swap(index, parent(index));

            index = parent(index)   } }

    public void deleteRoot() {

        if (size == 0) {

            System.out.println("Heap is empty!");

            return;

        }

        heap[0] = heap[size - 1];

        size--;

        heapifyDown(0);

    }

    private void heapifyDown(int index) {

        int smallest = index;

        int left = leftChild(index);

        int right = rightChild(index);

        if (left < size && heap[left] < heap[smallest]) {

            smallest = left;

        }

        if (right < size && heap[right] < heap[smallest]) {

            smallest = right;

        }

        if (smallest != index) {

            swap(index, smallest);

            heapifyDown(smallest);

        }

    }

    private void swap(int i, int j) {

        int temp = heap[i];

        heap[i] = heap[j];

        heap[j] = temp;

    }

    public void display() {

        if (size == 0) {

            System.out.println("Heap is empty!");

            return;

        }

        for (int i = 0; i < size; i++) {

            System.out.print(heap[i] + " ");

        }

        System.out.println();

    }

}

public class HeapImplementation {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        MinHeap minHeap = new MinHeap();

        int choice, val;

        do {

            System.out.println("\n1. Insert\n2. Delete Root\n3. Display\n4. Exit\nEnter choice: ");

            choice = scanner.nextInt();

            switch (choice) {

                case 1:

                    System.out.print("Enter value to insert: ");

                    val = scanner.nextInt();

                    minHeap.insert(val);

                    break;

                case 2:

                    minHeap.deleteRoot();

                    System.out.println("Root deleted.");

                    break;

                case 3:

                    System.out.print("Heap: ");

                    minHeap.display();

                    break;

                case 4:

                    System.out.println("Exiting...");

                    break;

                default:

                    System.out.println("Invalid choice.");

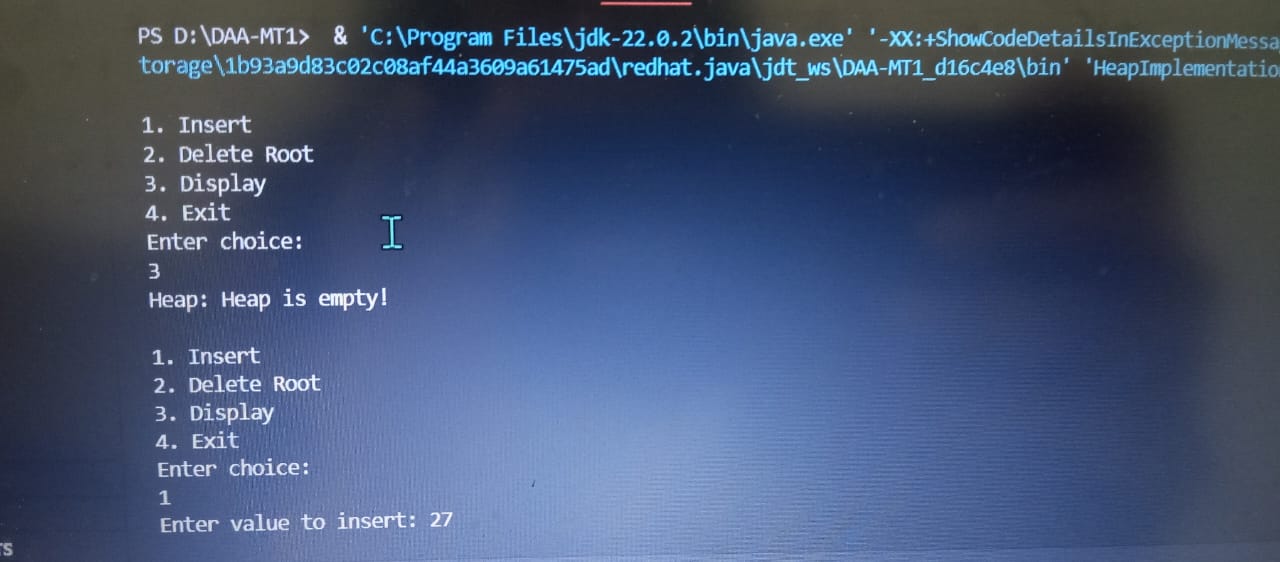
            }

        } while (choice != 4);

        scanner.close();

    }

}



import java.util.Scanner;

public class SecondMinMax {

    public static double[] findMinMax(double[] arr, int left, int right) {

        if (left == right) {

            return new double[]{arr[left], arr[left], Double.MAX\_VALUE, Double.MIN\_VALUE};

        }

        if (right - left == 1) {

            double min = Math.min(arr[left], arr[right]);

            double max = Math.max(arr[left], arr[right]);

            double secondMin = Math.max(arr[left], arr[right]);

            double secondMax = Math.min(arr[left], arr[right]);

            return new double[]{min, max, secondMin, secondMax};

        }

        int mid = left + (right - left) / 2;

        double[] leftPart = findMinMax(arr, left, mid);

        double[] rightPart = findMinMax(arr, mid + 1, right);

        double min = Math.min(leftPart[0], rightPart[0]);

        double max = Math.max(leftPart[1], rightPart[1]);

        double secondMin;

        if (leftPart[0] == min) secondMin = Math.min(leftPart[2], rightPart[0]);

        else secondMin = Math.min(leftPart[0], rightPart[2]);

        double secondMax;

        if (leftPart[1] == max) secondMax = Math.max(leftPart[3], rightPart[1]);

        else secondMax = Math.max(leftPart[1], rightPart[3]);

        return new double[]{min, max, secondMin, secondMax};

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter number of elements: ");

        int n = scanner.nextInt();

        double[] arr = new double[n];

        System.out.println("Enter elements:");

        for (int i = 0; i < n; i++) {

            arr[i] = scanner.nextDouble();

        }

        if (n < 2) {

            System.out.println("Not enough elements to find second min and max.");

        } else {

            double[] result = findMinMax(arr, 0, n - 1);

            System.out.println("Minimum: " + result[0]);

            System.out.println("Maximum: " + result[1]);

            System.out.println("Second Minimum: " + result[2]);

            System.out.println("Second Maximum: " + result[3]);

        } }

    }