**ALGORITHMS DATASTRUCTURES**

**Exercise 3: Sorting Customer Orders**

**Scenario:**

You are tasked with sorting customer orders by their total price on an e-commerce platform. This helps in prioritizing high-value orders.

**Steps:**

1. **Understand Sorting Algorithms:**
   * Explain different sorting algorithms (Bubble Sort, Insertion Sort, Quick Sort, Merge Sort).
2. **Setup:**
   * Create a class Order with attributes like orderId, customerName, and totalPrice.
3. **Implementation:**
   * Implement Bubble Sort to sort orders by totalPrice.
   * Implement Quick Sort to sort orders by totalPrice.
4. **Analysis:**
   * Compare the performance (time complexity) of Bubble Sort and Quick Sort.
   * Discuss why Quick Sort is generally preferred over Bubble Sort.

**DIFFERENT SORTING ALGORITHMS:**

**Bubble Sort:**

* Simple, compares adjacent elements and swaps.
* Time Complexity: Worst/Average — O(n²)
* Use: Small datasets only

**Insertion Sort:**

* Builds sorted array one item at a time.
* Time Complexity: O(n²)
* Use: Small, mostly sorted data

**Quick Sort:**

* Divide-and-conquer using a pivot.
* Time Complexity:
  + Average — O(n log n)
  + Worst — O(n²) (rare)
* Use: General-purpose, fast in practice

**Merge Sort:**

* Recursively splits and merges sorted arrays.
* Time Complexity: O(n log n)
* Use: Stable sort, consistent performance

**IMPLEMENTATION BY BUBBLE SORT:**

**PROGRAM:**

class Order {

String orderId;

String customerName;

double totalPrice;

public Order(String orderId, String customerName, double totalPrice) {

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

public String toString() {

return "OrderID: " + orderId + ", Name: " + customerName + ", Price: ₹" + totalPrice;

}

}

public class BubbleSortOrders {

public static void main(String[] args) {

Order[] orders = {

new Order("O1", "Keerthi", 2200),

new Order("O2", "Riya", 1250),

new Order("O3", "Varun", 3800),

new Order("O4", "Ajay", 1600)

};

bubbleSort(orders);

System.out.println("Sorted Orders (Bubble Sort by Price):");

for (Order order : orders) {

System.out.println(order);

}

}

static void bubbleSort(Order[] arr) {

int n = arr.length;

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

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

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

Order temp = arr[j];

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

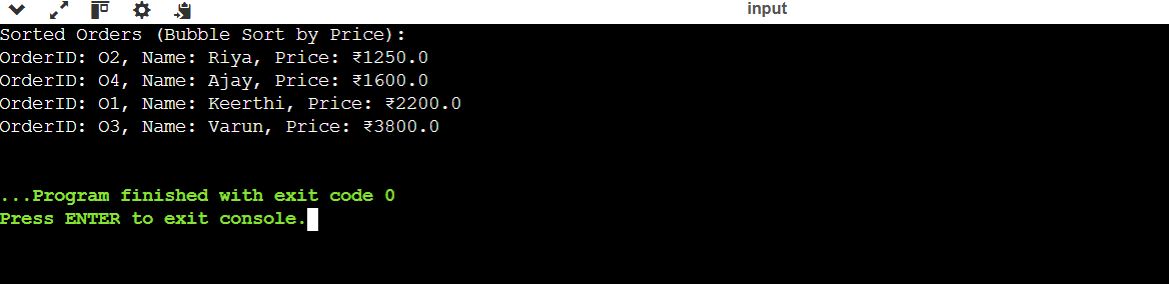
arr[j + 1] = temp;

}

}

}

**OUTPUT:**

****

**IMPLEMENTATION BY QUICK SORT:**

**PROGRAM:**

class Order {

String orderId;

String customerName;

double totalPrice;

public Order(String orderId, String customerName, double totalPrice) {

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

public String toString() {

return "OrderID: " + orderId + ", Name: " + customerName + ", Price: ₹" + totalPrice;

}

}

public class QuickSortOrders {

public static void main(String[] args) {

Order[] orders = {

new Order("O1", "Keerthi", 2200),

new Order("O2", "Riya", 1250),

new Order("O3", "Varun", 3800),

new Order("O4", "Ajay", 1600)

};

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

System.out.println("Sorted Orders (Quick Sort by Price):");

for (Order order : orders) {

System.out.println(order);

}

}

static void quickSort(Order[] arr, int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

static int partition(Order[] arr, int low, int high) {

double pivot = arr[high].totalPrice;

int i = low - 1;

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

if (arr[j].totalPrice < pivot) {

i++;

Order temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

Order temp = arr[i + 1];

arr[i + 1] = arr[high];

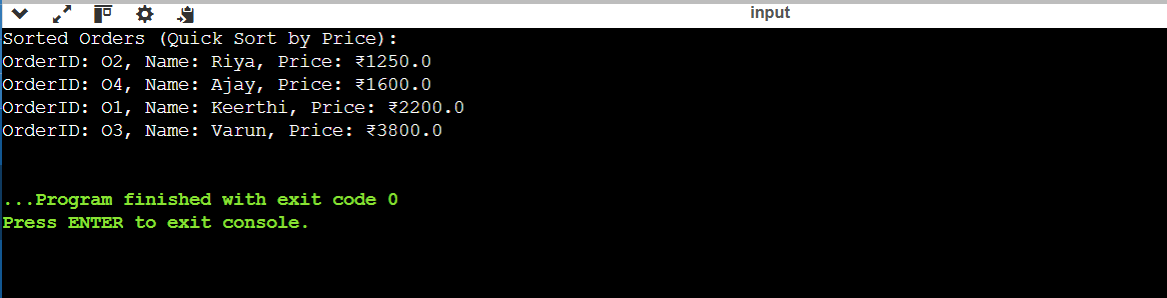
arr[high] = temp;

return i + 1;

}

}

**OUTPUT:**

****

**ANALYSIS:**

**Time Complexity**

* **Bubble Sort:**
  + Best Case: O(n)
  + Average Case: O(n²)
  + Worst Case: O(n²)
  + Space: O(1) (in-place)
* **Quick Sort:**
  + Best Case: O(n log n)
  + Average Case: O(n log n)
  + Worst Case: O(n²)\*
  + Space: O(log n) (due to recursion)

**REASON BEHIND USING QUICKSORT:**

* Fast Performance: Offers average-case time complexity of O(n log n), much faster than Bubble Sort (O(n²)).
* In-Place Sorting: Uses minimal extra memory (space complexity: O(log n) due to recursion).
* Efficient in Practice: Performs well with large datasets due to good cache usage and fewer swaps.
* Widely Used: Adopted in many programming languages’ standard libraries (e.g., Java, C, Python).
* Customizable Pivot Strategy: Allows optimization (e.g., median-of-three) to avoid worst-case scenarios.
* Scalable: Handles large data efficiently, making it ideal for real-world applications.