**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. **Bubble Sort:**

Bubble Sort compares adjacent elements and swaps them if they are in the wrong order. This process is repeated until the list is sorted.

**Time Complexity:**

* Best – O(n)
* Average – O(n²)
* Worst – O(n²)

**Space Complexity:**

O(1) – In-place sorting

1. **Insertion Sort**

**Definition:**

Insertion Sort picks elements one by one and inserts each into its correct position among the already sorted part of the array.

**Time Complexity:**

* Best – O(n)
* Average – O(n²)
* Worst – O(n²)

**Space Complexity:**

O(1) – In-place sorting

1. **Quick Sort**

Quick Sort selects a pivot element and partitions the array around it. It recursively applies this process to subarrays on the left and right of the pivot.

**Time Complexity:**

* Best – O(n log n)
* Average – O(n log n)
* Worst – O(n²)

**Space Complexity:**

O(log n) – Due to recursion stack

1. **Merge Sort**

Merge Sort divides the array into two halves, recursively sorts them, and merges the sorted halves to produce the final sorted array.

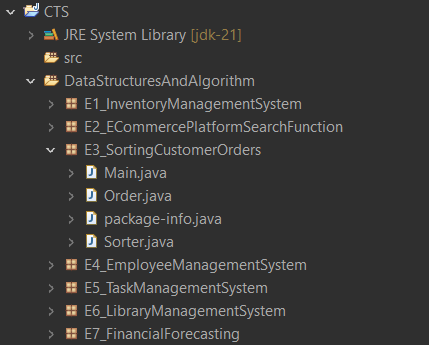
**Time Complexity:**

* Best – O(n log n)
* Average – O(n log n)
* Worst – O(n log n)

**Space Complexity:**

O(n) – Extra space for merging

1. **Setup:**
   * **Create a class Order with attributes like orderId, customerName, and totalPrice.**

****

1. **Implementation:**
   * **Implement Bubble Sort to sort orders by totalPrice.**
   * **Implement Quick Sort to sort orders by totalPrice.**

**Order.java**

package E3\_SortingCustomerOrders;

public class Order {

int orderId;

String customerName;

double totalPrice;

public Order(int id, String name, double price) {

this.orderId = id;

this.customerName = name;

this.totalPrice = price;

}

public String toString() {

return orderId + " - " + customerName + " - ₹" + totalPrice;

}

}

**Sorter.java**

package E3\_SortingCustomerOrders;

public class Sorter {

//bubble sort

public static void bubbleSort(Order[] orders) {

int n = orders.length;

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

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

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

Order temp = orders[j];

orders[j] = orders[j + 1];

orders[j + 1] = temp;

}

}

}

}

//quick sort

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

if (low < high) {

int pi = *partition*(orders, low, high);

*quickSort*(orders, low, pi - 1);

*quickSort*(orders, pi + 1, high);

}

}

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

double pivot = orders[high].totalPrice;

int i = low - 1;

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

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

i++;

Order temp = orders[i];

orders[i] = orders[j];

orders[j] = temp;

}

}

Order temp = orders[i + 1];

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

orders[high] = temp;

return i + 1;

}

}

**Main.java**

package E3\_SortingCustomerOrders;

public class Main {

public static void main(String[] args) {

Order[] orders = {

new Order(1, "Sona", 150.0),

new Order(2, "Sri", 300.0),

new Order(3, "Rudra", 50.0),

new Order(4, "Dev", 400.0)

};

// Bubble Sort

System.*out*.println("Bubble Sort by Total Price:");

Sorter.*bubbleSort*(orders);

for (Order o : orders) System.*out*.println(o);

// Reset for Quick Sort

orders = new Order[]{

new Order(1, "Sona", 150.0),

new Order(2, "Sri", 300.0),

new Order(3, "Rudra", 50.0),

new Order(4, "Dev", 400.0)

};

System.*out*.println("\nQuick Sort by Total Price:");

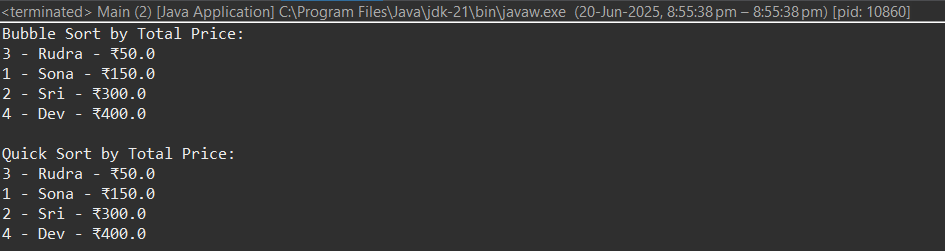
Sorter.*quickSort*(orders, 0, orders.length - 1);

for (Order o : orders) System.*out*.println(o);

}

}

**Output:**

****

1. **Analysis:**
   * **Compare the performance (time complexity) of Bubble Sort and Quick Sort.**

|  |  |  |
| --- | --- | --- |
| Case | Bubble Sort | Quick Sort |
| Best Case | **O(n)** | **O(n log n)** |
| Average Case | **O(n²)** | **O(n log n)** |
| Worst Case | **O(n²)** | **O(n²) (rare, can be optimized)** |
| Space | **O(1)** | **O(log n) (due to recursion)** |

* + **Discuss why Quick Sort is generally preferred over Bubble Sort.**
* Quick Sort has better average performance (O(n log n) vs Bubble Sort’s O(n²)).
* Quick Sort performs fewer swaps and comparisons overall.
* It efficiently handles large, unsorted data using partitioning and recursion.
* Many standard libraries (e.g., Java, C++, Python) internally use optimized Quick Sort or hybrid versions.
* Bubble Sort is mainly used for teaching; it’s inefficient in real-world applications.