**Exercise 3: Sorting Customer Orders**

1. Explain different sorting algorithms (Bubble Sort, Insertion Sort, Quick Sort, Merge Sort)

**Bubble Sort:**

* **Description:** A simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted.
* **Time Complexity:**
  + Best Case: O(n) (when the list is already sorted)
  + Average Case: O(n2)
  + Worst Case: O(n2)

**Insertion Sort:**

* **Description:** Builds the final sorted array one item at a time, taking each element and inserting it into its correct position among the already sorted elements.
* **Time Complexity:**
  + Best Case: O(n) (when the list is already sorted)
  + Average Case: O(n2)
  + Worst Case: O(n2)

**Quick Sort:**

* **Description:** A divide-and-conquer algorithm that picks an element as a pivot and partitions the array around the pivot. The process is then repeated recursively for the sub-arrays.
* **Time Complexity:**
  + Best Case: O(nlog⁡n)
  + Average Case: O(nlog⁡n)
  + Worst Case: O(n2) (when the pivot selection is poor, e.g., always picking the smallest or largest element)

**Merge Sort:**

* **Description:** A divide-and-conquer algorithm that divides the array into two halves, recursively sorts each half, and then merges the two sorted halves.
* **Time Complexity:**
  + Best Case: O(nlog⁡n)
  + Average Case: O(nlog⁡n)
  + Worst Case: O(nlog⁡n)

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

public class Order {

private int id;

private String customerName;

private double totalPrice;

static{

System.out.println("in static");

}

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

this.id = id;

this.customerName = name;

this.totalPrice = totalPrice;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return customerName;

}

public void setName(String name) {

this.customerName = name;

}

public double getTotalPrice() {

return totalPrice;

}

public void setTotalPrice(double totalPrice) {

this.totalPrice = totalPrice;

}

@Override

public String toString(){

return this.id + " " + this.customerName + " " + this.totalPrice;

}

}

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

public class Sort {

public void bubbleSort(Order arr[]){

int len = arr.length;

Order temp = null;

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

boolean swapped = false;

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

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

temp = arr[j];

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

arr[j + 1] = temp;

swapped = true;

}

}

if(!swapped)

break;

}

}

public static void swap(Order[] orders,int i,int j){

Order temp = orders[i];

orders[i] = orders[j];

orders[j] = temp;

}

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

int i = low - 1;

double pivot = orders[high].getTotalPrice();

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

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

i++;

swap(orders,i,j);

}

}

swap(orders,i + 1,high);

return i + 1;

}

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

if(low < high){

int pi = partion(orders,low,high);

quickSort(orders,low,pi-1);

quickSort(orders,pi + 1,high);

}

}

}

1. Comparison and Suitability

**Time Complexity:**

* **Bubble Sort:**
  + Best Case: O(n)O(n)O(n) (when the array is already sorted)
  + Average Case: O(n2)O(n^2)O(n2)
  + Worst Case: O(n2)O(n^2)O(n2)
* **Quick Sort:**
  + Best Case: O(nlog⁡n)O(n \log n)O(nlogn)
  + Average Case: O(nlog⁡n)O(n \log n)O(nlogn)
  + Worst Case: O(n2)O(n^2)O(n2) (when the pivot selection is poor)

**Comparison and Suitability:**

* **Bubble Sort:**
  + Easy to implement but inefficient for large datasets.
  + Suitable only for small datasets or when simplicity is more important than performance.
* **Quick Sort:**
  + Efficient and performs well on average for large datasets.
  + More complex to implement but generally preferred for its better average-case performance.