Code for Bubble Sort Algorithm:

package com.wipro.day03;

public class Assign4 {

public static void bubbleSort(int[] arr) {

int n = arr.length;

boolean swapped;

do {

swapped = false;

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

if (arr[i - 1] > arr[i]) {

// Swap the elements

int temp = arr[i - 1];

arr[i - 1] = arr[i];

arr[i] = temp;

swapped = true;

n = n - 1;

} while (swapped);

}

public static void main(String[] args) {

int[] arr = {64, 34, 12, 25};

*bubbleSort*(arr);

System.***out***.println("Sorted array: ");

for (int i = 0; i < arr.length; i++) {

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

}

}

Bubble Sort Algorithm Explanation:

The bubble sort algorithm using the array arr = {64, 34, 12, 25}. Bubble sort is a comparison-based algorithm that sorts a list by repeatedly stepping through the list, comparing adjacent elements, and swapping them if they are in the wrong order. The process is repeated until no more swaps are needed, indicating that the list is sorted.

Bubble Sort Process on the Array:

1. First Pass:

o Compare 64 and 34, swap since 64 > 34. arr = {34, 64, 12, 25}

o Compare 64 and 12, swap since 64 > 12. arr = {34, 12, 64, 25}

o Compare 64 and 25, swap since 64 > 25. arr = {34, 12, 25, 64}

2. Second Pass:

o Compare 34 and 12, swap since 34 > 12. arr = {12, 34, 25, 64}

o Compare 34 and 25, swap since 34 > 25. arr = {12, 25, 34, 64}

o No need to compare 34 and 64 since 34 < 64.

3. Third Pass:

o Compare 12 and 25, no swap needed since 12 < 25. arr = {12, 25, 34, 64}

o Compare 25 and 34, no swap needed since 25 < 34. arr = {12, 25, 34, 64}

After the third pass, the array is already sorted, and no further passes are necessary. The final sorted array is: arr = {12, 25, 34, 64}

Explanation: In each pass, the algorithm compares each pair of adjacent elements and swaps them if the first element is greater than the second. This causes larger values to “bubble up” to the end of the array, while smaller values “sink” towards the beginning. The process repeats, making fewer comparisons each time, until no swaps are needed, indicating that the array is sorted. The algorithm is efficient for small datasets but less so for larger lists. It has an average and worst-case complexity of O(n^2), where n is the number of items being sorted.