public class Selectionsort {

    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 main(String[] args) {

        int[] arr = {64, 25, 12, 22, 11};

        selectionSort(arr);

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

        for (int num : arr) {

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

        }

    }

}

Output:



Analysis:

1. Best Case Scenario

* **Description**: The array is already sorted in ascending order.
* **Time Complexity**: O(n2)*O*(*n*2)
* **Explanation**: Even in this optimal situation, selection sort performs n(n−1)22*n*(*n*−1)​ comparisons to ensure that each element is in the correct position. Thus, it still incurs a quadratic time complexity due to the necessity of checking each element.

2. Average Case Scenario

* **Description**: The elements of the array are in random order.
* **Time Complexity**: O(n2)*O*(*n*2)
* **Explanation**: On average, selection sort will still perform approximately n(n−1)44*n*(*n*−1)​ comparisons as it systematically scans through the unsorted portion to identify the minimum element for each iteration. This leads to a similar quadratic time complexity as in other cases.

3. Worst Case Scenario

* **Description**: The array is sorted in descending order.
* **Time Complexity**: O(n2)*O*(*n*2)
* **Explanation**: In this case, selection sort must perform the maximum number of comparisons and swaps to reorder the elements into ascending order. Each element must be checked against all others, resulting in a total of n(n−1)22*n*(*n*−1)​ comparisons.