Sorting Algorithms Analysis

Here, I have analysed Selection Sort, Bubble Sort and Insertion Sort algorithms with the Step Counting method and then plotted the results for different input size vs steps counted in graphs for both ascending and descending order sorting.

Code

```
#include <iostream>
#include <vector>
#include <iomanip>

using namespace std;

// Global variable to count steps
long long stepCount = 0;

enum SortOrder {
  Ascending,
  Descending
};

void insertionSort(vector<int>& arr, SortOrder order){
  int n = arr.size();
  // looping from 0 to n-1 to loop over the array
    for(int i = 0; i <= n-1; i++){
    stepCount++;
  // inner loop to select the element and insert it in its correct position
    for(int j = i; j>0; j--){
      stepCount++;
          if ((order == SortOrder::Ascending && arr[j] < arr[j - 1]) ||
                (order == SortOrder::Descending && arr[j] > arr[j - 1])) {
                // Swap if condition is met
                int temp = arr[j];  // Use temp variable to swap
                arr[j] = arr[j - 1];
                arr[j - 1] = temp;
                stepCount += 3;
          }
      stepCount+=2;
        }
    stepCount++;
    }
}

void selection_sort(vector<int>& arr, SortOrder order){
  int n = arr.size();
  // looping till second last element as last element will automatically be sorted
  // by the time we reach it
    for(int i = 0; i < n-1; i++){
        int min = i; // selecting the current element's index as minimum at start
    stepCount+=2;
    // looping from the selected element till end of array
        for(int j = i; j <= n-1 ; j++){
      stepCount++;
      if ((order == SortOrder::Ascending && arr[j] < arr[min]) ||
                (order == SortOrder::Descending && arr[j] > arr[min])) {
                min = j;
```

```
            }
        stepCount+=2;
            }
        // after the inner loop we have the minimum element in the unsorted part
        // now we swap the current element with the minimum element
        if (min != i) {
              int temp = arr[min];
              arr[min] = arr[i];
              arr[i] = temp;
          stepCount += 3;
        }
        stepCount+=2;
        }
      // from the next iteration
      //find the min element in the unsorted part of the array and repeat the process
}

void bubble_sort(vector<int>& arr, SortOrder order) {
    int n = arr.size();
    bool did_swap;

    for (int i = n - 1; i >= 1; i--) {
        did_swap = false;
        stepCount += 2;

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

            // Comparison logic based on sort order
            if ((order == SortOrder::Ascending && arr[j] > arr[j + 1]) ||
                (order == SortOrder::Descending && arr[j] < arr[j + 1])) {
                // Swap elements
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                did_swap = true;
                stepCount += 4;
            }
            stepCount += 2;
        }
        stepCount++;

        if (!did_swap) break;
        stepCount++;
    }
}

// Function to generate specific types of input
vector<int> generateInput(int size, string type) {
    vector<int> arr(size);
    if (type == "random") {
        for (int i = 0; i < size; ++i) {
            arr[i] = rand() % size;
        }
    } else if (type == "sorted") {
        for (int i = 0; i < size; ++i) {
            arr[i] = i;
        }
    } else if (type == "reversed") {
        for (int i = 0; i < size; ++i) {
            arr[i] = size - i;
        }
    }
```

```
    return arr;
}


// Function to analyze step count and output Big O notation
void analyzeStepCount(void (*sortFunction)(vector<int>&, SortOrder), vector<int>& inputSizes,
SortOrder order) {
    cout << "Step Count Analysis for "
         << (order == SortOrder::Ascending ? "Ascending" : "Descending")
         << " Order:\n";
    cout << "-------------------------------------------------------------------------\n";
    cout << "| Input Size | Sort Case Steps | Rev. Sort Case Steps | Random Case Steps |\n";
    cout << "-------------------------------------------------------------------------\n";

    vector<long long> bestSteps, worstSteps, averageSteps; // Store steps for all cases

    for (int size : inputSizes) {
        // Best Case
        vector<int> bestCaseInput = generateInput(size, "sorted");
        stepCount = 0;
        sortFunction(bestCaseInput, order);
        bestSteps.push_back(stepCount);

        // Worst Case
        vector<int> worstCaseInput = generateInput(size, "reversed");
        stepCount = 0;
        sortFunction(worstCaseInput, order);
        worstSteps.push_back(stepCount);

        // Average Case
        vector<int> averageCaseInput = generateInput(size, "random");
        stepCount = 0;
        sortFunction(averageCaseInput, order);
        averageSteps.push_back(stepCount);

        cout << "| " << setw(10) << size
             << " | " << setw(15) << bestSteps.back()
             << " | " << setw(15) << worstSteps.back()
             << " | " << setw(18) << averageSteps.back()
             << " |\n";
    }


}

void testCustomArray() {
    char response;
    cout << "Do you want to test a custom array? (y/n): ";
    cin >> response;

    if (response == 'y' || response == 'Y') {
        // Get the custom array from the user
        cout << "Enter the size of the array: ";
        int size;
        cin >> size;

        vector<int> customArray(size);
        cout << "Enter " << size << " elements for the array:\n";
        for (int i = 0; i < size; ++i) {
            cin >> customArray[i];
        }

        // Sort the array using all three algorithms
```

```cpp
    vector<int> arr1 = customArray; // Copy for each algorithm
    vector<int> arr2 = customArray;
    vector<int> arr3 = customArray;

    // Ascending Order
    cout << "\nSorting in Ascending Order:\n";
    cout << "---------------------------\n";

    stepCount = 0;
    bubble_sort(arr1, SortOrder::Ascending);
    cout << "Bubble Sort: Steps = " << stepCount << ", Sorted Array: ";
    for (int num : arr1) cout << num << " ";
    cout << "\n";

    stepCount = 0;
    selection_sort(arr2, SortOrder::Ascending);
    cout << "Selection Sort: Steps = " << stepCount << ", Sorted Array: ";
    for (int num : arr2) cout << num << " ";
    cout << "\n";

    stepCount = 0;
    insertionSort(arr3, SortOrder::Ascending);
    cout << "Insertion Sort: Steps = " << stepCount << ", Sorted Array: ";
    for (int num : arr3) cout << num << " ";
    cout << "\n";

    // Descending Order
    cout << "\nSorting in Descending Order:\n";
    cout << "----------------------------\n";

    arr1 = customArray;
    arr2 = customArray;
    arr3 = customArray;

    stepCount = 0;
    bubble_sort(arr1, SortOrder::Descending);
    cout << "Bubble Sort: Steps = " << stepCount << ", Sorted Array: ";
    for (int num : arr1) cout << num << " ";
    cout << "\n";

    stepCount = 0;
    selection_sort(arr2, SortOrder::Descending);
    cout << "Selection Sort: Steps = " << stepCount << ", Sorted Array: ";
    for (int num : arr2) cout << num << " ";
    cout << "\n";

    stepCount = 0;
    insertionSort(arr3, SortOrder::Descending);
    cout << "Insertion Sort: Steps = " << stepCount << ", Sorted Array: ";
    for (int num : arr3) cout << num << " ";
    cout << "\n";

    // Print step counts summary
    cout << "\nStep Count Summary for Input Size " << size << ":\n";
    cout << "-------------------------------------------------\n";
    cout << "| Algorithm       | Ascending Steps | Descending Steps |\n";
    cout << "-------------------------------------------------\n";

    // Ascending Steps
    stepCount = 0;
    bubble_sort(customArray, SortOrder::Ascending);
    long long bubbleAscendingSteps = stepCount;
```

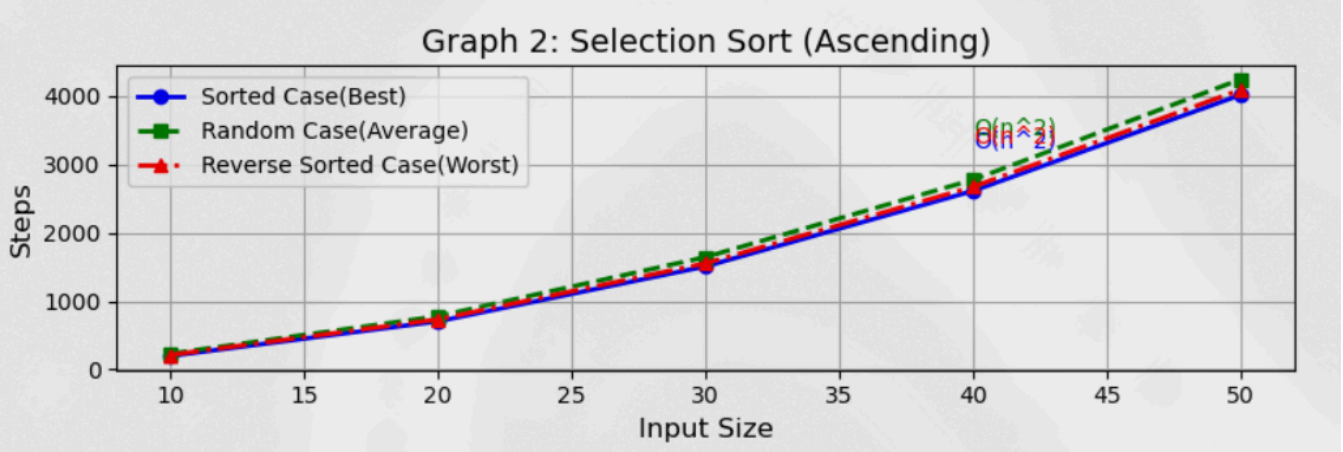
```
        stepCount = 0;
        selection_sort(customArray, SortOrder::Ascending);
        long long selectionAscendingSteps = stepCount;

        stepCount = 0;
        insertionSort(customArray, SortOrder::Ascending);
        long long insertionAscendingSteps = stepCount;

        // Descending Steps
        stepCount = 0;
        bubble_sort(customArray, SortOrder::Descending);
        long long bubbleDescendingSteps = stepCount;

        stepCount = 0;
        selection_sort(customArray, SortOrder::Descending);
        long long selectionDescendingSteps = stepCount;

        stepCount = 0;
        insertionSort(customArray, SortOrder::Descending);
        long long insertionDescendingSteps = stepCount;

        cout << "| Bubble Sort     | " << setw(15) << bubbleAscendingSteps
             << " | " << setw(15) << bubbleDescendingSteps << " |\n";
        cout << "| Selection Sort  | " << setw(15) << selectionAscendingSteps
             << " | " << setw(15) << selectionDescendingSteps << " |\n";
        cout << "| Insertion Sort  | " << setw(15) << insertionAscendingSteps
             << " | " << setw(15) << insertionDescendingSteps << " |\n";
        cout << "-------------------------------------------------\n";

    } else {
        cout << "No custom array testing requested. Exiting...\n";
    }
}

int main() {
    // Define input sizes for testing
    vector<int> inputSizes = {10, 20, 30, 40, 50};

    // Analyze step count for insertion sort
    cout << "\nSelection sort: "<< endl;
    analyzeStepCount(selection_sort, inputSizes, SortOrder::Ascending);
    cout << "\n";
    analyzeStepCount(selection_sort, inputSizes, SortOrder::Descending);
    cout << "\nBubble sort: "<< endl;
    analyzeStepCount(bubble_sort, inputSizes, SortOrder::Ascending);
    cout << "\n";
    analyzeStepCount(bubble_sort, inputSizes, SortOrder::Descending);
    cout << "\nInsertion sort: "<< endl;
    analyzeStepCount(insertionSort, inputSizes, SortOrder::Ascending);
    cout << "\n";
    analyzeStepCount(insertionSort, inputSizes, SortOrder::Descending);
    cout << "\n";
    testCustomArray();
    return 0;
}
```

Sorting Algorithm Step Count Analysis

Output for Selection Sort

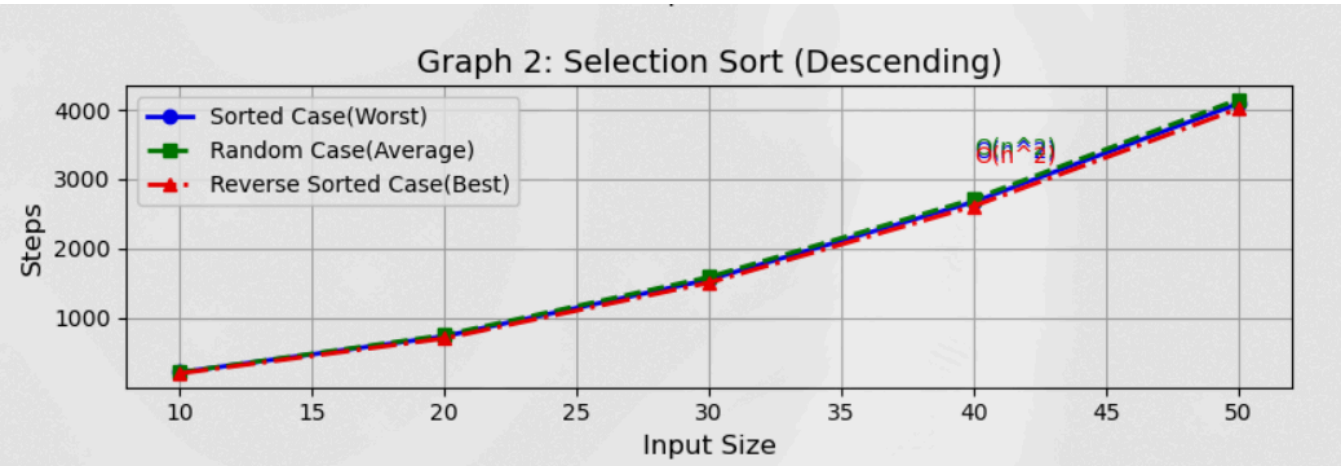
Ascending Order:

```
Step Count Analysis for Ascending Order:
------------------------------------------------------------------------
| Input Size | Sort Case Steps | Rev. Sort Case Steps | Random Case Steps |
------------------------------------------------------------------------
|         10 |             198 |              213 |                222 |
|         20 |             703 |              733 |                751 |
|         30 |            1508 |             1553 |               1586 |
|         40 |            2613 |             2673 |               2721 |
|         50 |            4018 |             4093 |               4156 |
```



Descending Order:

```
------------------------------------------------------------------------
| Input Size | Sort Case Steps | Rev. Sort Case Steps | Random Case Steps |
------------------------------------------------------------------------
|         10 |             213 |              198 |                207 |
|         20 |             733 |              703 |                760 |
|         30 |            1553 |             1508 |               1580 |
|         40 |            2673 |             2613 |               2721 |
|         50 |            4093 |             4018 |               4159 |
```

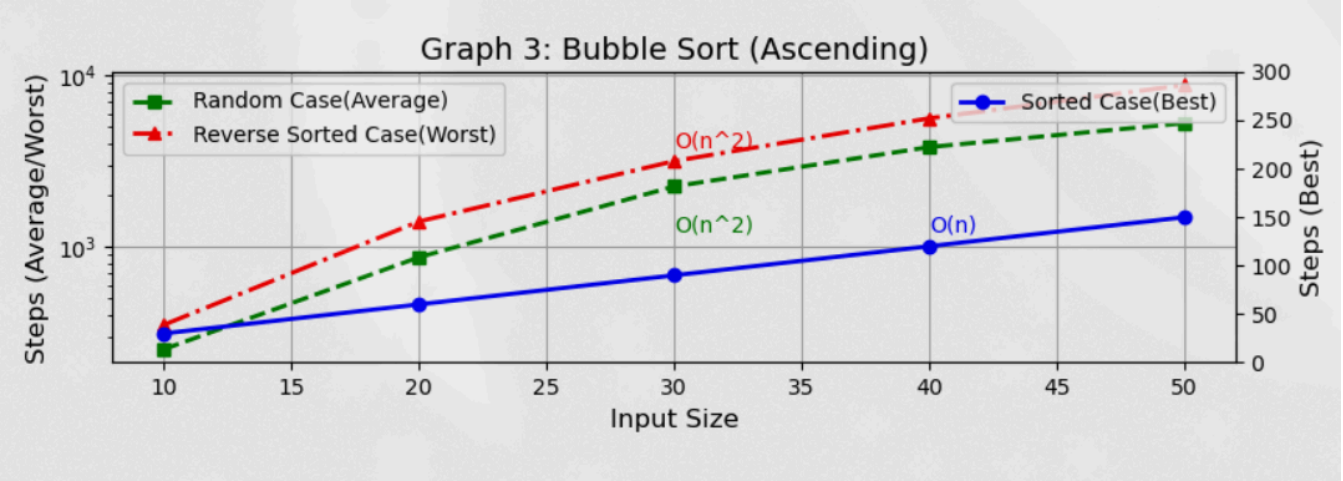


Output for Bubble Sort

Ascending Order:

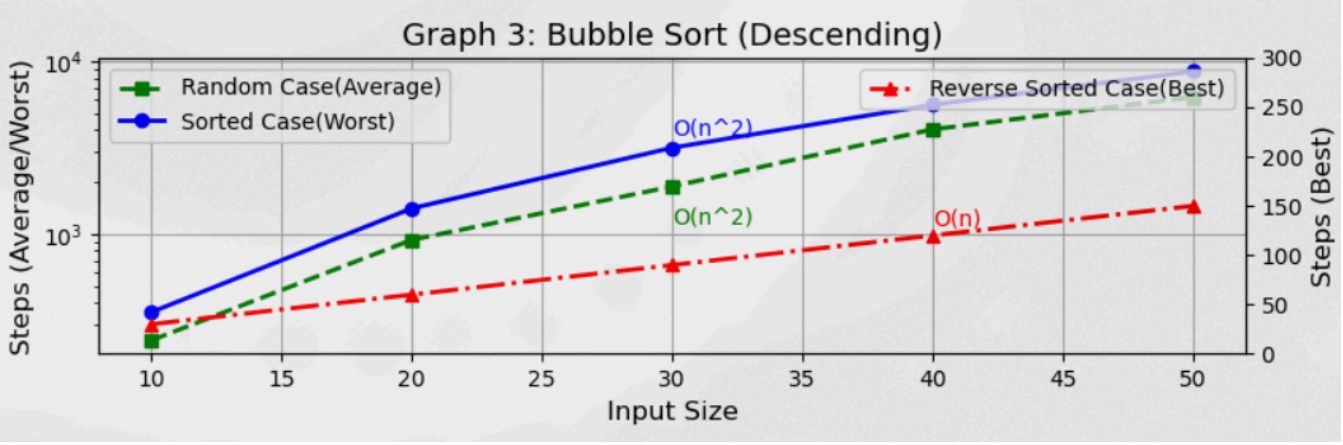
```
Step Count Analysis for Ascending Order:
------------------------------------------------------------------------
```

```
| Input Size | Sort Case Steps | Rev. Sort Case Steps | Random Case Steps |
-----------------------------------------------------------------------
|         10 |              30 |              351 |                290 |
|         20 |              60 |             1406 |                988 |
|         30 |              90 |             3161 |               2446 |
|         40 |             120 |             5616 |               3749 |
|         50 |             150 |             8771 |               6064 |
```



Descending Order:

```
Step Count Analysis for Descending Order:
-----------------------------------------------------------------------
| Input Size | Sort Case Steps | Rev. Sort Case Steps | Random Case Steps |
-----------------------------------------------------------------------
|         10 |             351 |               30 |                217 |
|         20 |            1406 |               60 |                983 |
|         30 |            3161 |               90 |               2156 |
|         40 |            5616 |              120 |               4141 |
|         50 |            8771 |              150 |               6520 |
```

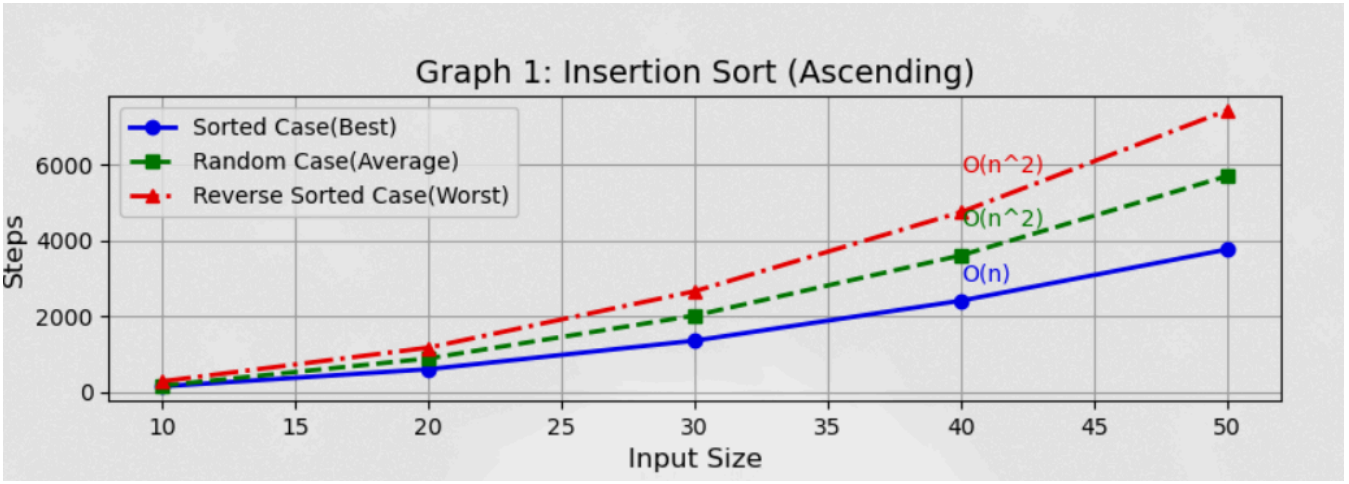


Output for Insertion Sort

Ascending Order:

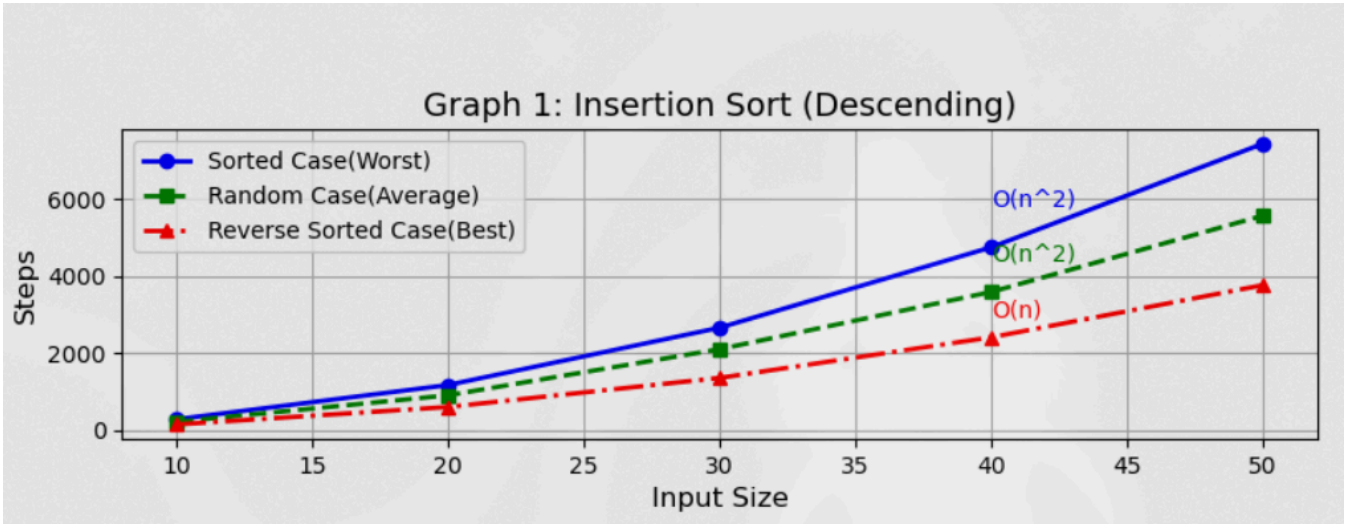
```
Step Count Analysis for Ascending Order:
-----------------------------------------------------------------------
| Input Size | Sort Case Steps | Rev. Sort Case Steps | Random Case Steps |
-----------------------------------------------------------------------
```

```
|         10 |             155 |             290 |             212 |
|         20 |             610 |            1180 |             955 |
|         30 |            1365 |            2670 |            1989 |
|         40 |            2420 |            4760 |            3674 |
|         50 |            3775 |            7450 |            5554 |
```



Descending Order:

```
Step Count Analysis for Descending Order:
|         10 |             290 |             155 |             203 |
|         20 |            1180 |             610 |             847 |
|         30 |            2670 |            1365 |            1983 |
|         40 |            4760 |            2420 |            3509 |
|         50 |            7450 |            3775 |            5284 |
```



Custom Array Testing

User Input:

```
Do you want to test a custom array? (y/n): y
Enter the size of the array: 5
Enter 5 elements for the array:
3 1 4 2 5
```

Output for Ascending Order:

```
Sorting in Ascending Order:
---------------------------
Bubble Sort: Steps = 50, Sorted Array: 1 2 3 4 5
Selection Sort: Steps = 67, Sorted Array: 1 2 3 4 5
Insertion Sort: Steps = 49, Sorted Array: 1 2 3 4 5
```

Output for Descending Order:

```
Sorting in Descending Order:
----------------------------
Bubble Sort: Steps = 74, Sorted Array: 5 4 3 2 1
Selection Sort: Steps =67, Sorted Array: 5 4 3 2 1
Insertion Sort: Steps = 61, Sorted Array: 5 4 3 2 1
```

Summary:

```
Step Count Summary for Input Size 5:
-------------------------------------------------
| Algorithm       | Ascending Steps | Descending Steps |
-------------------------------------------------
| Bubble Sort     |             50 |             86 |
| Selection Sort  |             58 |             58 |
| Insertion Sort  |             40 |             40 |
-------------------------------------------------
```