LAB TASK

TASK NO: 4

SUBJECT: DATA STRUCTURE AND ALGORITHM LAB

SUBMITTED TO: HIJAB DURRANI

SUBMITTED BY: MISBAH ULLAH JAN

CLASS CODE: BSSE-2024A

SEMESTER: 3RD

DATE: 11/20/2025

DEPARTMENT OF: SOFTWARE ENGINEERING

CECOS UNIVERSITY HAYATABAD PESHAWAR

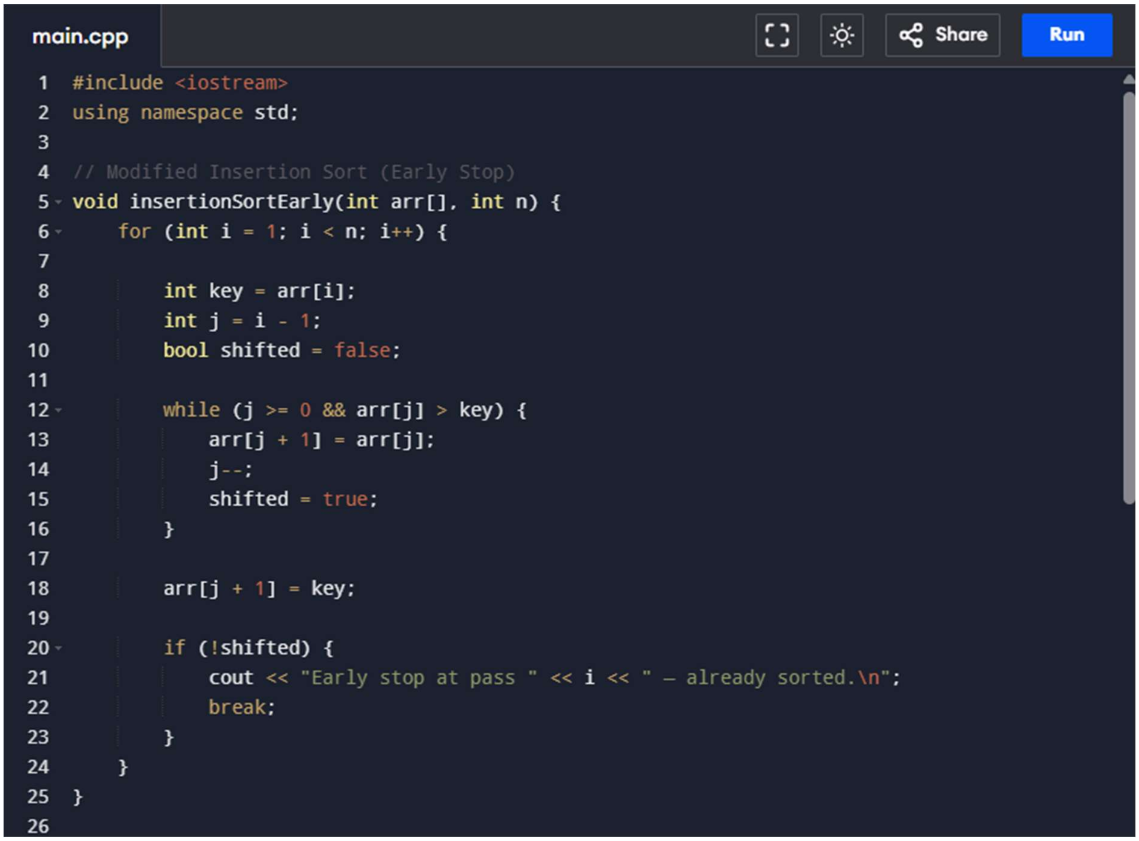
LAB TASK 1:

1. **Modify the Insertion Sort so it stops early if no shifting occurs during a pass (which means the list is already sorted).**

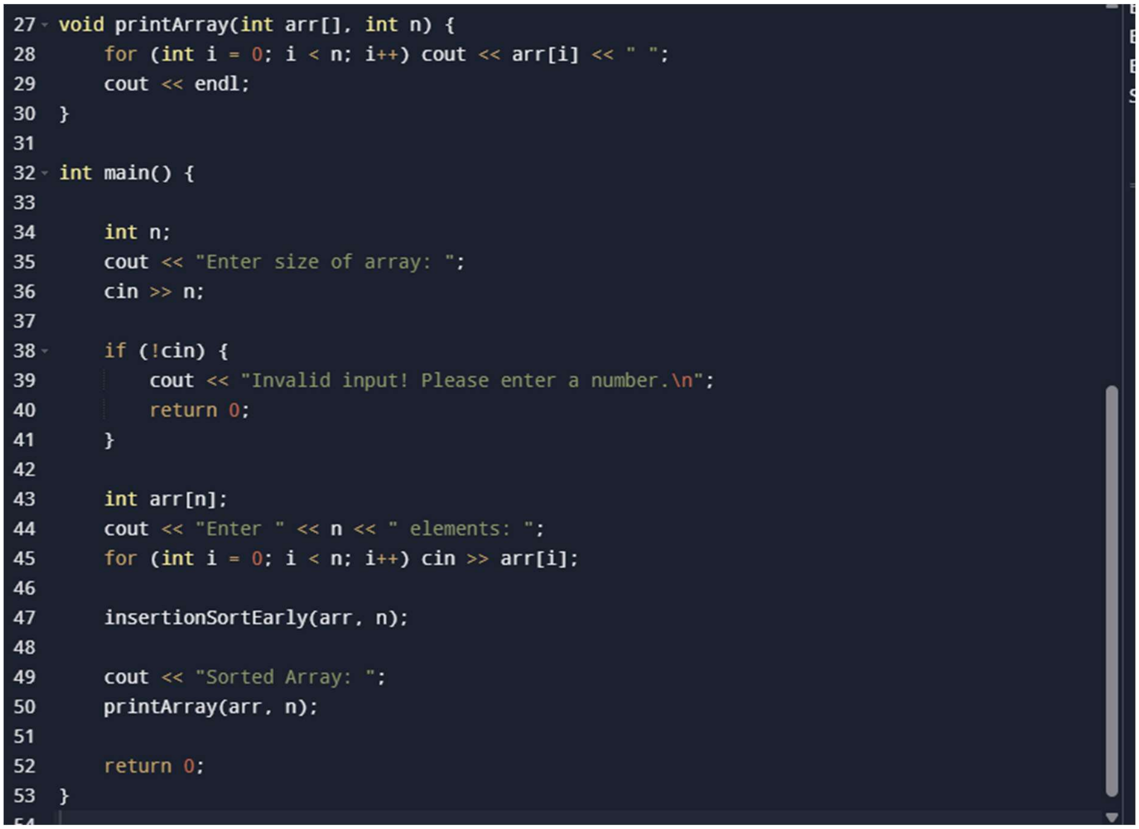
Modified Insertion Sort (Early Termination):

Algorithm: Modified Insertion Sort with Early Stop

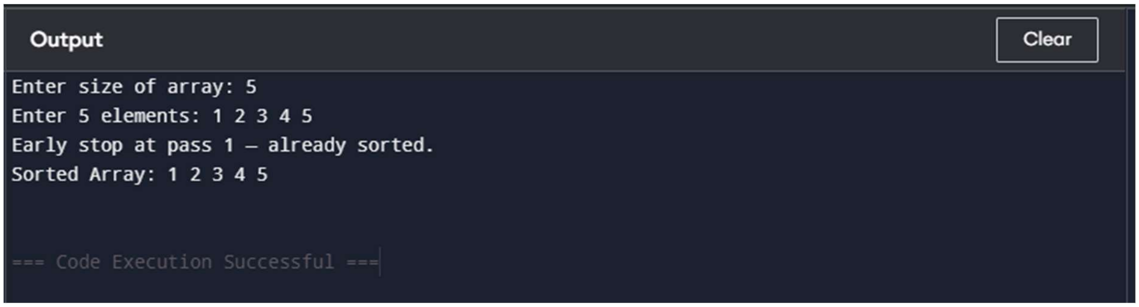
1. Start from the second element (index 1).
2. For each pass:
   a. Set a flag shifted = false.
   b. Store the current element in key.
   c. Compare key with elements in the sorted left part.
   d. While the element on the left is greater than key,
      - shift the element one position to the right,
      - set shifted = true.
        e. Insert key in its correct position.
3. After every outer loop pass:
   - If shifted == false, it means no movement happened → the array is already sorted.
   - Stop the algorithm early.
4. Continue until all elements are sorted or early-stop occurs.



```cpp
#include <iostream>
using namespace std;

// Modified Insertion Sort (Early Stop)
void insertionSortEarly(int arr[], int n) {
    for (int i = 1; i < n; i++) {

        int key = arr[i];
        int j = i - 1;
        bool shifted = false;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
            shifted = true;
        }

        arr[j + 1] = key;

        if (!shifted) {
            cout << "Early stop at pass " << i << " — already sorted.\n";
            break;
        }
    }
}

```



```cpp
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) cout << arr[i] << " ";
    cout << endl;
}

int main() {

    int n;
    cout << "Enter size of array: ";
    cin >> n;

    if (!cin) {
        cout << "Invalid input! Please enter a number.\n";
        return 0;
    }

    int arr[n];
    cout << "Enter " << n << " elements: ";
    for (int i = 0; i < n; i++) cin >> arr[i];

    insertionSortEarly(arr, n);

    cout << "Sorted Array: ";
    printArray(arr, n);

    return 0;
}
```



```
Enter size of array: 5
Enter 5 elements: 1 2 3 4 5
Early stop at pass 1 — already sorted.
Sorted Array: 1 2 3 4 5


=== Code Execution Successful ===
```

Task 2

2. Modify your Merge Sort, Quick Sort, algorithms to sort an array in descending order instead of ascending.

1. Algorithm for Descending Merge Sort

Algorithm: Merge Sort (Descending Order)

1. Divide the array into two halves.
2. Recursively divide until each subarray has 1 element.
3. Merge subarrays but while comparing,
   - choose the larger element first.
   - use >= instead of <=.
4. Continue merging until the full array is sorted in descending order.

2. Algorithm for Descending Quick Sort

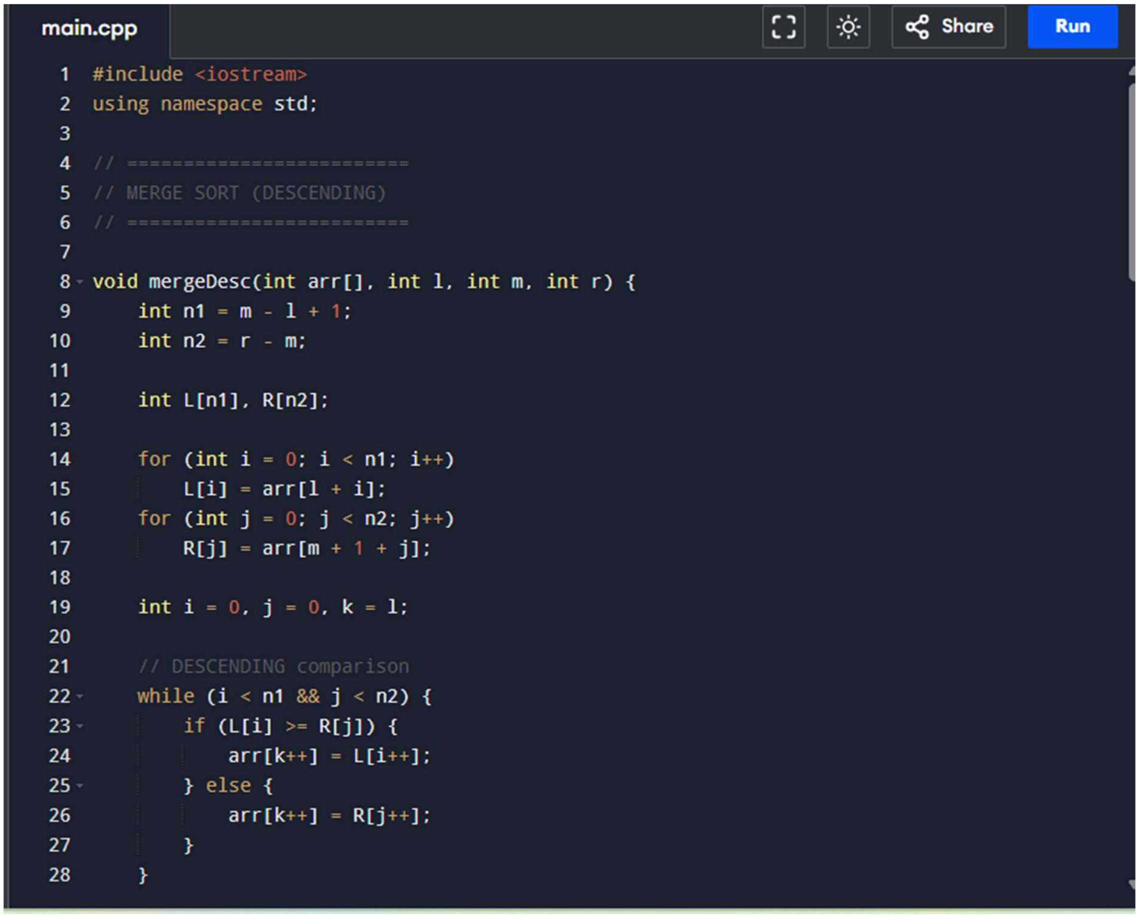
Algorithm: Quick Sort (Descending Order)

1. Select a pivot (last element usually).
2. Rearrange array so that:
   - elements greater than pivot go to the left,
   - elements smaller go to the right.

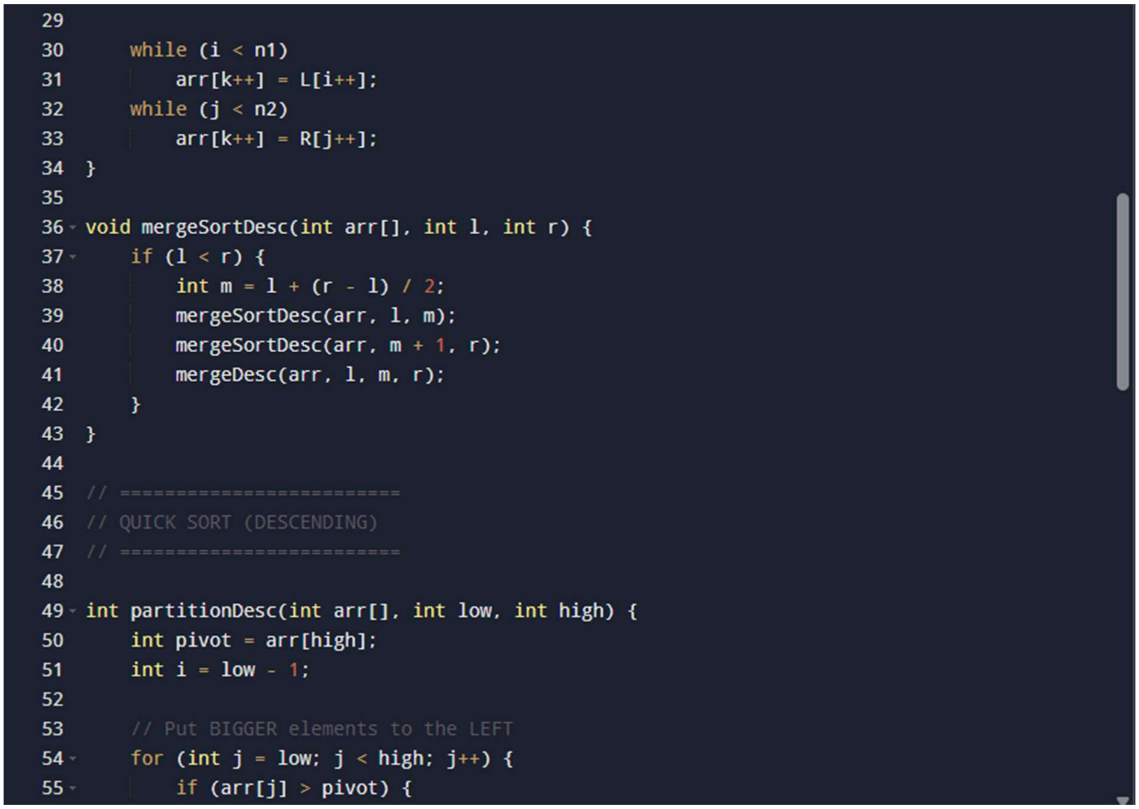
3. Put pivot in its correct descending position.

4. Recursively apply Quick Sort on left and right partitions.

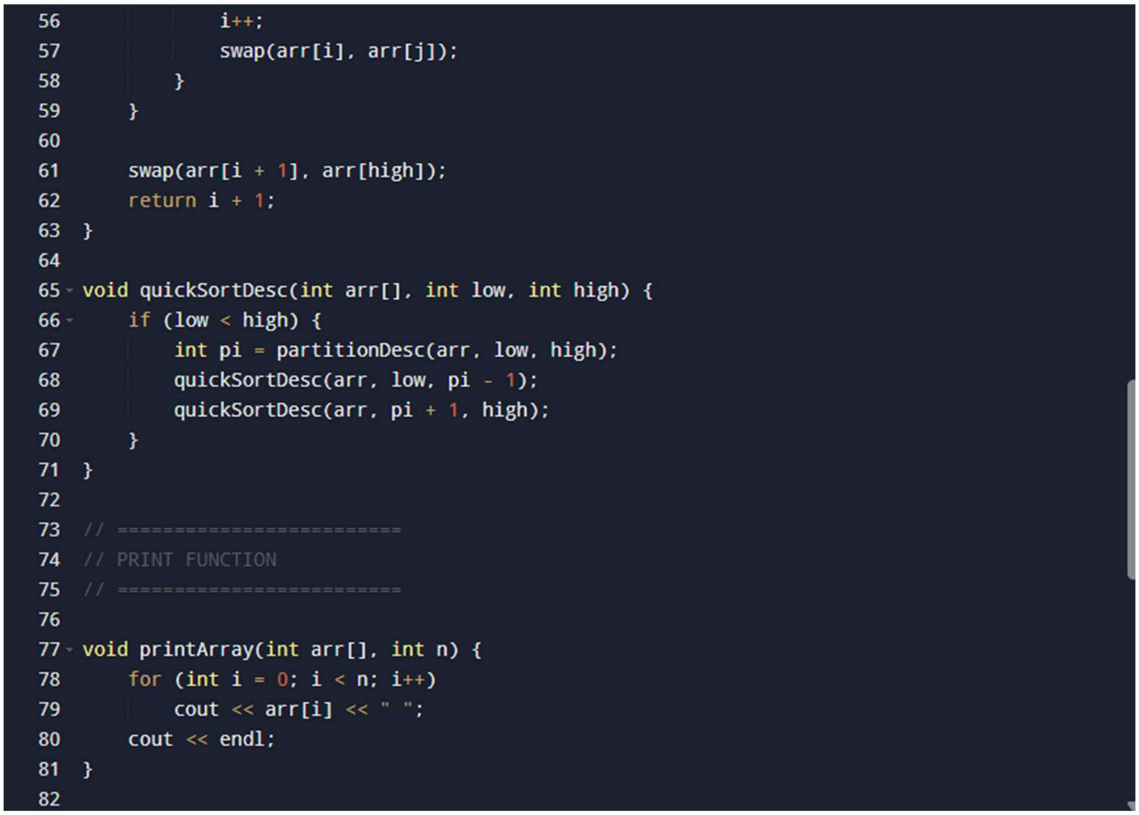
5. Stop when subarrays have size 0 or 1.



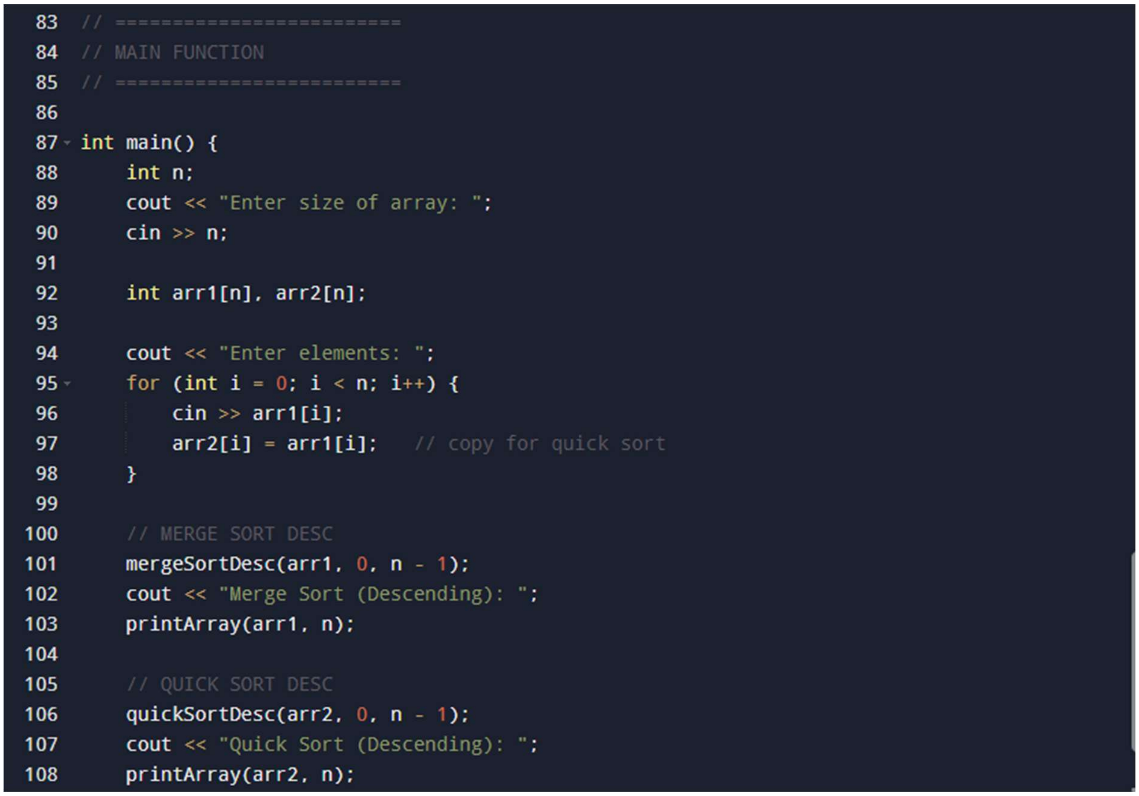
```cpp
#include <iostream>
using namespace std;

// ========================
// MERGE SORT (DESCENDING)
// ========================

void mergeDesc(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[n1], R[n2];

    for (int i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    int i = 0, j = 0, k = l;

    // DESCENDING comparison
    while (i < n1 && j < n2) {
        if (L[i] >= R[j]) {
            arr[k++] = L[i++];
        } else {
            arr[k++] = R[j++];
        }
    }
```



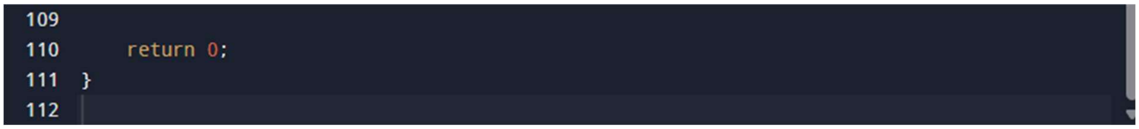
```cpp

    while (i < n1)
        arr[k++] = L[i++];
    while (j < n2)
        arr[k++] = R[j++];
}

void mergeSortDesc(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSortDesc(arr, l, m);
        mergeSortDesc(arr, m + 1, r);
        mergeDesc(arr, l, m, r);
    }
}

// ========================
// QUICK SORT (DESCENDING)
// ========================

int partitionDesc(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;

    // Put BIGGER elements to the LEFT
    for (int j = low; j < high; j++) {
        if (arr[j] > pivot) {
```



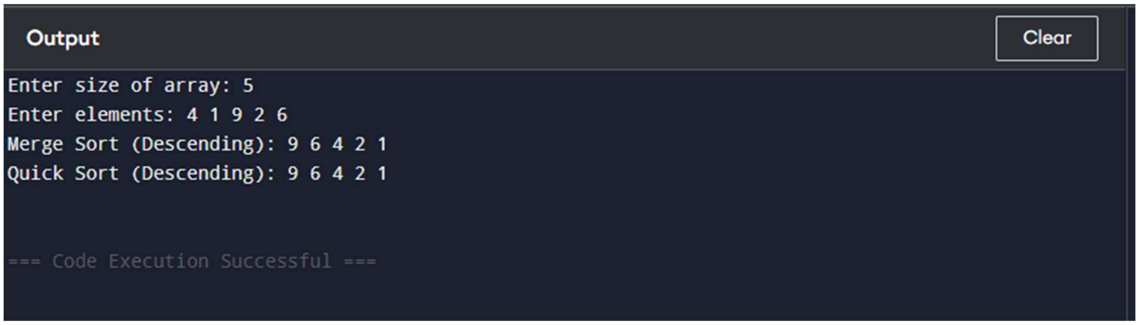
```cpp
            i++;
            swap(arr[i], arr[j]);
        }
    }

    swap(arr[i + 1], arr[high]);
    return i + 1;
}

void quickSortDesc(int arr[], int low, int high) {
    if (low < high) {
        int pi = partitionDesc(arr, low, high);
        quickSortDesc(arr, low, pi - 1);
        quickSortDesc(arr, pi + 1, high);
    }
}

// ========================
// PRINT FUNCTION
// ========================

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

```



```
// ========================
// MAIN FUNCTION
// ========================

int main() {
    int n;
    cout << "Enter size of array: ";
    cin >> n;

    int arr1[n], arr2[n];

    cout << "Enter elements: ";
    for (int i = 0; i < n; i++) {
        cin >> arr1[i];
        arr2[i] = arr1[i];   // copy for quick sort
    }

    // MERGE SORT DESC
    mergeSortDesc(arr1, 0, n - 1);
    cout << "Merge Sort (Descending): ";
    printArray(arr1, n);

    // QUICK SORT DESC
    quickSortDesc(arr2, 0, n - 1);
    cout << "Quick Sort (Descending): ";
    printArray(arr2, n);
```



```

    return 0;
}

```



```
Enter size of array: 5
Enter elements: 4 1 9 2 6
Merge Sort (Descending): 9 6 4 2 1
Quick Sort (Descending): 9 6 4 2 1


=== Code Execution Successful ===
```

Task 3

Write a program that:

1. Takes a list of 10 random numbers.

2. Sorts it using Insertion Sort, Merge Sort, Quick Sort(all ascending).

3. Counts and prints the number of comparisons and swaps/moves each algorithm performs.

Count Comparisons & Swaps/Moves for Insertion, Merge, and Quick Sort (Ascending Order)

We must:

1. Take 10 random numbers from user.
2. Sort them using:
   - Insertion Sort
   - Merge Sort
   - Quick Sort
3. Count:
   - Comparisons
   - Moves or Swaps
4. Print results exactly like the example.

Algorithm Summary for Task 3

Insertion Sort Counting

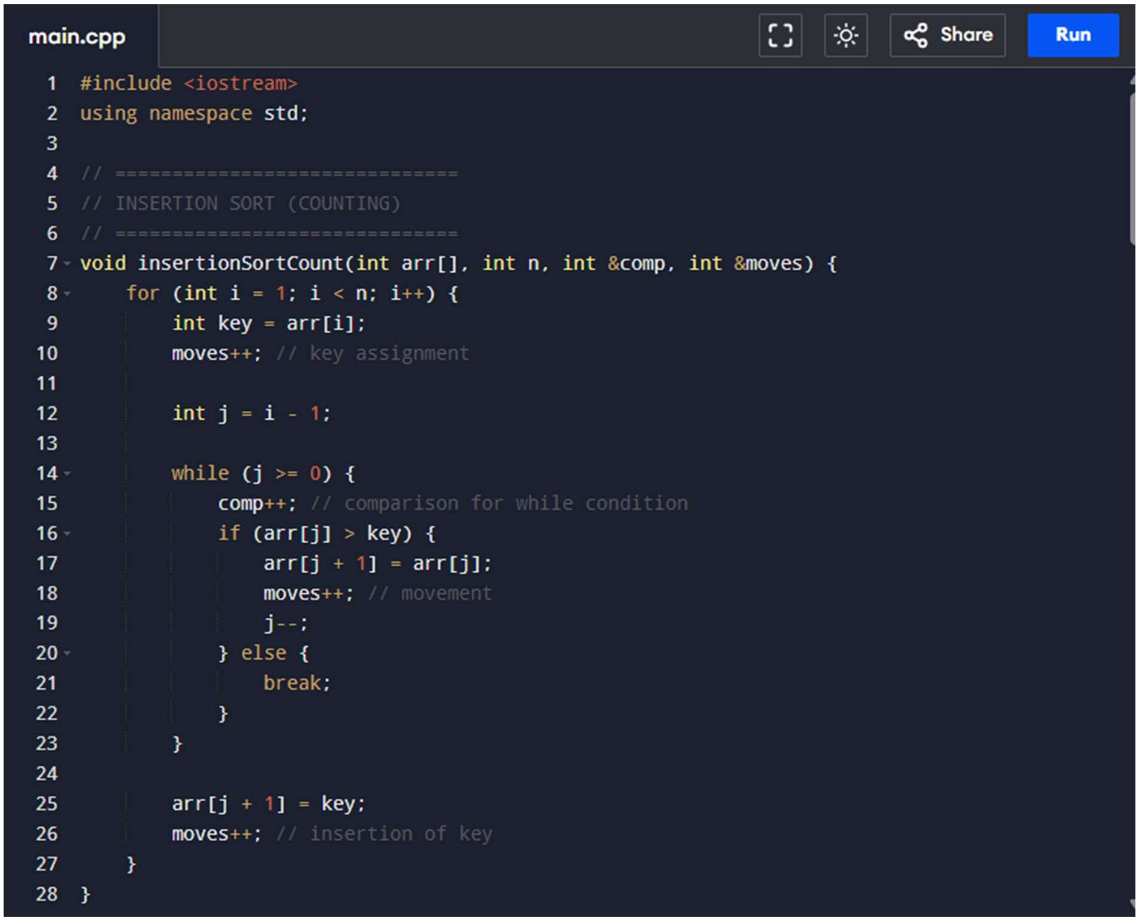
- Every time you compare → comp++
- Every time you move (shift) an element → moves++
- Every time you insert key → moves++

Merge Sort Counting

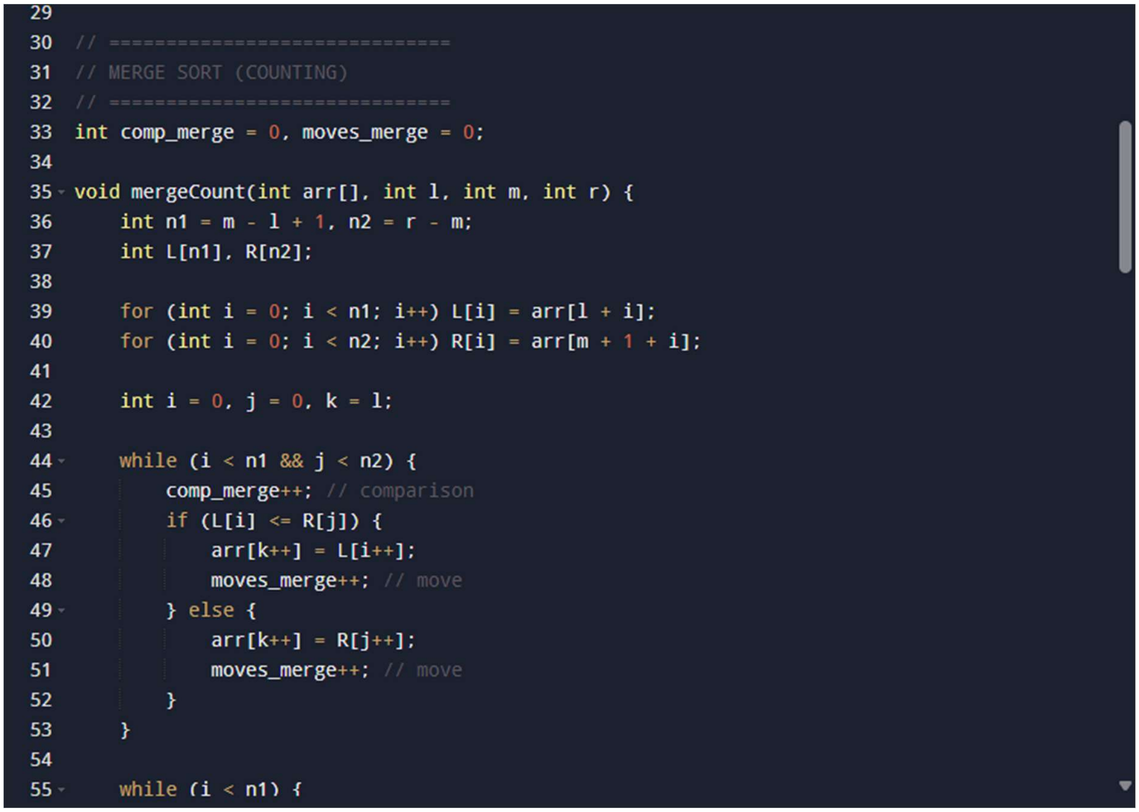
- Every comparison in merging → comp++
- Every time we copy an element into main array → moves++

Quick Sort Counting

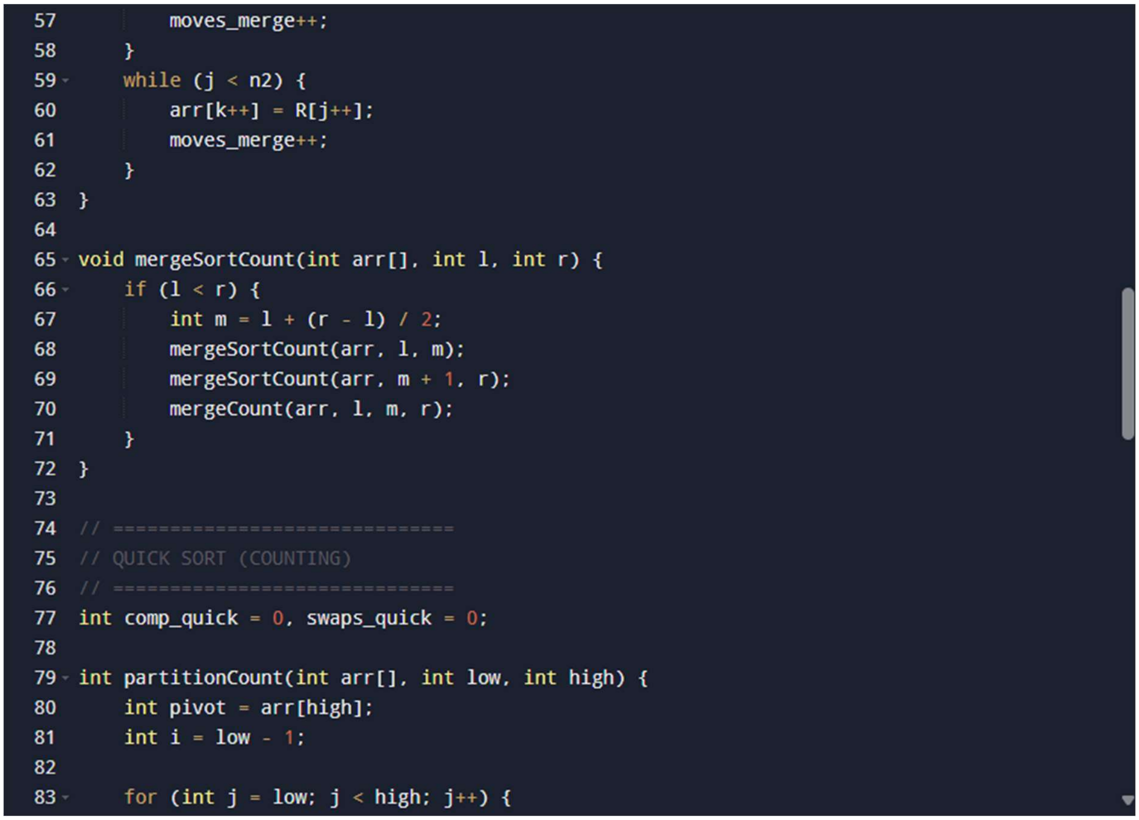
- Every comparison inside partition → comp++
- Every physical swap → swaps++



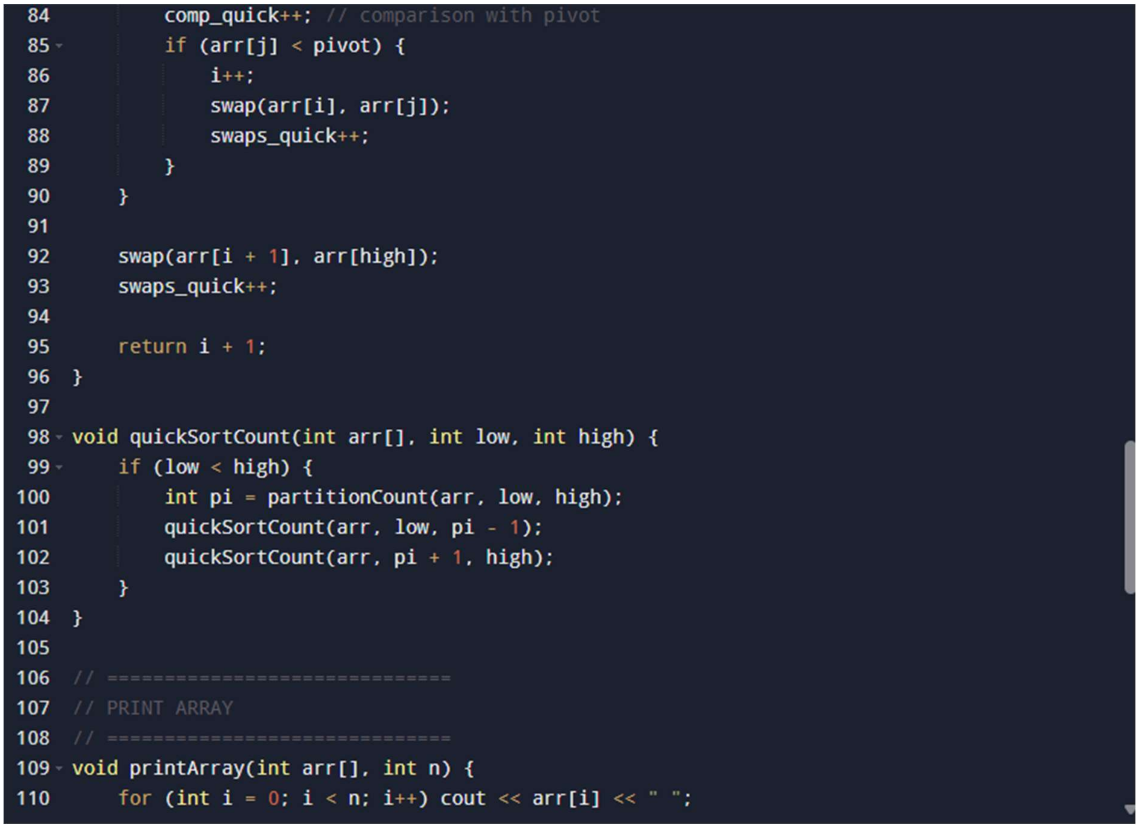
```cpp
#include <iostream>
using namespace std;

// ==============================
// INSERTION SORT (COUNTING)
// ==============================
void insertionSortCount(int arr[], int n, int &comp, int &moves) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        moves++; // key assignment

        int j = i - 1;

        while (j >= 0) {
            comp++; // comparison for while condition
            if (arr[j] > key) {
                arr[j + 1] = arr[j];
                moves++; // movement
                j--;
            } else {
                break;
            }
        }

        arr[j + 1] = key;
        moves++; // insertion of key
    }
}
```



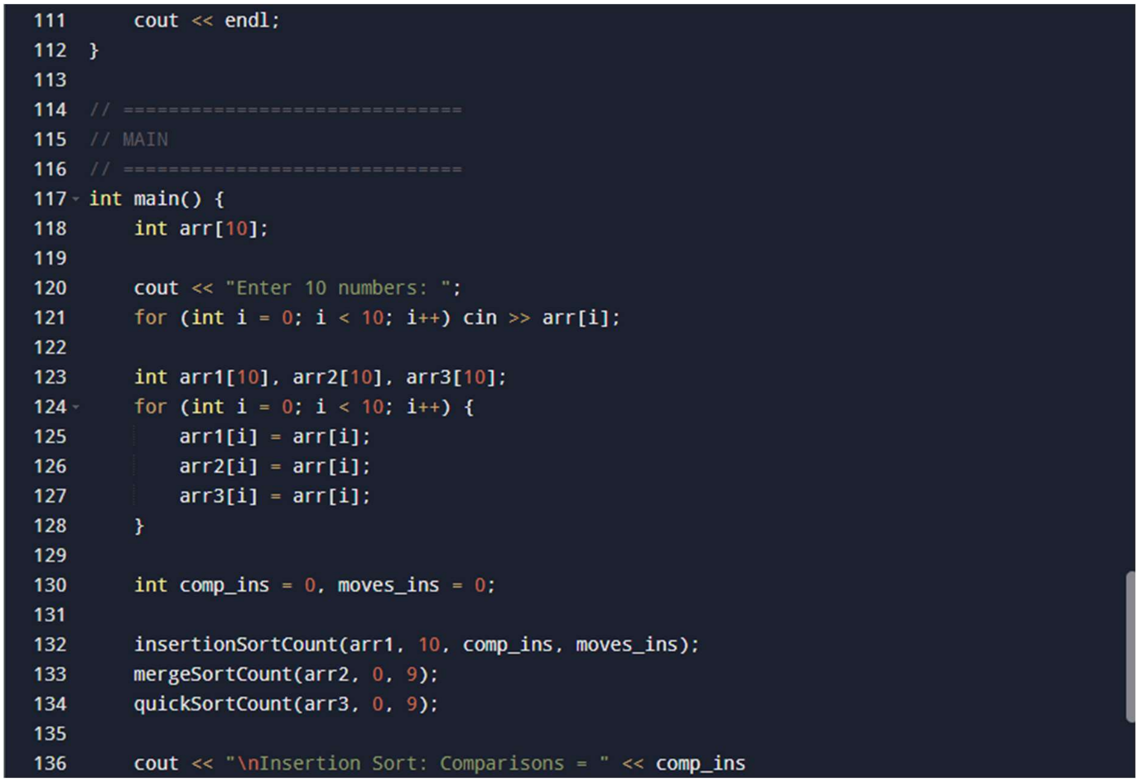
```

// ==============================
// MERGE SORT (COUNTING)
// ==============================
int comp_merge = 0, moves_merge = 0;

void mergeCount(int arr[], int l, int m, int r) {
    int n1 = m - l + 1, n2 = r - m;
    int L[n1], R[n2];

    for (int i = 0; i < n1; i++) L[i] = arr[l + i];
    for (int i = 0; i < n2; i++) R[i] = arr[m + 1 + i];

    int i = 0, j = 0, k = l;

    while (i < n1 && j < n2) {
        comp_merge++; // comparison
        if (L[i] <= R[j]) {
            arr[k++] = L[i++];
            moves_merge++; // move
        } else {
            arr[k++] = R[j++];
            moves_merge++; // move
        }
    }

    while (i < n1) {
```



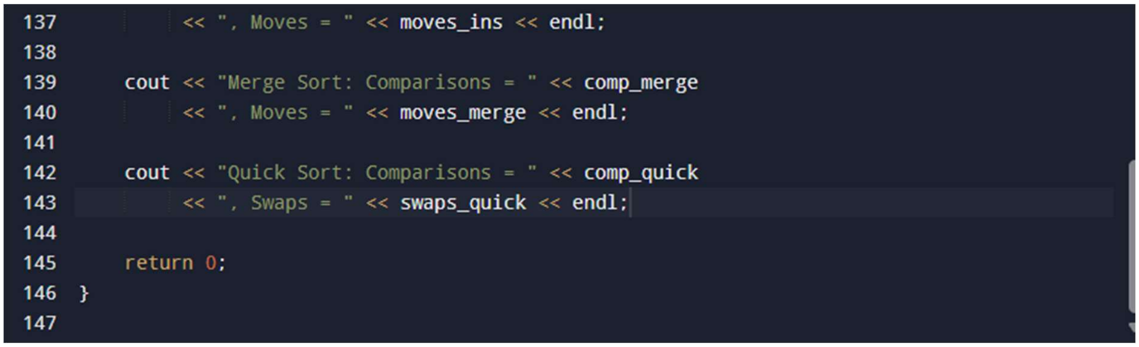
```
        moves_merge++;
    }
    while (j < n2) {
        arr[k++] = R[j++];
        moves_merge++;
    }
}

void mergeSortCount(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSortCount(arr, l, m);
        mergeSortCount(arr, m + 1, r);
        mergeCount(arr, l, m, r);
    }
}

// ==============================
// QUICK SORT (COUNTING)
// ==============================
int comp_quick = 0, swaps_quick = 0;

int partitionCount(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;

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



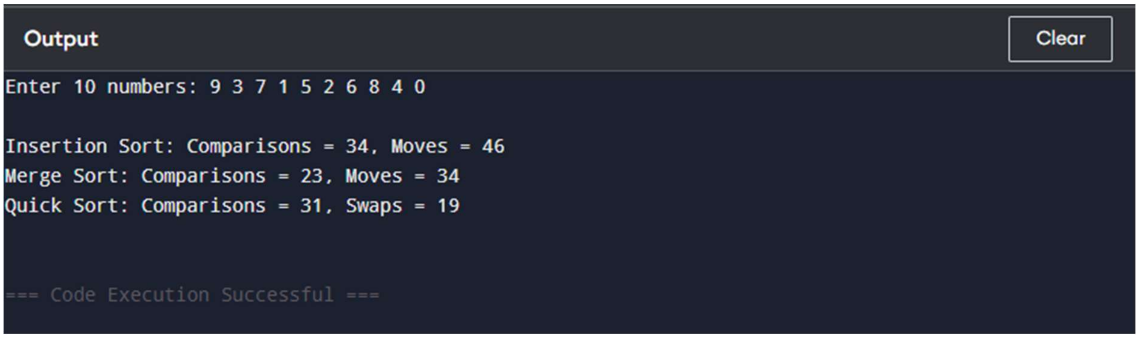
```
        comp_quick++; // comparison with pivot
        if (arr[j] < pivot) {
            i++;
            swap(arr[i], arr[j]);
            swaps_quick++;
        }
    }

    swap(arr[i + 1], arr[high]);
    swaps_quick++;

    return i + 1;
}

void quickSortCount(int arr[], int low, int high) {
    if (low < high) {
        int pi = partitionCount(arr, low, high);
        quickSortCount(arr, low, pi - 1);
        quickSortCount(arr, pi + 1, high);
    }
}

// ==============================
// PRINT ARRAY
// ==============================
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) cout << arr[i] << " ";
```



```
    cout << endl;
}

// ==============================
// MAIN
// ==============================
int main() {
    int arr[10];

    cout << "Enter 10 numbers: ";
    for (int i = 0; i < 10; i++) cin >> arr[i];

    int arr1[10], arr2[10], arr3[10];
    for (int i = 0; i < 10; i++) {
        arr1[i] = arr[i];
        arr2[i] = arr[i];
        arr3[i] = arr[i];
    }

    int comp_ins = 0, moves_ins = 0;

    insertionSortCount(arr1, 10, comp_ins, moves_ins);
    mergeSortCount(arr2, 0, 9);
    quickSortCount(arr3, 0, 9);

    cout << "\nInsertion Sort: Comparisons = " << comp_ins
```



```
        << ", Moves = " << moves_ins << endl;

    cout << "Merge Sort: Comparisons = " << comp_merge
        << ", Moves = " << moves_merge << endl;

    cout << "Quick Sort: Comparisons = " << comp_quick
        << ", Swaps = " << swaps_quick << endl;

    return 0;
}

```

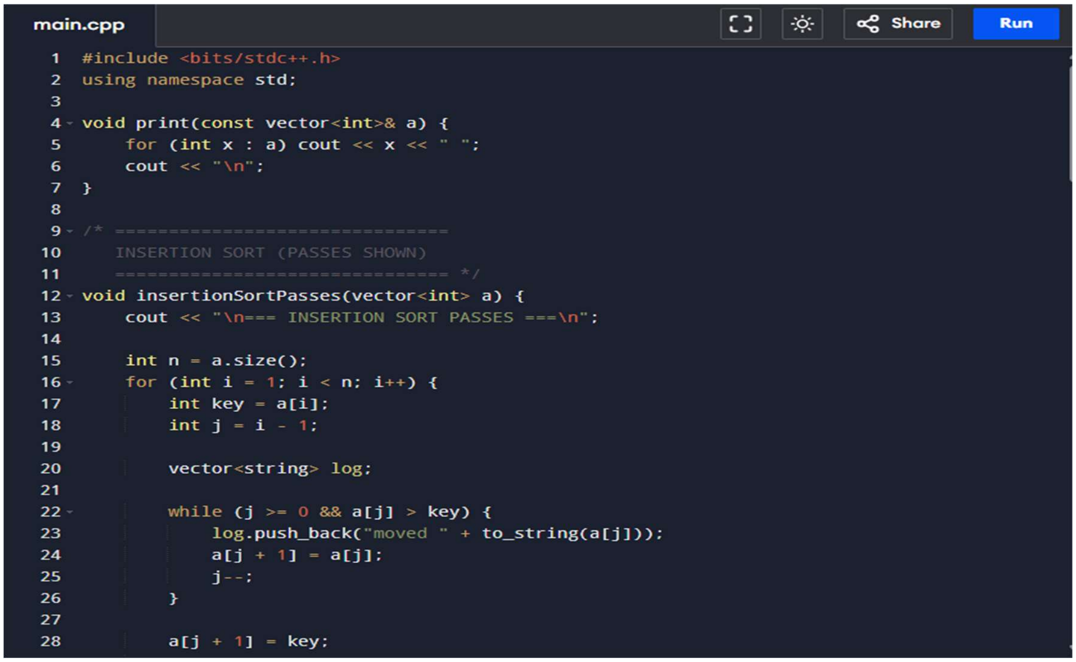


```
Enter 10 numbers: 9 3 7 1 5 2 6 8 4 0

Insertion Sort: Comparisons = 34, Moves = 46
Merge Sort: Comparisons = 23, Moves = 34
Quick Sort: Comparisons = 31, Swaps = 19


=== Code Execution Successful ===
```

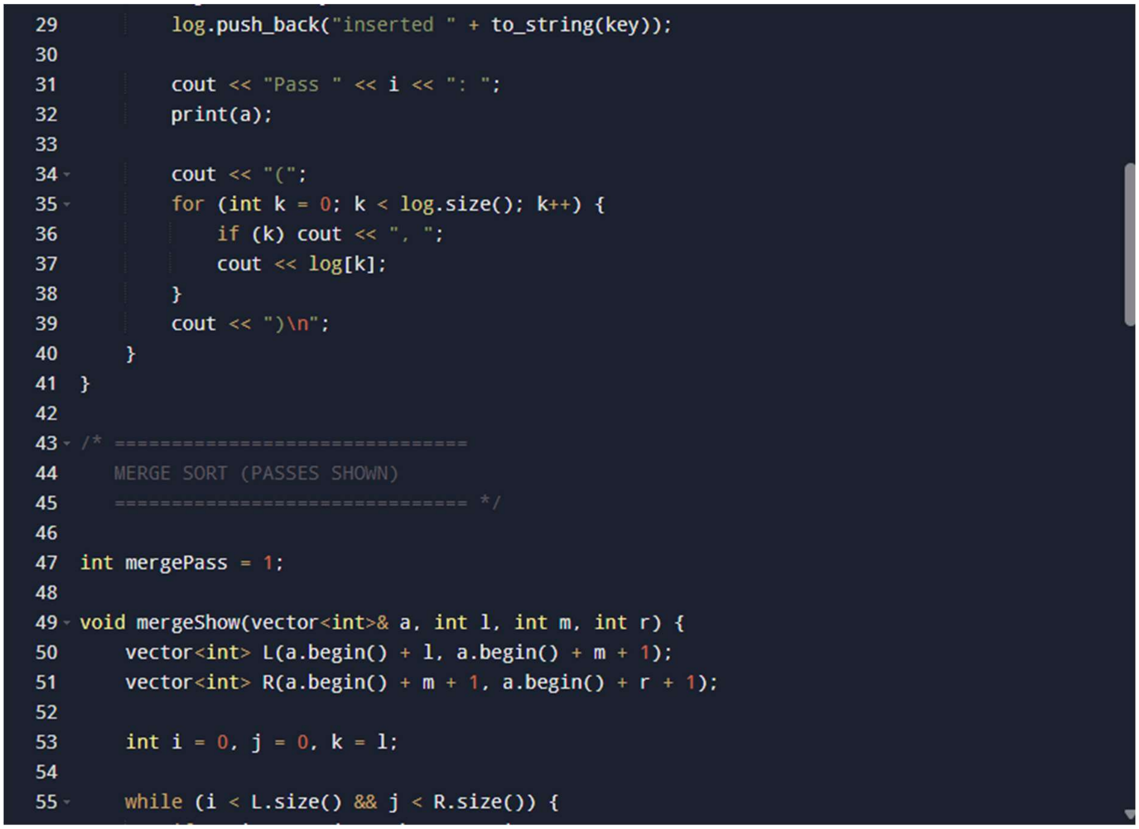
Task 4

4. Modify your Insertion Sort, Merge Sort, Quick Sort to:

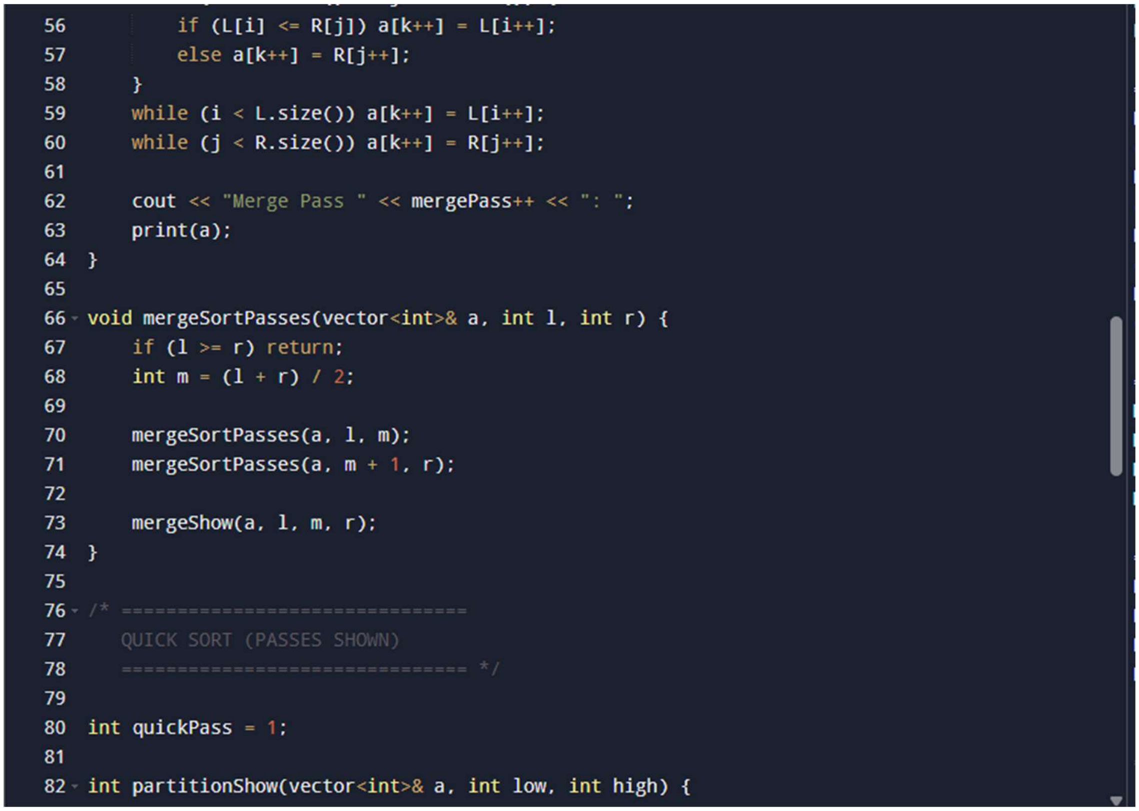
- **Print the array after each pass of the outer loop.**
- **Display which elements were swapped or moved in that pass.**



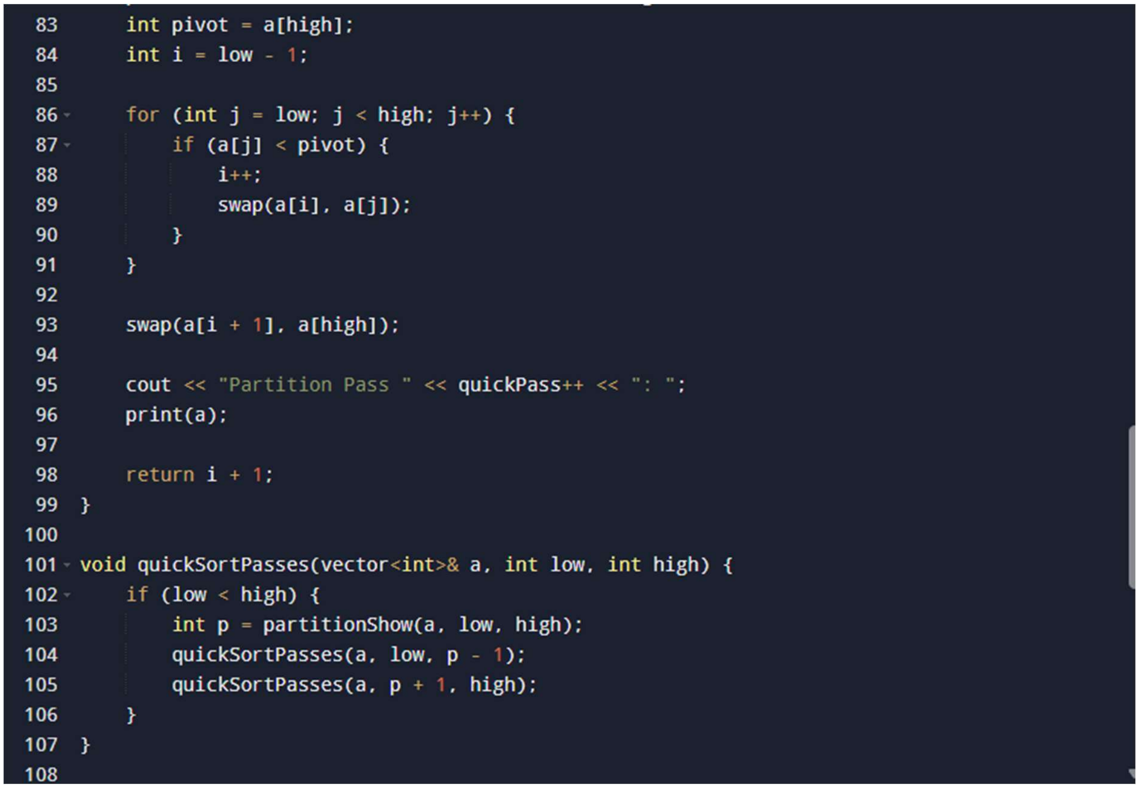
```
#include <bits/stdc++.h>
using namespace std;

void print(const vector<int>& a) {
    for (int x : a) cout << x << " ";
    cout << "\n";
}

/* ==============================
   INSERTION SORT (PASSES SHOWN)
   ============================== */
void insertionSortPasses(vector<int> a) {
    cout << "\n=== INSERTION SORT PASSES ===\n";

    int n = a.size();
    for (int i = 1; i < n; i++) {
        int key = a[i];
        int j = i - 1;

        vector<string> log;

        while (j >= 0 && a[j] > key) {
            log.push_back("moved " + to_string(a[j]));
            a[j + 1] = a[j];
            j--;
        }

        a[j + 1] = key;
```



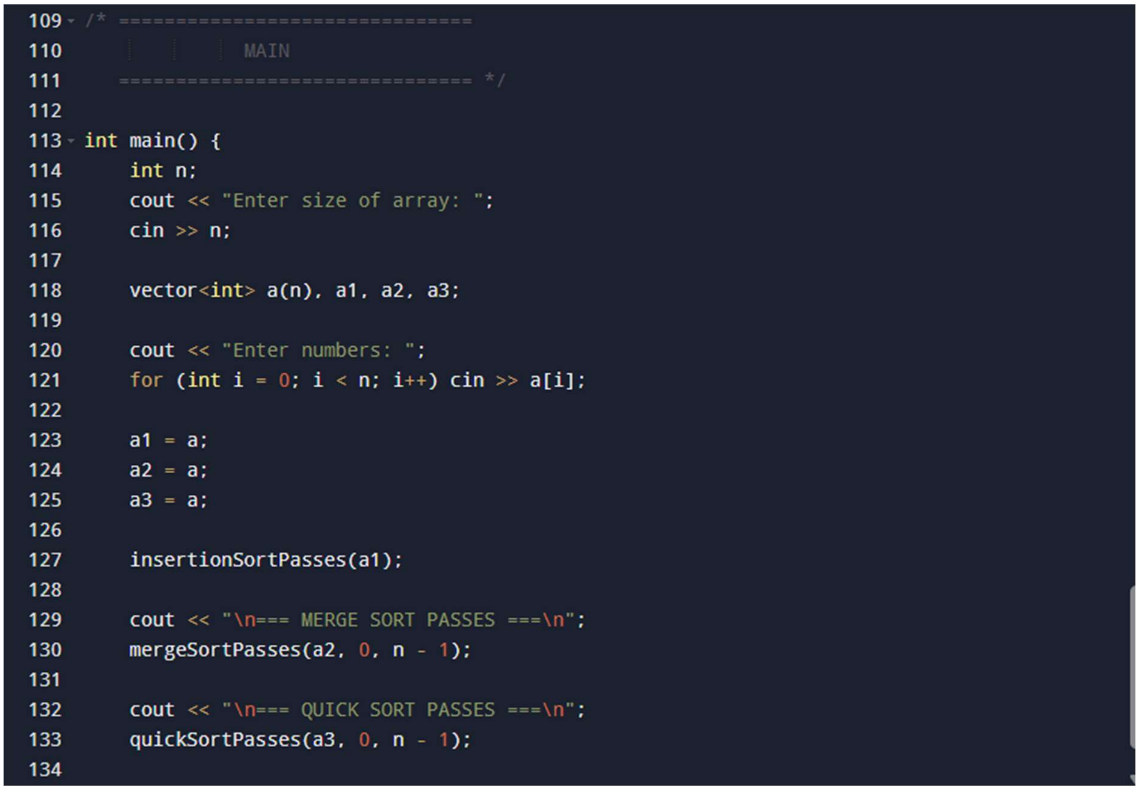
```
        log.push_back("inserted " + to_string(key));

        cout << "Pass " << i << ": ";
        print(a);

        cout << "(";
        for (int k = 0; k < log.size(); k++) {
            if (k) cout << ", ";
            cout << log[k];
        }
        cout << ")\n";
    }
}

/* ===============================
   MERGE SORT (PASSES SHOWN)
   =============================== */

int mergePass = 1;

void mergeShow(vector<int>& a, int l, int m, int r) {
    vector<int> L(a.begin() + l, a.begin() + m + 1);
    vector<int> R(a.begin() + m + 1, a.begin() + r + 1);

    int i = 0, j = 0, k = l;

    while (i < L.size() && j < R.size()) {
```



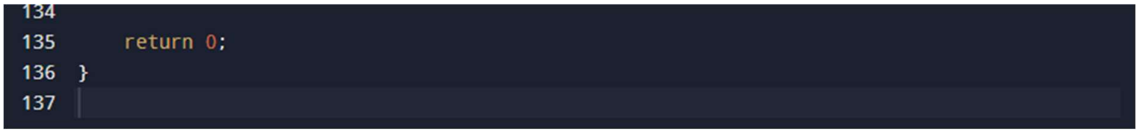
```
        if (L[i] <= R[j]) a[k++] = L[i++];
        else a[k++] = R[j++];
    }
    while (i < L.size()) a[k++] = L[i++];
    while (j < R.size()) a[k++] = R[j++];

    cout << "Merge Pass " << mergePass++ << ": ";
    print(a);
}

void mergeSortPasses(vector<int>& a, int l, int r) {
    if (l >= r) return;
    int m = (l + r) / 2;

    mergeSortPasses(a, l, m);
    mergeSortPasses(a, m + 1, r);

    mergeShow(a, l, m, r);
}

/* ===============================
   QUICK SORT (PASSES SHOWN)
   =============================== */

int quickPass = 1;

int partitionShow(vector<int>& a, int low, int high) {
```



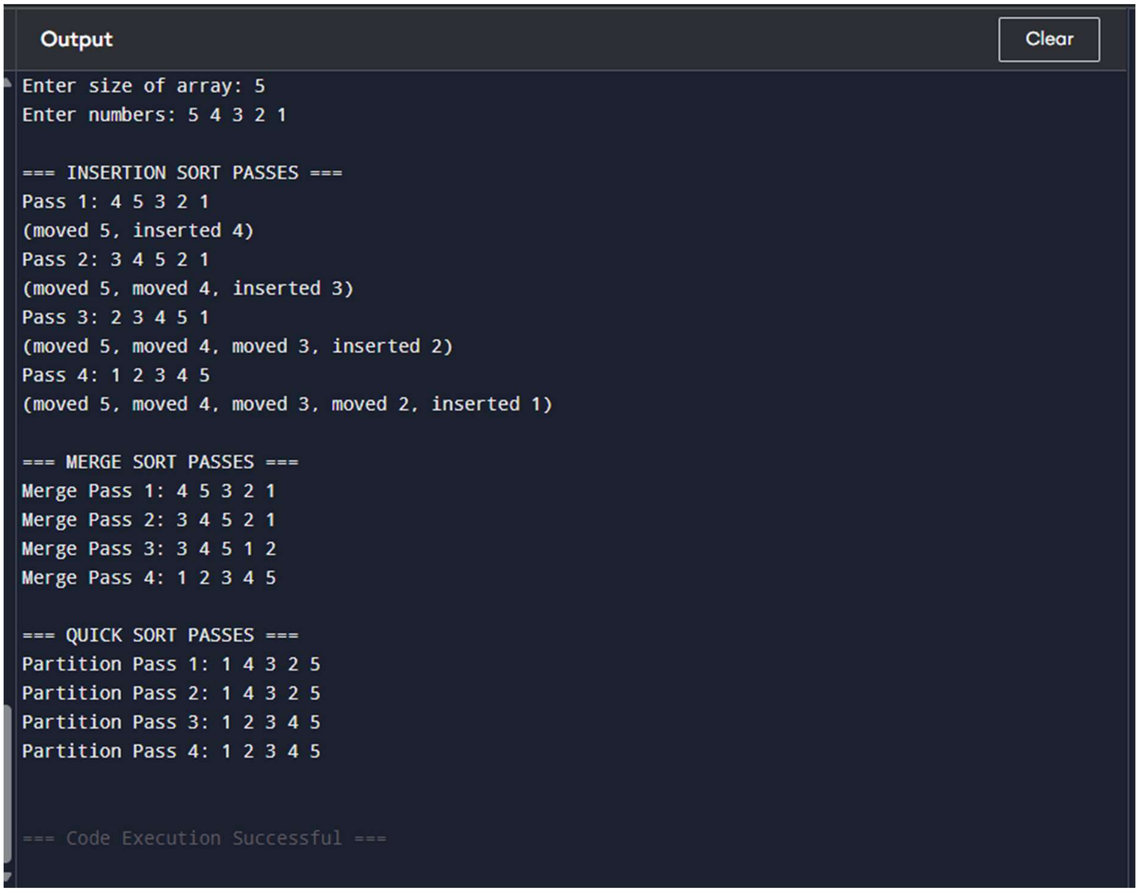
```cpp
    int pivot = a[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (a[j] < pivot) {
            i++;
            swap(a[i], a[j]);
        }
    }

    swap(a[i + 1], a[high]);

    cout << "Partition Pass " << quickPass++ << ": ";
    print(a);

    return i + 1;
}

void quickSortPasses(vector<int>& a, int low, int high) {
    if (low < high) {
        int p = partitionShow(a, low, high);
        quickSortPasses(a, low, p - 1);
        quickSortPasses(a, p + 1, high);
    }
}

```



```cpp
/* ================================
                MAIN
   ================================ */

int main() {
    int n;
    cout << "Enter size of array: ";
    cin >> n;

    vector<int> a(n), a1, a2, a3;

    cout << "Enter numbers: ";
    for (int i = 0; i < n; i++) cin >> a[i];

    a1 = a;
    a2 = a;
    a3 = a;

    insertionSortPasses(a1);

    cout << "\n=== MERGE SORT PASSES ===\n";
    mergeSortPasses(a2, 0, n - 1);

    cout << "\n=== QUICK SORT PASSES ===\n";
    quickSortPasses(a3, 0, n - 1);

```



```cpp

    return 0;
}

```



```
Enter size of array: 5
Enter numbers: 5 4 3 2 1

=== INSERTION SORT PASSES ===
Pass 1: 4 5 3 2 1
(moved 5, inserted 4)
Pass 2: 3 4 5 2 1
(moved 5, moved 4, inserted 3)
Pass 3: 2 3 4 5 1
(moved 5, moved 4, moved 3, inserted 2)
Pass 4: 1 2 3 4 5
(moved 5, moved 4, moved 3, moved 2, inserted 1)

=== MERGE SORT PASSES ===
Merge Pass 1: 4 5 3 2 1
Merge Pass 2: 3 4 5 2 1
Merge Pass 3: 3 4 5 1 2
Merge Pass 4: 1 2 3 4 5

=== QUICK SORT PASSES ===
Partition Pass 1: 1 4 3 2 5
Partition Pass 2: 1 4 3 2 5
Partition Pass 3: 1 2 3 4 5
Partition Pass 4: 1 2 3 4 5


=== Code Execution Successful ===
```

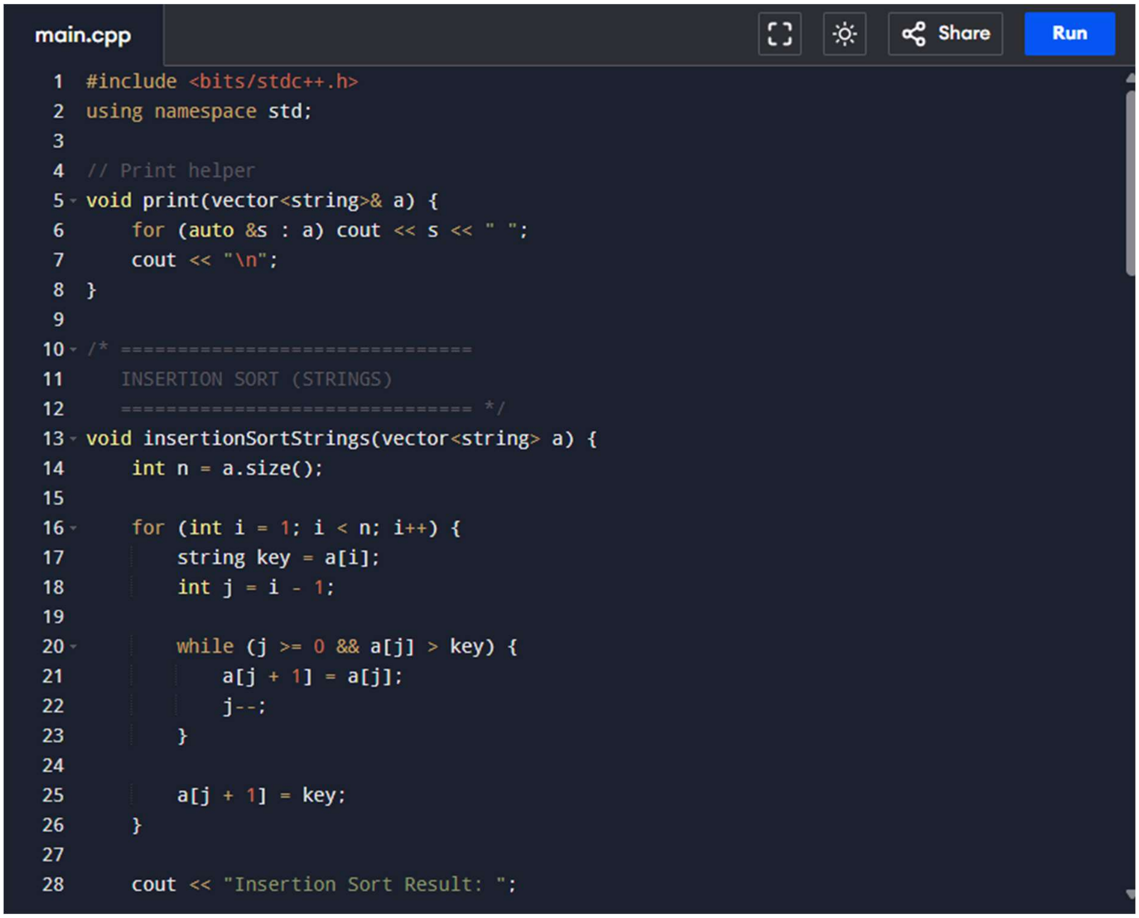
Task 5

5. Use Insertion Sort, Merge Sort, Quick Sort, to sort a list of strings

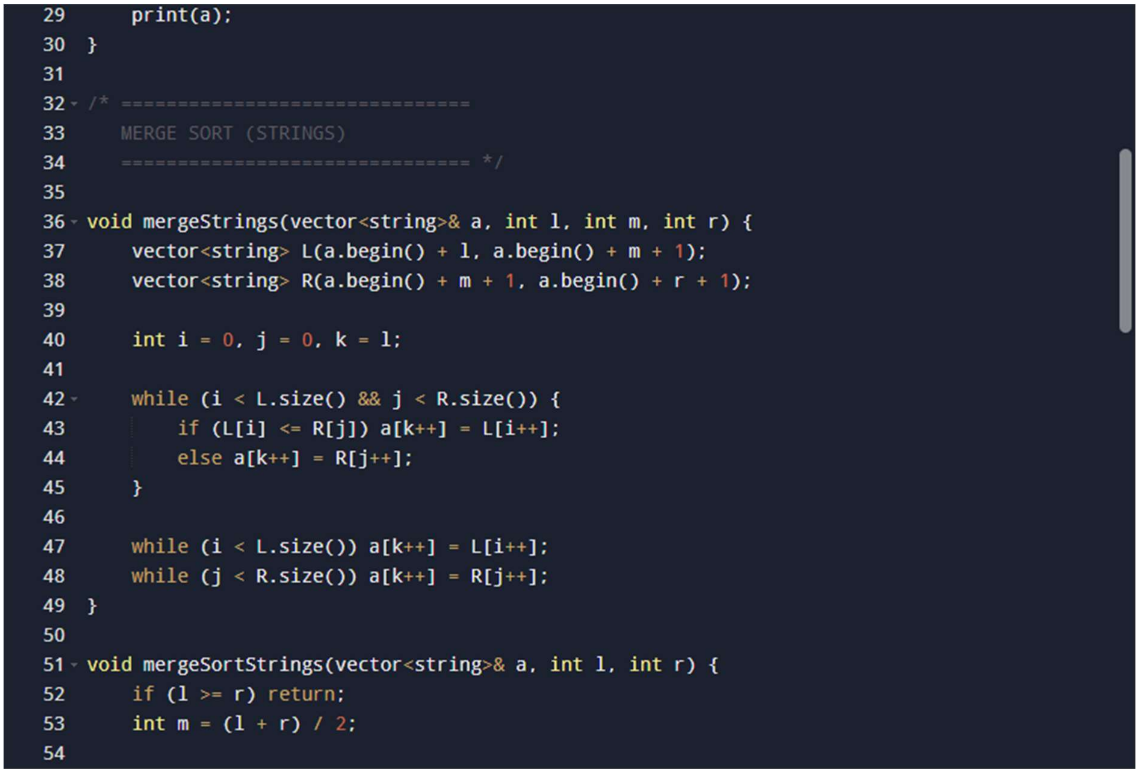
alphabetically.

Example Input:

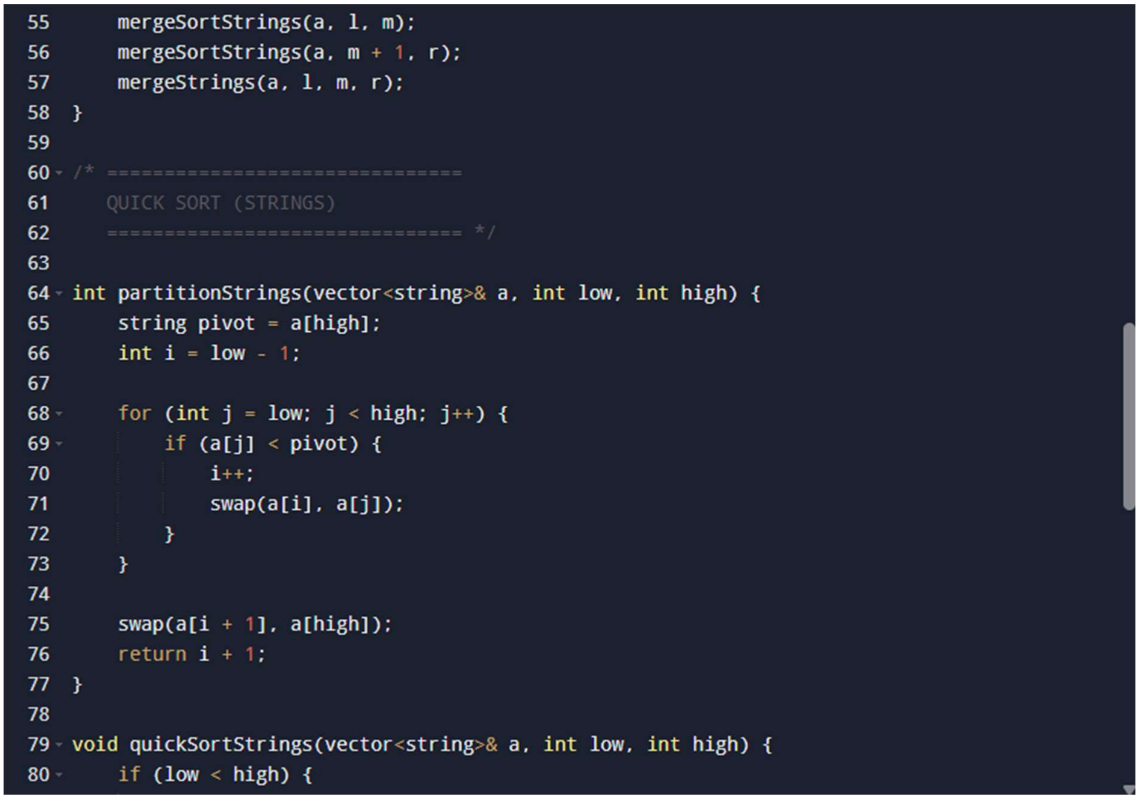
[pear, apple, banana,mango]



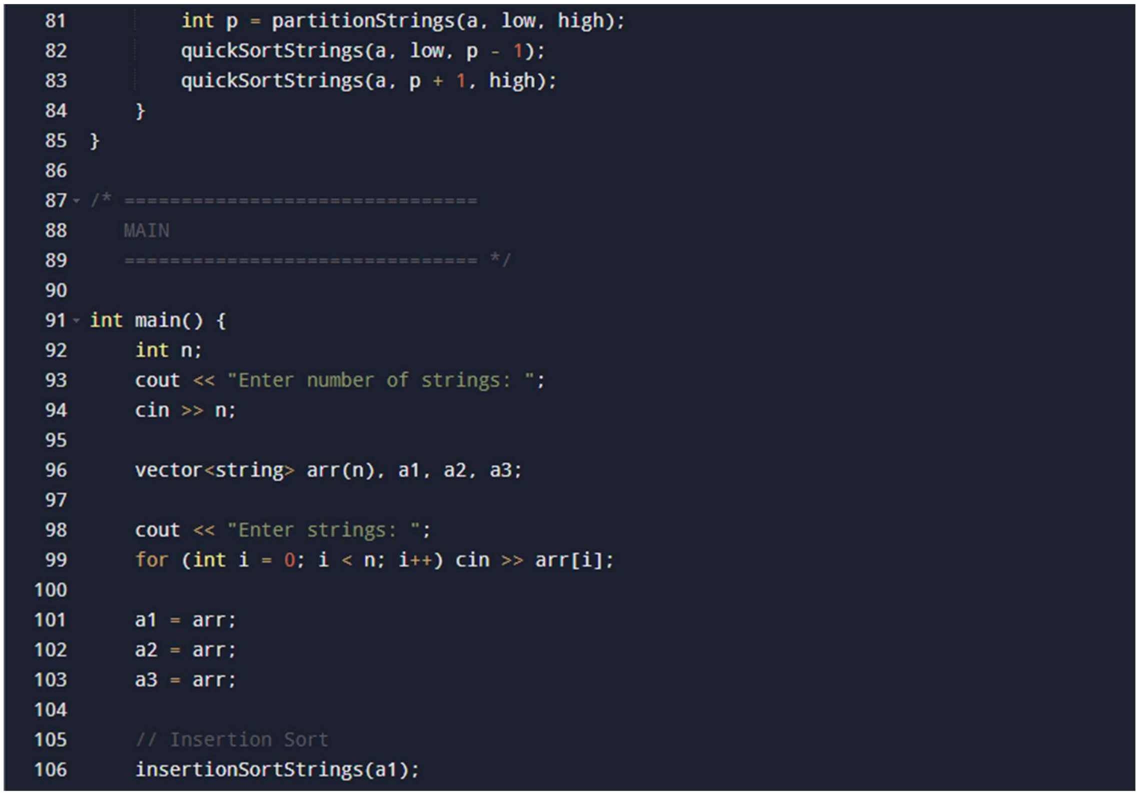
```cpp
#include <bits/stdc++.h>
using namespace std;

// Print helper
void print(vector<string>& a) {
    for (auto &s : a) cout << s << " ";
    cout << "\n";
}

/* ===============================
   INSERTION SORT (STRINGS)
   =============================== */
void insertionSortStrings(vector<string> a) {
    int n = a.size();

    for (int i = 1; i < n; i++) {
        string key = a[i];
        int j = i - 1;

        while (j >= 0 && a[j] > key) {
            a[j + 1] = a[j];
            j--;
        }

        a[j + 1] = key;
    }

    cout << "Insertion Sort Result: ";
```



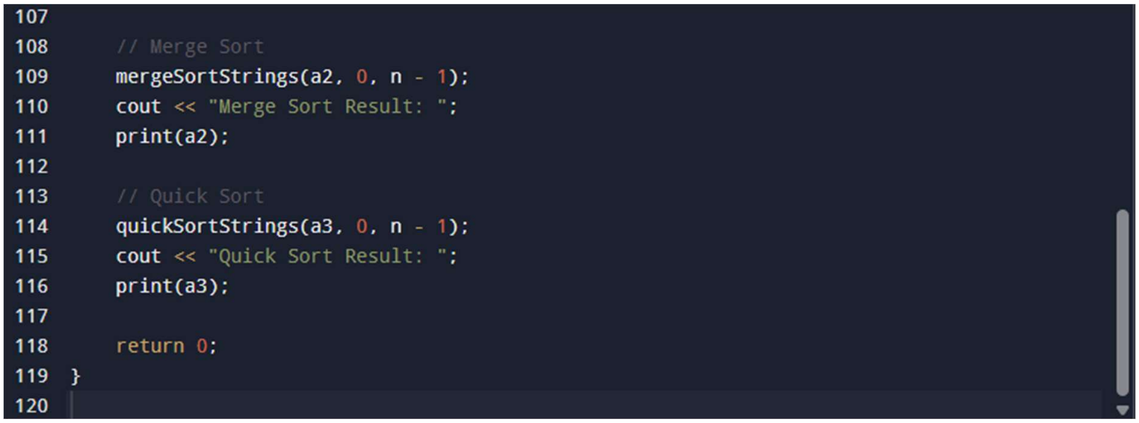
```cpp
    print(a);
}

/* ===============================
   MERGE SORT (STRINGS)
   =============================== */

void mergeStrings(vector<string>& a, int l, int m, int r) {
    vector<string> L(a.begin() + l, a.begin() + m + 1);
    vector<string> R(a.begin() + m + 1, a.begin() + r + 1);

    int i = 0, j = 0, k = l;

    while (i < L.size() && j < R.size()) {
        if (L[i] <= R[j]) a[k++] = L[i++];
        else a[k++] = R[j++];
    }

    while (i < L.size()) a[k++] = L[i++];
    while (j < R.size()) a[k++] = R[j++];
}

void mergeSortStrings(vector<string>& a, int l, int r) {
    if (l >= r) return;
    int m = (l + r) / 2;

```



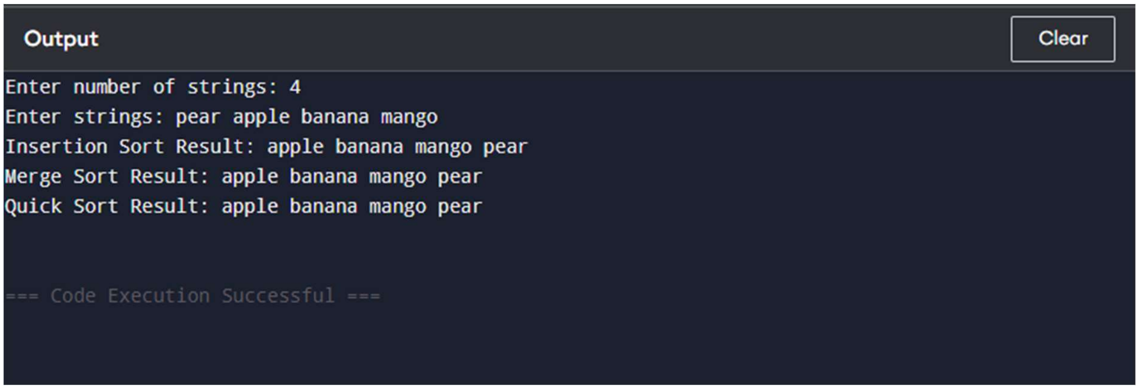
```cpp
    mergeSortStrings(a, l, m);
    mergeSortStrings(a, m + 1, r);
    mergeStrings(a, l, m, r);
}

/* ===============================
   QUICK SORT (STRINGS)
   =============================== */

int partitionStrings(vector<string>& a, int low, int high) {
    string pivot = a[high];
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (a[j] < pivot) {
            i++;
            swap(a[i], a[j]);
        }
    }

    swap(a[i + 1], a[high]);
    return i + 1;
}

void quickSortStrings(vector<string>& a, int low, int high) {
    if (low < high) {
```



```cpp
        int p = partitionStrings(a, low, high);
        quickSortStrings(a, low, p - 1);
        quickSortStrings(a, p + 1, high);
    }
}

/* ===============================
   MAIN
   =============================== */

int main() {
    int n;
    cout << "Enter number of strings: ";
    cin >> n;

    vector<string> arr(n), a1, a2, a3;

    cout << "Enter strings: ";
    for (int i = 0; i < n; i++) cin >> arr[i];

    a1 = arr;
    a2 = arr;
    a3 = arr;

    // Insertion Sort
    insertionSortStrings(a1);
```



```cpp

    // Merge Sort
    mergeSortStrings(a2, 0, n - 1);
    cout << "Merge Sort Result: ";
    print(a2);

    // Quick Sort
    quickSortStrings(a3, 0, n - 1);
    cout << "Quick Sort Result: ";
    print(a3);

    return 0;
}

```



```
Enter number of strings: 4
Enter strings: pear apple banana mango
Insertion Sort Result: apple banana mango pear
Merge Sort Result: apple banana mango pear
Quick Sort Result: apple banana mango pear


=== Code Execution Successful ===
```