

LAB TASK



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,
 - o shift the element one position to the right,
 - o set shifted = true.
 - e. Insert key in its correct position.
3. After every outer loop pass:
 - o If shifted == false, it means no movement happened → the array is already sorted.
 - o Stop the algorithm early.
4. Continue until all elements are sorted or early-stop occurs.

main.cpp

The screenshot shows a code editor window with the file name "main.cpp" at the top left. At the top right, there are four icons: a copy symbol, a sun-like icon, a "Share" button with a person icon, and a blue "Run" button. The code itself is a C++ program. It includes a header for *iostream*, uses the *std* namespace, and defines a function *insertionSortEarly* that sorts an array using insertion sort with an early stop condition. If an element is found to be in its correct position during the pass, the loop breaks. The program also includes a *printArray* function to display the contents of the array and a *main* function to handle user input for the array size and elements.

```
1 #include <iostream>
2 using namespace std;
3
4 // Modified Insertion Sort (Early Stop)
5 void insertionSortEarly(int arr[], int n) {
6     for (int i = 1; i < n; i++) {
7
8         int key = arr[i];
9         int j = i - 1;
10        bool shifted = false;
11
12        while (j >= 0 && arr[j] > key) {
13            arr[j + 1] = arr[j];
14            j--;
15            shifted = true;
16        }
17
18        arr[j + 1] = key;
19
20        if (!shifted) {
21            cout << "Early stop at pass " << i << " - already sorted.\n";
22            break;
23        }
24    }
25 }
26
27 void printArray(int arr[], int n) {
28     for (int i = 0; i < n; i++) cout << arr[i] << " ";
29     cout << endl;
30 }
31
32 int main() {
33
34     int n;
35     cout << "Enter size of array: ";
36     cin >> n;
37
38     if (!cin) {
39         cout << "Invalid input! Please enter a number.\n";
40         return 0;
41     }
42
43     int arr[n];
44     cout << "Enter " << n << " elements: ";
45     for (int i = 0; i < n; i++) cin >> arr[i];
46
47     insertionSortEarly(arr, n);
48
49     cout << "Sorted Array: ";
50     printArray(arr, n);
51
52     return 0;
53 }
```

```
Output  
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)

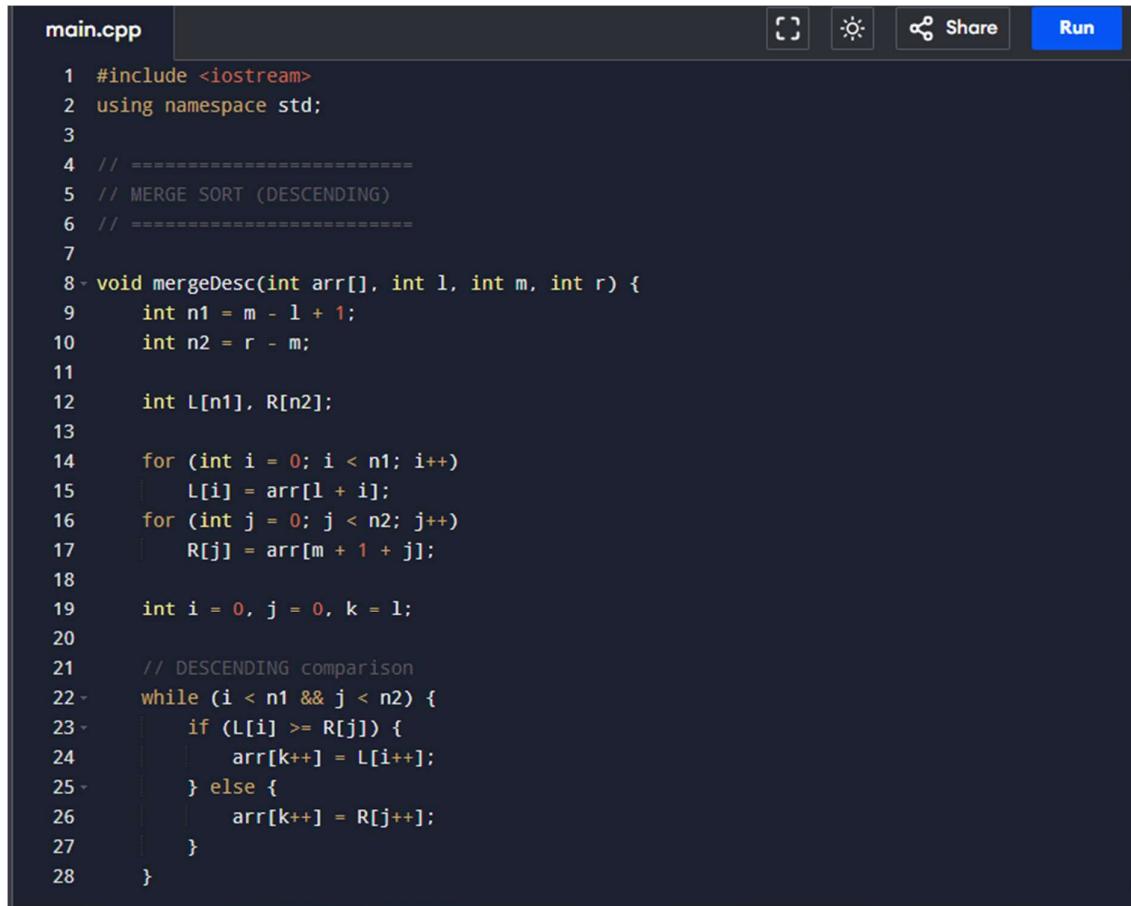
1. Divide the array into two halves.
2. Recursively divide until each subarray has 1 element.
3. Merge subarrays but while comparing,
 - o choose the larger element first.
 - o use \geq instead of \leq .
4. Continue merging until the full array is sorted in descending order.

2. Algorithm for Descending Quick Sort

Algorithm: Quick Sort (Descending Order)

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



The screenshot shows a code editor window with a dark theme. The file is named "main.cpp". The code implements a merge sort algorithm for descending order. It includes comments explaining the merge step and the comparison logic. The code uses arrays L and R to store elements from two halves of the input array arr, and then merges them back into arr in descending order. The editor has standard toolbar icons for copy, paste, share, and run.

```
main.cpp

1 #include <iostream>
2 using namespace std;
3
4 // =====
5 // MERGE SORT (DESCENDING)
6 // =====
7
8 void mergeDesc(int arr[], int l, int m, int r) {
9     int n1 = m - l + 1;
10    int n2 = r - m;
11
12    int L[n1], R[n2];
13
14    for (int i = 0; i < n1; i++)
15        L[i] = arr[l + i];
16    for (int j = 0; j < n2; j++)
17        R[j] = arr[m + 1 + j];
18
19    int i = 0, j = 0, k = l;
20
21    // DESCENDING comparison
22    while (i < n1 && j < n2) {
23        if (L[i] >= R[j]) {
24            arr[k++] = L[i++];
25        } else {
26            arr[k++] = R[j++];
27        }
28    }
}
```

```
29
30     while (i < n1)
31     |     arr[k++] = L[i++];
32     while (j < n2)
33     |     arr[k++] = R[j++];
34 }
35
36 - void mergeSortDesc(int arr[], int l, int r) {
37 -     if (l < r) {
38         int m = l + (r - 1) / 2;
39         mergeSortDesc(arr, l, m);
40         mergeSortDesc(arr, m + 1, r);
41         mergeDesc(arr, l, m, r);
42     }
43 }
44
45 // =====
46 // QUICK SORT (DESCENDING)
47 // =====
48
49 - int partitionDesc(int arr[], int low, int high) {
50     int pivot = arr[high];
51     int i = low - 1;
52
53     // Put BIGGER elements to the LEFT
54 -     for (int j = low; j < high; j++) {
55 -         if (arr[j] > pivot) {
56             |             i++;
57             |             swap(arr[i], arr[j]);
58         }
59     }
60
61     swap(arr[i + 1], arr[high]);
62     return i + 1;
63 }
64
65 - void quickSortDesc(int arr[], int low, int high) {
66 -     if (low < high) {
67         int pi = partitionDesc(arr, low, high);
68         quickSortDesc(arr, low, pi - 1);
69         quickSortDesc(arr, pi + 1, high);
70     }
71 }
72
73 // =====
74 // PRINT FUNCTION
75 // =====
76
77 - void printArray(int arr[], int n) {
78     for (int i = 0; i < n; i++)
79     |     cout << arr[i] << " ";
80     cout << endl;
81 }
82
```

```

83 // =====
84 // MAIN FUNCTION
85 // =====
86
87 int main() {
88     int n;
89     cout << "Enter size of array: ";
90     cin >> n;
91
92     int arr1[n], arr2[n];
93
94     cout << "Enter elements: ";
95     for (int i = 0; i < n; i++) {
96         cin >> arr1[i];
97         arr2[i] = arr1[i]; // copy for quick sort
98     }
99
100    // MERGE SORT DESC
101    mergeSortDesc(arr1, 0, n - 1);
102    cout << "Merge Sort (Descending): ";
103    printArray(arr1, n);
104
105    // QUICK SORT DESC
106    quickSortDesc(arr2, 0, n - 1);
107    cout << "Quick Sort (Descending): ";
108    printArray(arr2, n);
109
110    return 0;
111 }
112

```

Output Clear

```

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:
 - o Insertion Sort
 - o Merge Sort
 - o Quick Sort
3. Count:
 - o Comparisons
 - o Moves or Swaps
4. Print results exactly like the example.

Algorithm Summary for Task 3

Insertion Sort Counting

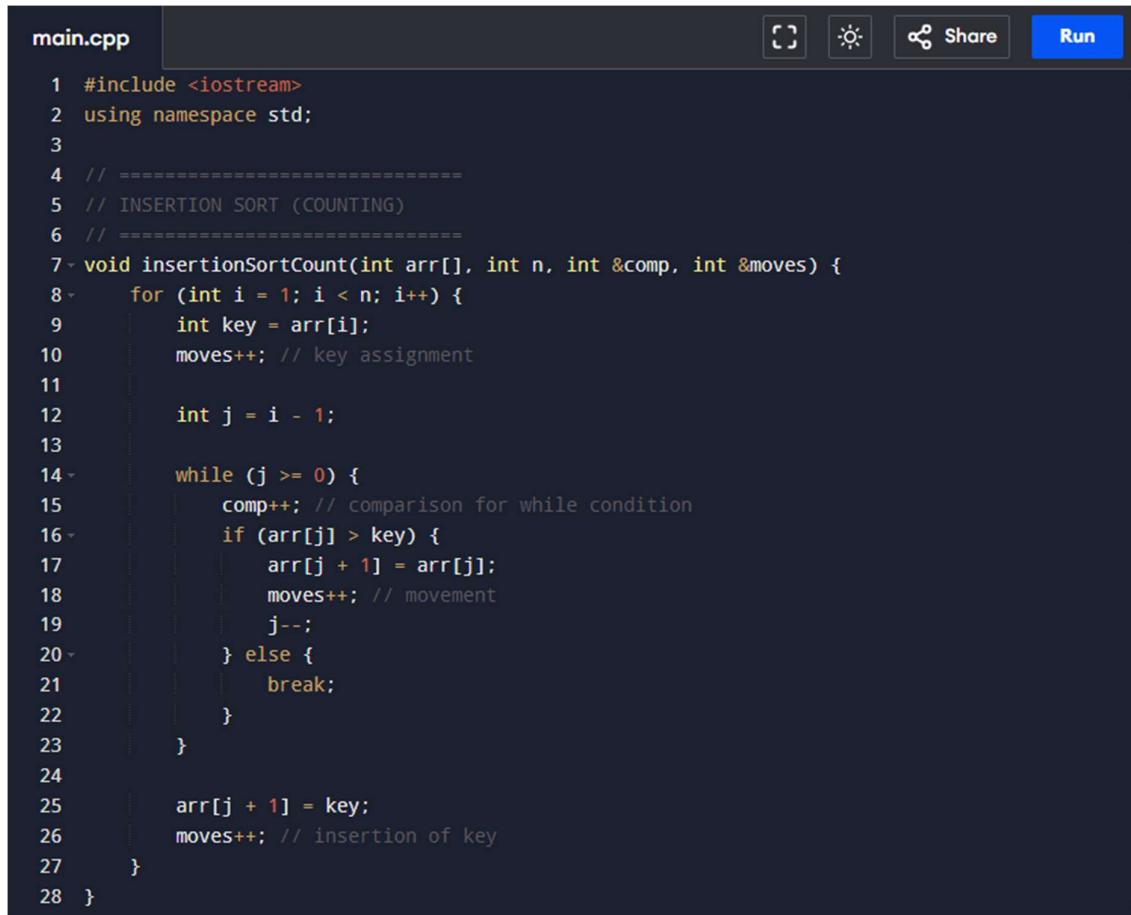
- Every time you compare → comp++
- Every time you move (shift) an element → moves++
- Every time you insert key → moves++

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++



The screenshot shows a code editor window titled "main.cpp". The code is a C++ implementation of insertion sort that counts the number of comparisons (comp) and moves (moves). The code is as follows:

```
1 #include <iostream>
2 using namespace std;
3
4 // =====
5 // INSERTION SORT (COUNTING)
6 // =====
7 void insertionSortCount(int arr[], int n, int &comp, int &moves) {
8     for (int i = 1; i < n; i++) {
9         int key = arr[i];
10        moves++; // key assignment
11
12        int j = i - 1;
13
14        while (j >= 0) {
15            comp++; // comparison for while condition
16            if (arr[j] > key) {
17                arr[j + 1] = arr[j];
18                moves++; // movement
19                j--;
20            } else {
21                break;
22            }
23        }
24
25        arr[j + 1] = key;
26        moves++; // insertion of key
27    }
28 }
```

The code editor interface includes tabs for "main.cpp", "Run" button, and other standard editor icons.

```
29
30 // =====
31 // MERGE SORT (COUNTING)
32 // =====
33 int comp_merge = 0, moves_merge = 0;
34
35 void mergeCount(int arr[], int l, int m, int r) {
36     int n1 = m - l + 1, n2 = r - m;
37     int L[n1], R[n2];
38
39     for (int i = 0; i < n1; i++) L[i] = arr[l + i];
40     for (int i = 0; i < n2; i++) R[i] = arr[m + 1 + i];
41
42     int i = 0, j = 0, k = l;
43
44     while (i < n1 && j < n2) {
45         comp_merge++; // comparison
46         if (L[i] <= R[j]) {
47             arr[k++] = L[i++];
48             moves_merge++; // move
49         } else {
50             arr[k++] = R[j++];
51             moves_merge++; // move
52         }
53     }
54
55     while (i < n1) {
56         moves_merge++;
57     }
58     while (j < n2) {
59         arr[k++] = R[j++];
60         moves_merge++;
61     }
62 }
63 }
64
65 void mergeSortCount(int arr[], int l, int r) {
66     if (l < r) {
67         int m = l + (r - l) / 2;
68         mergeSortCount(arr, l, m);
69         mergeSortCount(arr, m + 1, r);
70         mergeCount(arr, l, m, r);
71     }
72 }
73
74 // =====
75 // QUICK SORT (COUNTING)
76 // =====
77 int comp_quick = 0, swaps_quick = 0;
78
79 int partitionCount(int arr[], int low, int high) {
80     int pivot = arr[high];
81     int i = low - 1;
82
83     for (int j = low; j < high; j++) {
```

```
84     comp_quick++; // comparison with pivot
85     if (arr[j] < pivot) {
86         i++;
87         swap(arr[i], arr[j]);
88         swaps_quick++;
89     }
90 }
91
92 swap(arr[i + 1], arr[high]);
93 swaps_quick++;
94
95 return i + 1;
96 }
97
98 void quickSortCount(int arr[], int low, int high) {
99     if (low < high) {
100         int pi = partitionCount(arr, low, high);
101         quickSortCount(arr, low, pi - 1);
102         quickSortCount(arr, pi + 1, high);
103     }
104 }
105
106 // =====
107 // PRINT ARRAY
108 // =====
109 void printArray(int arr[], int n) {
110     for (int i = 0; i < n; i++) cout << arr[i] << " ";

```

```
111     cout << endl;
112 }
113
114 // =====
115 // MAIN
116 // =====
117 int main() {
118     int arr[10];
119
120     cout << "Enter 10 numbers: ";
121     for (int i = 0; i < 10; i++) cin >> arr[i];
122
123     int arr1[10], arr2[10], arr3[10];
124     for (int i = 0; i < 10; i++) {
125         arr1[i] = arr[i];
126         arr2[i] = arr[i];
127         arr3[i] = arr[i];
128     }
129
130     int comp_ins = 0, moves_ins = 0;
131
132     insertionSortCount(arr1, 10, comp_ins, moves_ins);
133     mergeSortCount(arr2, 0, 9);
134     quickSortCount(arr3, 0, 9);
135
136     cout << "\nInsertion Sort: Comparisons = " << comp_ins

```

```

137     |     << ", Moves = " << moves_ins << endl;
138
139     cout << "Merge Sort: Comparisons = " << comp_merge
140     |     << ", Moves = " << moves_merge << endl;
141
142     cout << "Quick Sort: Comparisons = " << comp_quick
143     |     << ", Swaps = " << swaps_quick << endl;
144
145     return 0;
146 }
147

```

Output Clear

```

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.

main.cpp

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 void print(const vector<int>& a) {
5     for (int x : a) cout << x << " ";
6     cout << "\n";
7 }
8
9 /**
10  * -----  

11  *      INSERTION SORT (PASSES SHOWN)  

12  * ----- */
12 void insertionSortPasses(vector<int> a) {
13     cout << "\n==== INSERTION SORT PASSES ===\n";
14
15     int n = a.size();
16     for (int i = 1; i < n; i++) {
17         int key = a[i];
18         int j = i - 1;
19
20         vector<string> log;
21
22         while (j >= 0 && a[j] > key) {
23             log.push_back("moved " + to_string(a[j]));
24             a[j + 1] = a[j];
25             j--;
26         }
27         a[j + 1] = key;
28     }
29
30     cout << "Final Array: ";
31     for (int x : a) cout << x << " ";
32     cout << "\n";
33 }
34
35 int main() {
36     vector<int> arr = {9, 3, 7, 1, 5, 2, 6, 8, 4, 0};
37
38     insertionSortPasses(arr);
39
40     cout << "Sorted Array: ";
41     for (int x : arr) cout << x << " ";
42     cout << "\n";
43 }

```

```
29         log.push_back("inserted " + to_string(key));
30
31         cout << "Pass " << i << ":" ;
32         print(a);
33
34         cout << "(";
35         for (int k = 0; k < log.size(); k++) {
36             if (k) cout << ", ";
37             cout << log[k];
38         }
39         cout << ")"\n";
40     }
41 }
42
43 /* =====
44 MERGE SORT (PASSES SHOWN)
45 ===== */
46
47 int mergePass = 1;
48
49 void mergeShow(vector<int>& a, int l, int m, int r) {
50     vector<int> L(a.begin() + l, a.begin() + m + 1);
51     vector<int> R(a.begin() + m + 1, a.begin() + r + 1);
52
53     int i = 0, j = 0, k = l;
54
55     while (i < L.size() && j < R.size()) {
56
57         if (L[i] <= R[j]) a[k++] = L[i++];
58         else a[k++] = R[j++];
59     }
60     while (i < L.size()) a[k++] = L[i++];
61     while (j < R.size()) a[k++] = R[j++];
62
63     cout << "Merge Pass " << mergePass++ << ":" ;
64     print(a);
65 }
66
67 void mergeSortPasses(vector<int>& a, int l, int r) {
68     if (l >= r) return;
69     int m = (l + r) / 2;
70
71     mergeSortPasses(a, l, m);
72     mergeSortPasses(a, m + 1, r);
73
74     mergeShow(a, l, m, r);
75 }
76
77 /* =====
78 QUICK SORT (PASSES SHOWN)
79 ===== */
80
81 int quickPass = 1;
82
83 int partitionShow(vector<int>& a, int low, int high) {
```

```
83     int pivot = a[high];
84     int i = low - 1;
85
86     for (int j = low; j < high; j++) {
87         if (a[j] < pivot) {
88             i++;
89             swap(a[i], a[j]);
90         }
91     }
92
93     swap(a[i + 1], a[high]);
94
95     cout << "Partition Pass " << quickPass++ << ":" ;
96     print(a);
97
98     return i + 1;
99 }
100
101 void quickSortPasses(vector<int>& a, int low, int high) {
102     if (low < high) {
103         int p = partitionShow(a, low, high);
104         quickSortPasses(a, low, p - 1);
105         quickSortPasses(a, p + 1, high);
106     }
107 }
108
```

```
109 /* =====
110 | | | | MAIN
111 | ===== */
112
113 int main() {
114     int n;
115     cout << "Enter size of array: ";
116     cin >> n;
117
118     vector<int> a(n), a1, a2, a3;
119
120     cout << "Enter numbers: ";
121     for (int i = 0; i < n; i++) cin >> a[i];
122
123     a1 = a;
124     a2 = a;
125     a3 = a;
126
127     insertionSortPasses(a1);
128
129     cout << "\n==== MERGE SORT PASSES ===\n";
130     mergeSortPasses(a2, 0, n - 1);
131
132     cout << "\n==== QUICK SORT PASSES ===\n";
133     quickSortPasses(a3, 0, n - 1);
134 }
```

```
134
135     return 0;
136 }
137
```

```
Output Clear
▶ 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]

main.cpp

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 // Print helper
5 void print(vector<string>& a) {
6     for (auto &s : a) cout << s << " ";
7     cout << "\n";
8 }
9
10 /* =====
11     INSERTION SORT (STRINGS)
12 ===== */
13 void insertionSortStrings(vector<string> a) {
14     int n = a.size();
15
16     for (int i = 1; i < n; i++) {
17         string key = a[i];
18         int j = i - 1;
19
20         while (j >= 0 && a[j] > key) {
21             a[j + 1] = a[j];
22             j--;
23         }
24
25         a[j + 1] = key;
26     }
27
28     cout << "Insertion Sort Result: ";
```

```
29     print(a);
30 }
31
32 /* =====
33     MERGE SORT (STRINGS)
34 ===== */
35
36 void mergeStrings(vector<string>& a, int l, int m, int r) {
37     vector<string> L(a.begin() + l, a.begin() + m + 1);
38     vector<string> R(a.begin() + m + 1, a.begin() + r + 1);
39
40     int i = 0, j = 0, k = l;
41
42     while (i < L.size() && j < R.size()) {
43         if (L[i] <= R[j]) a[k++] = L[i++];
44         else a[k++] = R[j++];
45     }
46
47     while (i < L.size()) a[k++] = L[i++];
48     while (j < R.size()) a[k++] = R[j++];
49 }
50
51 void mergeSortStrings(vector<string>& a, int l, int r) {
52     if (l >= r) return;
53     int m = (l + r) / 2;
```

```
55     mergeSortStrings(a, l, m);
56     mergeSortStrings(a, m + 1, r);
57     mergeStrings(a, l, m, r);
58 }
59
60 /* =====
61  QUICK SORT (STRINGS)
62 ===== */
63
64 int partitionStrings(vector<string>& a, int low, int high) {
65     string pivot = a[high];
66     int i = low - 1;
67
68     for (int j = low; j < high; j++) {
69         if (a[j] < pivot) {
70             i++;
71             swap(a[i], a[j]);
72         }
73     }
74
75     swap(a[i + 1], a[high]);
76     return i + 1;
77 }
78
79 void quickSortStrings(vector<string>& a, int low, int high) {
80     if (low < high) {
```

```
81         int p = partitionStrings(a, low, high);
82         quickSortStrings(a, low, p - 1);
83         quickSortStrings(a, p + 1, high);
84     }
85 }
86
87 /* =====
88  MAIN
89 ===== */
90
91 int main() {
92     int n;
93     cout << "Enter number of strings: ";
94     cin >> n;
95
96     vector<string> arr(n), a1, a2, a3;
97
98     cout << "Enter strings: ";
99     for (int i = 0; i < n; i++) cin >> arr[i];
100
101    a1 = arr;
102    a2 = arr;
103    a3 = arr;
104
105    // Insertion Sort
106    insertionSortStrings(a1);
```

```
107     // Merge Sort
108     mergeSortStrings(a2, 0, n - 1);
109     cout << "Merge Sort Result: ";
110     print(a2);
111
112     // Quick Sort
113     quickSortStrings(a3, 0, n - 1);
114     cout << "Quick Sort Result: ";
115     print(a3);
116
117     return 0;
118 }
120
```

Output

```
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 ===
```