

Q1) Write a program to sort the elements of an array using Insertion Sort (The program should report the number of comparisons).

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

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

    int arr[n];

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

    int comparisons = 0;

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

        while (j >= 0) {
            comparisons++;

            if (arr[j] > key) {
                arr[j + 1] = arr[j];
                j--;
            } else {
                break;
            }
        }
        arr[j + 1] = key;
    }

    cout << "\nSorted Array:\n";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
}
```

```
Enter 10 elements:
100
90
80
770
60
50
40
30
20
3

Sorted Array:
3 20 30 40 50 60 80 90 100 770

Total Comparisons: 43
PS C:\Users\Ritesh\OneDrive\Desktop\DAA Practical File> |
```

2) Write a program to sort the elements of an array using Merge Sort (The program should report the number of comparisons).

```

1  #include <iostream>
2  using namespace std;
3
4  long long comparisons = 0;
5
6  void merge(int arr[], int left, int mid, int right) {
7      int n1 = mid - left + 1;
8      int n2 = right - mid;
9
10     int L[n1], R[n2];
11
12     for (int i = 0; i < n1; i++)
13         L[i] = arr[left + i];
14     for (int j = 0; j < n2; j++)
15         R[j] = arr[mid + 1 + j];
16
17     int i = 0, j = 0, k = left;
18
19     while (i < n1 && j < n2) {
20         comparisons++;
21
22         if (L[i] <= R[j]) {
23             arr[k] = L[i];
24             i++;
25         } else {
26             arr[k] = R[j];
27             j++;
28         }
29         k++;
30     }
31
32     while (i < n1) {
33         arr[k] = L[i];
34         i++;
35         k++;
36     }
37
38     while (j < n2) {
39         arr[k] = R[j];
40         j++;
41         k++;
42     }
43 }

```

```

45 void mergeSort(int arr[], int left, int right) {
46     if (left < right) {
47         int mid = left + (right - left) / 2;
48
49         mergeSort(arr, left, mid);
50         mergeSort(arr, mid + 1, right);
51
52         merge(arr, left, mid, right);
53     }
54 }
55
56 int main() {
57     int n;
58     cout << "Enter number of elements: ";
59     cin >> n;
60
61     int arr[n];
62     cout << "Enter " << n << " elements:\n";
63     for (int i = 0; i < n; i++) {
64         cin >> arr[i];
65     }
66
67     mergeSort(arr, 0, n - 1);
68
69     cout << "\nSorted Array:\n";
70     for (int i = 0; i < n; i++) {
71         cout << arr[i] << " ";
72     }
73
74     cout << "\n\nTotal Comparisons: " << comparisons << endl;
75
76     return 0;
77 }
78

```

```

Enter number of elements: 10
Enter 10 elements:
100 90 80 70 60 50 44 33 22 11 1

Sorted Array:
11 22 33 44 50 60 70 80 90 100

Total Comparisons: 15
PS C:\Users\Ritesh\OneDrive\Desktop\DAI Practical File>

```

3) Write a program to sort the elements of an array using Heap Sort (The program should report the number of comparisons).

```

Q3.cpp > heapSort(int [], int)
1  #include <iostream>
2  using namespace std;
3
4  long long comparisons = 0;
5
6
7  void heapify(int arr[], int n, int i) {
8      int largest = i;
9      int left = 2 * i + 1;
10     int right = 2 * i + 2;
11
12     if (left < n) {
13         comparisons++;
14         if (arr[left] > arr[largest])
15             largest = left;
16     }
17
18     if (right < n) {
19         comparisons++;
20         if (arr[right] > arr[largest])
21             largest = right;
22     }
23
24     if (largest != i) {
25         swap(arr[i], arr[largest]);
26         heapify(arr, n, largest);
27     }
28 }
29
30 void heapSort(int arr[], int n) {
31     for (int i = n / 2 - 1; i >= 0; i--)
32         heapify(arr, n, i);
33
34     for (int i = n - 1; i > 0; i--) {
35         swap(arr[0], arr[i]);
36         heapify(arr, i, 0);
37     }
38 }
39

```

```

40  int main() {
41      int n;
42      cout << "Enter number of elements: ";
43      cin >> n;
44
45      int arr[n];
46      cout << "Enter " << n << " elements:\n";
47      for (int i = 0; i < n; i++) {
48          cin >> arr[i];
49      }
50
51      heapSort(arr, n);
52
53      cout << "\nSorted Array:\n";
54      for (int i = 0; i < n; i++) {
55          cout << arr[i] << " ";
56      }
57
58      cout << "\n\nTotal Comparisons: " << comparisons << endl;
59
60      return 0;
61  }
62

```

```
g++ Q3.cpp -o Q3 } ; if ($?) { .\Q3 }
```

```

Enter number of elements: 10
Enter 10 elements:
100 90 80 70 60 50 44 33 22 11 1

```

```

Sorted Array:
11 22 33 44 50 60 70 80 90 100

```

```
Total Comparisons: 35
```

```
D:\C:\Users\Ritosh\OneDrive\Desktop\DA-Practical-Files>
```

4) Write a program to sort the elements of an array using Quick Sort (The program should report the number of comparisons).

Q4.cpp > ...

```
1  #include <iostream>
2  using namespace std;
3
4  long long comparisons = 0;
5
6  int partition(int arr[], int low, int high) {
7      int pivot = arr[high];
8      int i = low - 1;
9
10     for (int j = low; j < high; j++) {
11         comparisons++;
12
13         if (arr[j] <= pivot) {
14             i++;
15             swap(arr[i], arr[j]);
16         }
17     }
18
19     swap(arr[i + 1], arr[high]);
20     return (i + 1);
21 }
22
23 void quickSort(int arr[], int low, int high) {
24     if (low < high) {
25         int pi = partition(arr, low, high);
26
27         quickSort(arr, low, pi - 1);
28         quickSort(arr, pi + 1, high);
29     }
30 }
31
```



```

31
32  int main() {
33      int n;
34      cout << "Enter number of elements: ";
35      cin >> n;
36
37      int arr[n];
38      cout << "Enter " << n << " elements:\n";
39      for (int i = 0; i < n; i++) {
40          cin >> arr[i];
41      }
42
43      quickSort(arr, 0, n - 1);
44
45      cout << "\nSorted Array:\n";
46      for (int i = 0; i < n; i++) {
47          cout << arr[i] << " ";
48      }
49
50      cout << "\n\nTotal Comparisons: " << comparisons << endl;
51
52      return 0;
53  }
54

```

```

> cd
g++ Q4.cpp -o Q4 } ; if ($?) { .\Q4 }
Enter number of elements: 10
Enter 10 elements:
101 132 43 65 63 87 32 15 85 90

Sorted Array:
15 32 43 63 65 85 87 90 101 132

Total Comparisons: 24
PS C:\Users\Ritesh\OneDrive\Desktop\DAA Practical File>

```

5) Write a program to multiply two matrices using the Strassen's algorithm for matrix multiplication.

```

Q5.cpp > strassen(const vector<vector<int>>& A, const vector<vector<int>>& B)
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  vector<vector<int>> add(const vector<vector<int>>& A, const vector<vector<int>>& B) {
6      int n = A.size();
7      vector<vector<int>> C(n, vector<int>(n));
8      for (int i = 0; i < n; i++)
9          for (int j = 0; j < n; j++)
10             C[i][j] = A[i][j] + B[i][j];
11     return C;
12 }
13
14 vector<vector<int>> subtract(const vector<vector<int>>& A, const vector<vector<int>>& B) {
15     int n = A.size();
16     vector<vector<int>> C(n, vector<int>(n));
17     for (int i = 0; i < n; i++)
18         for (int j = 0; j < n; j++)
19             C[i][j] = A[i][j] - B[i][j];
20     return C;
21 }
22
23 vector<vector<int>> strassen(const vector<vector<int>>& A, const vector<vector<int>>& B) {
24     int n = A.size();
25
26     if (n == 1) {
27         return {{A[0][0] * B[0][0]}};
28     }
29
30     int k = n / 2;
31
32     vector<vector<int>>
33         A11(k, vector<int>(k)), A12(k, vector<int>(k)),
34         A21(k, vector<int>(k)), A22(k, vector<int>(k)),
35         B11(k, vector<int>(k)), B12(k, vector<int>(k)),
36         B21(k, vector<int>(k)), B22(k, vector<int>(k));
37
38     for (int i = 0; i < k; i++) {
39         for (int j = 0; j < k; j++) {
40             A11[i][j] = A[i][j];
41             A12[i][j] = A[i][j + k];
42             A21[i][j] = A[i + k][j];
43             A22[i][j] = A[i + k][j + k];
44
45             B11[i][j] = B[i][j];
46             B12[i][j] = B[i][j + k];
47             B21[i][j] = B[i + k][j];
48             B22[i][j] = B[i + k][j + k];
49         }
50     }
51

```

```

51
52     auto M1 = strassen(add(A11, A22), add(B11, B22));
53     auto M2 = strassen(add(A21, A22), B11);
54     auto M3 = strassen(A11, subtract(B12, B22));
55     auto M4 = strassen(A22, subtract(B21, B11));
56     auto M5 = strassen(add(A11, A12), B22);
57     auto M6 = strassen(subtract(A21, A11), add(B11, B12));
58     auto M7 = strassen(subtract(A12, A22), add(B21, B22));
59
60     vector<vector<int>> C(n, vector<int>(n));
61
62     auto C11 = add(subtract(add(M1, M4), M5), M7);
63     auto C12 = add(M3, M5);
64     auto C21 = add(M2, M4);
65     auto C22 = add(subtract(add(M1, M3), M2), M6);
66
67     for (int i = 0; i < k; i++) {
68         for (int j = 0; j < k; j++) {
69             C[i][j] = C11[i][j];
70             C[i][j + k] = C12[i][j];
71             C[i + k][j] = C21[i][j];
72             C[i + k][j + k] = C22[i][j];
73         }
74     }
75
76     return C;
77 }

```

```

79 int main() {
80     int n;
81     cout << "Enter matrix size (must be power of 2): ";
82     cin >> n;
83
84     vector<vector<int>> A(n, vector<int>(n));
85     vector<vector<int>> B(n, vector<int>(n));
86
87     cout << "Enter Matrix A:\n";
88     for (int i = 0; i < n; i++)
89         for (int j = 0; j < n; j++)
90             cin >> A[i][j];
91
92     cout << "Enter Matrix B:\n";
93     for (int i = 0; i < n; i++)
94         for (int j = 0; j < n; j++)
95             cin >> B[i][j];
96
97     auto C = strassen(A, B);
98
99     cout << "\nResult (A x B):\n";
100    for (int i = 0; i < n; i++) {
101        for (int j = 0; j < n; j++)
102            cout << C[i][j] << " ";
103        cout << endl;
104    }
105
106    return 0;
107 }

```

```
{ .\Q5 }  
Enter matrix size (must be power of 2): 4  
Enter Matrix A:  
1 0 1 4 3 6 4 3 2 6 3 0 7 4 2 5  
Enter Matrix B:  
4 2 7 4 7 7 4 2 7 4 2 1 0 6 7 3  
  
Result (A X B):  
11 30 37 17  
82 82 74 37  
71 58 44 23  
70 80 104 53  
PS C:\Users\Ritesh\OneDrive\Desktop\DAA Practical File> |
```

6) Write a program to sort the elements of an array using Count Sort.

Q6.cpp > main()

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6      int n;
7      cout << "Enter number of elements: ";
8      cin >> n;
9
10     vector<int> arr(n);
11     cout << "Enter " << n << " elements:\n";
12     for (int i = 0; i < n; i++)
13         cin >> arr[i];
14
15     int maxVal = arr[0];
16     for (int i = 1; i < n; i++)
17         if (arr[i] > maxVal)
18             maxVal = arr[i];
19
20     vector<int> count(maxVal + 1, 0);
21
22     for (int i = 0; i < n; i++)
23         count[arr[i]]++;
24
```

```

25     for (int i = 1; i <= maxVal; i++)
26         count[i] += count[i - 1];
27
28     vector<int> sorted(n);
29
30     for (int i = n - 1; i >= 0; i--) {
31         sorted[count[arr[i]] - 1] = arr[i];
32         count[arr[i]]--;
33     }
34
35     cout << "\nSorted Array:\n";
36     for (int i = 0; i < n; i++)
37         cout << sorted[i] << " ";
38
39     cout << endl;
40     return 0;
41 }
42

```

Enter number of elements: 7

Enter 7 elements:

6 3 2 4 6 2 6

Sorted Array:

2 2 3 4 6 6 6

PS C:\Users\Ritesh\OneDrive\Desktop\DAA Practical File>

7) Display the data stored in a given graph using the Breadth-First Search algorithm.

Q7.cpp

Q7.cpp > BFS(int, vector<vector<int>>&, int)

```
1  #include <iostream>
2  #include <vector>
3  #include <queue>
4  using namespace std;
5
6  void BFS(int start, vector<vector<int>>& graph, int n) {
7      vector<bool> visited(n, false);
8      queue<int> q;
9
10     visited[start] = true;
11     q.push(start);
12
13     cout << "BFS Traversal: ";
14
15     while (!q.empty()) {
16         int node = q.front();
17         q.pop();
18         cout << node << " ";
19
20         for (int neighbor : graph[node]) {
21             if (!visited[neighbor]) {
22                 visited[neighbor] = true;
23                 q.push(neighbor);
24             }
25         }
26     }
27     cout << endl;
28 }
29
```

```

29
30 ✓ int main() {
31     int n, edges;
32     cout << "Enter number of vertices: ";
33     cin >> n;
34
35     cout << "Enter number of edges: ";
36     cin >> edges;
37
38     vector<vector<int>> graph(n);
39
40     cout << "Enter edges (u v):\n";
41 ✓ for (int i = 0; i < edges; i++) {
42     int u, v;
43     cin >> u >> v;
44     graph[u].push_back(v);
45     graph[v].push_back(u);
46 }
47
48     int start;
49     cout << "Enter starting vertex for BFS: ";
50     cin >> start;
51
52     BFS(start, graph, n);
53
54     return 0;
55 }
56

```

```

PS C:\Users\Ritesh\OneDrive\Desktop\DAA Practical File> cd "c:\Users\Ritesh\OneDrive\Desktop\DAA P
{ .\Q7 }
Enter number of vertices: 5
Enter number of edges: 6
Enter edges (u v):
0 1
0 2
1 3
1 4
2 3
3 4
Enter starting vertex for BFS: 0
BFS Traversal: 0 1 2 3 4
PS C:\Users\Ritesh\OneDrive\Desktop\DAA Practical File>

```

8) Display the data stored in a given graph using the Depth-First Search algorithm.

Q7.cpp

Q8.cpp

Q8.cpp > DFS(int, vector<vector<int>>&, vector<bool>&)

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  void DFS(int node, vector<vector<int>>& graph, vector<bool>& visited) {
6      visited[node] = true;
7      cout << node << " ";
8
9      for (int neighbor : graph[node]) {
10         if (!visited[neighbor]) {
11             DFS(neighbor, graph, visited);
12         }
13     }
14 }
```

```
15
16 int main() {
17     int n, edges;
18     cout << "Enter number of vertices: ";
19     cin >> n;
20
21     cout << "Enter number of edges: ";
22     cin >> edges;
23
24     vector<vector<int>> graph(n);
25
26     cout << "Enter edges (u v):\n";
27     for (int i = 0; i < edges; i++) {
28         int u, v;
29         cin >> u >> v;
30         graph[u].push_back(v);
31         graph[v].push_back(u);
32     }
33
34     int start;
35     cout << "Enter starting vertex for DFS: ";
36     cin >> start;
37
38     vector<bool> visited(n, false);
39
40     cout << "DFS Traversal: ";
41     DFS(start, graph, visited);
42     cout << endl;
43
44     return 0;
45 }
```

```
PS C:\Users\Ritesh\OneDrive\Desktop\DAA Practical File> cd "c:\Users\Ritesh\OneDrive\Desktop\DAA Practical File\" ;  
{ .\Q8 }  
Enter number of vertices: 5  
Enter number of edges: 6  
Enter edges (u v):  
0 1  
0 2  
1 3  
1 4  
2 3  
3 4  
Enter starting vertex for DFS: 0  
DFS Traversal: 0 1 3 2 4  
PS C:\Users\Ritesh\OneDrive\Desktop\DAA Practical File> |
```

9) Write a program to determine a minimum spanning tree of a graph using the Prim's algorithm.

Q7.cpp

Q8.cpp

Q9.cpp

Q9.cpp > main()

```
1  #include <iostream>
2  using namespace std;
3
4  #define INF 999999
5
6  int main() {
7      int n;
8      cout << "Enter number of vertices: ";
9      cin >> n;
10
11     int graph[50][50];
12
13     cout << "Enter adjacency matrix (use 0 if no edge):\n";
14     for (int i = 0; i < n; i++) {
15         for (int j = 0; j < n; j++) {
16             cin >> graph[i][j];
17             if (graph[i][j] == 0 && i != j)
18                 graph[i][j] = INF;
19         }
20     }
21
22     int selected[50] = {0};
23     selected[0] = 1;
24
25     int edgeCount = 0;
26     int totalCost = 0;
27
28     cout << "\nEdges in MST:\n";
```

Q7.cpp

Q8.cpp

Q9.cpp

Q9.cpp > main()

```
1  #include <iostream>
2  using namespace std;
3
4  #define INF 999999
5
6  int main() {
7      int n;
8      cout << "Enter number of vertices: ";
9      cin >> n;
10
11     int graph[50][50];
12
13     cout << "Enter adjacency matrix (use 0 if no edge):\n";
14     for (int i = 0; i < n; i++) {
15         for (int j = 0; j < n; j++) {
16             cin >> graph[i][j];
17             if (graph[i][j] == 0 && i != j)
18                 graph[i][j] = INF;
19         }
20     }
21
22     int selected[50] = {0};
23     selected[0] = 1;
24
25     int edgeCount = 0;
26     int totalCost = 0;
27
28     cout << "\nEdges in MST:\n";
```

```

29
30 while (edgeCount < n - 1) {
31     int minWeight = INF;
32     int u = -1, v = -1;
33
34     for (int i = 0; i < n; i++) {
35         if (selected[i]) {
36             for (int j = 0; j < n; j++) {
37                 if (!selected[j] && graph[i][j] < minWeight) {
38                     minWeight = graph[i][j];
39                     u = i;
40                     v = j;
41                 }
42             }
43         }
44     }
45
46     cout << u << " - " << v << " (weight = " << minWeight << ")\n";
47     totalCost += minWeight;
48     selected[v] = 1;
49     edgeCount++;
50 }
51
52 cout << "\nTotal cost of MST = " << totalCost << endl;
53
54 return 0;
55 }
56

```

```

PS C:\Users\Ritesh\OneDrive\Desktop\DAA Practical File> cd "c:\Users\Ritesh\OneDrive\Desktop\DAA Practical File\" ;
{ .\Q9 }
Enter number of vertices: 5
Enter adjacency matrix (use 0 if no edge):
0 2 0 6 0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9
0 5 7 9 0

Edges in MST:
0 - 1 (weight = 2)
1 - 2 (weight = 3)
1 - 4 (weight = 5)
0 - 3 (weight = 6)

Total cost of MST = 16
PS C:\Users\Ritesh\OneDrive\Desktop\DAA Practical File>

```

10) Write a program to determine the shortest path from a given node s to the other nodes of a graph using the Dijkstra's algorithm.

```
Q10.cpp > main()
1  #include <iostream>
2  using namespace std;
3
4  #define INF 999999
5
6  int main() {
7      int n;
8      cout << "Enter number of vertices: ";
9      cin >> n;
10
11     int graph[50][50];
12
13     cout << "Enter adjacency matrix (0 if no edge):\n";
14     for (int i = 0; i < n; i++) {
15         for (int j = 0; j < n; j++) {
16             cin >> graph[i][j];
17             if (graph[i][j] == 0 && i != j)
18                 graph[i][j] = INF;
19         }
20     }
21
22     int source;
23     cout << "Enter source vertex: ";
24     cin >> source;
25
26     int dist[50];
27     bool visited[50];
28
29     for (int i = 0; i < n; i++) {
30         dist[i] = graph[source][i];
31         visited[i] = false;
32     }
```

```

33     dist[source] = 0;
34     visited[source] = true;
35
36     for (int count = 1; count < n; count++) {
37         int minDist = INF, u = -1;
38
39         for (int i = 0; i < n; i++) {
40             if (!visited[i] && dist[i] < minDist) {
41                 minDist = dist[i];
42                 u = i;
43             }
44         }
45
46         visited[u] = true;
47
48         for (int v = 0; v < n; v++) {
49             if (!visited[v] && graph[u][v] != INF &&
50                 dist[u] + graph[u][v] < dist[v]) {
51                 dist[v] = dist[u] + graph[u][v];
52             }
53         }
54     }
55
56     // Output result
57     cout << "\nShortest distances from source " << source << ":\n";
58     for (int i = 0; i < n; i++) {
59         cout << "To vertex " << i << " = " << dist[i] << endl;
60     }
61
62     return 0;
63 }
64

```

```

PS C:\Users\Ritesh\OneDrive\Desktop\DAA Practical File> cd "c:\Users\Ritesh\O
) { .\Q10 }
Enter number of vertices: 5
Enter adjacency matrix (0 if no edge):
0 10 0 5 0
10 0 1 2 0
0 1 0 9 0
5 2 9 0 2
0 0 0 2 0
Enter source vertex: 0

Shortest distances from source 0:
To vertex 0 = 0
To vertex 1 = 7
To vertex 2 = 8
To vertex 3 = 5
To vertex 4 = 7
PS C:\Users\Ritesh\OneDrive\Desktop\DAA Practical File>

```

11) Write a program to solve the 0-1 knapsack problem using Dynamic Programming.

Q11.cpp > main()

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6      int n, W;
7      cout << "Enter number of items: ";
8      cin >> n;
9      cout << "Enter maximum weight of knapsack: ";
10     cin >> W;
11
12     vector<int> weight(n), value(n);
13     cout << "Enter weights of items:\n";
14     for (int i = 0; i < n; i++)
15         cin >> weight[i];
16
17     cout << "Enter values of items:\n";
18     for (int i = 0; i < n; i++)
19         cin >> value[i];
20
21     vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));
22
23     for (int i = 1; i <= n; i++) {
24         for (int w = 1; w <= W; w++) {
25             if (weight[i - 1] <= w) {
26                 dp[i][w] = max(value[i - 1] + dp[i - 1][w - weight[i - 1]], dp[i - 1][w]);
27             } else {
28                 dp[i][w] = dp[i - 1][w];
29             }
30         }
31     }
32
33     cout << "\nMaximum value in knapsack = " << dp[n][W] << endl;
34
35     cout << "Items included (0-based index): ";
36     int w = W;
37     for (int i = n; i > 0 && w > 0; i--) {
38         if (dp[i][w] != dp[i - 1][w]) {
39             cout << (i - 1) << " ";
40             w -= weight[i - 1];
41         }
42     }
43     cout << endl;
44
45     return 0;
46 }
```



```
PS C:\Users\Ritesh\OneDrive\Desktop\DAA Practical File> cd C:\Users\Ritesh\OneDrive\Desktop\DAA Practical File
) { .\Q11 }
Enter number of items: 4
Enter maximum weight of knapsack: 7
Enter weights of items:
1 3 4 5
Enter values of items:
1 4 5 7

Maximum value in knapsack = 9
Items included (0-based index): 2 1
PS C:\Users\Ritesh\OneDrive\Desktop\DAA Practical File>
```

