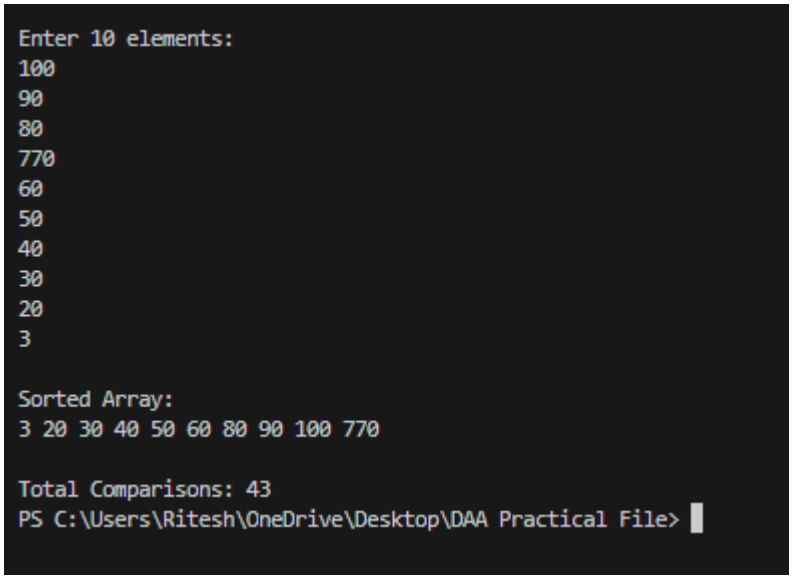
Q1) Write a program to sort the elements of an array using Insertion Sort (The program should report the number of comparisons).

```cpp
#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] << " ";
    }
```



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

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

long long comparisons = 0;

void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1], R[n2];

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

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

    while (i < n1 && j < n2) {
        comparisons++;

        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

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

```cpp
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);
    }
}

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];
    }

    mergeSort(arr, 0, n - 1);

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

    cout << "\n\nTotal Comparisons: " << comparisons << endl;

    return 0;
}
```

```
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\Bitesh\OneDrive\Desktop\DAA 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)

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

long long comparisons = 0;


void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n) {
        comparisons++;
        if (arr[left] > arr[largest])
            largest = left;
    }

    if (right < n) {
        comparisons++;
        if (arr[right] > arr[largest])
            largest = right;
    }

    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}

void heapSort(int arr[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    for (int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}
```

```cpp
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];
    }

    heapSort(arr, n);

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

    cout << "\n\nTotal Comparisons: " << comparisons << endl;

    return 0;
}
```

```
 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
```

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

Q4.cpp > ...

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

long long comparisons = 0;

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

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

        if (arr[j] <= pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }

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

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

```cpp

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];
    }

    quickSort(arr, 0, n - 1);

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

    cout << "\n\nTotal Comparisons: " << comparisons << endl;

    return 0;
}
```

```
                                                    > 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>
```

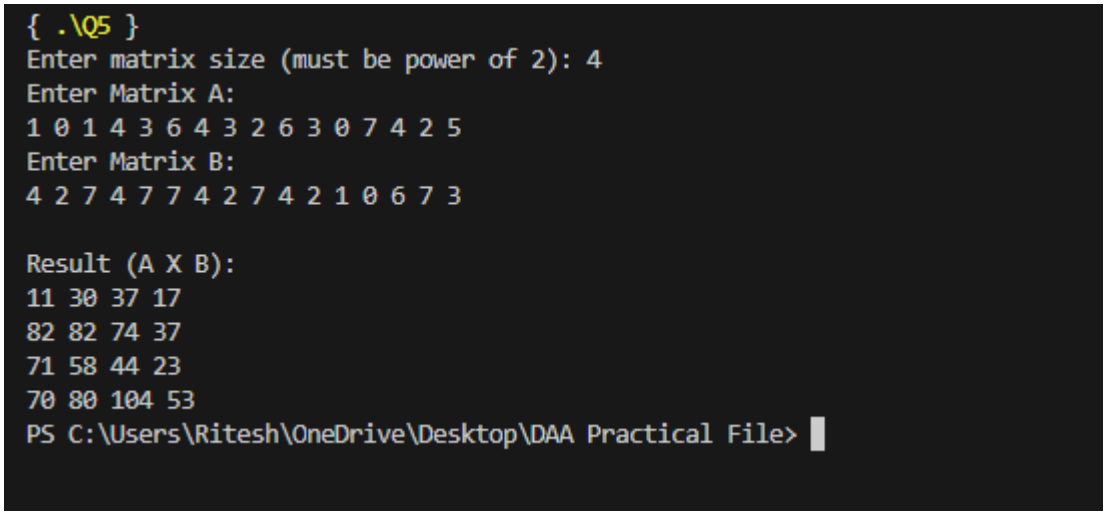
5) Write a program to multiply two matrices using the Strassen's algorithm for matrix multiplication.

Q5.cpp > strassen(const vector<vector<int>>&, const vector<vector<int>>&)

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

vector<vector<int>> add(const vector<vector<int>>& A, const vector<vector<int>>& B) {
    int n = A.size();
    vector<vector<int>> C(n, vector<int>(n));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            C[i][j] = A[i][j] + B[i][j];
    return C;
}

vector<vector<int>> subtract(const vector<vector<int>>& A, const vector<vector<int>>& B) {
    int n = A.size();
    vector<vector<int>> C(n, vector<int>(n));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            C[i][j] = A[i][j] - B[i][j];
    return C;
}

vector<vector<int>> strassen(const vector<vector<int>>& A, const vector<vector<int>>& B) {
    int n = A.size();

    if (n == 1) {
        return {{A[0][0] * B[0][0]}};
    }

    int k = n / 2;

    vector<vector<int>>
        A11(k, vector<int>(k)), A12(k, vector<int>(k)),
        A21(k, vector<int>(k)), A22(k, vector<int>(k)),
        B11(k, vector<int>(k)), B12(k, vector<int>(k)),
        B21(k, vector<int>(k)), B22(k, vector<int>(k));

    for (int i = 0; i < k; i++) {
        for (int j = 0; j < k; j++) {
            A11[i][j] = A[i][j];
            A12[i][j] = A[i][j + k];
            A21[i][j] = A[i + k][j];
            A22[i][j] = A[i + k][j + k];

            B11[i][j] = B[i][j];
            B12[i][j] = B[i][j + k];
            B21[i][j] = B[i + k][j];
            B22[i][j] = B[i + k][j + k];
        }
    }
```

```
    auto M1 = strassen(add(A11, A22), add(B11, B22));
    auto M2 = strassen(add(A21, A22), B11);
    auto M3 = strassen(A11, subtract(B12, B22));
    auto M4 = strassen(A22, subtract(B21, B11));
    auto M5 = strassen(add(A11, A12), B22);
    auto M6 = strassen(subtract(A21, A11), add(B11, B12));
    auto M7 = strassen(subtract(A12, A22), add(B21, B22));

    vector<vector<int>> C(n, vector<int>(n));

    auto C11 = add(subtract(add(M1, M4), M5), M7);
    auto C12 = add(M3, M5);
    auto C21 = add(M2, M4);
    auto C22 = add(subtract(add(M1, M3), M2), M6);

    for (int i = 0; i < k; i++) {
        for (int j = 0; j < k; j++) {
            C[i][j] = C11[i][j];
            C[i][j + k] = C12[i][j];
            C[i + k][j] = C21[i][j];
            C[i + k][j + k] = C22[i][j];
        }
    }

    return C;
}
```

```
int main() {
    int n;
    cout << "Enter matrix size (must be power of 2): ";
    cin >> n;

    vector<vector<int>> A(n, vector<int>(n));
    vector<vector<int>> B(n, vector<int>(n));

    cout << "Enter Matrix A:\n";
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            cin >> A[i][j];

    cout << "Enter Matrix B:\n";
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            cin >> B[i][j];

    auto C = strassen(A, B);

    cout << "\nResult (A x B):\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++)
            cout << C[i][j] << " ";
        cout << endl;
    }

    return 0;
}
```



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

Q6.cpp > main()

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

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

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

    int maxVal = arr[0];
    for (int i = 1; i < n; i++)
        if (arr[i] > maxVal)
            maxVal = arr[i];

    vector<int> count(maxVal + 1, 0);

    for (int i = 0; i < n; i++)
        count[arr[i]]++;

```

```
    for (int i = 1; i <= maxVal; i++)
        count[i] += count[i - 1];

    vector<int> sorted(n);

    for (int i = n - 1; i >= 0; i--) {
        sorted[count[arr[i]] - 1] = arr[i];
        count[arr[i]]--;
    }

    cout << "\nSorted Array:\n";
    for (int i = 0; i < n; i++)
        cout << sorted[i] << " ";

    cout << endl;
    return 0;
}
```

```
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)

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

void BFS(int start, vector<vector<int>>& graph, int n) {
    vector<bool> visited(n, false);
    queue<int> q;

    visited[start] = true;
    q.push(start);

    cout << "BFS Traversal: ";

    while (!q.empty()) {
        int node = q.front();
        q.pop();
        cout << node << " ";

        for (int neighbor : graph[node]) {
            if (!visited[neighbor]) {
                visited[neighbor] = true;
                q.push(neighbor);
            }
        }
    }
    cout << endl;
}

```

```cpp
int main() {
    int n, edges;
    cout << "Enter number of vertices: ";
    cin >> n;

    cout << "Enter number of edges: ";
    cin >> edges;

    vector<vector<int>> graph(n);

    cout << "Enter edges (u v):\n";
    for (int i = 0; i < edges; i++) {
        int u, v;
        cin >> u >> v;
        graph[u].push_back(v);
        graph[v].push_back(u);
    }

    int start;
    cout << "Enter starting vertex for BFS: ";
    cin >> start;

    BFS(start, graph, n);

    return 0;
}
```

```
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>&)

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

void DFS(int node, vector<vector<int>>& graph, vector<bool>& visited) {
    visited[node] = true;
    cout << node << " ";

    for (int neighbor : graph[node]) {
        if (!visited[neighbor]) {
            DFS(neighbor, graph, visited);
        }
    }
}

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

    cout << "Enter number of edges: ";
    cin >> edges;

    vector<vector<int>> graph(n);

    cout << "Enter edges (u v):\n";
    for (int i = 0; i < edges; i++) {
        int u, v;
        cin >> u >> v;
        graph[u].push_back(v);
        graph[v].push_back(u);
    }

    int start;
    cout << "Enter starting vertex for DFS: ";
    cin >> start;

    vector<bool> visited(n, false);

    cout << "DFS Traversal: ";
    DFS(start, graph, visited);
    cout << endl;

    return 0;
}
```

```
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()

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

#define INF 999999

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

    int graph[50][50];

    cout << "Enter adjacency matrix (use 0 if no edge):\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> graph[i][j];
            if (graph[i][j] == 0 && i != j)
                graph[i][j] = INF;
        }
    }

    int selected[50] = {0};
    selected[0] = 1;

    int edgeCount = 0;
    int totalCost = 0;

    cout << "\nEdges in MST:\n";
```

Q7.cpp | Q8.cpp | Q9.cpp ●

Q9.cpp > main()

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

#define INF 999999

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

    int graph[50][50];

    cout << "Enter adjacency matrix (use 0 if no edge):\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> graph[i][j];
            if (graph[i][j] == 0 && i != j)
                graph[i][j] = INF;
        }
    }

    int selected[50] = {0};
    selected[0] = 1;

    int edgeCount = 0;
    int totalCost = 0;

    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()

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

#define INF 999999

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

    int graph[50][50];

    cout << "Enter adjacency matrix (0 if no edge):\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> graph[i][j];
            if (graph[i][j] == 0 && i != j)
                graph[i][j] = INF;
        }
    }

    int source;
    cout << "Enter source vertex: ";
    cin >> source;

    int dist[50];
    bool visited[50];

    for (int i = 0; i < n; i++) {
        dist[i] = graph[source][i];
        visited[i] = false;
    }
```

```cpp
    dist[source] = 0;
    visited[source] = true;

    for (int count = 1; count < n; count++) {
        int minDist = INF, u = -1;

        for (int i = 0; i < n; i++) {
            if (!visited[i] && dist[i] < minDist) {
                minDist = dist[i];
                u = i;
            }
        }

        visited[u] = true;

        for (int v = 0; v < n; v++) {
            if (!visited[v] && graph[u][v] != INF &&
                dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }

    // Output result
    cout << "\nShortest distances from source " << source << ":\n";
    for (int i = 0; i < n; i++) {
        cout << "To vertex " << i << " = " << dist[i] << endl;
    }

    return 0;
}

```

```
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()

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

int main() {
    int n, W;
    cout << "Enter number of items: ";
    cin >> n;
    cout << "Enter maximum weight of knapsack: ";
    cin >> W;

    vector<int> weight(n), value(n);
    cout << "Enter weights of items:\n";
    for (int i = 0; i < n; i++)
        cin >> weight[i];

    cout << "Enter values of items:\n";
    for (int i = 0; i < n; i++)
        cin >> value[i];

    vector<vector<int>> dp(n + 1, vector<int>(W + 1, 0));

    for (int i = 1; i <= n; i++) {
        for (int w = 1; w <= W; w++) {
            if (weight[i - 1] <= w) {
                dp[i][w] = max(value[i - 1] + dp[i - 1][w - weight[i - 1]], dp[i - 1][w]);
            } else {
                dp[i][w] = dp[i - 1][w];
            }
        }
    }

    cout << "\nMaximum value in knapsack = " << dp[n][W] << endl;

    cout << "Items included (0-based index): ";
    int w = W;
    for (int i = n; i > 0 && w > 0; i--) {
        if (dp[i][w] != dp[i - 1][w]) {
            cout << (i - 1) << " ";
            w -= weight[i - 1];
        }
    }
    cout << endl;

    return 0;
}
```

```
) { .\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>
```