BINARY AND LINEAR SEARCH

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

// Linear Search Function
int linearSearch(int arr[], int n, int key) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == key)
            return i;  // return index
    }
    return -1;
}

// Binary Search Function (array must be sorted)
int binarySearch(int arr[], int n, int key) {
    int low = 0, high = n - 1;

    while (low <= high) {
        int mid = (low + high) / 2;

        if (arr[mid] == key)
            return mid;
        else if (arr[mid] < key)
            low = mid + 1;
        else
            high = mid - 1;
    }

    return -1;
}

int main() {
    int arr[100], n, key, choice;

    cout << "Enter number of elements: ";
    cin >> n;

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

    cout << "Enter value to search: ";
    cin >> key;

    cout << "\nChoose search method:\n";
    cout << "1. Linear Search\n2. Binary Search (array must be sorted)\n";
    cin >> choice;

    int result = -1;

    if (choice == 1)
        result = linearSearch(arr, n, key);
    else if (choice == 2)
        result = binarySearch(arr, n, key);
    else
        cout << "Invalid choice.\n";

    if (result != -1)
        cout << "Element found at index " << result << ".\n";
    else
        cout << "Element not found.\n";

    return 0;
}
```

TREE TRAVERSAL(IN-ORDER , PRE-ORDER , POST-ORDER)

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

// Define a Node
struct Node {
   int data;
   Node* left;
   Node* right;

   // Constructor
   Node(int value) {
      data = value;
      left = right = nullptr;
   }
};

// Inorder Traversal (Left, Root, Right)
void inorder(Node* root) {
   if (root != nullptr) {
      inorder(root->left);
      cout << root->data << " ";
      inorder(root->right);
   }
}

// Preorder Traversal (Root, Left, Right)
void preorder(Node* root) {
   if (root != nullptr) {
      cout << root->data << " ";
      preorder(root->left);
      preorder(root->right);
   }
}

// Postorder Traversal (Left, Right, Root)
void postorder(Node* root) {
   if (root != nullptr) {
      postorder(root->left);
      postorder(root->right);
      cout << root->data << " ";
   }
}

// Driver code
```

```
int main() {
    /*
        Sample Binary Tree:
              1
             / \
            2   3
           / \   \
          4   5   6
    */
    Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    root->right->right = new Node(6);

    cout << "Inorder Traversal: ";
    inorder(root);
    cout << "\n";

    cout << "Preorder Traversal: ";
    preorder(root);
    cout << "\n";

    cout << "Postorder Traversal: ";
    postorder(root);
    cout << "\n";

    return 0;
}
```

STACK

```
#include <iostream>
using namespace std;
#define MAX 100                          // Maximum size of the stack
class Stack {
private:
  int arr[MAX];                          // Array to store stack elements
  int top;                               // Index of the top element
public:
  Stack() {
    top = -1;                            // Initialize stack as empty
  }
  // Push operation
  void push(int value) {
    if (top >= MAX - 1) {
      cout << "Stack Overflow! Cannot push " << value << endl;
      return;
    }
    top++;
    arr[top] = value;
    cout << value << " pushed into the stack.\n";
  }

  // Pop operation
  void pop() {
    if (top < 0) {
      cout << "Stack Underflow! Cannot pop.\n";
      return;
    }
    cout << arr[top] << " popped from the stack.\n";
    top--;
  }

  // Display operation
  void display() {
    if (top < 0) {
      cout << "Stack is empty.\n";
      return;
    }
    cout << "Stack elements are:\n";
    for (int i = top; i >= 0; i--) {
      cout << arr[i] << " ";
    }
    cout << endl;
  }
};
// Driver code
```

```
int main() {
    Stack s;
    int choice, value;

    do {
        cout << "\n--- Stack Menu ---\n";
        cout << "1. Push\n2. Pop\n3. Display\n4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
        case 1:
            cout << "Enter value to push: ";
            cin >> value;
            s.push(value);
            break;
        case 2:
            s.pop();
            break;
        case 3:
            s.display();
            break;
        case 4:
            cout << "Exiting...\n";
            break;
        default:
            cout << "Invalid choice. Please try again.\n";
        }
    } while (choice != 4);
    return 0;
}
```

SINGLE LINKED LIST

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

// Node structure
struct Node {
    int data;
    Node* next;
};

// Linked List class
class LinkedList {
private:
    Node* head;

public:
    // Constructor
    LinkedList() {
        head = nullptr;
    }

    // Create/Insert at end
    void insert(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = nullptr;

        if (head == nullptr) {
            head = newNode;
        } else {
            Node* temp = head;
            while (temp->next != nullptr)
                temp = temp->next;
            temp->next = newNode;
        }
        cout << value << " inserted into the list.\n";
    }

    // Delete a node by value
    void deleteNode(int value) {
        if (head == nullptr) {
            cout << "List is empty. Cannot delete.\n";
            return;
        }

        // If head needs to be deleted
        if (head->data == value) {
            Node* temp = head;
            head = head->next;
            delete temp;
            cout << value << " deleted from the list.\n";
            return;
        }

        // Traverse and delete
        Node* temp = head;
        Node* prev = nullptr;
        while (temp != nullptr && temp->data != value) {
            prev = temp;
            temp = temp->next;
        }

        if (temp == nullptr) {
            cout << "Value not found in the list.\n";
            return;
        }

        prev->next = temp->next;
        delete temp;
        cout << value << " deleted from the list.\n";
    }

    // Display the list
    void display() {
        if (head == nullptr) {
            cout << "List is empty.\n";
            return;
        }

        Node* temp = head;
        cout << "Linked List: ";
        while (temp != nullptr) {
            cout << temp->data << " -> ";
            temp = temp->next;
        }
        cout << "NULL\n";
    }
};

// Driver code
int main() {
    LinkedList list;
    int choice, value;

    do {
        cout << "\n--- Linked List Menu ---\n";
        cout << "1. Insert\n2. Delete\n3. Display\n4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
        case 1:
            cout << "Enter value to insert: ";
            cin >> value;
            list.insert(value);
            break;
        case 2:
            cout << "Enter value to delete: ";
            cin >> value;
            list.deleteNode(value);
            break;
        case 3:
            list.display();
            break;
        case 4:
            cout << "Exiting...\n";
            break;
        default:
            cout << "Invalid choice. Try again.\n";
        }
    } while (choice != 4);

    return 0;
}
```

BUBBLE SORT

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

void bubbleSort(int arr[], int n) {
  bool swapped;

  for (int i = 0; i < n - 1; i++) {
    swapped = false;

    // Last i elements are already in place
    for (int j = 0; j < n - i - 1; j++) {
      // Swap if the element is greater than the next
      if (arr[j] > arr[j + 1]) {
        swap(arr[j], arr[j + 1]);
        swapped = true;
      }
    }
    // If no two elements were swapped in inner loop, array is sorted
    if (!swapped)
      break;
  }
}
void display(int arr[], int n) {
  cout << "Sorted Array: ";
  for (int i = 0; i < n; i++)
    cout << arr[i] << " ";
  cout << endl;
}

// Driver code
int main() {
  int arr[100], n;

  cout << "Enter the number of elements: ";
  cin >> n;

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

  bubbleSort(arr, n);
                                                                           display(arr, n);

  return 0;
}
```

QUEUE

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

#define MAX 100

class Queue {
private:
    int arr[MAX];
    int front, rear;

public:
    Queue() {
        front = -1;
        rear = -1;
    }

    // Enqueue operation
    void enqueue(int value) {
        if (rear == MAX - 1) {
            cout << "Queue Overflow! Cannot insert " << value << endl;
            return;
        }

        if (front == -1) front = 0;              // First insertion
        rear++;
        arr[rear] = value;
        cout << value << " enqueued into the queue.\n";
    }

    // Dequeue operation
    void dequeue() {
        if (front == -1 || front > rear) {
            cout << "Queue Underflow! Cannot dequeue.\n";
            return;
        }

        cout << arr[front] << " dequeued from the queue.\n";
        front++;
    }

    // Display operation
    void display() {
        if (front == -1 || front > rear) {
            cout << "Queue is empty.\n";
            return;
        }

        cout << "Queue elements are: ";
        for (int i = front; i <= rear; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};

// Driver code
int main() {
    Queue q;
    int choice, value;

    do {
        cout << "\n--- Queue Menu ---\n";
        cout << "1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
        case 1:
            cout << "Enter value to enqueue: ";
            cin >> value;
            q.enqueue(value);
            break;
        case 2:
            q.dequeue();
            break;
        case 3:
            q.display();
            break;
        case 4:
            cout << "Exiting...\n";
            break;
        default:
            cout << "Invalid choice. Try again.\n";
        }
    } while (choice != 4);

    return 0;
}
```

SINGLE LINKED LIST (CREATE , SEARCH , DISPLAY)

```
#include <iostream>
using namespace std;
// Node structure
struct Node {
    int data;
    Node* next;
};

// Linked List class
class LinkedList {
private:
    Node* head;

public:
    LinkedList() {
        head = nullptr;
    }

    // Create/Insert at end
    void insert(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = nullptr;

        if (head == nullptr) {
            head = newNode;
        } else {
            Node* temp = head;
            while (temp->next != nullptr)
                temp = temp->next;
            temp->next = newNode;
        }
        cout << value << " inserted into the list.\n";
    }

    // Search for a value
    void search(int value) {
        Node* temp = head;
        int position = 1;
        bool found = false;

        while (temp != nullptr) {
            if (temp->data == value) {
                cout << value << " found at position " << position << ".\n";
                found = true;
                break;
            }
            temp = temp->next;
            position++;
        }

        if (!found) {
            cout << value << " not found in the list.\n";
        }
    }

    // Display the list
    void display() {
        if (head == nullptr) {
            cout << "List is empty.\n";
            return;
        }

        Node* temp = head;
        cout << "Linked List: ";
        while (temp != nullptr) {
            cout << temp->data << " -> ";
            temp = temp->next;
        }
        cout << "NULL\n";
    }
};

// Driver code
int main() {
    LinkedList list;
    int choice, value;

    do {
        cout << "\n--- Singly Linked List Menu ---\n";
        cout << "1. Insert\n2. Search\n3. Display\n4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
        case 1:
            cout << "Enter value to insert: ";
            cin >> value;
            list.insert(value);
            break;
        case 2:
            cout << "Enter value to search: ";
            cin >> value;
            list.search(value);
            break;
        case 3:
            list.display();
            break;
        case 4:
            cout << "Exiting...\n";
            break;
        default:
            cout << "Invalid choice. Try again.\n";
        }
    } while (choice != 4);

    return 0;
}
```

DIJIKSTRA’S ALGORITHM

```
#include <iostream>
#include <limits.h>
using namespace std;

#define V 5  // Number of vertices in the graph

// Find the vertex with minimum distance value
int minDistance(int dist[], bool visited[]) {
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++) {
        if (!visited[v] && dist[v] <= min) {
            min = dist[v], min_index = v;
        }
    }

    return min_index;
}

// Dijkstra's Algorithm
void dijkstra(int graph[V][V], int src) {
    int dist[V];       // Output array. dist[i] holds the shortest distance from src to i
    bool visited[V];   // visited[i] will be true if vertex i is included in shortest path tree

    // Initialize all distances as INFINITE and visited[] as false
    for (int i = 0; i < V; i++) {
        dist[i] = INT_MAX;
        visited[i] = false;
    }

    // Distance from source to itself is always 0
    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, visited);

        visited[u] = true;

        // Update distance of adjacent vertices of the picked vertex
        for (int v = 0; v < V; v++) {
            if (!visited[v] && graph[u][v] && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
            }
        }
    }

    // Print the result
    cout << "Vertex\tDistance from Source " << src << endl;
    for (int i = 0; i < V; i++)
        cout << i << "\t" << dist[i] << endl;
}

// Driver code
int main() {
    // Example graph represented as an adjacency matrix
    int graph[V][V] = {
        {0, 10, 0, 0, 5},
        {0, 0, 1, 0, 2},
        {0, 0, 0, 4, 0},
        {7, 0, 6, 0, 0},
        {0, 3, 9, 2, 0}
    };

    int source;
    cout << "Enter source vertex (0 to " << V-1 << "): ";
    cin >> source;

    dijkstra(graph, source);

    return 0;
}
```

ADJACENCY MATRIX

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

class Graph {
private:
    int adjMatrix[10][10]; // Maximum 10 nodes for simplicity
    int numVertices;

public:
    // Constructor
    Graph(int vertices) {
        numVertices = vertices;
        // Initialize matrix with 0s
        for (int i = 0; i < numVertices; i++) {
            for (int j = 0; j < numVertices; j++) {
                adjMatrix[i][j] = 0;
            }
        }
    }

    // Add edge
    void addEdge(int i, int j) {
        if (i >= numVertices || j >= numVertices || i < 0 || j < 0) {
            cout << "Invalid edge!\n";
        } else {
            adjMatrix[i][j] = 1;
            adjMatrix[j][i] = 1; // For undirected graph
        }
    }

    // Display matrix
    void display() {
        cout << "\nAdjacency Matrix:\n";
        for (int i = 0; i < numVertices; i++) {
            for (int j = 0; j < numVertices; j++) {
                cout << adjMatrix[i][j] << " ";
            }
            cout << endl;
        }
    }
};

// Driver code
int main() {
    int vertices, edges, v1, v2;
    cout << "Enter number of vertices: ";
    cin >> vertices;

    Graph g(vertices);

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

    cout << "Enter edges (format: vertex1 vertex2):\n";
    for (int i = 0; i < edges; i++) {
        cin >> v1 >> v2;
        g.addEdge(v1, v2);
    }

    g.display();

    return 0;
}
```