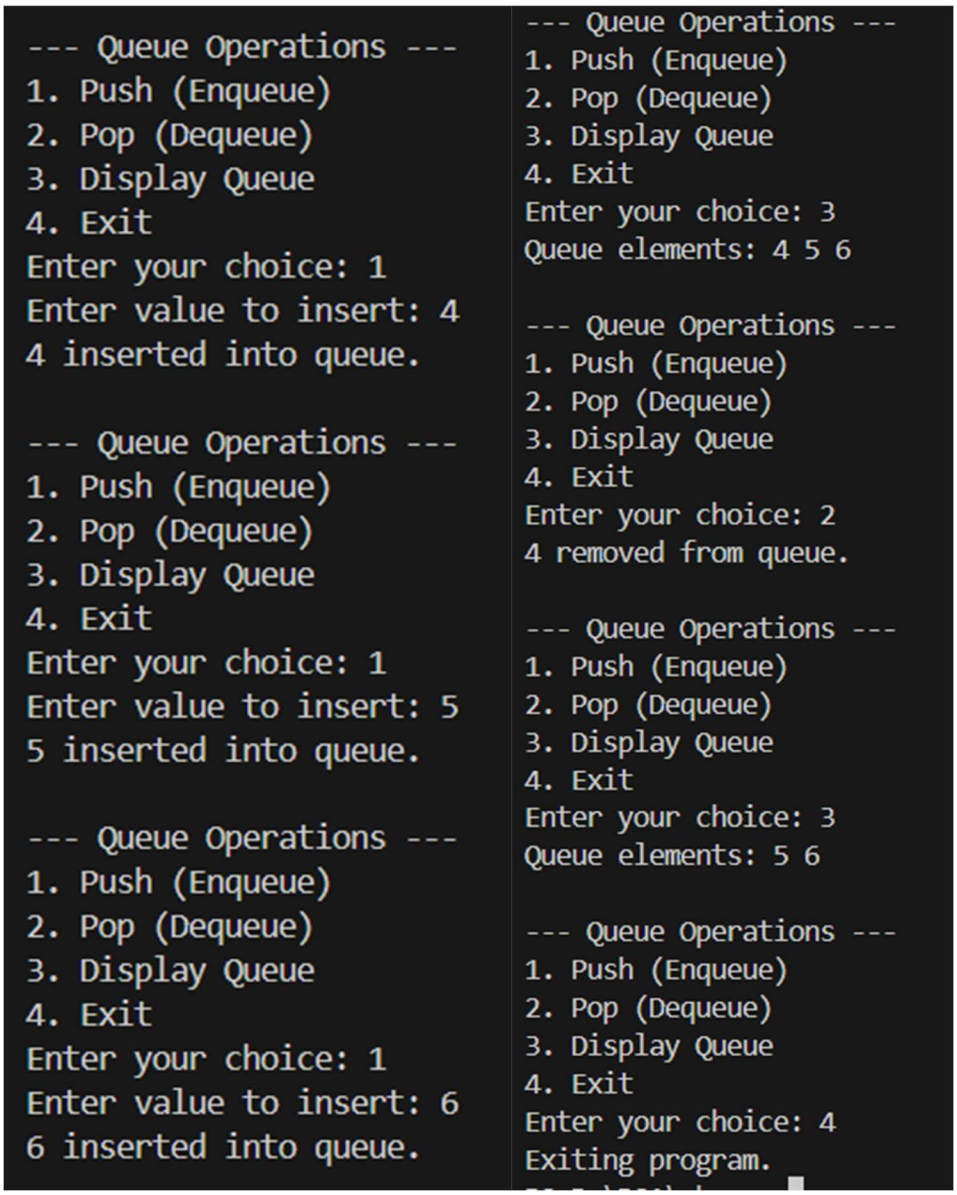
Q.13 WAP to implement push pop operations on a queue using liked list.

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   Node* next;
};
class Queue {
private:
   Node* front;
   Node* rear;
public:
  Queue() {
      front = rear = nullptr;
   }
   void push(int value) {
      Node* newNode = new Node();
      newNode->data = value;
      newNode->next = nullptr;
      if (rear == nullptr) {
         front = rear = newNode;
         cout << value << " inserted into queue.\n";
         return; }
      rear->next = newNode;
      rear = newNode;
      cout << value << " inserted into queue.\n";
 }
```

```
    void pop() {
        if (front == nullptr) {
            cout << "Queue Underflow! No elements to delete.\n";
            return;
        }
        Node* temp = front;
        front = front->next;
        if (front == nullptr)
            rear = nullptr;
        cout << temp->data << " removed from queue.\n";
        delete temp;
    }
    void display() {
        if (front == nullptr) {
            cout << "Queue is empty.\n";
            return;
        }
        Node* temp = front;
        cout << "Queue elements: ";
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;   }
        cout << "\n";  }
};
int main() {
    Queue q;
    int choice, value;
    do {
        cout << "\n--- Queue Operations ---\n";
```

```
        cout << "1. Push (Enqueue)\n";
        cout << "2. Pop (Dequeue)\n";
        cout << "3. Display Queue\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice) {
        case 1:
            cout << "Enter value to insert: ";
            cin >> value;
            q.push(value);
            break;
        case 2:
            q.pop();
            break;
        case 3:
            q.display();
            break;
        case 4:
            cout << "Exiting program.\n";
            break;
        default:
            cout << "Invalid choice! Try again.\n";
     }
    } while (choice != 4);
    return 0;
}
```

Output:



```
--- Queue Operations ---
1. Push (Enqueue)
2. Pop (Dequeue)
3. Display Queue
4. Exit
Enter your choice: 1
Enter value to insert: 4
4 inserted into queue.

--- Queue Operations ---
1. Push (Enqueue)
2. Pop (Dequeue)
3. Display Queue
4. Exit
Enter your choice: 1
Enter value to insert: 5
5 inserted into queue.

--- Queue Operations ---
1. Push (Enqueue)
2. Pop (Dequeue)
3. Display Queue
4. Exit
Enter your choice: 1
Enter value to insert: 6
6 inserted into queue.
```

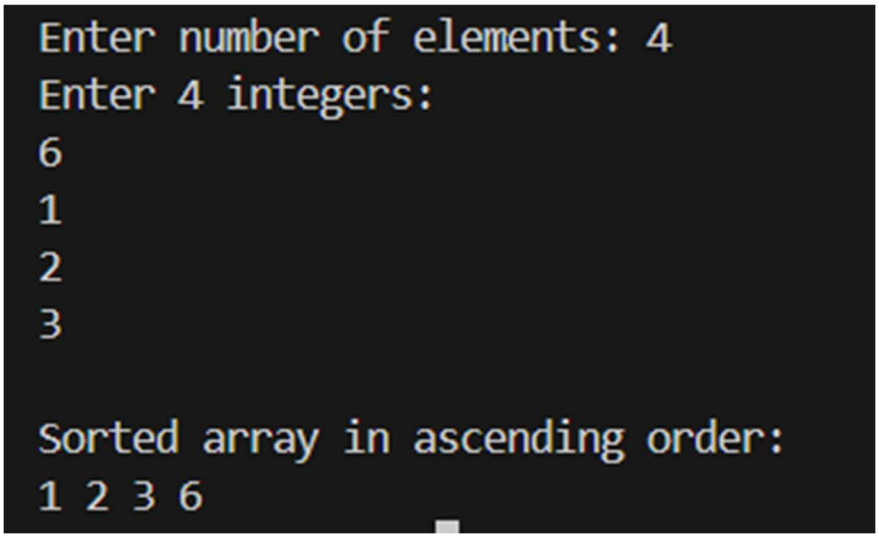
```
--- Queue Operations ---
1. Push (Enqueue)
2. Pop (Dequeue)
3. Display Queue
4. Exit
Enter your choice: 3
Queue elements: 4 5 6

--- Queue Operations ---
1. Push (Enqueue)
2. Pop (Dequeue)
3. Display Queue
4. Exit
Enter your choice: 2
4 removed from queue.

--- Queue Operations ---
1. Push (Enqueue)
2. Pop (Dequeue)
3. Display Queue
4. Exit
Enter your choice: 3
Queue elements: 5 6

--- Queue Operations ---
1. Push (Enqueue)
2. Pop (Dequeue)
3. Display Queue
4. Exit
Enter your choice: 4
Exiting program.
```

Q.14 program to sort an array of integers in ascending order using bubble sort.

```
#include <iostream>

using namespace std;


int main() {

   int n;

   cout << "Enter number of elements: ";

   cin >> n

   int arr[n];

   cout << "Enter " << n << " integers:\n";

   for (int i = 0; i < n; i++) {

      cin >> arr[i];

   }


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

      bool swapped = false;

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

         if (arr[j] > arr[j + 1]) {

            int temp = arr[j];

            arr[j] = arr[j + 1];

            arr[j + 1] = temp;

            swapped = true;

    }

}

      if (!swapped)

         break;

   }

   cout << "\nSorted array in ascending order:\n";

   for (int i = 0; i < n; i++) {
```

```
        cout << arr[i] << " ";
    }
    cout << endl;
    return 0;
}
```

Output:

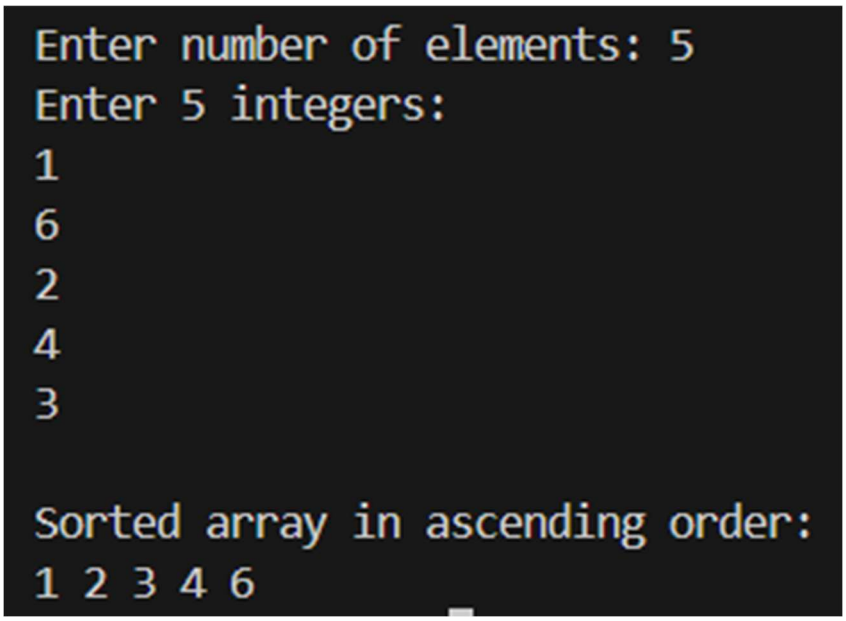


Q.15 Program to sort an array of integers in ascending order using selection sort.

```
#include <iostream>

using namespace std;


int main() {
   int n;
   cout << "Enter number of elements: ";
   cin >> n;
   int arr[n];
   cout << "Enter " << n << " integers:\n";
   for (int i = 0; i < n; i++) {
      cin >> arr[i];
   }
   for (int i = 0; i < n - 1; i++) {
      int minIndex = i;
      for (int j = i + 1; j < n; j++) {
         if (arr[j] < arr[minIndex]) {
            minIndex = j;
         }
      }


      int temp = arr[i];
      arr[i] = arr[minIndex];
      arr[minIndex] = temp;
   }


   cout << "\nSorted array in ascending order:\n";
   for (int i = 0; i < n; i++) {
      cout << arr[i] << " ";
```

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

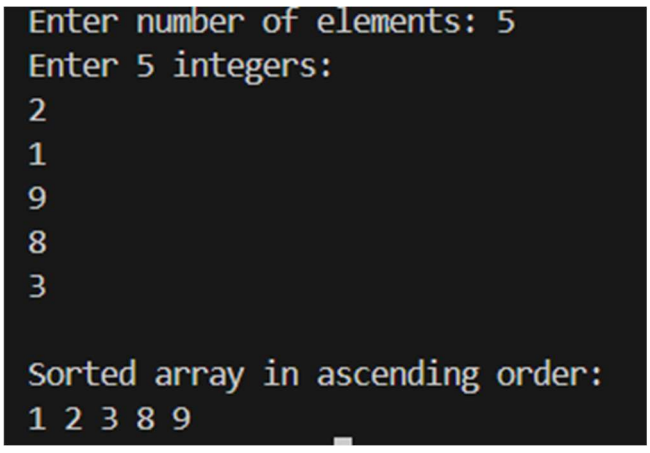
Output:



Q.16 Program to sort an array of integers in ascending order using insertion sort.

```
#include <iostream>

using namespace std;

int main() {
   int n;
   cout << "Enter number of elements: ";
   cin >> n;
   int arr[n];
   cout << "Enter " << n << " integers:\n";
   for (int i = 0; i < n; i++) {
      cin >> arr[i];
   }
   for (int i = 1; i < n; i++) {
      int key = arr[i];
      int j = i - 1;
      while (j >= 0 && arr[j] > key) {
         arr[j + 1] = arr[j];
         j--;
      }
      arr[j + 1] = key;
   }
   cout << "\nSorted array in ascending order:\n";
   for (int i = 0; i < n; i++) {
      cout << arr[i] << " ";
   }
   cout << endl;

   return 0;
}
```

output:

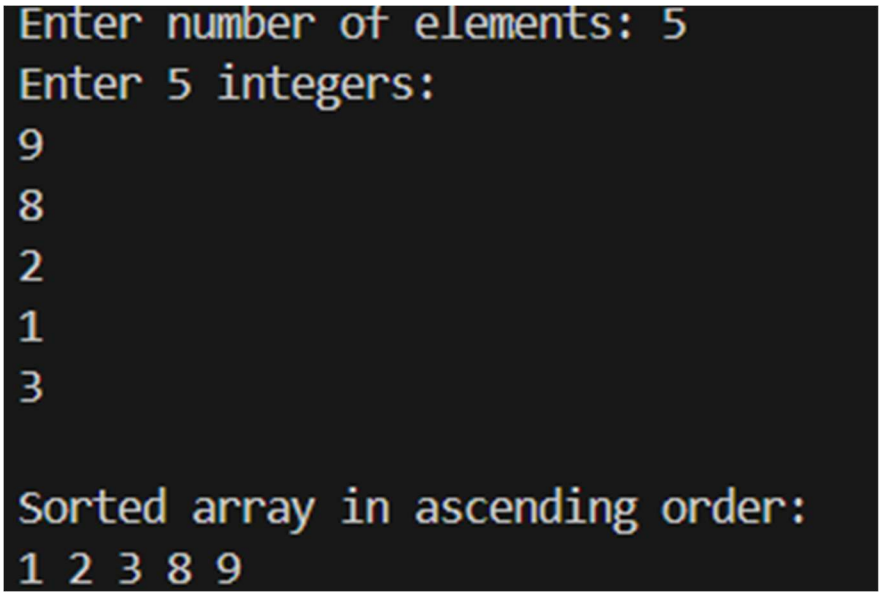


Q.17 Program to sort an array of integers in ascending order using quick sort.

```
#include <iostream>

using namespace std;

void swap(int &a, int &b) {
   int temp = a;
   a = b;
   b = temp;
}
int partition(int arr[], int low, int high) {
   int pivot = arr[high];
   int i = low - 1;
   for (int j = low; j < high; j++) {
      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);
   }
}
```

```
int main() {
    int n;
    cout << "Enter number of elements: ";
    cin >> n;
    int arr[n];
    cout << "Enter " << n << " integers:\n";
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    quickSort(arr, 0, n - 1);
    cout << "\nSorted array in ascending order:\n";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    return 0;
}
```

Output:



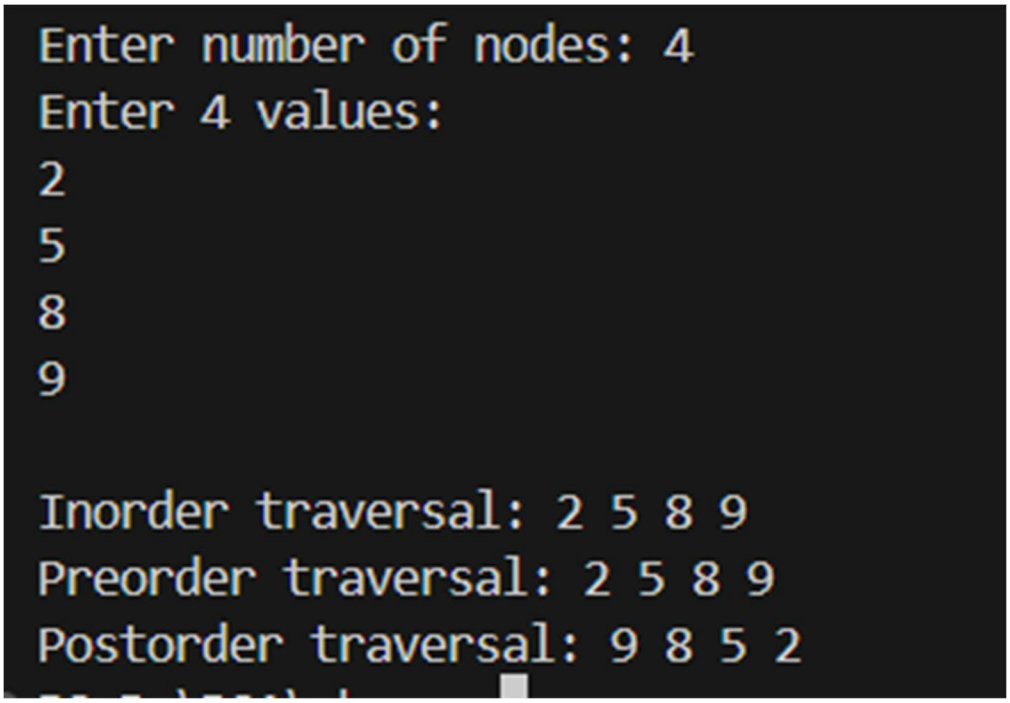
Q.18 Program to traverse a Binary search tree in Pre-order, In-order and Post-order.

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   Node* left;
   Node* right;
};
Node* createNode(int value) {
   Node* newNode = new Node();
   newNode->data = value;
   newNode->left = newNode->right = nullptr;
   return newNode;
}
Node* insert(Node* root, int value) {
   if (root == nullptr) {
      return createNode(value);
   }
   if (value < root->data)
      root->left = insert(root->left, value);
   else if (value > root->data)
      root->right = insert(root->right, value);
   return root;
}
void inorder(Node* root) {
   if (root == nullptr)
      return;
   inorder(root->left);
   cout << root->data << " ";
```

```
    inorder(root->right);
}
void preorder(Node* root) {
    if (root == nullptr)
        return;
    cout << root->data << " ";
    preorder(root->left);
    preorder(root->right);
}
void postorder(Node* root) {
    if (root == nullptr)
        return;
    postorder(root->left);
    postorder(root->right);
    cout << root->data << " ";
}

int main() {
    Node* root = nullptr;
    int n, value;
    cout << "Enter number of nodes: ";
    cin >> n;
    cout << "Enter " << n << " values:\n";
    for (int i = 0; i < n; i++) {
        cin >> value;
        root = insert(root, value);
    }

    cout << "\nInorder traversal: ";
```

```
    inorder(root);
    cout << "\nPreorder traversal: ";
    preorder(root);
    cout << "\nPostorder traversal: ";
    postorder(root);
    cout << endl;
    return 0;
}
```

Output:

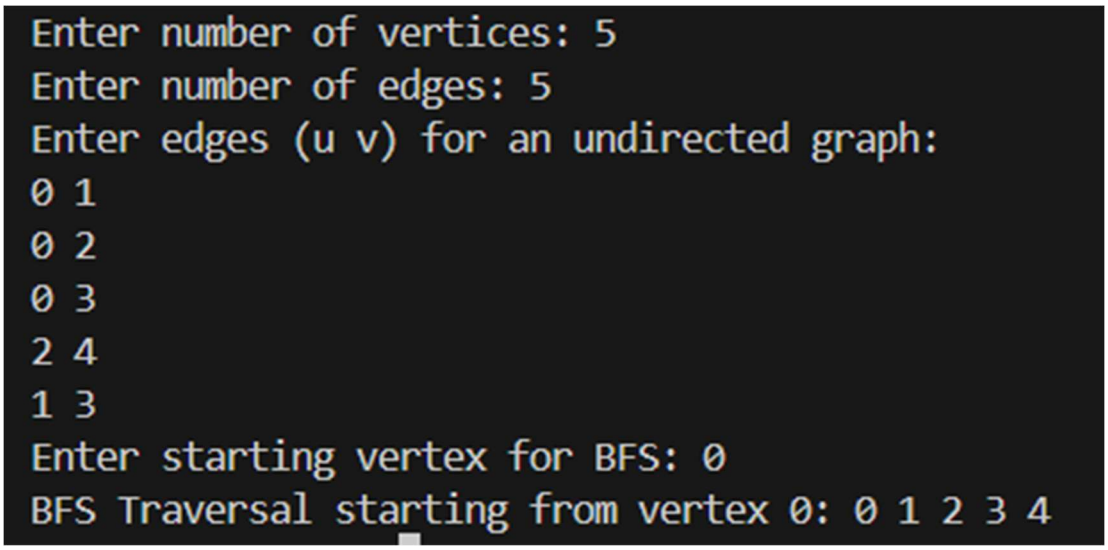


Q.19 Program to traverse graphs using BFS.

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;
void bfs(int start, vector<vector<int>>& adj, int vertices) {
    vector<bool> visited(vertices, false);
    queue<int> q;
    visited[start] = true;
    q.push(start);
    cout << "BFS Traversal starting from vertex " << start << ": ";
    while (!q.empty()) {
        int node = q.front();
        q.pop();
        cout << node << " ";
        for (int neighbor : adj[node]) {
            if (!visited[neighbor]) {
                visited[neighbor] = true;
                q.push(neighbor);
            }
        }
    }
    cout << endl;
}
int main() {
    int vertices, edges;
    cout << "Enter number of vertices: ";
    cin >> vertices;
    cout << "Enter number of edges: ";
```

```
    cin >> edges;
    vector<vector<int>> adj(vertices);
    cout << "Enter edges (u v) for an undirected graph:\n";
    for (int i = 0; i < edges; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u); // remove this line if graph is directed
    }
    int start;
    cout << "Enter starting vertex for BFS: ";
    cin >> start;
    bfs(start, adj, vertices);
    return 0;
}
```

Output:



Q.20 Program to traverse graphs using DFS.

```
#include <iostream>
#include <vector>
using namespace std;
void dfs(int node, vector<vector<int>>& adj, vector<bool>& visited) {
    visited[node] = true;
    cout << node << " ";
    for (int neighbor : adj[node]) {
        if (!visited[neighbor]) {
            dfs(neighbor, adj, visited);
        }
    }
}

int main() {
    int vertices, edges;
    cout << "Enter number of vertices: ";
    cin >> vertices;
    cout << "Enter number of edges: ";
    cin >> edges;
    vector<vector<int>> adj(vertices);
    cout << "Enter edges (u v) for an undirected graph:\n";
    for (int i = 0; i < edges; i++) {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
```

```
    int start;
    cout << "Enter starting vertex for DFS: ";
    cin >> start;
    vector<bool> visited(vertices, false);
    cout << "DFS Traversal starting from vertex " << start << ": ";
    dfs(start, adj, visited);
    cout << endl;
    return 0;
}
```

Output: