1. **Problem Name:** Write a C++ program to implement a Binary Search Tree with Insertion, Traversal(In-Order, Pre-Order, Post-Order) and Search operation.

Start here | Untitled1.cpp

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

class Node {
 public:
  int data;
  Node* left;
  Node* right;

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

class BST {
 public:
  Node* root;

  BST() {
    root = nullptr;
  }

  // Insert function
  Node* insert(Node* node, int value) {
    if (node == nullptr) {
      return new Node(value);
    }

    if (value < node->data) {
      node->left = insert(node->left, value);
    } else if (value > node->data) {
      node->right = insert(node->right, value);
    }

    return node;
  }
```

Start here | Untitled1.cpp

```cpp
  // In-order traversal (LNR)
  void inOrder(Node* node) {
    if (node == nullptr) return;
    inOrder(node->left);
    cout << node->data << " ";
    inOrder(node->right);
  }

  // Pre-order traversal (NLR)
  void preOrder(Node* node) {
    if (node == nullptr) return;
    cout << node->data << " ";
    preOrder(node->left);
    preOrder(node->right);
  }

  // Post-order traversal (LRN)
  void postOrder(Node* node) {
    if (node == nullptr) return;
    postOrder(node->left);
    postOrder(node->right);
    cout << node->data << " ";
  }

  // Search function
  bool search(Node* node, int key) {
    if (node == nullptr) return false;
    if (node->data == key) return true;
    if (key < node->data) return search(node->left, key);
    return search(node->right, key);
  }
};

```

```cpp
int main() {
  BST tree;
  int n;

  cout << "Enter number of values to insert: ";
  cin >> n;

  int* values = new int[n];
  cout << "Enter the values: ";
  for (int i = 0; i < n; ++i) {
    cin >> values[i];
    tree.root = tree.insert(tree.root, values[i]);
  }

  cout << "\nIn-order Traversal: ";
  tree.inOrder(tree.root);
  cout << "\nPre-order Traversal: ";
  tree.preOrder(tree.root);
  cout << "\nPost-order Traversal: ";
  tree.postOrder(tree.root);
  cout << endl;

  int key;
  cout << "\nEnter value to search: ";
  cin >> key;

  if (tree.search(tree.root, key)) {
    cout << "Found" << endl;
  } else {
    cout << "Not Found" << endl;
  }

  delete[] values;
  return 0;
}
```

**Output:**