// // 1. You are given a text file, named “students.txt” that contains students’ records. Each Line

// // contains information of a single student in the form of

// <Student Name, Roll No,

// // Department>.

// // A. Read the records from the file into an array of structures.

// // B. Three Options will turn up: (1) Bubble Sort, (2) Binary Search, and (3) Quit. In the

// // following we describe what your C/C++ program will do on Selecting the options.

// // (1) Bubble Sort: Sorts the records based on Student Name. If more than One students has

// // the same name, then sort them on their roll no.

// // (2) Binary Search: Given a student name, the function will return all the Student records

// // <Student Name, Roll No, Department> having the Student name.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include <algorithm>

using namespace std;

// Structure to store student records
struct Student {
    string name;
    int rollNo;
    string department;
};
```

```
// Function to read student records from a CSV file
void readStudents(vector<Student>& students, const string& filename) {
    ifstream file(filename);
    if (!file) {
        cout << "Error: Unable to open file!" << endl;
        return;
    }

    string line;
    while (getline(file, line)) {
        stringstream ss(line);
        string name, rollNoStr, department;

        // Read name, roll number, department
        getline(ss, name, ',');
        getline(ss, rollNoStr, ',');
        getline(ss, department, ',');

        // Convert roll number from string to integer
        Student s;
        s.name = name;
        s.rollNo = stoi(rollNoStr);
        s.department = department;

        students.push_back(s);
    }
    file.close();
}
```

```
// Bubble Sort function (Sorts by Name, then Roll No)
void bubbleSort(vector<Student>& students) {
    int n = students.size();
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (students[j].name > students[j + 1].name ||
                (students[j].name == students[j + 1].name && students[j].rollNo > students[j + 1].rollNo)) {
                swap(students[j], students[j + 1]);
            }
        }
    }
    cout << "Records sorted successfully.\n";
}

// Function to display student records
void displayStudents(const vector<Student>& students) {
    for (const auto& s : students) {
        cout << s.name << ", " << s.rollNo << ", " << s.department << endl;
    }
}

// Binary Search function for student name
void binarySearch(const vector<Student>& students, const string& key) {
    int left = 0, right = students.size() - 1;
    bool found = false;
```

```
    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (students[mid].name == key) {
            found = true;
            int i = mid;

            // Print left occurrences
            while (i >= 0 && students[i].name == key) {
                cout << students[i].name << ", " << students[i].rollNo << ", " <<
students[i].department << endl;
                i--;
            }

            // Print right occurrences
            i = mid + 1;
            while (i < students.size() && students[i].name == key) {
                cout << students[i].name << ", " << students[i].rollNo << ", " <<
students[i].department << endl;
                i++;
            }
            break;
        }
        else if (students[mid].name < key) {
            left = mid + 1;
        }
        else {
            right = mid - 1;
        }
```

```
    }

    if (!found) {
        cout << "No student found with the name '" << key << "'.\n";
    }
}

int main() {
    vector<Student> students;
    readStudents(students, "students.csv");

    if (students.empty()) {
        cout << "No student records found.\n";
        return 0;
    }

    int choice;
    do {
        cout << "\nChoose an option:\n";
        cout << "1. Bubble Sort\n2. Binary Search\n3. Quit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        cin.ignore(); // To handle newline character from input buffer

        switch (choice) {
            case 1:
                bubbleSort(students);
                cout << "Sorted Student Records:\n";
```

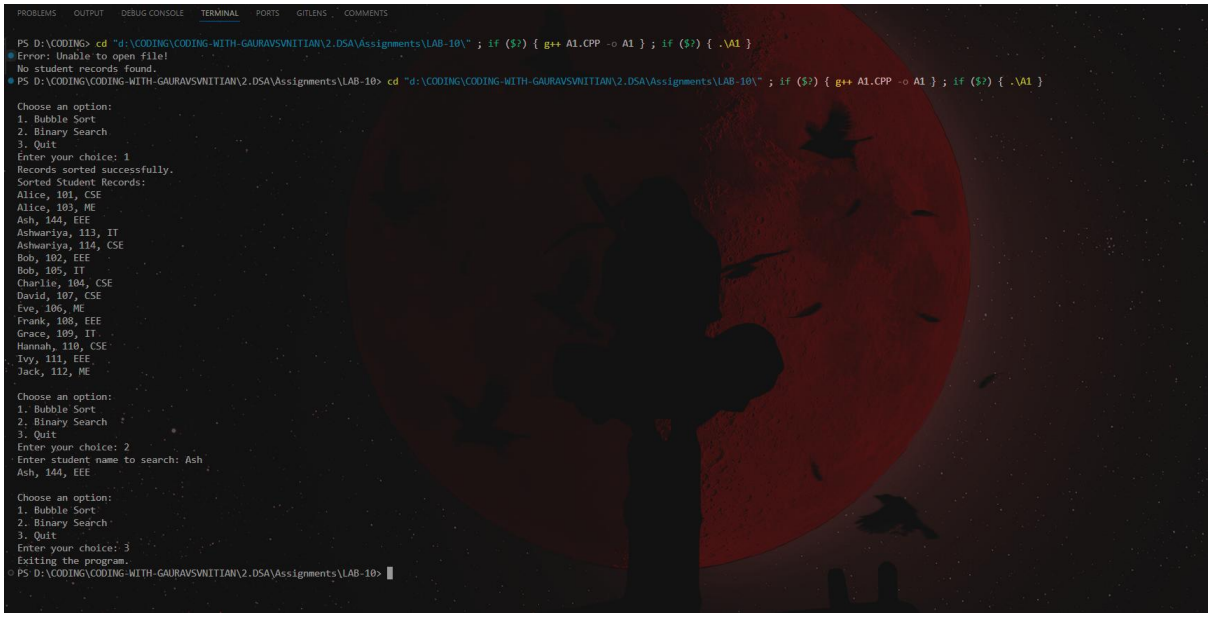
```
            displayStudents(students);
            break;

        case 2: {
            string searchName;
            cout << "Enter student name to search: ";
            getline(cin, searchName);
            binarySearch(students, searchName);
            break;
        }

        case 3:
            cout << "Exiting the program.....\n";
            break;

        default:
            cout << "Invalid choice! Please try again.\n";
      }
   } while (choice != 3);

   return 0;
}
```




```
// 2. Let A[n] be an array of n distinct integers. If I < j and A[i] > A[j], then the pair (I, j) Is
// called an inversion of A. Write a C/C++ program that determines the number of
// Inversions in any permutation on n elements in O(n lg n) worst-case time. (Hint: Modify
// merge sort)
// Example: A = {4, 1, 3, 2} output is 4

#include <iostream>
#include <vector>

using namespace std;

int merge(vector<int>& v, int low, int mid, int high) {
    int n1 = mid - low + 1;
    int n2 = high - mid;

    vector<int> v1(n1);
    vector<int> v2(n2);
```

```
// Copy elements into left and right subarrays
for (int i = 0; i < n1; i++) {
    v1[i] = v[low + i];
}

for (int i = 0; i < n2; i++) {
    v2[i] = v[mid + 1 + i];
}

int i = 0, j = 0, k = low;
int inversions = 0;
while (i < n1 && j < n2) {
    if (v1[i] <= v2[j]) {
        v[k++] = v1[i++];
    } else {
        v[k++] = v2[j++];
        inversions += n1 - i; // Count inversions
    }
}

while (i < n1) {
    v[k++] = v1[i++];
}

while (j < n2) {
    v[k++] = v2[j++];
}
```

```
    return inversions;
}

int mergesort(vector<int>& v, int low, int high) {
    int inversions = 0;
    if (low < high) {
        int mid = low + (high - low) / 2; // Correct mid calculation

        // Recursively sort both halves
        inversions += mergesort(v, low, mid);
        inversions += mergesort(v, mid + 1, high);

        // Merge sorted halves
        inversions += merge(v, low, mid, high);
    }
    return inversions;
}

int main() {
    int n;
    cout << "Enter size of Array: ";
    cin >> n;

    vector<int> v(n);

    cout << "Enter elements of Array: ";
    for (int i = 0; i < n; i++) {
        cin >> v[i];
```

```
    }

    int inversions = mergesort(v, 0, n - 1);

    cout << "Number of Inversions: " << inversions << endl;

    return 0;
}
```