***Assignment 4***

Name: Karan P. Shah

Regis. No.: 2310700758

Aim:

Task 1: Consider first/second year course-code choices of 100 students.   
Find inversion count of these choices.  
Find students with zero, one, two, three inversion counts comment on your result.

Task 2: Consider large integers of size 10, 50, 100, 500 and 1000 digits.   
Write integer multiplication program   
Write integer multiplication program using divide and conquer technique.

Task 1 **(ASSUMPTION: COURSE CODES ARE 5 DIGIT POSITIVE NUMBERS ONLY)**

Program

#include <bits/stdc++.h>

using namespace std;

// Brute force method to count inversions

int countInversionsBruteForce(const vector<int>& course\_codes) {

    int inversions = 0;

    int n = course\_codes.size();

    for (int i = 0; i < n; ++i) {

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

            if (course\_codes[i] > course\_codes[j]) {

                inversions++;

            }

        }

    }

    return inversions;

}

// Merge function used in merge sort to count inversions

int mergeAndCount(vector<int>& arr, vector<int>& temp, int left, int mid, int right) {

    int i = left;  // Starting index for left subarray

    int j = mid + 1;  // Starting index for right subarray

    int k = left;  // Starting index to be sorted

    int inv\_count = 0;

    while (i <= mid && j <= right) {

        if (arr[i] <= arr[j]) {

            temp[k++] = arr[i++];

        }

        else {

            temp[k++] = arr[j++];

            inv\_count += (mid - i + 1);  // Increment inversion count

        }

    }

    while (i <= mid)

        temp[k++] = arr[i++];

    while (j <= right)

        temp[k++] = arr[j++];

    for (i = left; i <= right; i++)

        arr[i] = temp[i];

    return inv\_count;

}

// Divide and Conquer approach (Merge Sort) to count inversions

int mergeSortAndCount(vector<int>& arr, vector<int>& temp, int left, int right) {

    int mid, inv\_count = 0;

    if (right > left) {

        mid = (right + left) / 2;

        inv\_count += mergeSortAndCount(arr, temp, left, mid);

        inv\_count += mergeSortAndCount(arr, temp, mid + 1, right);

        inv\_count += mergeAndCount(arr, temp, left, mid, right);

    }

    return inv\_count;

}

// Function to use divide and conquer to count inversions

int countInversionsOptimized(vector<int> course\_codes) {

    vector<int> temp(course\_codes.size());

    return mergeSortAndCount(course\_codes, temp, 0, course\_codes.size() - 1);

}

// Function to check if a course code is valid (positive integer)

bool isValidCourseCode(const string& code) {

    if (code.empty()) return false;

    // Ensure the course code is numeric and not negative

    for (char c : code) {

        if (!isdigit(c)) return false;

    }

    return true;

}

// Function to read the CSV file and parse student course choices

vector<vector<string>> readCSV(const string& filename) {

    vector<vector<string>> students;

    ifstream file(filename);

    string line;

    if (!file.is\_open()) {

        cerr << "Error: Unable to open the file " << filename << endl;

        return students;

    }

    // Skip the header

    getline(file, line);

    // Read each student's data

    int line\_number = 2; // Start after the header

    while (getline(file, line)) {

        stringstream ss(line);

        vector<string> course\_codes;

        string value;

        // Skip the first column (student ID)

        getline(ss, value, ',');

        if (value.empty()) {

            cout << "Error: Missing student ID at line " << line\_number << ". Skipping this student's entry." << endl;

            line\_number++;

            continue;

        }

        bool is\_valid = true;

        // Read the course codes

        while (getline(ss, value, ',')) {

            // Trim leading/trailing spaces

            value.erase(0, value.find\_first\_not\_of(" \t"));

            value.erase(value.find\_last\_not\_of(" \t") + 1);

            course\_codes.push\_back(value);

        }

        if (course\_codes.size() != 5) {  // Each student should have exactly 5 course codes

            cout << "Error: Inconsistent number of course codes at line " << line\_number << ". Expected 5, but got " << course\_codes.size() << ". Skipping this student's entry." << endl;

            is\_valid = false;

        }

        if (is\_valid) {

            students.push\_back(course\_codes);

        }

        line\_number++;

    }

    file.close();

    return students;

}

int main() {

    // File path for the CSV file

    string file\_path = "student\_course\_choices.csv";

    // Read the CSV file and load student course choices

    vector<vector<string>> students = readCSV(file\_path);

    if (students.empty()) {

        cout << "No valid student course data available for processing." << endl;

        return 1;

    }

    // Map to store the count of students with a certain number of inversions (for both methods)

    map<int, int> inversion\_count\_map\_brute\_force;

    map<int, int> inversion\_count\_map\_optimized;

    // Loop over each student and calculate inversions using both methods

    int student\_number = 1;

    for (const auto& course\_codes\_str : students) {

        bool has\_invalid\_code = false;

        vector<int> course\_codes;

        cout << "Student " << student\_number << ": ";

        for (const string& code\_str : course\_codes\_str) {

            if (!isValidCourseCode(code\_str)) {

                cout << "Non-numeric or invalid course code '" << code\_str << "' found. Skipping inversion calculation." << endl;

                has\_invalid\_code = true;

                break;

            }

            int code = stoi(code\_str);

            if (code < 0) {

                cout << "Negative course code " << code << " found. Skipping inversion calculation." << endl;

                has\_invalid\_code = true;

                break;

            }

            course\_codes.push\_back(code);

        }

        if (!has\_invalid\_code) {

            int inversions\_brute\_force = countInversionsBruteForce(course\_codes);

            int inversions\_optimized = countInversionsOptimized(course\_codes);

            // Print the inversion counts for each student (both methods)

            cout << "Brute-force inversions = " << inversions\_brute\_force

                 << ", Optimized inversions = " << inversions\_optimized << endl;

            // Count inversions for brute-force method

            inversion\_count\_map\_brute\_force[inversions\_brute\_force]++;

            // Count inversions for optimized method

            inversion\_count\_map\_optimized[inversions\_optimized]++;

        }

        student\_number++;

    }

    // Output the inversion count summary (brute force)

    cout << "\nSummary of Inversion Counts (Brute Force):" << endl;

    for (const auto& pair : inversion\_count\_map\_brute\_force) {

        cout << pair.first << " inversions: " << pair.second << " students" << endl;

    }

    // Output the inversion count summary (optimized approach)

    cout << "\nSummary of Inversion Counts (Optimized Divide-and-Conquer):" << endl;

    for (const auto& pair : inversion\_count\_map\_optimized) {

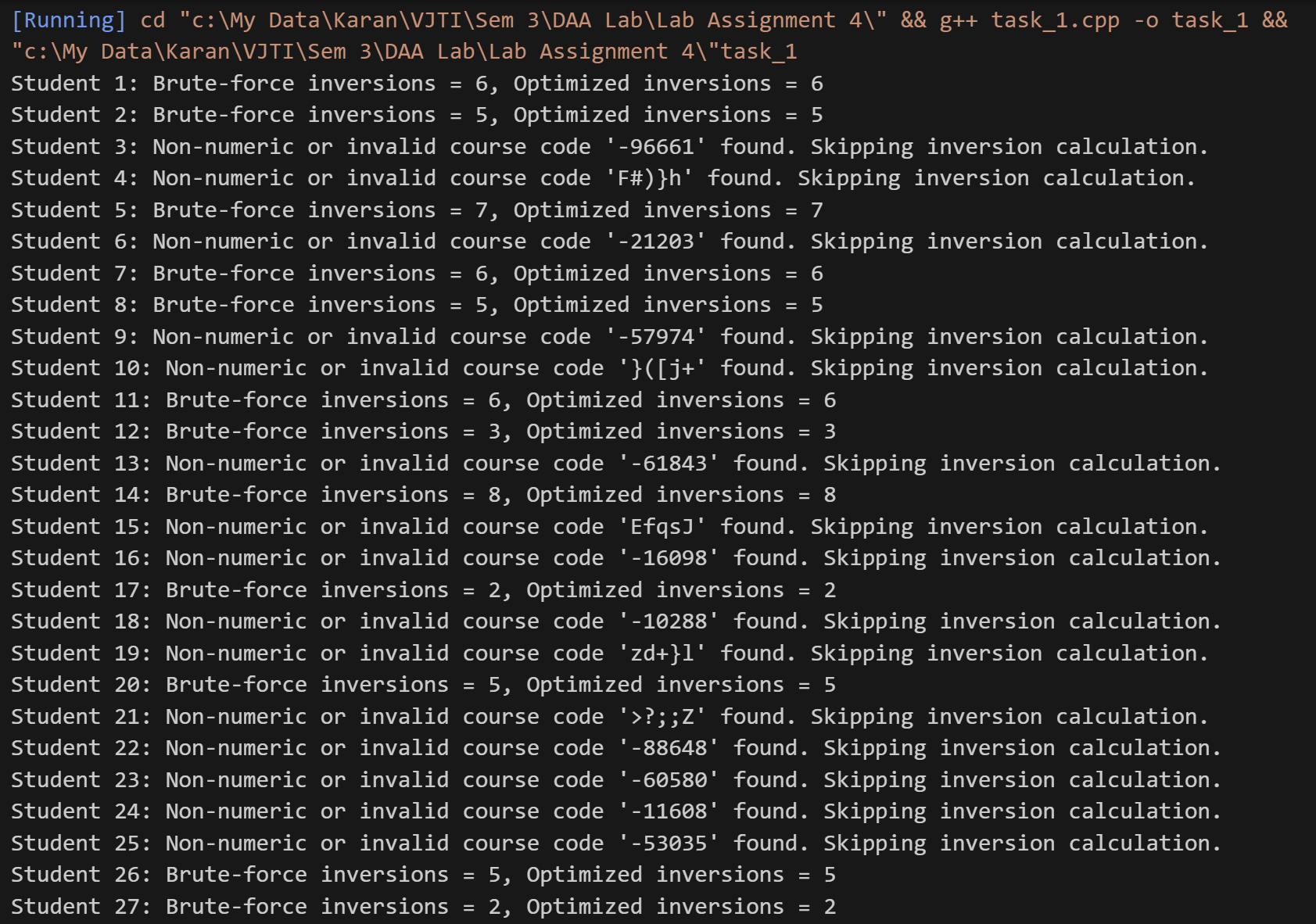
        cout << pair.first << " inversions: " << pair.second << " students" << endl;

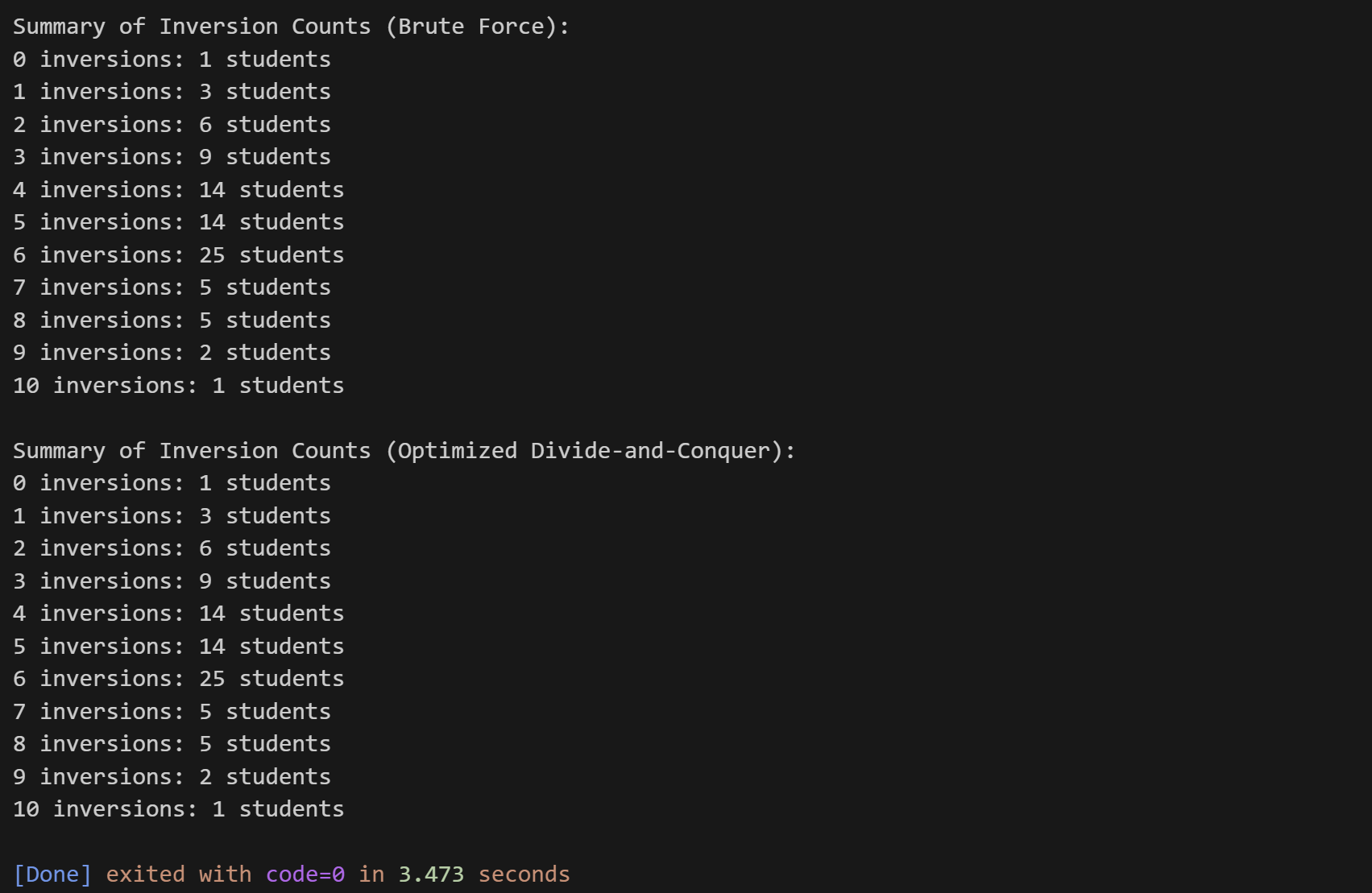
    }

    return 0;

}

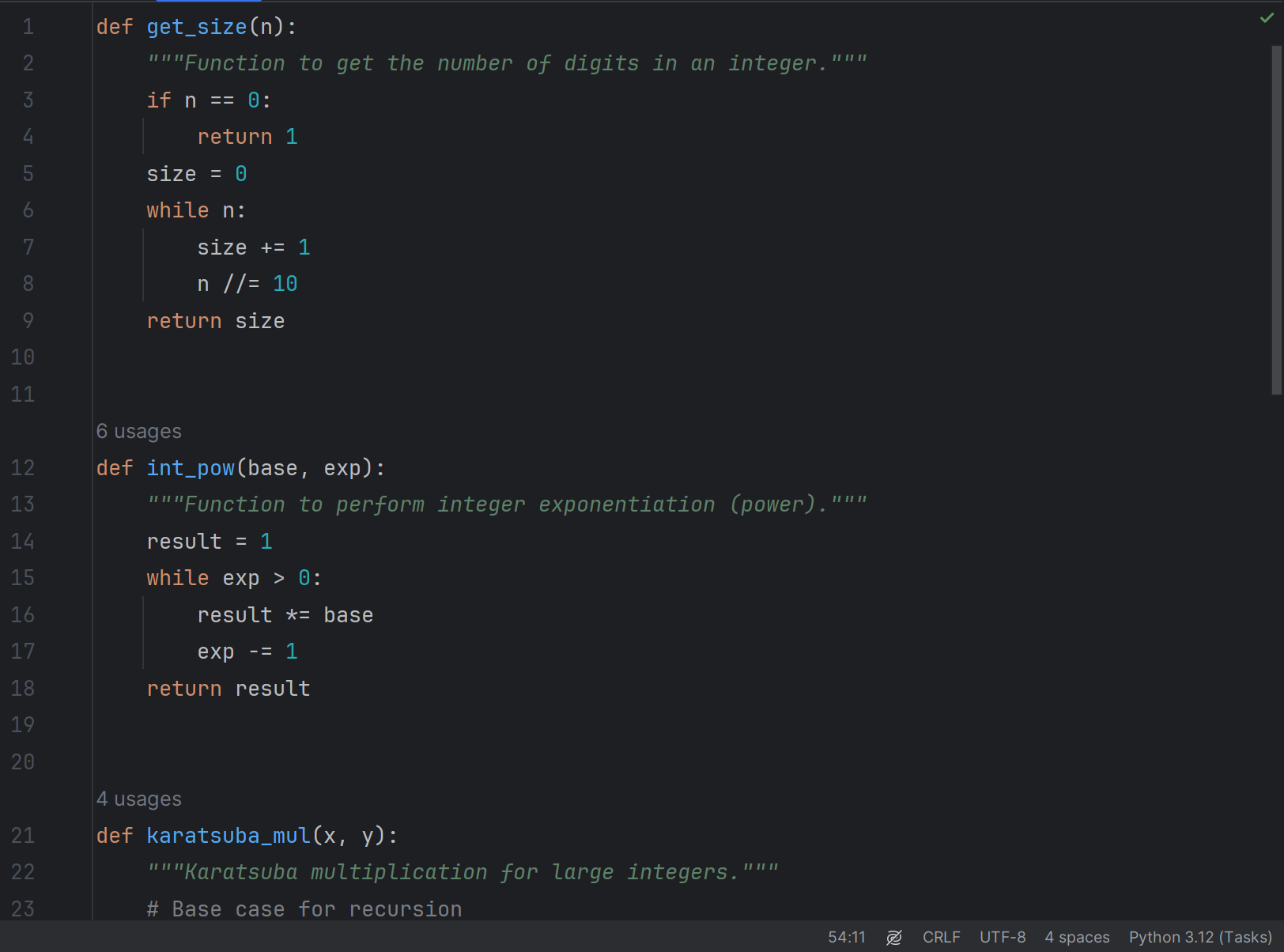
Output



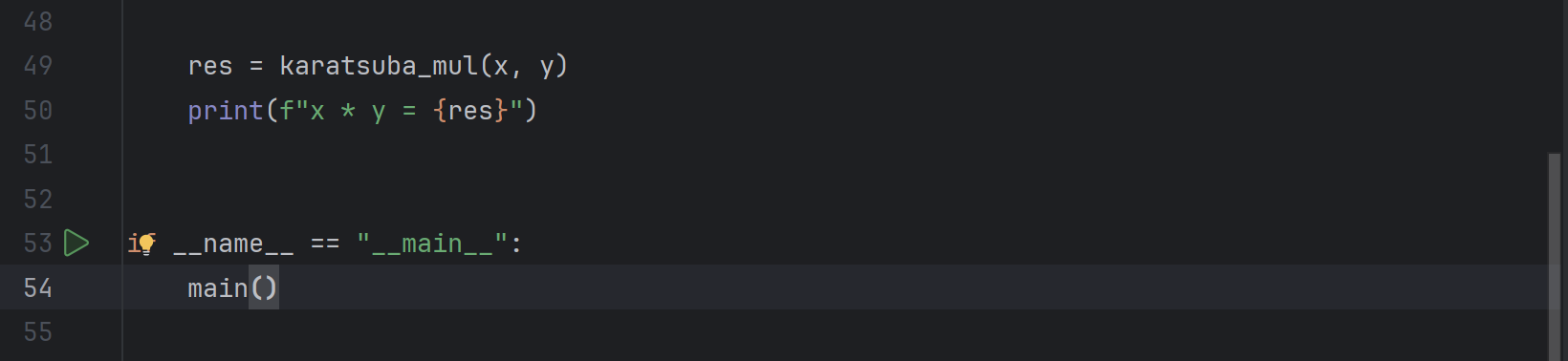


Task 2

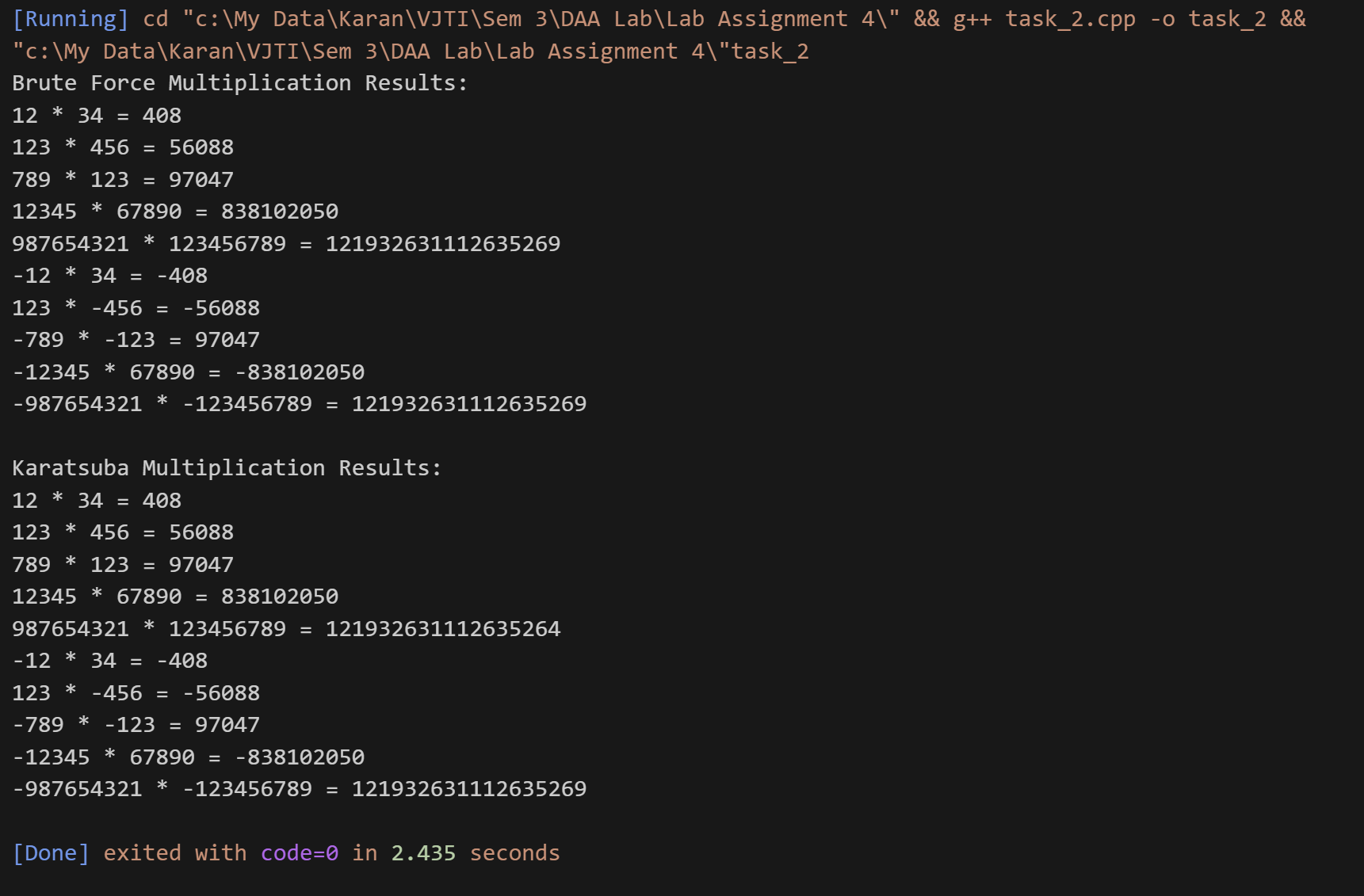
Program







Output



Conclusion

Task 1: Hence, we have found out no. of counting inversions in student’s course codes using Brute-force method and Divide and Conques Technique. We have also seen that Divide and Conquer Method takes less time as compared to Bryte force method.

TC of Brute Force: O(n2)

TC of Divide and Conquer: O(n logn)

Task 2: We have calculated the product of 2 integers using Brute-Force and Divide and Conquer method (Karatsuba Algorithm). We have also seen that for large integers Brute force may not work but those products can be found out using Karatsuba Algorithm method.

TC of Brute Force: O(n2)

TC of Karatsuba Algorithm: O(n1.585)