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++;
            }
        }
```

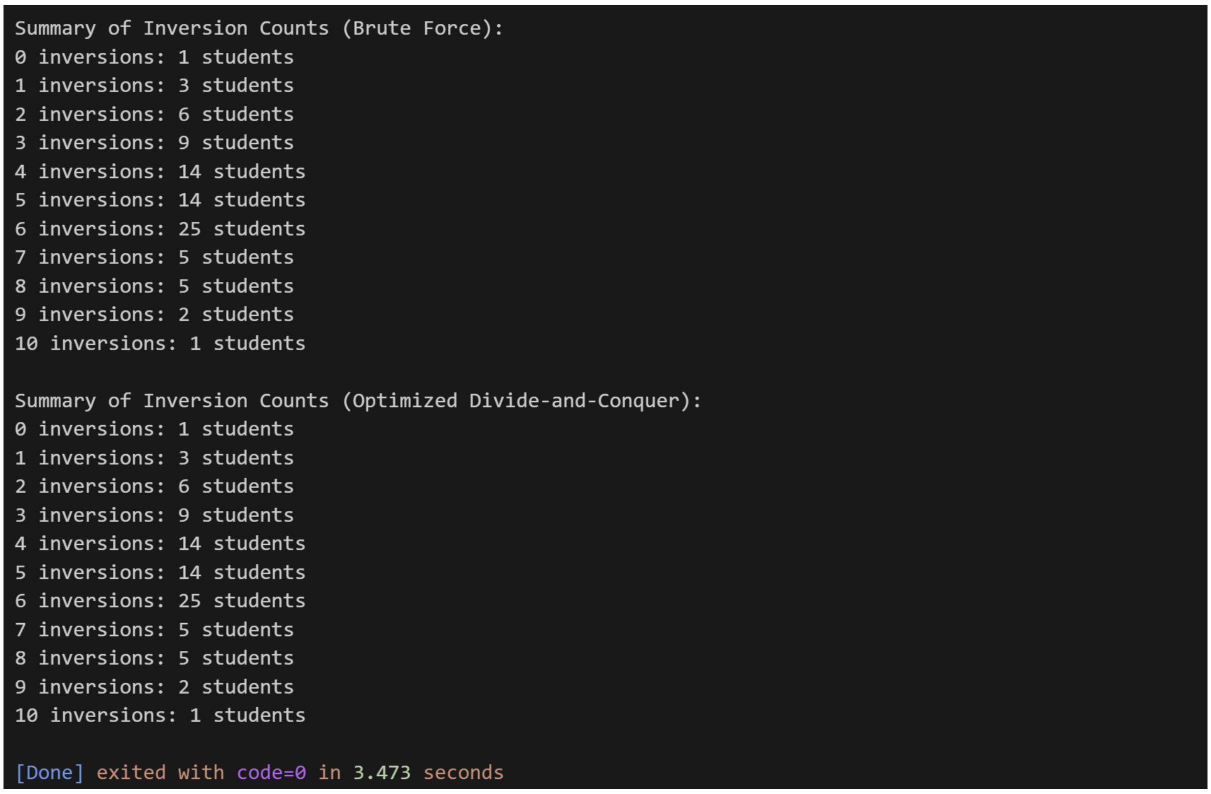
```cpp
    }

    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;
```

```cpp
}

// 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++;
    }
```

```cpp

    // 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;
}
```

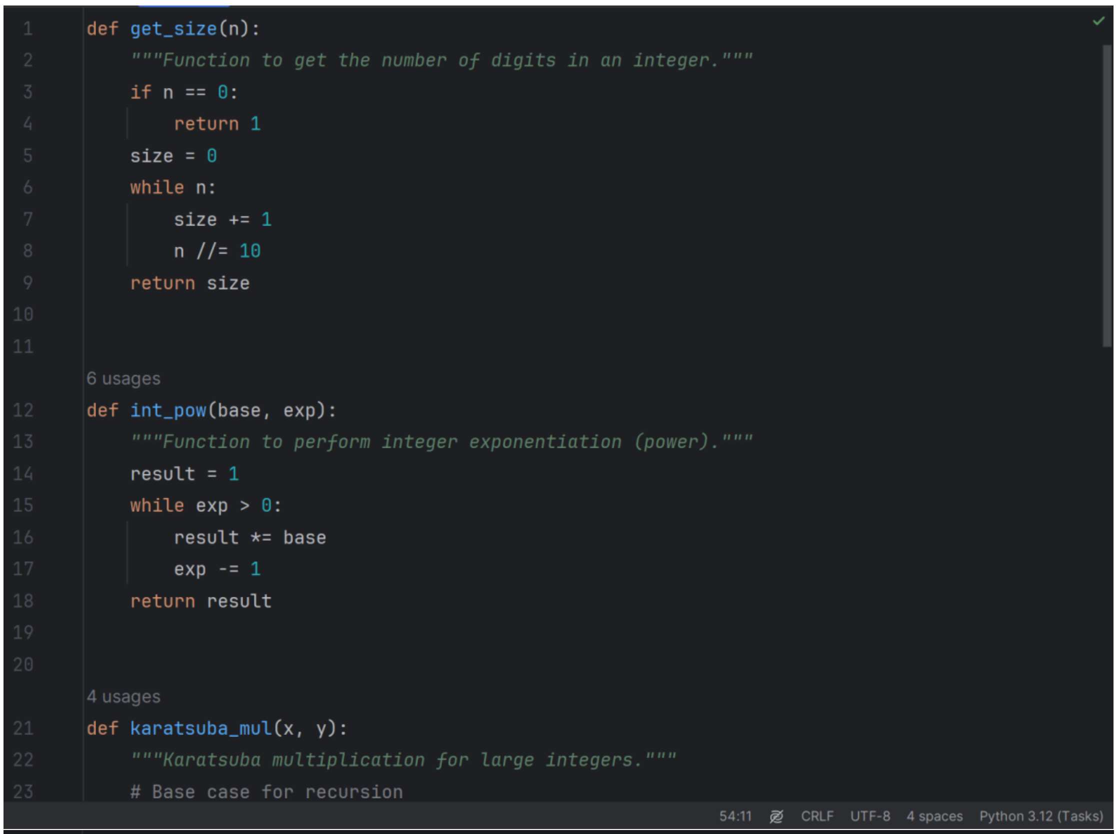
Output

```
[Running] cd "c:\My Data\Karan\VJTI\Sem 3\DAA Lab\Lab Assignment 4\" && g++ task_1.cpp -o task_1 &&
"c:\My Data\Karan\VJTI\Sem 3\DAA Lab\Lab Assignment 4\"task_1
Student 1: Brute-force inversions = 6, Optimized inversions = 6
Student 2: Brute-force inversions = 5, Optimized inversions = 5
Student 3: Non-numeric or invalid course code '-96661' found. Skipping inversion calculation.
Student 4: Non-numeric or invalid course code 'F#)}h' found. Skipping inversion calculation.
Student 5: Brute-force inversions = 7, Optimized inversions = 7
Student 6: Non-numeric or invalid course code '-21203' found. Skipping inversion calculation.
Student 7: Brute-force inversions = 6, Optimized inversions = 6
Student 8: Brute-force inversions = 5, Optimized inversions = 5
Student 9: Non-numeric or invalid course code '-57974' found. Skipping inversion calculation.
Student 10: Non-numeric or invalid course code '}([j+' found. Skipping inversion calculation.
Student 11: Brute-force inversions = 6, Optimized inversions = 6
Student 12: Brute-force inversions = 3, Optimized inversions = 3
Student 13: Non-numeric or invalid course code '-61843' found. Skipping inversion calculation.
Student 14: Brute-force inversions = 8, Optimized inversions = 8
Student 15: Non-numeric or invalid course code 'EfqsJ' found. Skipping inversion calculation.
Student 16: Non-numeric or invalid course code '-16098' found. Skipping inversion calculation.
Student 17: Brute-force inversions = 2, Optimized inversions = 2
Student 18: Non-numeric or invalid course code '-10288' found. Skipping inversion calculation.
Student 19: Non-numeric or invalid course code 'zd+}l' found. Skipping inversion calculation.
Student 20: Brute-force inversions = 5, Optimized inversions = 5
Student 21: Non-numeric or invalid course code '>?;;Z' found. Skipping inversion calculation.
Student 22: Non-numeric or invalid course code '-88648' found. Skipping inversion calculation.
Student 23: Non-numeric or invalid course code '-60580' found. Skipping inversion calculation.
Student 24: Non-numeric or invalid course code '-11608' found. Skipping inversion calculation.
Student 25: Non-numeric or invalid course code '-53035' found. Skipping inversion calculation.
Student 26: Brute-force inversions = 5, Optimized inversions = 5
Student 27: Brute-force inversions = 2, Optimized inversions = 2
```

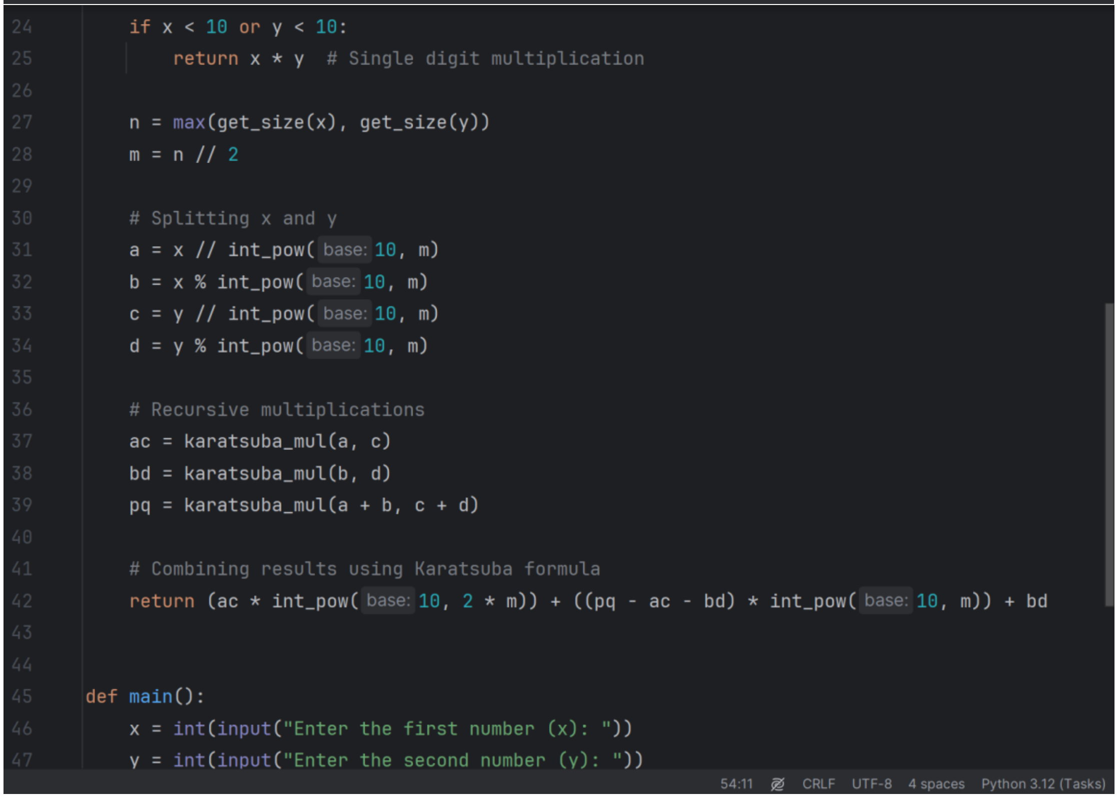

```
Summary of Inversion Counts (Brute Force):
0 inversions: 1 students
1 inversions: 3 students
2 inversions: 6 students
3 inversions: 9 students
4 inversions: 14 students
5 inversions: 14 students
6 inversions: 25 students
7 inversions: 5 students
8 inversions: 5 students
9 inversions: 2 students
10 inversions: 1 students

Summary of Inversion Counts (Optimized Divide-and-Conquer):
0 inversions: 1 students
1 inversions: 3 students
2 inversions: 6 students
3 inversions: 9 students
4 inversions: 14 students
5 inversions: 14 students
6 inversions: 25 students
7 inversions: 5 students
8 inversions: 5 students
9 inversions: 2 students
10 inversions: 1 students

[Done] exited with code=0 in 3.473 seconds
```

Task 2

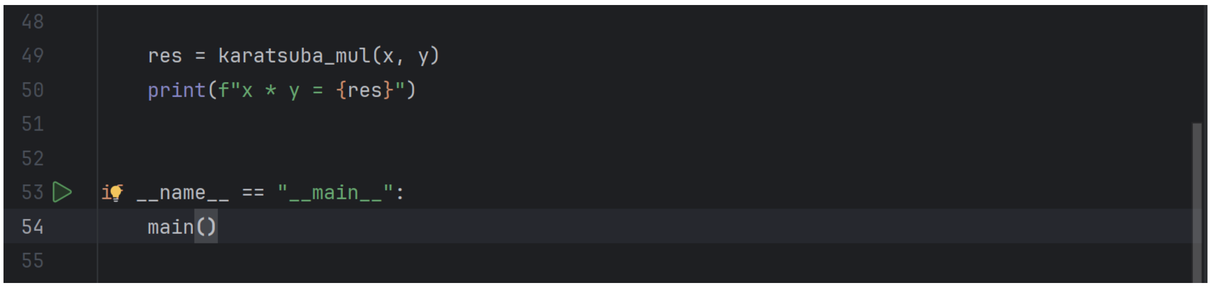
Program



```python
def get_size(n):
    """Function to get the number of digits in an integer."""
    if n == 0:
        return 1
    size = 0
    while n:
        size += 1
        n //= 10
    return size


6 usages
def int_pow(base, exp):
    """Function to perform integer exponentiation (power)."""
    result = 1
    while exp > 0:
        result *= base
        exp -= 1
    return result


4 usages
def karatsuba_mul(x, y):
    """Karatsuba multiplication for large integers."""
    # Base case for recursion
```

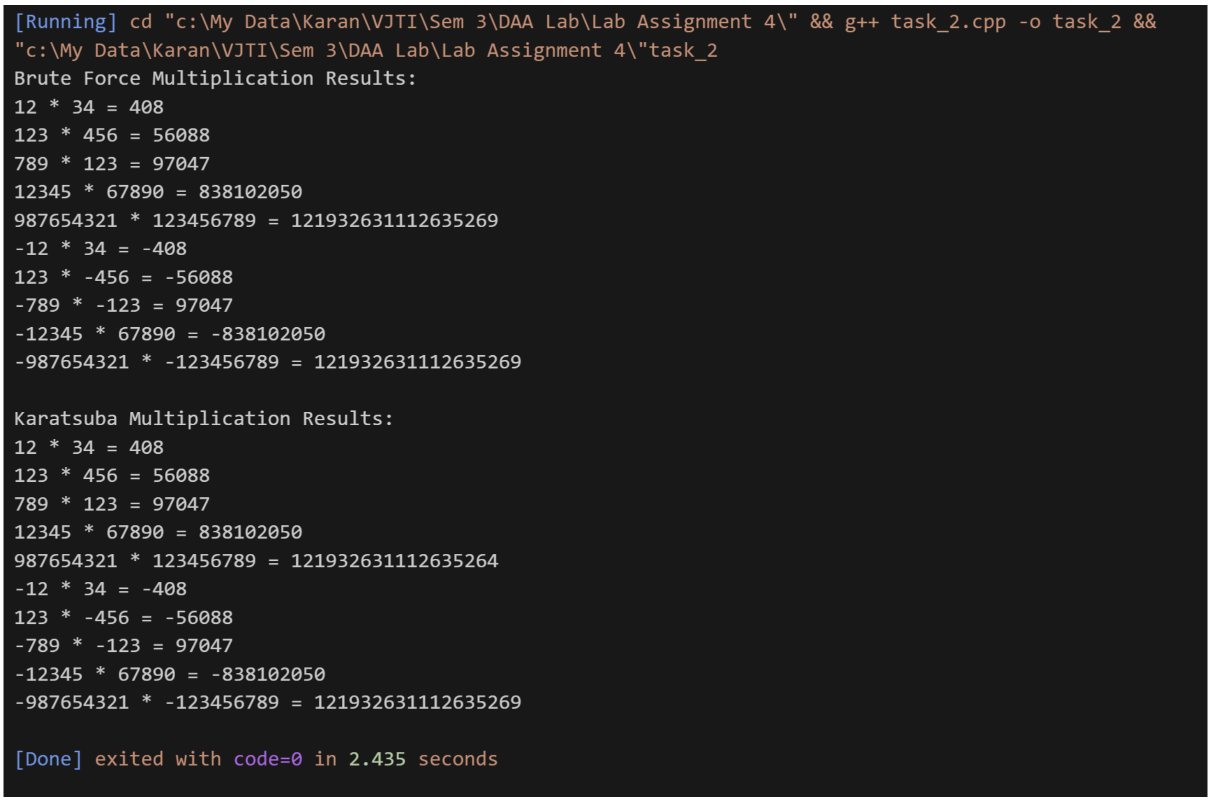


```python
    if x < 10 or y < 10:
        return x * y  # Single digit multiplication

    n = max(get_size(x), get_size(y))
    m = n // 2

    # Splitting x and y
    a = x // int_pow( base: 10, m)
    b = x % int_pow( base: 10, m)
    c = y // int_pow( base: 10, m)
    d = y % int_pow( base: 10, m)

    # Recursive multiplications
    ac = karatsuba_mul(a, c)
    bd = karatsuba_mul(b, d)
    pq = karatsuba_mul(a + b, c + d)

    # Combining results using Karatsuba formula
    return (ac * int_pow( base: 10, 2 * m)) + ((pq - ac - bd) * int_pow( base: 10, m)) + bd


def main():
    x = int(input("Enter the first number (x): "))
    y = int(input("Enter the second number (y): "))
```



```python

    res = karatsuba_mul(x, y)
    print(f"x * y = {res}")


if __name__ == "__main__":
    main()

```

Output



```
[Running] cd "c:\My Data\Karan\VJTI\Sem 3\DAA Lab\Lab Assignment 4\" && g++ task_2.cpp -o task_2 &&
"c:\My Data\Karan\VJTI\Sem 3\DAA Lab\Lab Assignment 4\"task_2
Brute Force Multiplication Results:
12 * 34 = 408
123 * 456 = 56088
789 * 123 = 97047
12345 * 67890 = 838102050
987654321 * 123456789 = 121932631112635269
-12 * 34 = -408
123 * -456 = -56088
-789 * -123 = 97047
-12345 * 67890 = -838102050
-987654321 * -123456789 = 121932631112635269

Karatsuba Multiplication Results:
12 * 34 = 408
123 * 456 = 56088
789 * 123 = 97047
12345 * 67890 = 838102050
987654321 * 123456789 = 121932631112635264
-12 * 34 = -408
123 * -456 = -56088
-789 * -123 = 97047
-12345 * 67890 = -838102050
-987654321 * -123456789 = 121932631112635269

[Done] exited with code=0 in 2.435 seconds
```

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(n^2)

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: $O(n^2)$

TC of Karatsuba Algorithm: $O(n^{1.585})$