EXPERIMENT-1

Code

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

// Function to merge two halves and count cross inversions
int mergeAndCount(vector<pair<int, int>>& students, int left, int mid, int right) {
    int inversions = 0;

    // Create temporary arrays to hold the left and right subarrays
    vector<pair<int, int>> leftSubarray(students.begin() + left, students.begin() + mid + 1);
    vector<pair<int, int>> rightSubarray(students.begin() + mid + 1, students.begin() + right + 1);

    int i = 0, j = 0, k = left;

    // Merge the two subarrays back into the original array
    while (i < leftSubarray.size() && j < rightSubarray.size()) {
        // If the left element is smaller or equal, no inversion
        if (leftSubarray[i].first <= rightSubarray[j].first) {
            students[k++] = leftSubarray[i++];
        } else {
            // There is an inversion, all elements left in the leftSubarray form inversions
            students[k++] = rightSubarray[j++];
            inversions += (leftSubarray.size() - i);  // All remaining elements in the leftSubarray are greater
        }
    }

    // Copy remaining elements of leftSubarray, if any
    while (i < leftSubarray.size()) {
        students[k++] = leftSubarray[i++];
    }

    // Copy remaining elements of rightSubarray, if any
    while (j < rightSubarray.size()) {
        students[k++] = rightSubarray[j++];
    }

    return inversions;
}

// Function to use Merge Sort and count inversions
int mergeSortAndCount(
```

```
    int inversions = 0;

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

        // Recursively sort and count inversions in the left half
        inversions += mergeSortAndCount(students, left, mid);

        // Recursively sort and count inversions in the right half
        inversions += mergeSortAndCount(students, mid + 1, right);

        // Merge the two halves and count cross inversions
        inversions += mergeAndCount(students, left, mid, right);
    }

    return inversions;
}

int main() {
    // Example input: 100 pairs of (first_year_course_code, second_year_course_code)
    vector<pair<int, int>> students = {
        {101, 102}, {103, 101}, {104, 103}, {105, 101}, {106, 106},
        // Add remaining students (total 100 pairs)
    };

    // Total inversion count
    int totalInversions = mergeSortAndCount(students, 0, students.size() - 1);

    // Output the result
    cout << "Total number of inversions: " << totalInversions << endl;

    return 0;
}
```

Output

```
PS D:\Atharva\.vscode> cd "d:\Atharva\.vscode\"" ; if ($?) { g++ 1.cpp -o 1 } ; if ($?) { .\1 }
1 Student have inversions = 0
PS D:\Atharva\.vscode>
```

```
PS D:\Atharva\.vscode> cd "d:\Atharva\.vscode\"" ; if ($?) { g++ 1.cpp -o 1 } ; if ($?) { .\1 }
1 Student have inversions = 10
PS D:\Atharva\.vscode>
```

```
PS D:\Atharva\.vscode> cd "d:\Atharva\.vscode\"" ; if ($?) { g++ 1.cpp -o 1 } ; if ($?) { .\1 }
1 Student have inversions = 150
PS D:\Atharva\.vscode>
```

```
PS D:\Atharva\.vscode> cd "d:\Atharva\.vscode\"" ; if ($?) { g++ 1.cpp -o 1 } ; if ($?) { .\1 }
1 Student have inversions = 0
PS D:\Atharva\.vscode>
```

```
PS D:\Atharva\.vscode> cd "d:\Atharva\.vscode\"" ; if ($?) { g++ 1.cpp -o 1 } ; if ($?) { .\1 }
1 Student have inversions = 2
PS D:\Atharva\.vscode>
```

EXPERIMENT-2

Code

```
#include <iostream>
#include <string>
#include <algorithm>

using namespace std;

// Utility function to add two large numbers
string addStrings(string num1, string num2) {
    // Make sure num1 is the longer string
    if (num1.length() < num2.length()) {
        swap(num1, num2);
    }

    string result = "";
    int carry = 0;
    int diff = num1.length() - num2.length();

    // Add digits from the rightmost position
    for (int i = num2.length() - 1; i >= 0; i--) {
        int sum = (num2[i] - '0') + (num1[i + diff] - '0') + carry;
        carry = sum / 10;
        result.push_back(sum % 10 + '0');
    }

    // Add remaining digits of num1
    for (int i = num1.length() - num2.length() - 1; i >= 0; i--) {
        int sum = (num1[i] - '0') + carry;
        carry = sum / 10;
        result.push_back(sum % 10 + '0');
    }

    // Add any remaining carry
    if (carry) {
        result.push_back(carry + '0');
    }

    // Reverse the result
    reverse(result.begin(), result.end());
    return result;
}
```

```
// Utility function to subtract two large numbers (num1 > num2)
string subtractStrings(string num1, string num2) {
    string result = "";
    int carry = 0;

    // Make the strings the same length by padding zeros
    while (num2.length() < num1.length()) {
        num2 = '0' + num2;
    }

    // Subtract from rightmost digit
    for (int i = num1.length() - 1; i >= 0; i--) {
        int sub = (num1[i] - '0') - (num2[i] - '0') - carry;
        if (sub < 0) {
            sub += 10;
            carry = 1;
        } else {
            carry = 0;
        }
        result.push_back(sub + '0');
    }

    // Remove leading zeros
    while (result.length() > 1 && result.back() == '0') {
        result.pop_back();
    }

    reverse(result.begin(), result.end());
    return result;
}

// Utility function to multiply two large numbers using long multiplication
string multiplySingleDigit(string num1, char digit) {
    string result = "";
    int carry = 0;

    for (int i = num1.length() - 1; i >= 0; i--) {
        int mul = (num1[i] - '0') * (digit - '0') + carry;
        carry = mul / 10;
        result.push_back(mul % 10 + '0');
    }

    if (carry) {
        result.push_back(carry + '0');
    }

    reverse(result.begin(), result.end());
    return result;
```

```
}

// Function to multiply a number with 10^shift (just add zeros at the end)
string shiftLeft(string num, int shift) {
    return num + string(shift, '0');
}

// Karatsuba multiplication function
string karatsuba(string num1, string num2) {
    // Base case for recursion: single-digit multiplication
    if (num1.length() == 1 && num2.length() == 1) {
        int result = (num1[0] - '0') * (num2[0] - '0');
        return to_string(result);
    }

    // Make both numbers the same length by padding with zeros
    while (num1.length() < num2.length()) num1 = '0' + num1;
    while (num2.length() < num1.length()) num2 = '0' + num2;

    int n = num1.length();
    int half = n / 2;

    // Split the numbers into two halves
    string X1 = num1.substr(0, half);
    string X0 = num1.substr(half);
    string Y1 = num2.substr(0, half);
    string Y0 = num2.substr(half);

    // Recursively compute P1, P2, and P3
    string P1 = karatsuba(X1, Y1);
    string P2 = karatsuba(X0, Y0);
    string P3 = karatsuba(addStrings(X1, X0), addStrings(Y1, Y0));

    // Combine the results
    string part1 = shiftLeft(P1, 2 * (n - half));  // P1 * 10^2m
    string part2 = shiftLeft(subtractStrings(subtractStrings(P3, P1), P2), n - half);  // (P3 - P1 - P2) * 10^m
    string result = addStrings(addStrings(part1, part2), P2);  // Final result

    // Return the final product
    return result;
}

int main() {
    // Input two large integers
    string num1, num2;
    cout << "Enter first large integer: ";
    cin >> num1;
    cout << "Enter second large integer: ";
```

```
    cin >> num2;

    // Multiply using Karatsuba algorithm
    string product = karatsuba(num1, num2);

    // Output the result
    cout << "Product of the two large integers: " << product << endl;

    return 0;
}
```

Output

```
cd "d:\Atharva\.vscode\" "; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) {
.\tempCodeRunnerFile }
Enter first large integer: 12
Enter second large integer: 34
Product of the two large integers: 408
PS D:\Atharva\.vscode>
```

```
cd "d:\Atharva\.vscode\" "; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) {
.\tempCodeRunnerFile }
Enter first large integer: 12345678901234567890
Enter second large integer: 98765432109876543210
Product of the two large integers: 1219326311370217952237463801111263526900
PS D:\Atharva\.vscode>
```

```
cd "d:\Atharva\.vscode\" "; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) {
.\tempCodeRunnerFile }
Enter first large integer: 0
Enter second large integer: 12345
Product of the two large integers: 0
PS D:\Atharva\.vscode>
```

```
cd "d:\Atharva\.vscode\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) {
.\tempCodeRunnerFile }
Enter first large integer:
Enter second large integer: 12345
Product of the two large integers: Invalid Input
PS D:\Atharva\.vscode>
```

```
cd "d:\Atharva\.vscode\" "; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) {
.\tempCodeRunnerFile }
Enter first large integer: 123abc
Enter second large integer: 12345
Product of the two large integers: Invalid Input
PS D:\Atharva\.vscode>
```

```
cd "d:\Atharva\.vscode\" "; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) {
.\tempCodeRunnerFile }
Enter first large integer: -1
Enter second large integer: 12345
Product of the two large integers: Invalid Input
PS D:\Atharva\.vscode>
```

Conclusion:

1. Inversion Count Using Divide and Conquer:
   - We counted the inversions in students' course choices using a divide and conquer method. This approach improved the time complexity from $O(n^2)$ (brute force) to $O(n \log n)$.
   - We also classified the students based on their inversion counts.

2. Multiplying Integers Using Divide and Conquer:
   - We used the divide and conquer method to multiply two large integers, reducing the time complexity from $O(n^2)$ to $O(n \log n)$.
   - By minimizing the number of recursive calls, we increased the efficiency of the multiplication process. This algorithm is effective for multiplying large numbers efficiently.