Name: Ashutosh kumar

Reg. Num: 231070006

Batch: A

DAA Lab Experiment 4

Aim:

1. To find inversion count of course choice of students.

2. To multiply two integers using brute force and divide-and-conquer methods

Program:

1. Inversion count:

Brute-force approach:

```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <string>

using namespace std;

// Function to read course choices from CSV file
vector<vector<int>> readCourseChoicesFromCSV(const string& filename) {
    ifstream file(filename);
    vector<vector<int>> courseChoices;
    string line, value;

    // Skip the header line
```

```
    getline(file, line);

    // Read each line (student's course choices)
    while (getline(file, line)) {
        vector<int> studentChoices;
        stringstream ss(line);

        // Ignore the first column (StudentID)
        getline(ss, value, ',');

        // Read each course choice and store in the vector
        while (getline(ss, value, ',')) {
            studentChoices.push_back(stoi(value)); // Convert string to
integer
        }

        // Add this student's course choices to the overall list
        courseChoices.push_back(studentChoices);
    }

    file.close();
    return courseChoices;
}

int main() {
    // Read course choices from the CSV file
    string filename = "students_course_choices.csv";
    vector<vector<int>> courseChoices = readCourseChoicesFromCSV(filename);

    // Print the course choices for each student
    for (int i = 0; i < courseChoices.size(); i++) {
        cout << "Student " << i + 1 << " course choices: ";
        for (int j = 0; j < courseChoices[i].size(); j++) {
            cout << courseChoices[i][j] << " ";
        }
        cout << endl;
    }

    // You can now pass a specific student's course choices to the inversion
count function
    // Example: vector<int> student1Choices = courseChoices[0]; // Course
choices of Student 1

    return 0;
}
```

Efficient Approach (Merge Sort):

```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <string>

using namespace std;

// Function to merge two halves and count inversions
int mergeAndCount(vector<int>& arr, int left, int mid, int right) {
    int count = 0;
    vector<int> temp(right - left + 1);
    int i = left, j = mid + 1, k = 0;

    // Merge the two halves while counting inversions
    while (i <= mid && j <= right) {
        if (arr[i] <= arr[j]) {
            temp[k++] = arr[i++];
        } else {
            temp[k++] = arr[j++];
            // Since arr[i] > arr[j], there are mid - i inversions
            count += (mid - i + 1);
        }
    }

    // Copy remaining elements from both halves
    while (i <= mid) {
        temp[k++] = arr[i++];
    }

    while (j <= right) {
        temp[k++] = arr[j++];
    }

    // Copy the sorted subarray back to the original array
    for (i = left, k = 0; i <= right; i++, k++) {
        arr[i] = temp[k];
    }

    return count;
}

// Function to perform merge sort and count inversions
int mergeSortAndCount(vector<int>& arr, int left, int right) {
    if (left >= right) {
        return 0;
```

```
    }

    int mid = left + (right - left) / 2;
    int count = 0;

    // Recursively sort and count inversions
    count += mergeSortAndCount(arr, left, mid);
    count += mergeSortAndCount(arr, mid + 1, right);
    count += mergeAndCount(arr, left, mid, right);

    return count;
}

// Function to read course choices from CSV file
vector<vector<int>> readCourseChoicesFromCSV(const string& filename) {
    ifstream file(filename);
    vector<vector<int>> courseChoices;
    string line, value;

    // Skip the header line
    getline(file, line);

    // Read each line (student's course choices)
    while (getline(file, line)) {
        vector<int> studentChoices;
        stringstream ss(line);

        // Ignore the first column (StudentID)
        getline(ss, value, ',');

        // Read each course choice and store in the vector
        while (getline(ss, value, ',')) {
            studentChoices.push_back(stoi(value)); // Convert string to
integer
        }

        // Add this student's course choices to the overall list
        courseChoices.push_back(studentChoices);
    }

    file.close();
    return courseChoices;
}

int main() {
    // Read course choices from the CSV file
    string filename = "students_course_choices.csv";
    vector<vector<int>> courseChoices = readCourseChoicesFromCSV(filename);
```

```
    // Process each student's course choices and calculate inversion count
    for (int i = 0; i < courseChoices.size(); i++) {
        int inversionCount = mergeSortAndCount(courseChoices[i], 0,
courseChoices[i].size() - 1);
        cout << "Student " << i + 1 << " inversion count: " << inversionCount
<< endl;
    }

    return 0;
}
```

Output:

```
Student 1 inversion count: 7
Student 2 inversion count: 7
Student 3 inversion count: 6
Student 4 inversion count: 3
Student 5 inversion count: 10
Student 6 inversion count: 9
Student 7 inversion count: 6
Student 8 inversion count: 6
Student 9 inversion count: 8
Student 10 inversion count: 5
```

2. Integer Multiplication

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

using namespace std;

// Function to perform brute force multiplication
int bruteForceMultiply(int a, int b) {
    return a * b;
```

```cpp
}

// Function to perform divide-and-conquer multiplication (Karatsuba Algorithm)
int multiplyDivideAndConquer(int x, int y) {
    // Base case
    if (x < 10 || y < 10) return x * y;

    int n = max(to_string(x).length(), to_string(y).length());
    int half = n / 2;

    // Split the digit sequences
    int a = x / pow(10, half);
    int b = x % static_cast<int>(pow(10, half));
    int c = y / pow(10, half);
    int d = y % static_cast<int>(pow(10, half));

    // Recursively calculate
    int ac = multiplyDivideAndConquer(a, c);
    int bd = multiplyDivideAndConquer(b, d);
    int ab_cd = multiplyDivideAndConquer(a + b, c + d);

    // Return the result
    return ac * pow(10, 2 * half) + (ab_cd - ac - bd) * pow(10, half) + bd;
}

// Function to read pairs of numbers from CSV
vector<pair<int, int>> readNumbersFromCSV(const string& filename) {
    ifstream file(filename);
    vector<pair<int, int>> numberPairs;
    string line, value;

    // Skip the header
    getline(file, line);

    // Read each line (pairs of numbers)
    while (getline(file, line)) {
        stringstream ss(line);
        string num1, num2;

        getline(ss, num1, ',');
        getline(ss, num2, ',');

        numberPairs.push_back(make_pair(stoi(num1), stoi(num2)));
    }

    file.close();
    return numberPairs;
}
```

```cpp
int main() {
    // File containing pairs of numbers
    string filename = "numbers_for_multiplication.csv";

    // Read the number pairs from the CSV file
    vector<pair<int, int>> numberPairs = readNumbersFromCSV(filename);

    // Perform multiplication using brute force and divide-and-conquer methods
    for (int i = 0; i < numberPairs.size(); i++) {
        int a = numberPairs[i].first;
        int b = numberPairs[i].second;

        // Brute-force multiplication
        int bruteResult = bruteForceMultiply(a, b);

        // Divide-and-conquer multiplication
        int divAndConqResult = multiplyDivideAndConquer(a, b);

        cout << "Pair " << i + 1 << ": " << a << " * " << b << endl;
        cout << "Brute-force result: " << bruteResult << endl;
        cout << "Divide-and-conquer result: " << divAndConqResult << endl;
        cout << "---------------------------------" << endl;
    }

    return 0;
}
```

Output:

```
ashutosh kumar@Ashutosh-PC MINGW64 /e/programs/code forces/output
Enter first large integer: 1234
Enter second large integer: 4321
Product of the two large integers: 5332114

ashutosh kumar@Ashutosh-PC MINGW64 /e/programs/code forces/output
Enter first large integer: 8765
Enter second large integer: 0
Product of the two large integers: 0000000

ashutosh kumar@Ashutosh-PC MINGW64 /e/programs/code forces/output
Enter first large integer: -9876
Enter second large integer: 3456
Product of the two large integers: 1929451456

ashutosh kumar@Ashutosh-PC MINGW64 /e/programs/code forces/output
Enter first large integer: -1234
```

```
Enter second large integer: -1234
Product of the two large integers: 1029682756

ashutosh kumar@Ashutosh-PC MINGW64 /e/programs/code forces/output
Enter first large integer: 1234
Enter second large integer: 1234
Product of the two large integers: 1522756
```

Conclusion:

1. We have found the inversion count of course choice of students using divide and conquer method, which reduces the time complexity to O(nlogn) from O(n^2) of the brute force method. We classified the students according to their inversion count.

2. We have multiplied two integers using the divide and conquer method, which reduces the time complexity to O(nlogn) from O(n^2) of the brute force method. We reduced the number of recursive calls to the function, which increased the efficiency. This algorithm can be used to multiply numbers of large size efficiently.

B

```
func CountBrut (arr):
// Input : Array of number
// Output : No. of inverted value & index pairs
int Count = 0
for (i: 0 → len (arr) − 1)
  { for (j : i+1 → len (arr) - 1)
  {  if arr [i] > arr [j] :
        int Count ++
  }
}
return invCount
}
```

Time Complexity
Since there are 2 Nested loops so time Complexity is $O(N^2)$ which is not optimal.

Optimal Solution :

```
func mergeAndCount (arr):
{ // input : Array of Numbers
  // output : Number of inversions of Value & index and
              Sorted array.
     if ( len (arr) < 2 )
        return arr, 0
  mid = length (arr)/2
l, leftInv = mergeAndCount (arr [0: mid])
r, RightInv = mergeAndCount (arr [0: lenfarr)])
  merged , Split Inv = merge (l, r)
  total Inv = left Inv + Right Inv + Split Inv
  return merged , total Inv
}
```

classmate
Date
Page

function to merge two array

```
funct merge [left, right] {
  // Input : 2 arrays
  // Output : Merged array and Count of Inversions Merged = []
  i = 0
  j = 0
  Split Inv = 0
  while i < len (left) && j < len (right)
  {
    if left [i] <= right [j]
    { merged . append ( left [i] )
        i++ }
  else { merged . append [ right [i] )
         j++ }

  append remaining elements of left or right to
  merged

return .merged , Split Inv
}
```

fn to Count no. of inversion :

```
func CountStudentInv (arr) :
   Zero Inv = 0
   One Inv = 0
   two Inv = 0
   three Inv = 0
```

```
for each Source Code in arr
    inversion = Merge And Count (arr)
  If inversions == 0 :
     ZeroInvt = 1
  Else If inversions == 1 :
     One Invt = 1
  Else If inversions == 2
        twoInv += 1
  Else if inversion == 3
     threeInv += 1.
```

Time Complexity using piggyback on merge sort:

Divide step: array is recursively divided in half until it reaches subarray of size 1.

$\therefore D(n) = O(\log n)$

Conquer step: During Merge process, two sorted halves are combined, there requires time Complexity of $O(n)$

$\therefore$ total time Complexity

$$T(n) = 2T(n/2) + O(n)$$

Using Master theorem

$a = 2, \quad b = 2, \quad d = 1$

$a = b^d$ as $2 = 2^1$

$\therefore T(n) = O(n \log n)$

Experiment task-2

```
func manual Multiply (num1, num)
// Input : 2 Integers with with digits in 2 arrays
// Output : product

for (i: len(Num) → 0)
 for (j: len(Num 2) -1 → 0 )
  mul = num1 [i] + num2 [j]
  Position 1 = i+j
  Position 2 = i+j+1
Sum = mul + result [position 2]
 result [Position 2] = Sum % 10
 resut [Position1] t = Sum/10
```

Time Complexity
$O(n_1 * n_2)$ where n_1 and n_2 are the length of the input arrays.

Optimal Solution

```
func Karatsuba (x,y)
{ // Input : 2 large integer as array of digits
  // Output : There product.

 if (x <10 or y <10)
    return x*y
 m = max (len (x), len(y))
 half = m/2
 high x, low x = div mod (x, 10^half)
 high y, low y = div mod (y, 10^half)
```

Z0 = Karatsuba (low x, low y)
Z1 = Karatsuba (low x + high x, low y + high y)
Z3 = Karatsuba (high x, high y)

return (z2 * 10^(2*half) + (z1 - z2 - z0) * 10^half + z0)

Test Cases :

1. Arr 1 = { 5, 4, 4, 3}
   Arr 2 = { 5, 4, 4, 3}
   output 2 9 6 2 6 2 4 9

2. Arr 2 = {0}
   Arr = { 9, 9, 9, 9, 9, 9}
   output = 0

3. Arr 1 = { 9, 9, 9, 9, 9}
   Arr 2 = { 1, 2, 3}
   Output = 1 2 2 9 9 8 7 7

4. Arr 1 = { 9, 9, 8, 7, 6, 5, 4}
   Arr 2 = {0}
   output = 0

5. Arr 1 = { -8, 6, 5, 4}
   Arr2 = { 6, 5, 4}
   Output = -5, 6, 5 9 7 1 6

6. Arr 1 = {-8, 6, 5, 4}
   Arr 2 = { -6, 5, 4}
   output = 5 6 5 9 7 1 6