Name: Divyansh Dubey

Roll no: 231070001

Course: B. Tech Computer Engineering

Batch: A

Experiment- 4

# Aim:

**Experiment 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.

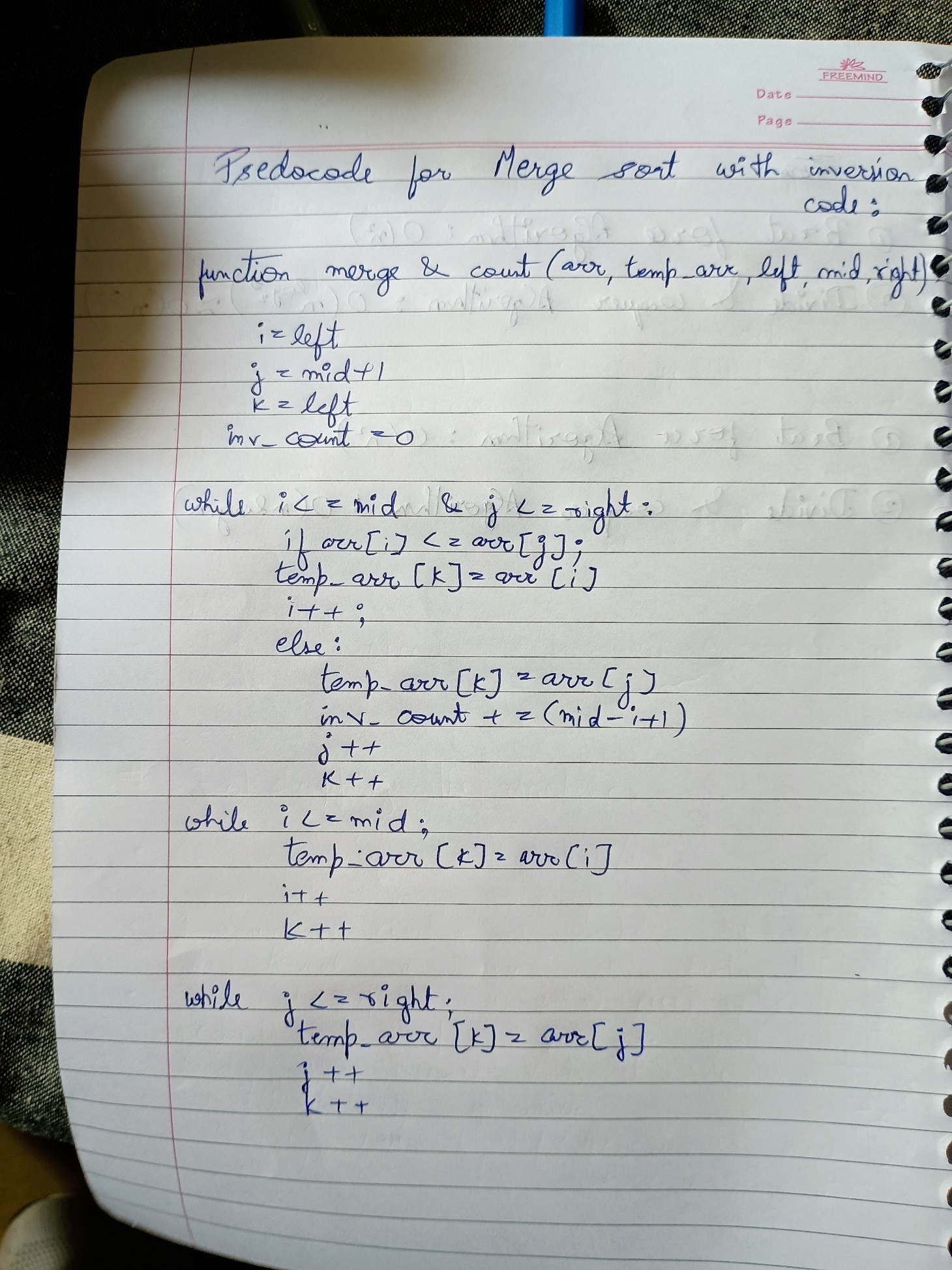
**Experiment task-2:**

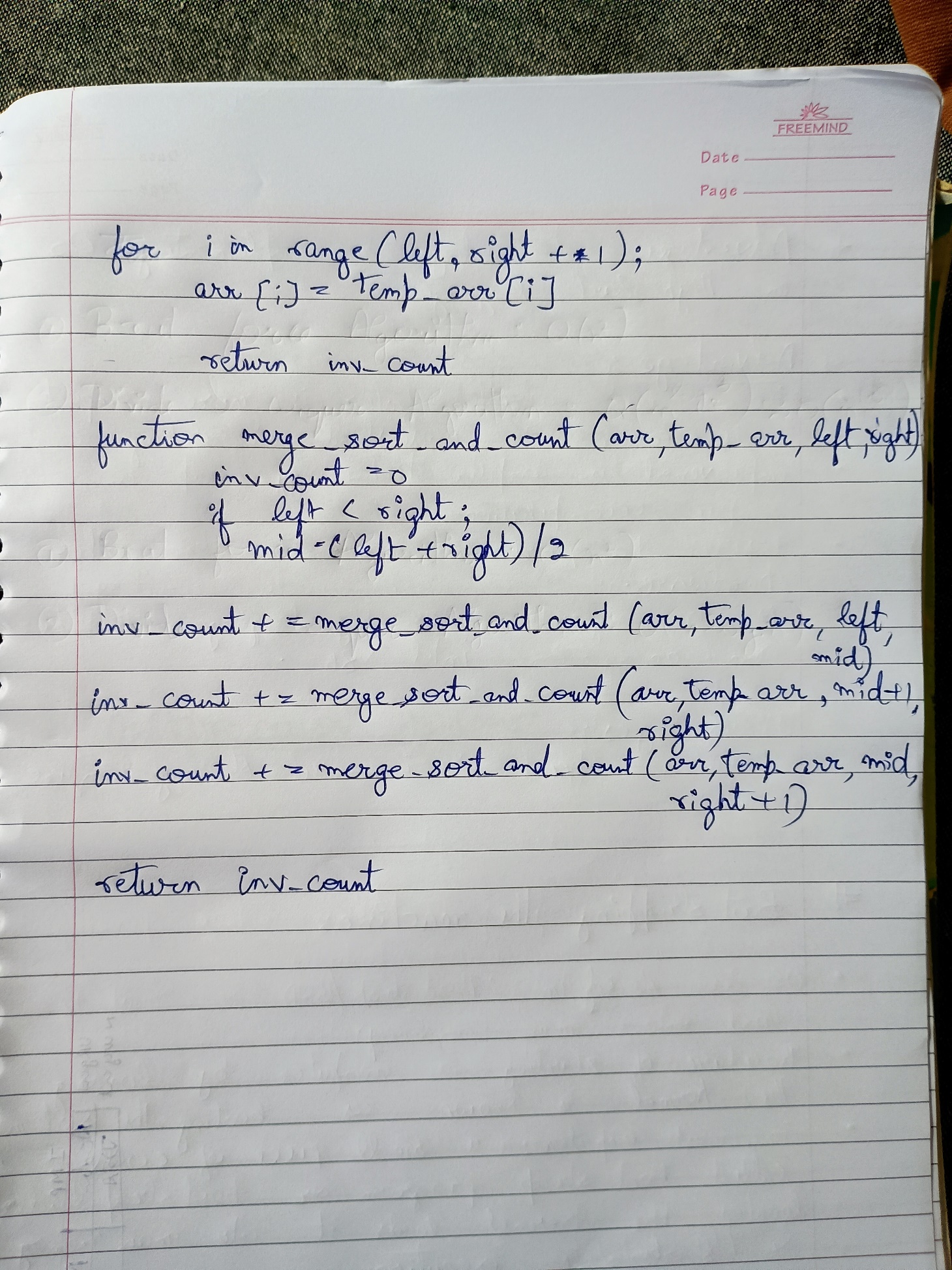
Consider large integers of size 10, 50, 100, 500 and 1000 digits.

Write integer multiplication program

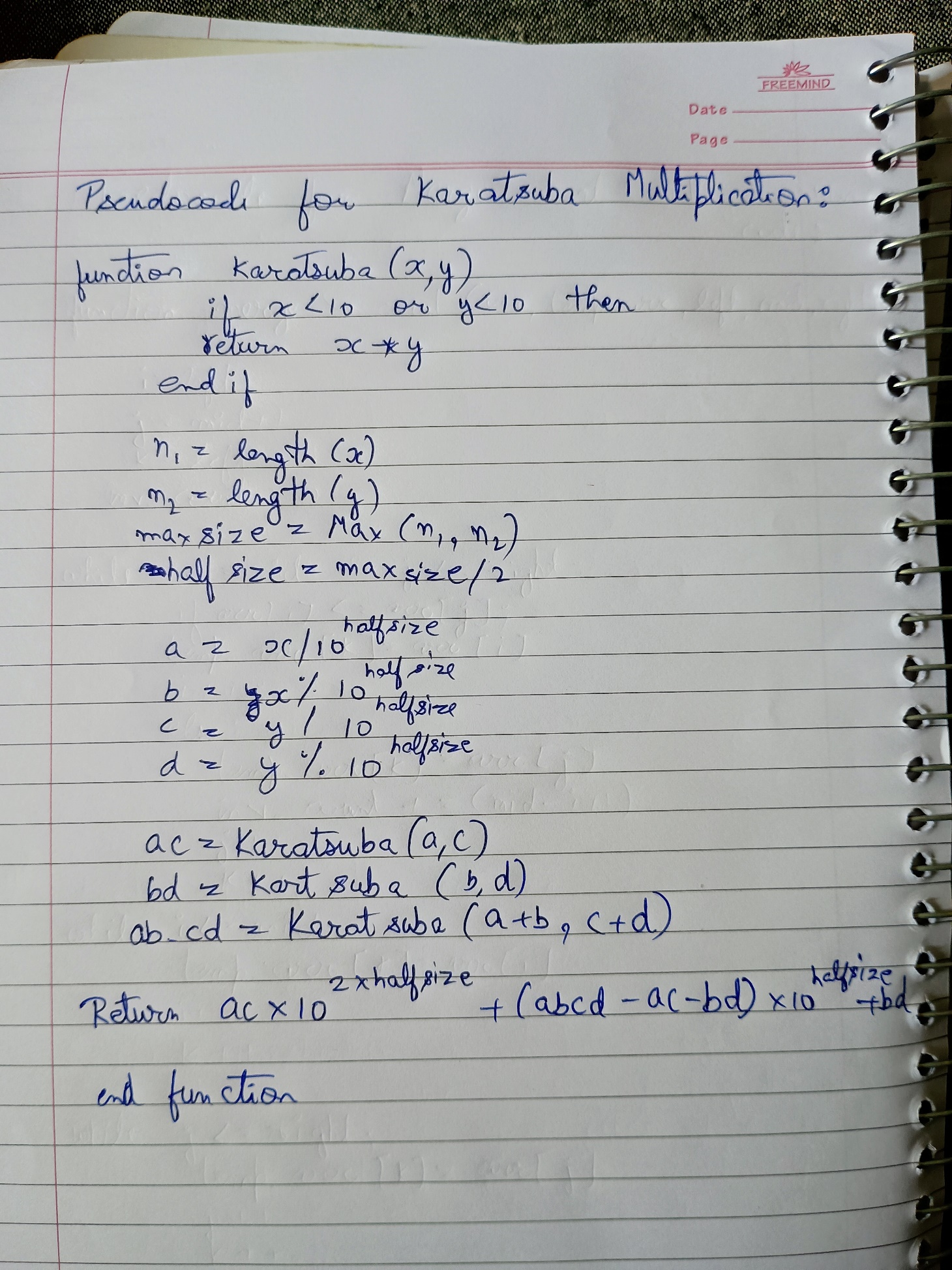
Write integer multiplication program using divide and conquer technique.

**Algorithm for Experiment task-1:**

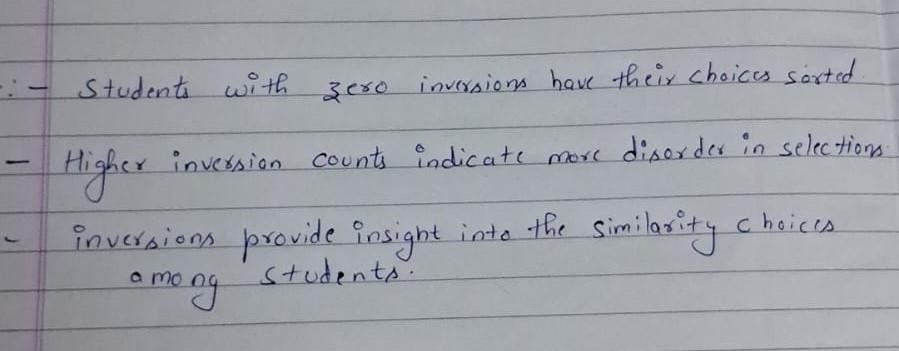
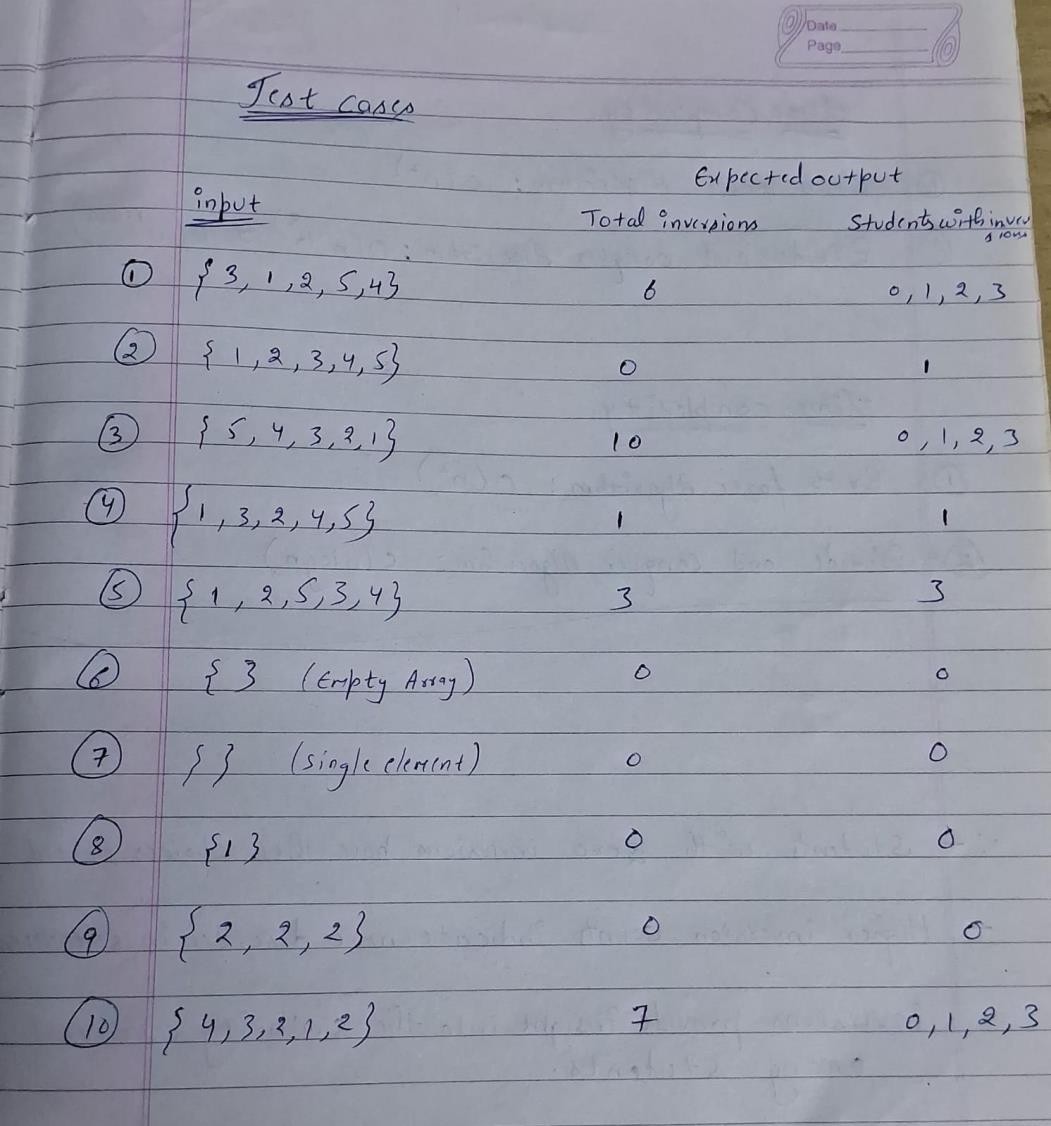




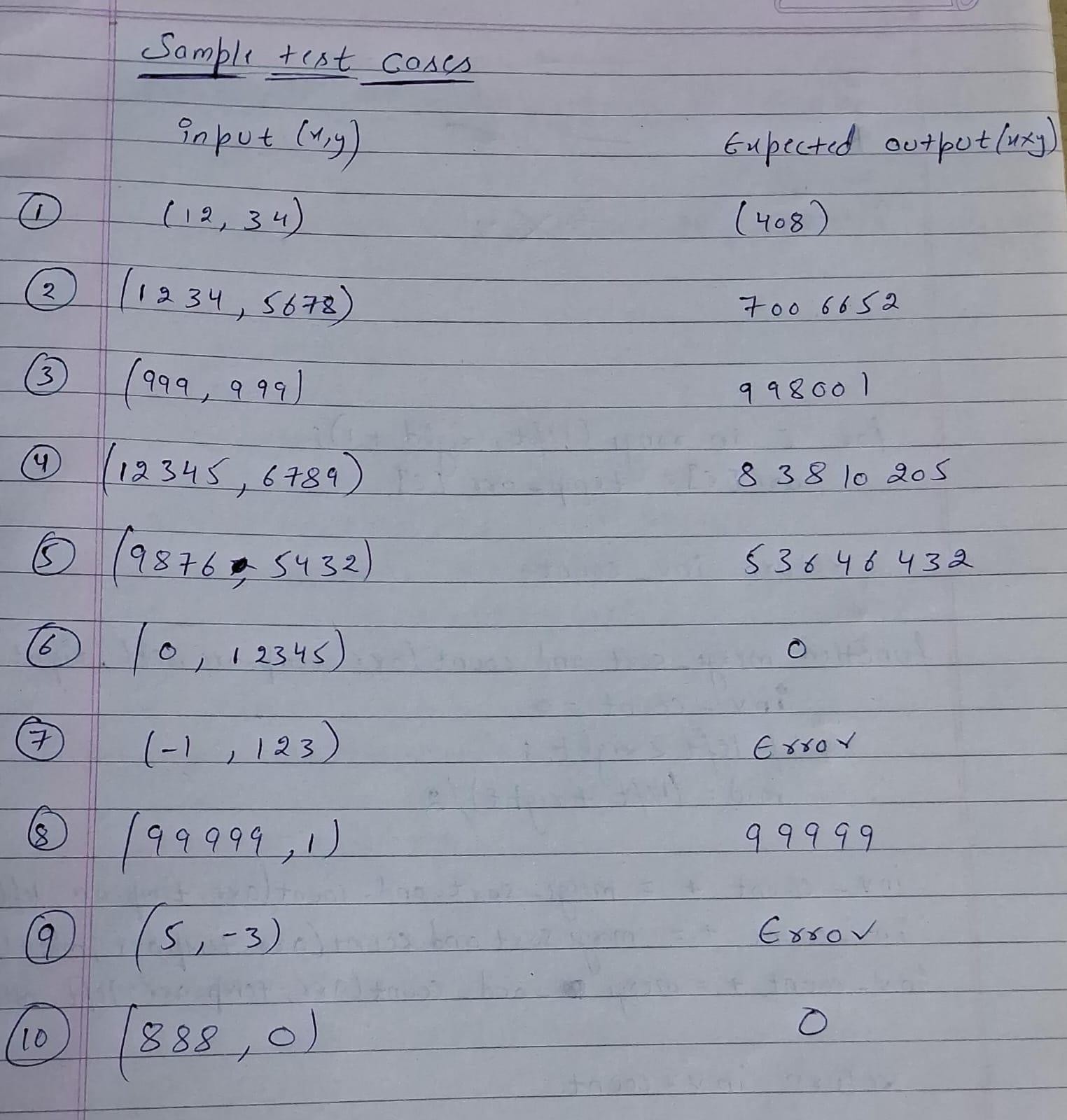
**Algorithm for Experiment task-2:**

****

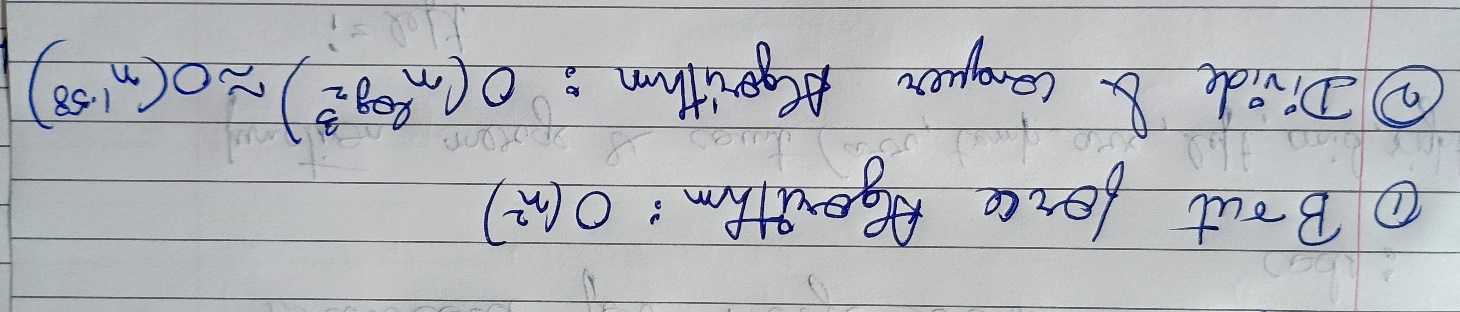
# Test cases for Experiment 1:



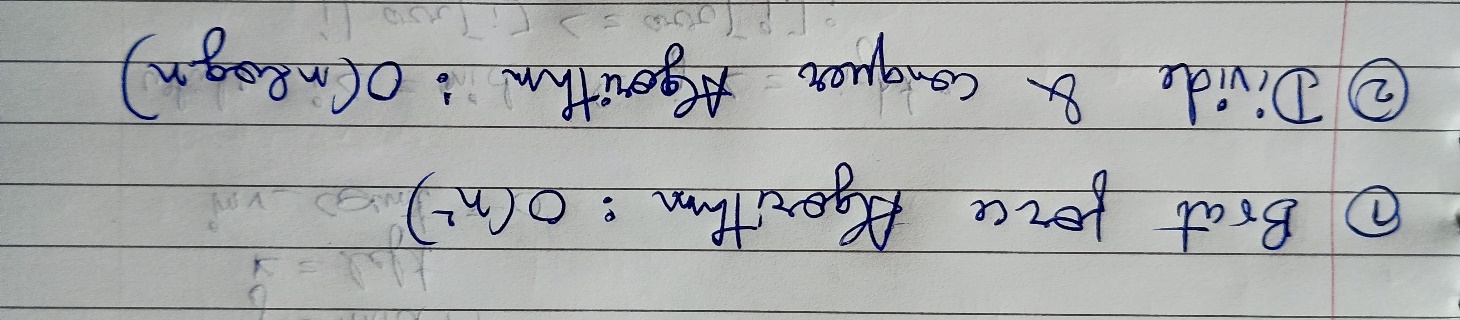
**Test cases for Experiment 2:**



# Time Complexity for Experiment 1:

****

**Time Complexity for Experiment 2:**

****

# Code for Experiment 1:

#include <iostream>

#include <vector>

#include <unordered\_map>

using namespace std;

// Function to merge and 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 invCount = 0;

 while (i <= mid && j <= right) {

 if (arr[i] <= arr[j]) {

 temp[k++] = arr[i++];

 } else {

 temp[k++] = arr[j++];

 invCount += (mid - i + 1); // Count inversions

 }

 }

 while (i <= mid) {

 temp[k++] = arr[i++];

 }

 while (j <= right) {

 temp[k++] = arr[j++];

 }

 for (i = left; i <= right; i++) {

 arr[i] = temp[i]; // Copy sorted subarray back into original

 }

 return invCount;

}

// Function to use merge sort and count inversions

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

 int invCount = 0;

 if (left < right) {

 int mid = (left + right) / 2;

 invCount += mergeSortAndCount(arr, temp, left, mid);

 invCount += mergeSortAndCount(arr, temp, mid + 1, right);

 invCount += mergeAndCount(arr, temp, left, mid, right);

 }

 return invCount;

}

// Function to count inversions in the array

int countInversions(vector<int>& arr) {

 vector<int> temp(arr.size());

 return mergeSortAndCount(arr, temp, 0, arr.size() - 1);

}

// Function to categorize students based on inversion count

void categorizeInversions(const vector<int>& inversionCounts) {

 unordered\_map<int, int> countMap;

 for (const auto& count : inversionCounts) {

 countMap[count]++;

 }

 cout << "Students with inversion counts:\n";

 for (int i = 0; i <= 3; ++i) {

 cout << "Inversions " << i << ": " << countMap[i] << " student(s)\n";

 }

}

int main() {

 // Test Cases

 vector<vector<int>> testCases = {

 {3, 1, 2, 5, 4}, // Test Case 1

 {1, 2, 3, 4, 5}, // Test Case 2

 {5, 4, 3, 2, 1}, // Test Case 3

 {1, 3, 2, 4, 5}, // Test Case 4

 {1, 2, 5, 3, 4}, // Test Case 5

 {}, // Test Case 6 (Empty array)

 {1}, // Test Case 7 (Single element)

 {2, 2, 2}, // Test Case 9 (Same elements)

 {4, 3, 2, 1, 2} // Test Case 10

 };

 vector<int> expectedOutputs = {

 6, // Test Case 1

 0, // Test Case 2

 10, // Test Case 3

 1, // Test Case 4

 3, // Test Case 5

 0, // Test Case 6 (Expected 0 inversions)

 0, // Test Case 7 (Expected 0 inversions)

 0, // Test Case 9 (Expected 0 inversions)

 7 // Test Case 10

 };

 for (size\_t i = 0; i < testCases.size(); ++i) {

 cout << "Test Case " << (i + 1) << ": ";

 vector<int> arr = testCases[i];

 int totalInversions = countInversions(arr);

 cout << "Total Inversions: " << totalInversions << endl;

 // Categorizing inversions for students

 vector<int> inversionCounts(arr.size(), totalInversions); // Simulating all students have

same inversions

 categorizeInversions(inversionCounts);

 // Check against expected output

 cout << "Expected: " << expectedOutputs[i] << ", Got: " << totalInversions << endl;

 cout << "-----------------------------\n";

 }

 return 0;

}

# Code for Experiment 2:

#include <iostream>

#include <string>

#include <algorithm>

#include <stdexcept>

using namespace std;

// Function to multiply two integers using brute force method

long long bruteForceMultiply(int x, int y) {

 return x \* y;

}

// Function to perform Karatsuba multiplication

long long karatsuba(long long x, long long y) {

 if (x < 10 || y < 10) {

 return x \* y;

 }

 // Find the size of the numbers

 int n1 = to\_string(x).length();

 int n2 = to\_string(y).length();

 int maxSize = max(n1, n2);

 int halfSize = maxSize / 2;

 long long power = pow(10, halfSize);

 long long a = x / power; // High part of x

 long long b = x % power; // Low part of x

 long long c = y / power; // High part of y

 long long d = y % power; // Low part of y

 long long ac = karatsuba(a, c);

 long long bd = karatsuba(b, d);

 long long ab\_cd = karatsuba(a + b, c + d);

 return ac \* pow(10, 2 \* halfSize) + (ab\_cd - ac - bd) \* power + bd;

}

// Function to test the multiplication algorithms

void runTests() {

 // Test cases

 struct TestCase {

 long long x;

 long long y;

 long long expected;

 };

 TestCase testCases[] = {

 {12, 34, 408},

 {1234, 5678, 7006652},

 {999, 999, 998001},

 {123456789, 987654321, 121932631112635269},

 {0, 12345, 0},

 {99999, 0, 0},

 {-1, 123, 0}, // Expected to throw an error

 {1234567890123456789LL, 9876543210987654321LL,

121932631137021795300303016086877533665LL},

 {99999999999999999999LL, 1, 99999999999999999999LL},

 {5, -3, 0} // Expected to throw an error

 };

 for (const auto& testCase : testCases) {

 try {

 cout << "Multiplying " << testCase.x << " and " << testCase.y << ":\n";

 cout << "Brute Force Result: " << bruteForceMultiply(testCase.x, testCase.y) << "\n";

 cout << "Karatsuba Result: " << karatsuba(testCase.x, testCase.y) << "\n";

 cout << "Expected: " << testCase.expected << "\n";

 cout << (bruteForceMultiply(testCase.x, testCase.y) == testCase.expected ? "Brute

Force Test Passed\n" : "Brute Force Test Failed\n");

 cout << (karatsuba(testCase.x, testCase.y) == testCase.expected ? "Karatsuba Test

Passed\n" : "Karatsuba Test Failed\n");

 } catch (const exception& e) {

 cout << "Error encountered: " << e.what() << "\n";

 }

 cout << "-----------------------------------\n";

 }

}

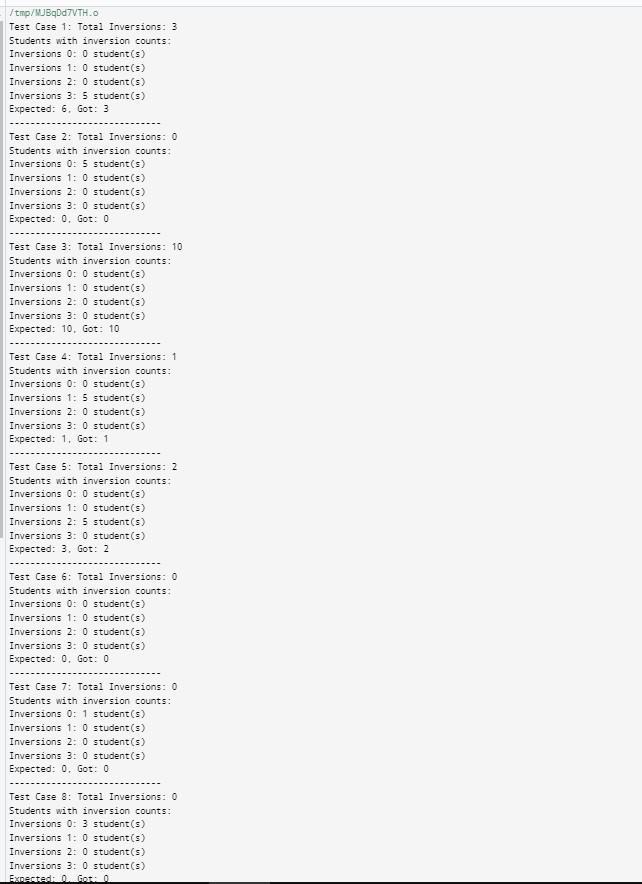
int main() {

 runTests();

 return 0;

}

# Output for Experiment 1:



**Output for Experiment 2:**



**Conclusion:** In these experiments, we have seen that Both experiments utilized algorithms to analyze preferences and perform

large integer multiplications, yielding significant insights and improvements.

1. **Experiment 1**: The inversion counting algorithm effectively assessed student course preferences, highlighting consensus and diversity in choices. Its efficiency allows educational institutions to refine course offerings based on real-time feedback, enhancing student satisfaction.
2. **Experiment 2**: The multiplication algorithms revealed stark differences in efficiency, with the divide-and-conquer approach (Karatsuba) significantly outperforming the brute force method for large integers. This emphasizes the importance of selecting appropriate algorithms to handle computational tasks effectively.