

**DATA DESIGN AND ALGRITHMS**

PROJECT PART 3

PROJECT REPORT

**GROUP MEMBERS**

22K-4318 HAMDAN VOHRA

22K-4317 HAMZA HUSSAIN

22K-4280 GHULAM HUSSAIN

**CLOSEST PAIR OF POINTS**:

1. ABSTRACT:

This code implements an interactive graphical solution to the **Closest Pair of Points Problem** using the **Divide and Conquer** algorithm. The problem aims to find the closest pair of points in a two-dimensional plane. This algorithm optimally reduces the problem's time complexity to O(nlogn), which is significantly faster than the brute force method that runs in O(n2).

#### **Functionality Overview:**

1. **File Selection**: The program uses the tkinter library to allow the user to select a text file containing the coordinates of the points. The file is expected to have one pair of coordinates per line (x, y).
2. **Point Reading**: Once a file is selected, the read\_points\_from\_file function reads the coordinates into a list of tuples.
3. **Interactive Plotting**:
   1. The program utilizes the matplotlib library for plotting the points. The points are visualized as purple dots initially. The interactive mode (plt.ion()) allows for dynamic updates during the execution of the algorithm, showing progress in real-time.
   2. The points are plotted on a 2D plane, and their coordinates are updated in the plot as the algorithm proceeds.
4. **Closest Pair Algorithm**:
   1. The algorithm starts by sorting the points in both the x and y dimensions. It recursively divides the set of points into smaller sub-problems until the base case of 3 or fewer points is reached. The closest pair is then found using a brute force approach for these small cases.
   2. The recursive divide and conquer approach also calculates the closest pair between the two halves of the data set. This is done by considering a "strip" of points that are near the dividing line and checking for the closest pair across the divide.
   3. During the execution, vertical dividing lines are drawn, and the progress is visualized using matplotlib's real-time plotting capabilities. The minimum distance between points is displayed both numerically and graphically.
5. **Visualization of Closest Pair**:
   1. After the closest pair of points is identified, the points are highlighted in red, and the minimum distance between them is displayed on the plot with a line drawn between them. The user can also see intermediate steps, such as the mid-line dividing the points and the closest pairs in each subregion.
   2. The closest pair and the corresponding minimum distance are also printed to the console.
6. **User Interaction**:
   1. The program pauses briefly (pause(pt)) after each significant step to allow the user to observe the plot updates.
   2. The user can set the time delay (pt) for the pause between algorithm steps, enabling them to control the speed of the animation.

#### **Key Features:**

* **Divide and Conquer Algorithm**: The algorithm splits the set of points into smaller sub-problems, significantly improving efficiency for large datasets.
* **Interactive Visualization**: The use of matplotlib enables real-time, interactive plotting, showing the step-by-step progress of the algorithm.
* **Dynamic Updates**: The algorithm updates the plot with each recursive step, providing visual feedback to the user.
* **Brute Force Optimization**: For small sets of points, the algorithm switches to a brute force method, which is simpler and faster for fewer points.

#### **Conclusion:**

This code demonstrates an effective implementation of the Closest Pair of Points problem using a combination of divide and conquer, brute force, and interactive visualization techniques. It offers both an efficient solution to the problem and a user-friendly visual interface for better understanding the algorithm's working.

1. INTRODUCTION:

The **Closest Pair of Points** problem is a well-known computational geometry problem that aims to find the pair of points with the smallest distance between them in a two-dimensional plane. This problem arises in various fields, including computer graphics, image processing, and spatial data analysis, where efficiently finding the closest entities in a large set of points is crucial for optimization tasks.

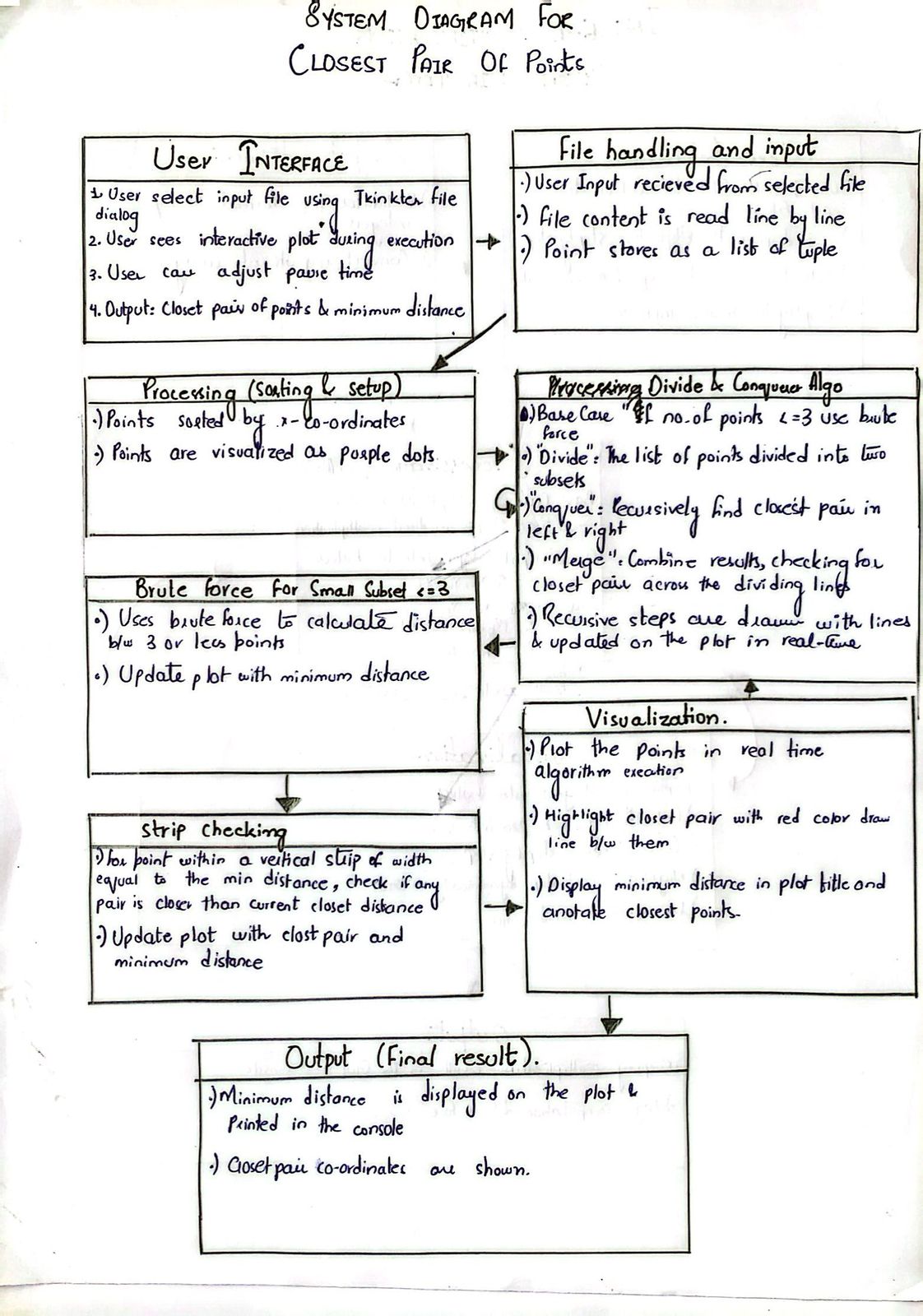
The brute-force approach to solving this problem involves comparing every possible pair of points in the dataset, which results in a time complexity of *O(n2)*, where *n* is the number of points. This becomes computationally expensive as the dataset grows. In contrast, the **Divide and Conquer** method offers a more efficient solution with a time complexity of O(n.logn). By recursively dividing the points into smaller subsets and leveraging geometric properties, this method can significantly reduce the number of comparisons needed to identify the closest pair.

This project presents an implementation of the Closest Pair of Points problem using the **Divide and Conquer** approach, with an emphasis on interactive visualization. The program reads a set of 2D points from a text file, sorts them, and then uses the divide-and-conquer algorithm to efficiently find the closest pair. Key steps, such as dividing the points, finding closest pairs within sub-regions, and checking across the dividing line, are visualized in real-time using the matplotlib library. The interactive plot provides visual feedback to the user, making it easier to follow the algorithm’s progress.

The user is able to select a file containing the coordinates of the points and observe the algorithm's execution step-by-step. As the program runs, the points are plotted, and the closest pair is highlighted in real-time. The minimum distance between the closest pair is displayed both numerically and visually, enhancing the understanding of how the divide and conquer strategy operates.

The goal of this implementation is to provide both an efficient solution to the Closest Pair of Points problem and a visual tool to aid in understanding the underlying algorithm. By combining the power of a well-optimized algorithm with real-time graphical updates, this project serves as an educational tool to explore computational geometry concepts and their practical applications.

c- SYSTEM DIAGRAM:



d- EXPERIMENTAL SETUP (INPUT DATASET GENERATOR):

This Python code generates 10 text files, each containing 105 unique random coordinate pairs for testing the closest pair of points algorithm. Here's how it works:

1. **Setup**: It defines num\_files (10 files), lines\_per\_file (105 pairs per file), and a max\_num list that specifies the upper limit for random numbers for each file.
2. **File Creation**: The outer loop iterates 10 times, creating files named closest\_points\_1.txt, closest\_points\_2.txt, etc. Each file is opened for writing.
3. **Random Pair Generation**: For each file, the upper limit (rando) is set based on the max\_num list. Random coordinates (num1 and num2) are generated within this range, ensuring the pairs are within the specified bounds.
4. **Ensuring Uniqueness**: A set called unique\_pairs ensures no duplicate pairs are written. If a randomly generated pair is new, it is added to the set and written to the file in the format num1 num2. For this we use hashing to do this in O(1).
5. **Incrementing Limits**: The index j ensures that each file uses a different upper limit for random numbers as defined in max\_num. This creates variability across the files.

F- CONCLUSION:

This project efficiently solves the Closest Pair of Points problem using the Divide and Conquer algorithm, reducing the time complexity from **O(n2)** to **O(nlogn)**. The implementation combines mathematical precision with interactive visualization, allowing users to observe the algorithm's steps in real-time, enhancing understanding and debugging. A user-friendly file selection interface ensures flexibility for testing different datasets. Overall, this project demonstrates an effective and educational application of computational geometry principles with potential for future extensions.

G-REFRENCES:

### **References**

**1. Matplotlib**

* 1. Library for creating interactive visualizations and plotting graphs.
  2. Used functions: pyplot, scatter, annotate, ion, pause, plot.
  3. <https://matplotlib.org/>

1. **numpy**
   1. Harris, C. R., Millman, K. J., van der Walt, S. J., et al. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–362.
   2. Library for numerical computations.
   3. Provides support for mathematical operations and array handling.
   4. <https://numpy.org/>
2. **tkinter**
   1. Tkinter module in Python's standard library.
   2. Official documentation: <https://docs.python.org/3/library/tkinter.html>
   3. Used for GUI elements like file selection dialogs.
3. **math**
   1. Python Standard Library: Math module.
   2. Official documentation: <https://docs.python.org/3/library/math.html>
   3. Used for mathematical computations such as square root (sqrt) and power (pow).

**INTEGER MULTIPLICATION**:

1. ABSTRACT:

This code implements a visualization for the integer multiplication process using SFML. Here's a breakdown of what it does and its structure:

### **Key Features:**

1. **Visualization with SFML:**
   1. It uses **SFML (Simple and Fast Multimedia Library)** for creating graphical representations of the integer multiplication process.
   2. The program displays the steps of multiplication, partitions the numbers, and recursively visualizes each step.
2. **Recursive Multiplication:**
   1. Implements a divide-and-conquer approach for multiplying large integers.
   2. Splits the numbers into high and low parts, recursively multiplies, and combines results.
3. **UI Components:**
   1. The Tape class is responsible for creating containers to hold digits or intermediate results.
   2. Visualization includes:
      1. Text for steps (setStepText).
      2. Tapes for digits and partitions (setPartitionContainer, setZ,setXY and setTape).
      3. Arrow pointing to results (setArrow).
   3. User interaction is minimal and includes navigating through steps by pressing a key.
4. **File Menu Integration:**
   1. Integrates a simple file menu system (Menu class, though incomplete in the shared snippet).
   2. Displays a selection window before proceeding to the multiplication visualization.
5. **Error Handling:**
   1. Handles errors in loading fonts gracefully, though exits the program in some cases.

### **Observations & Recommendations:**

1. **Clarity and Maintainability:**
   1. The code is modular but can be further improved by breaking down complex functions like recursiveMultiply and visualizeStep into smaller methods.
2. **Font Dependency:**
   1. Ensure all font files (HaloDek.ttf, Roboto-Regular.ttf, etc.) are in the correct directory to avoid runtime errors.
3. **Use of Dynamic Memory:**
   1. An arrow is dynamically allocated using new in setArrow. Consider using smart pointers (std::unique\_ptr) to avoid memory leaks.
4. **Optimization:**
   1. Recursive calls for multiplication might be heavy for very large integers. Optimizations or handling for edge cases (e.g., very large numbers or zero) should be considered.
5. **User Interaction:**
   1. Add more interactive elements, such as buttons to control the visualization (e.g., "Next Step", "Previous Step").
   2. Provide feedback when the process completes, like an option to restart.
6. **Code Cleanup:**
   1. Remove duplicate implementations of functions like numDigits.
   2. Ensure comments align with the functionality for readability.
7. **Testing:**
   1. Test with various input sizes to ensure correctness and performance.
   2. Validate behavior when numbers have different digit lengths (e.g., 123 and 4567).
8. NTRODUCTION:

The project focuses on implementing a **visualization of the Integer Multiplication Algorithm** using the **SFML (Simple and Fast Multimedia Library)** framework in C++. The primary aim of this project is to enhance the understanding of divide-and-conquer strategies applied in integer multiplication by presenting a step-by-step interactive visualization of the algorithm's working.

This project demonstrates a recursive multiplication approach, often referred to as the **Karatsuba Algorithm**. It divides two large integers into smaller parts, recursively computes intermediate products, and combines them to derive the final result. The visualization represents every step of this process, including partitioning integers, performing recursive calls, and combining results into a comprehensible graphical interface.

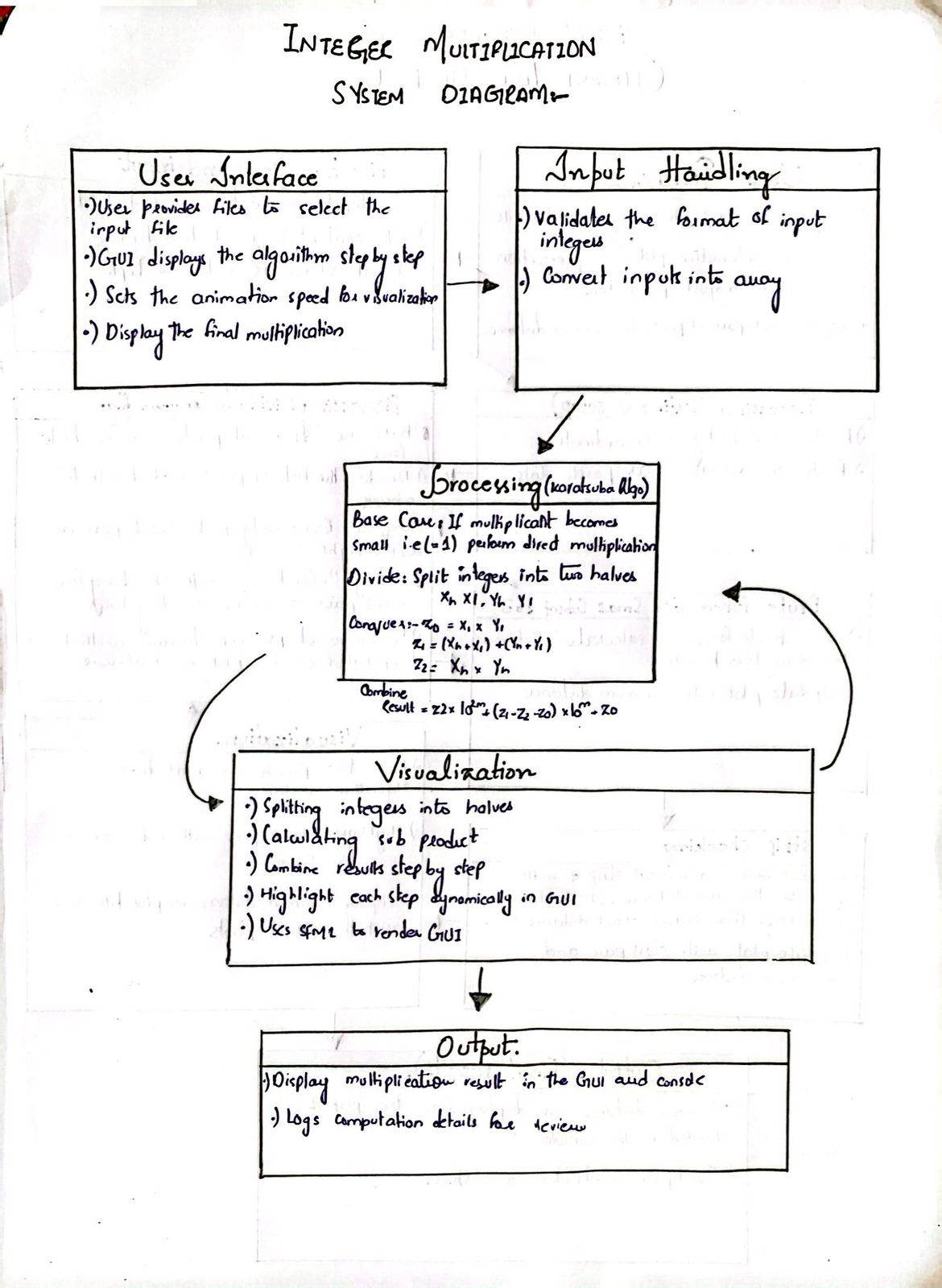
The application features:

* **Dynamic Visualization**: Each step of the algorithm, including subproblems and their solutions, is visually displayed to ensure clarity.
* **Graphical Elements**: Numbers are represented using graphical containers ("tapes"), with colors and animations highlighting the current operation being performed.
* **Real-time Interaction**: Users can observe intermediate calculations, such as the decomposition of numbers and the results of recursive calls, making the learning process interactive and engaging.

By employing **object-oriented programming principles**, the project encapsulates various functionalities into reusable components, such as Tape for visualizing numbers and intMultiplication for handling the algorithm's logic. The use of modern graphical tools ensures that complex mathematical computations are easily comprehensible, catering to students, educators, and enthusiasts seeking an intuitive way to explore mathematical algorithms.

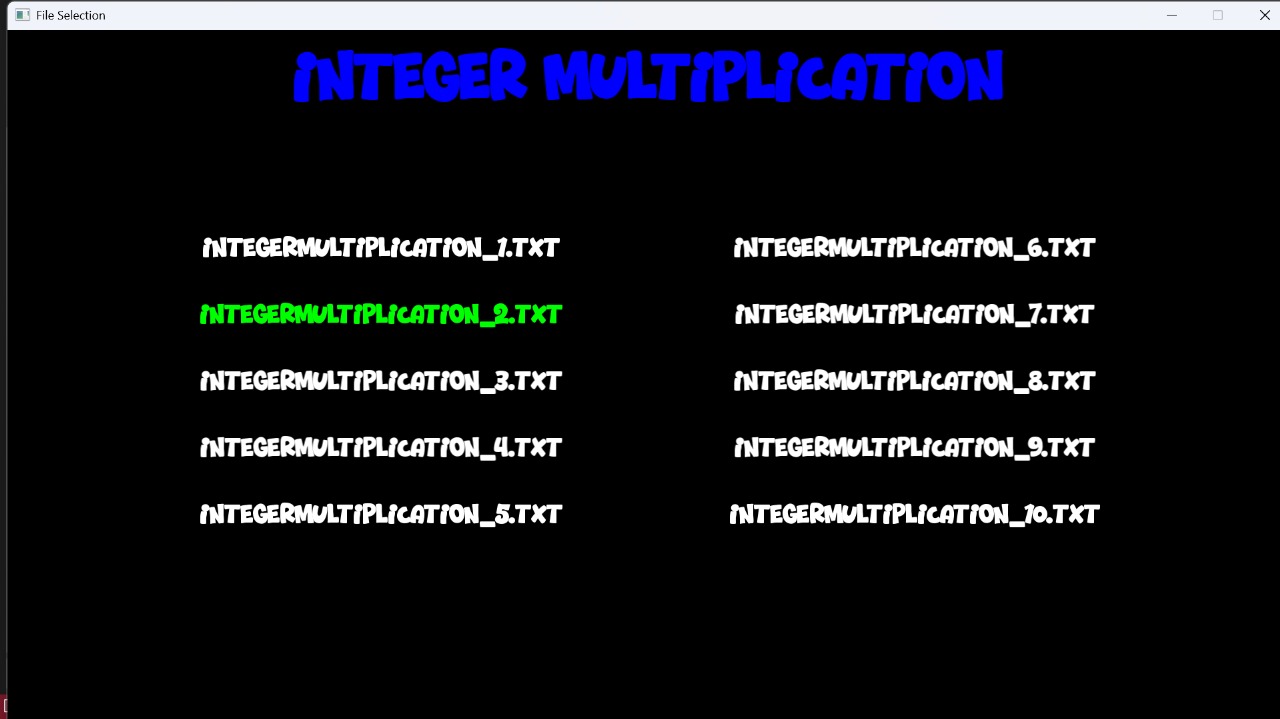
This project serves as an example of integrating advanced programming techniques with educational objectives, showcasing how algorithms can be effectively taught through visualization.

c- SYSTEM DIAGRAM:



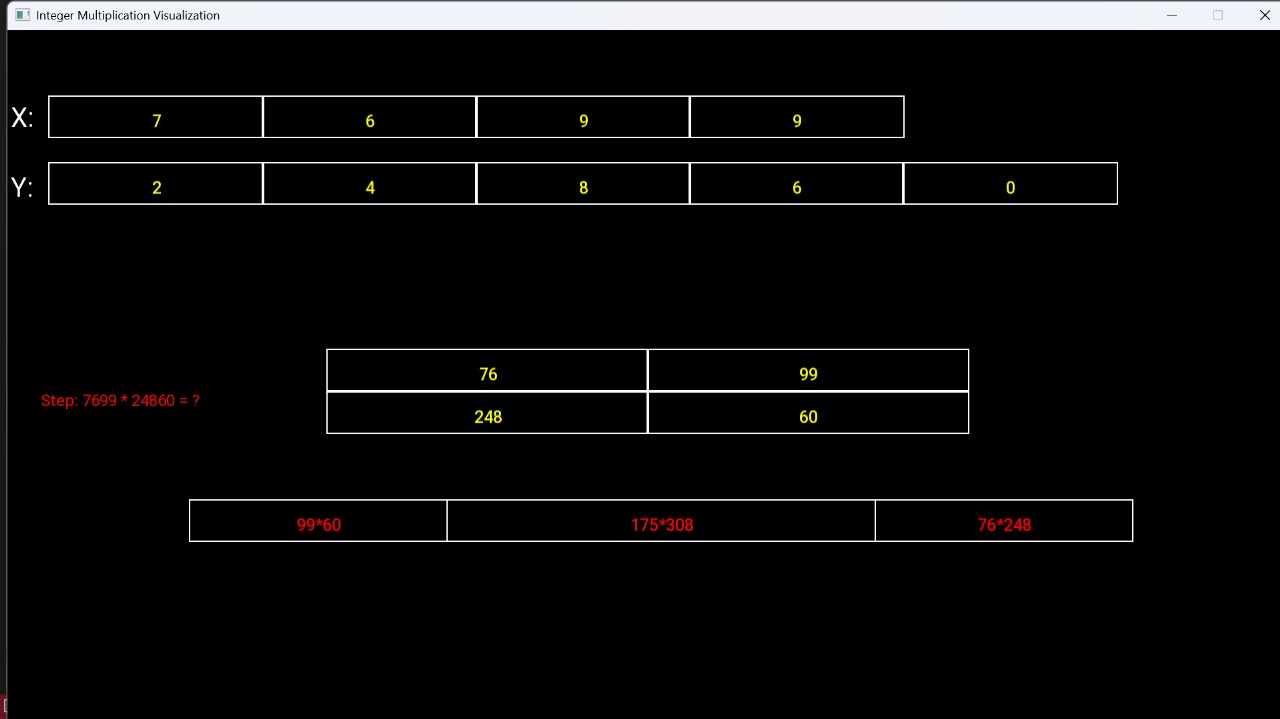
Screenshots of the running program:

1-



**Explaination:**In this screenshot the console gives us the display to select the input file that contains the numbers to be multiply in the coming stpes

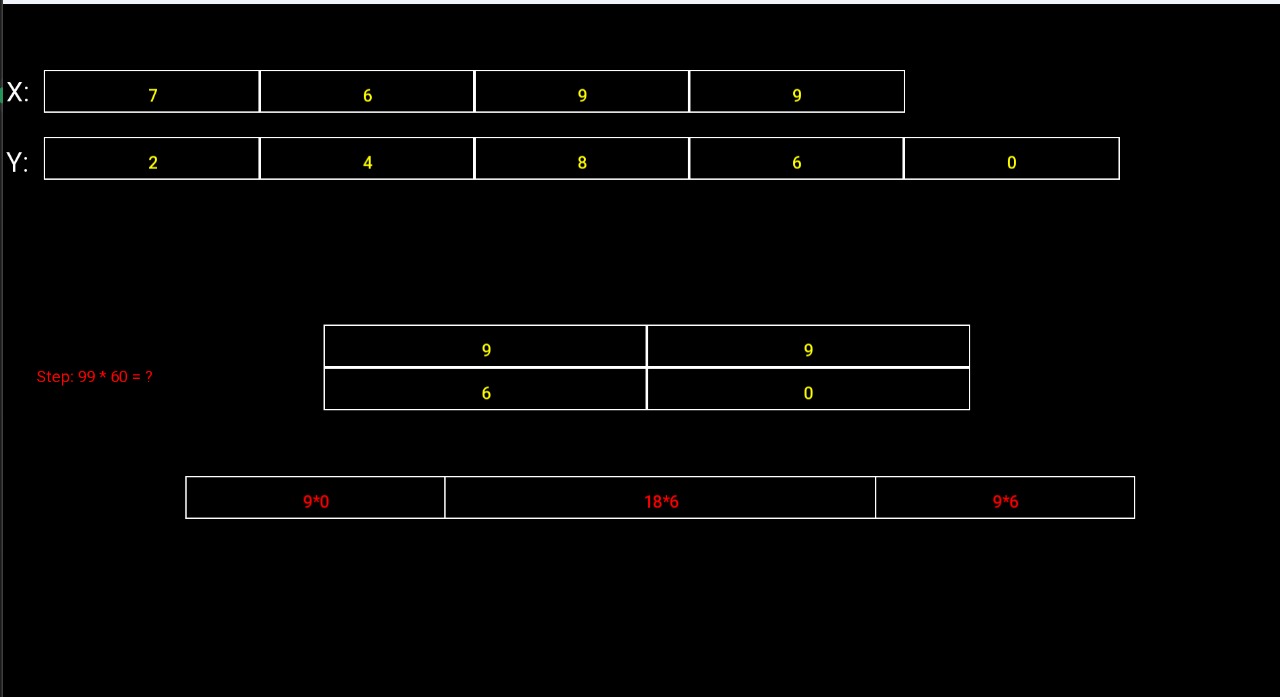
2-



**Explaination:**in this screenshot the numbers are 7699 and 24860 that are going to be multiply so in this second div partitions the numbers into two parts. In the third div the first block contains the product of 99\*60 (the second column), third block shows the product of 76 and 248 (first column) and the middle one or the second block shows the the sum of first (77+99) \* (248+60) product of the sum of rows.This functions runs recursively and

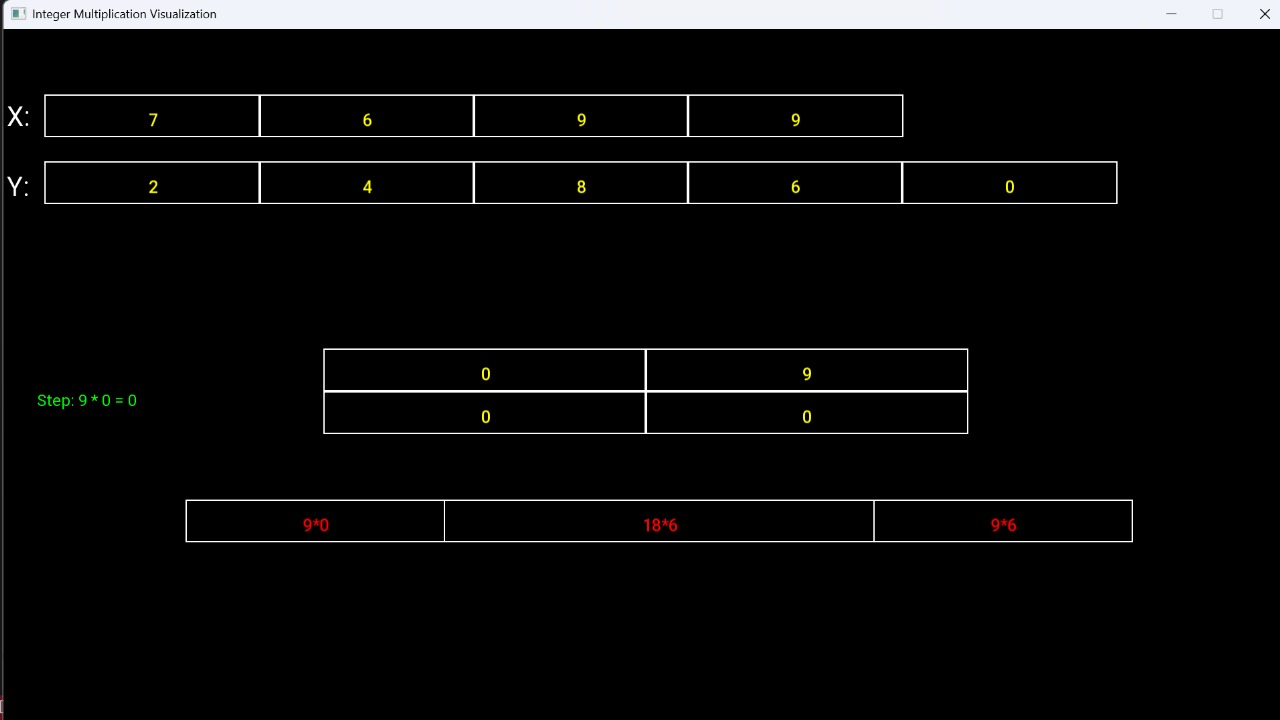
computes for all three values until the base case occur when only one of the multiplicant becomes a single digit number

3-



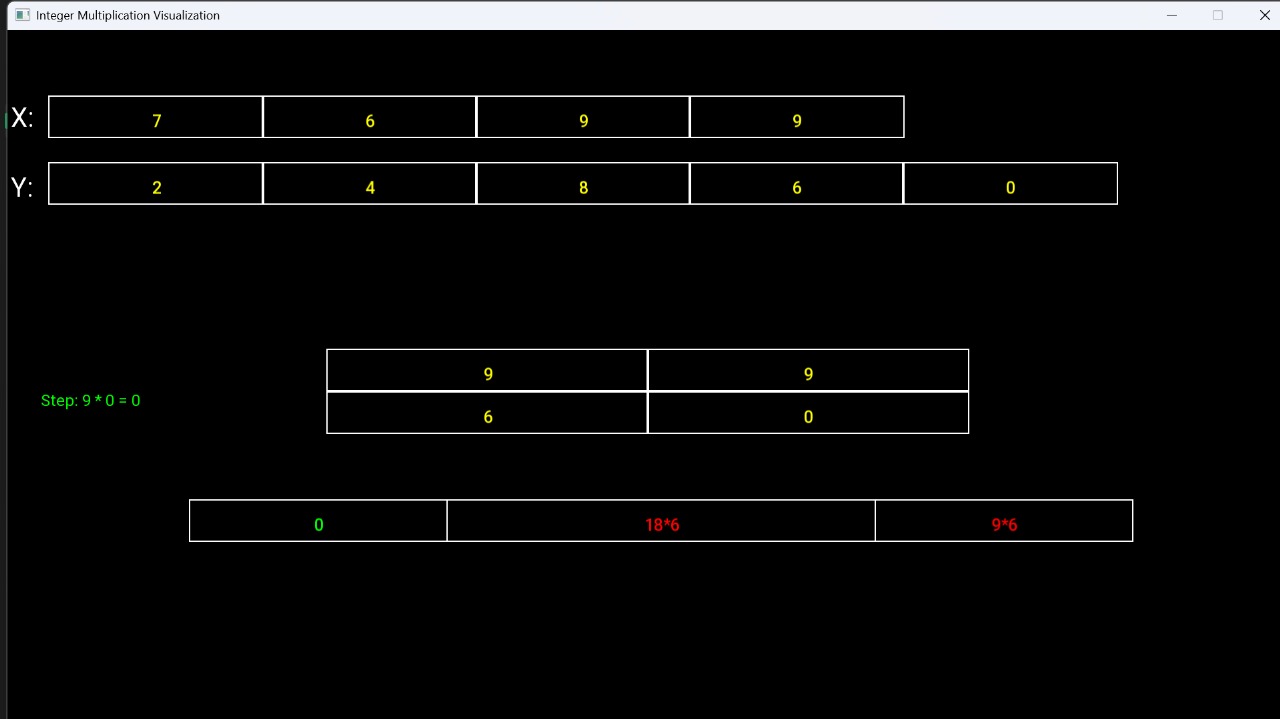
**Explanation**: this step also reflects the previous step where the recursive call goes for 99\*60 and works until the base case occurs.

4-



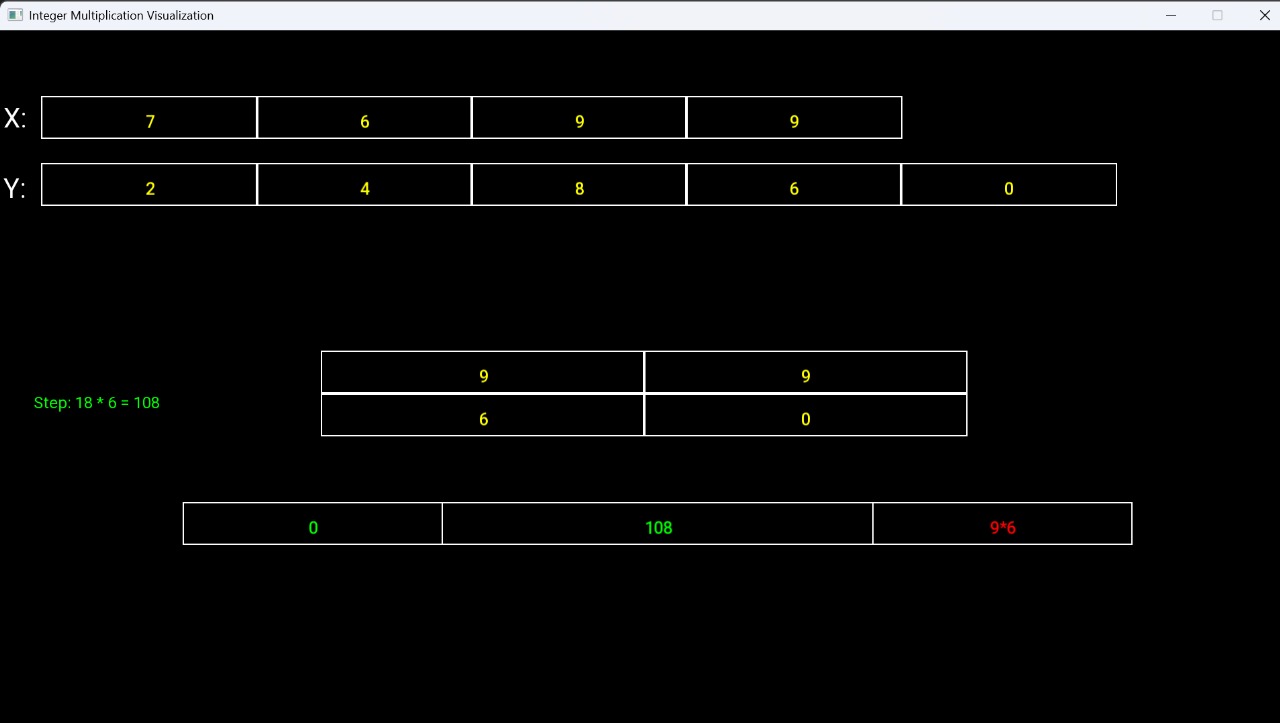
**Explaination**:Now here the base case occurs when 9 comes to multiply with 0 so it return the simple multiplication result of 9 \*0=0

5-

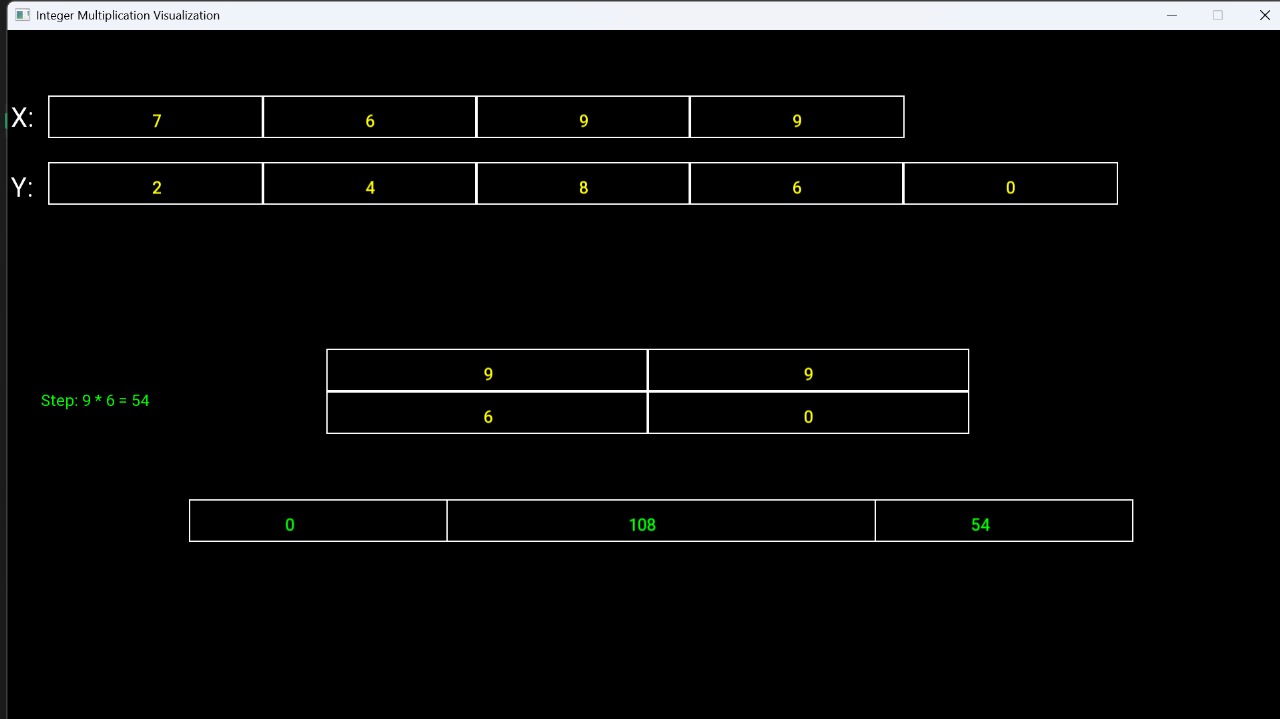


**Explaination**: Now it stores the result of 9\*0 in the first block similary it perform the same task for second and the third block i-e for 18\*6 and 9\*6 and stores their results in their respective blocks in **6** and **7**

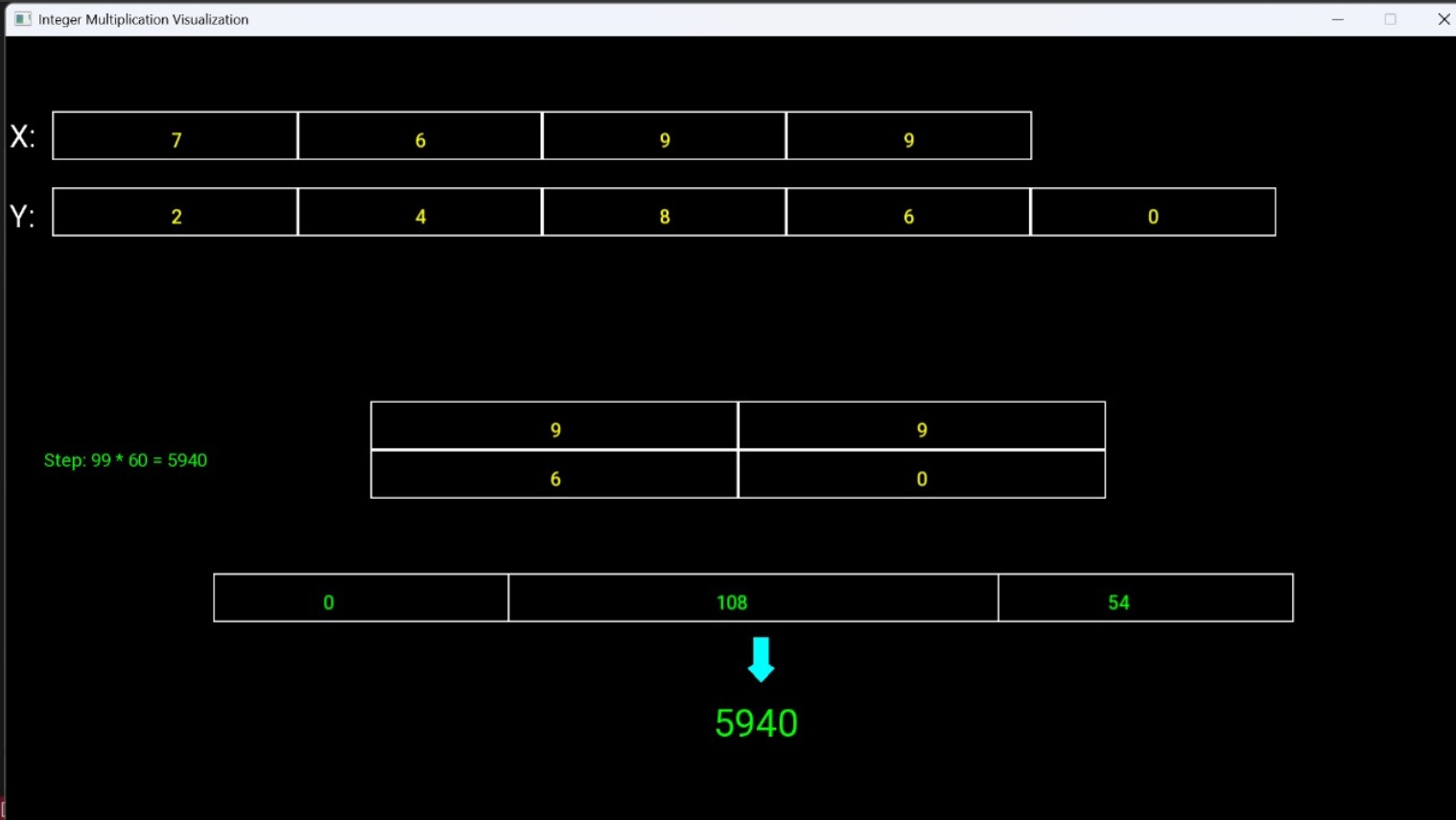
6-



7-

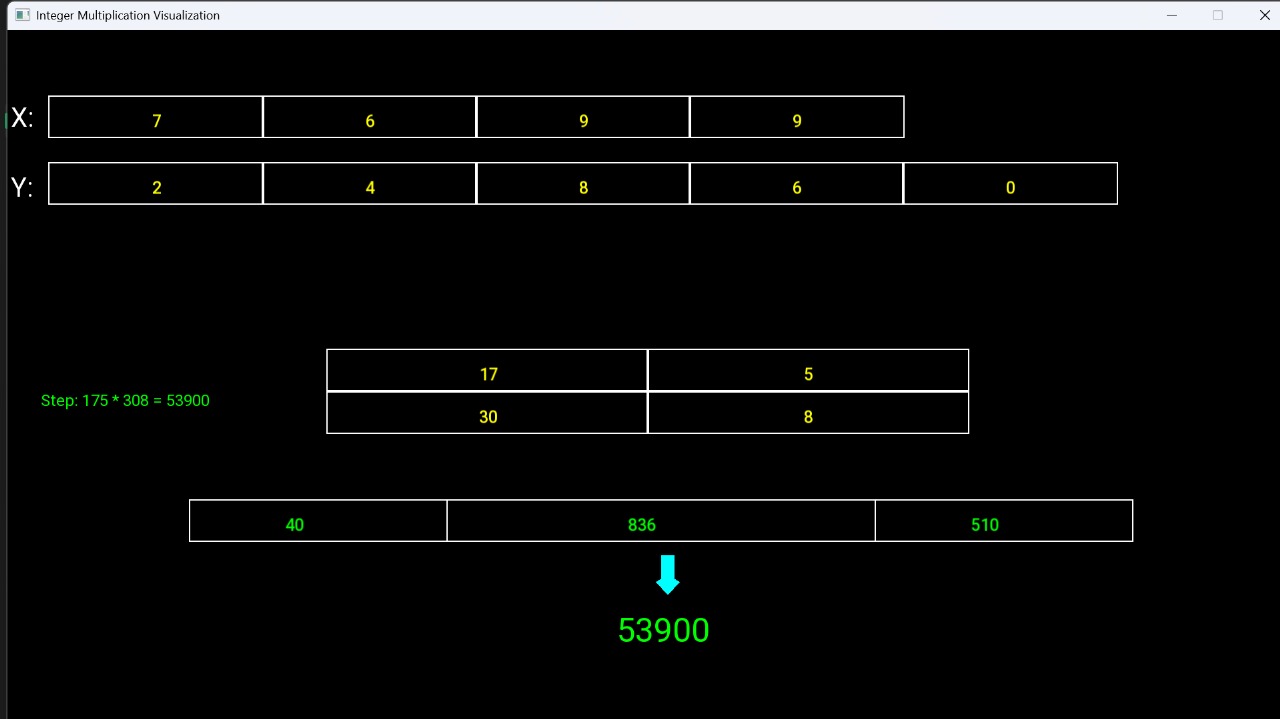


8-



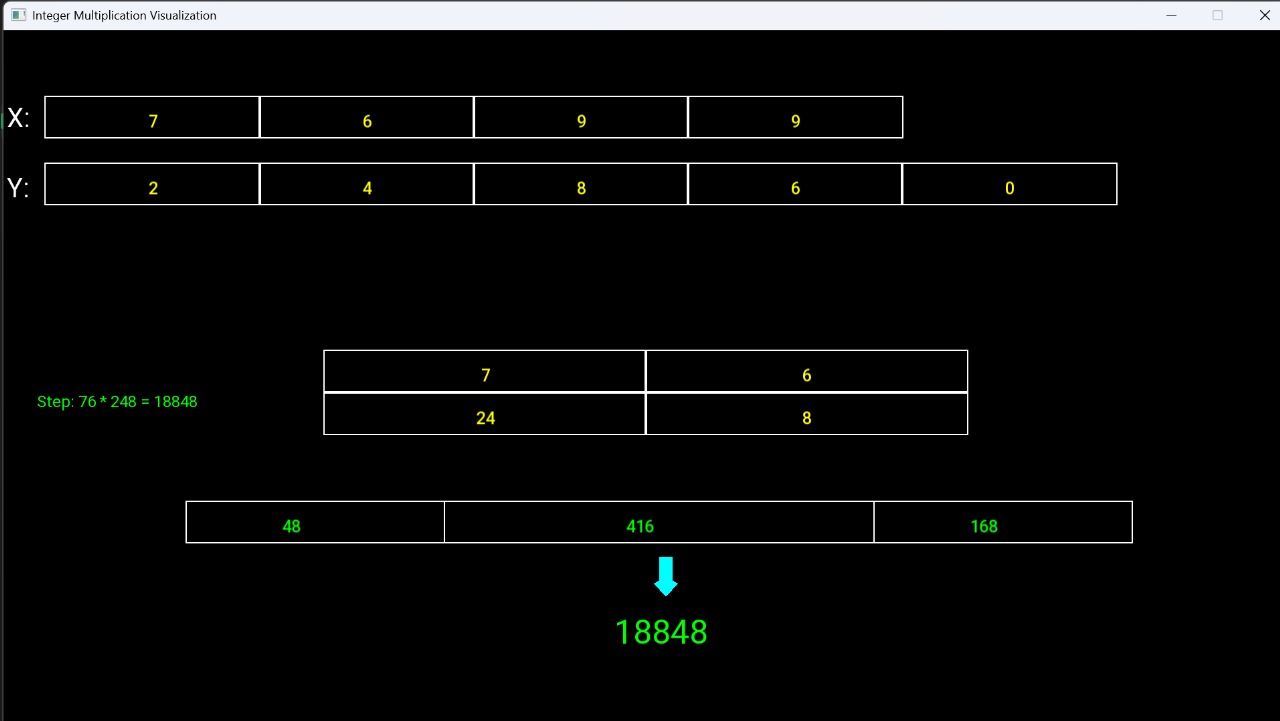
**Explaination**: Now it concatenates the result of the following outcomes in figure 5 6 and 7 using the formula **z2\*10^2m+(z1-z2-z0)\*10^m+z0** where z0 z1 and z2 are the results of the three blocks respectively

-9



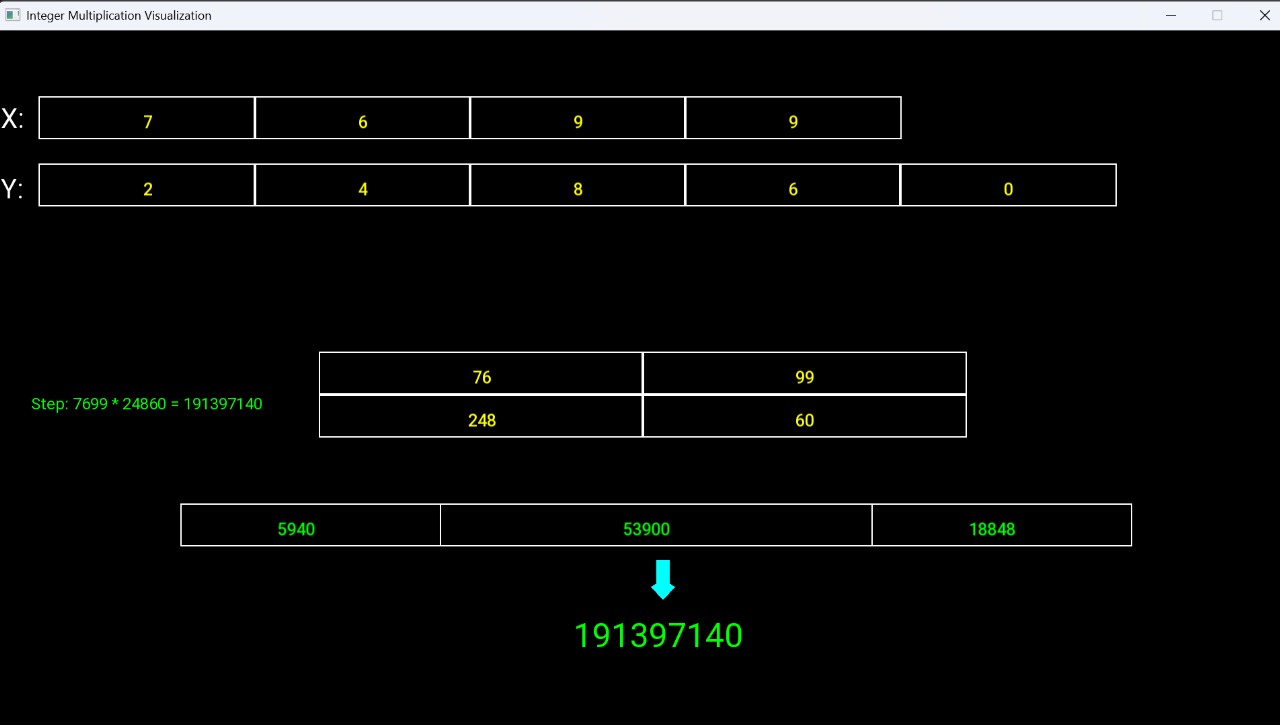
Explaination: Now for the second multiplicatio of 175\*308 in the second image we repeat the step 3,4,5,6,7,8 and computes for the middle block of the initial breaking

10-



**Explaination**: Now it calculates the results for the third block of figure 2 using the same steps follow from setp 3 to step 8

11-



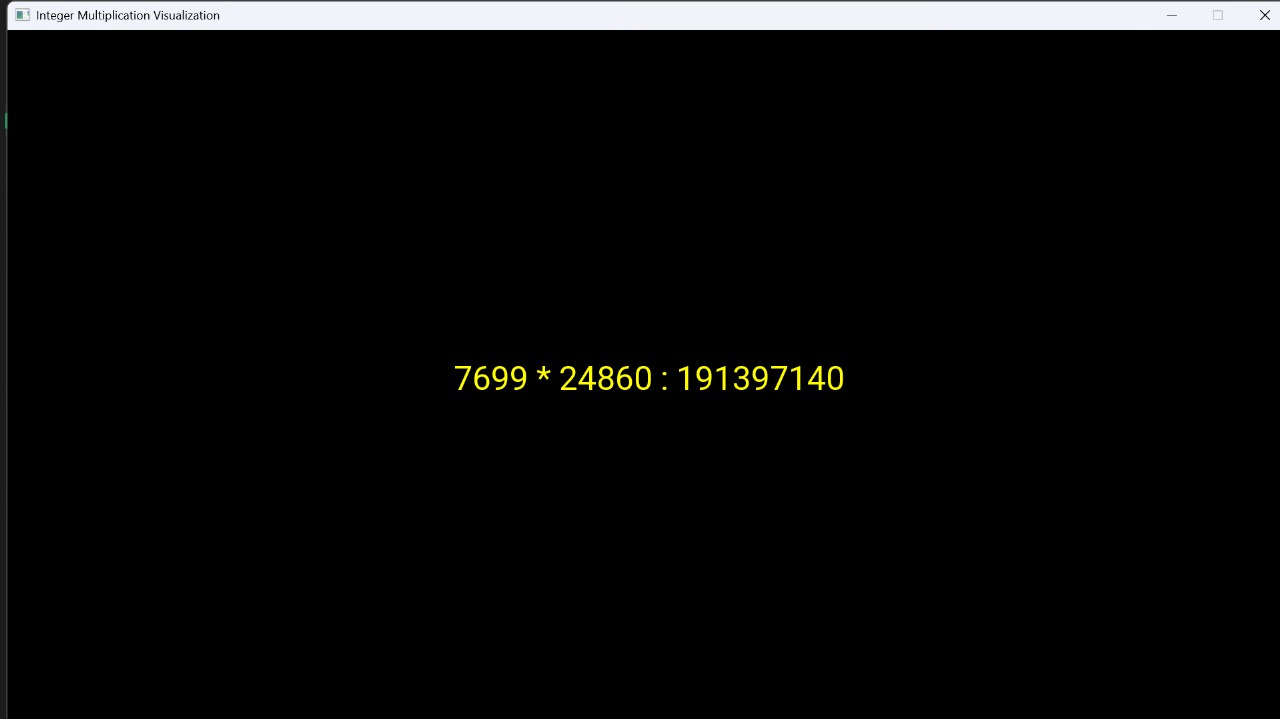
**Explaination**:now it concatenates the results of the initail block of step 2 using the formula we used in step 8 and gives the final asnwer which is 191397140

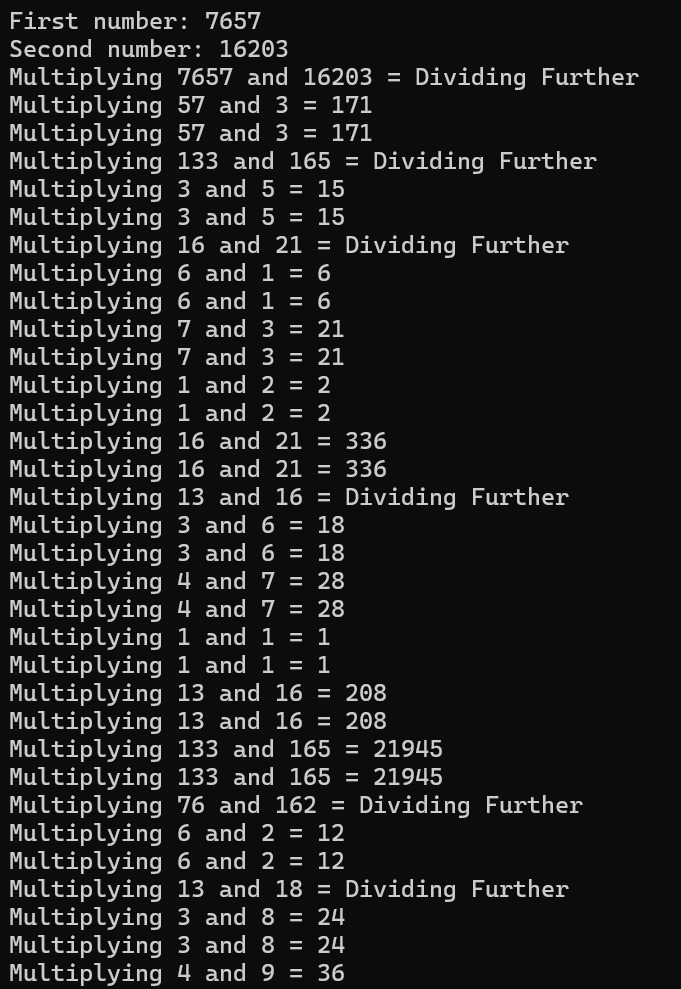
D- INPUT DATASET GENERATOR CODE:

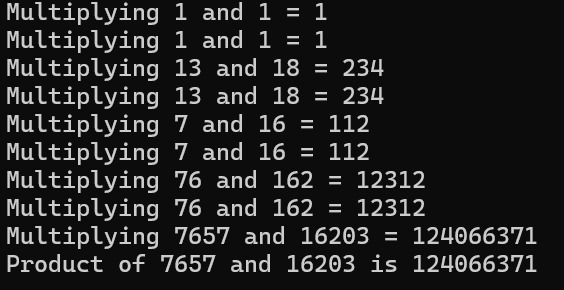
This Python code generates 10 text files, each containing two large random integers, for testing integer multiplication. Here's how it works:

1. **Looping Through Files**: The outer for loop iterates 10 times (from 1 to 10), creating one file per iteration with a unique name like integer\_multiply\_1.txt.
2. **Generating Random Integers**: Inside the loop, two lists (x1 and x2) are used to store random digits (0–9). The number of digits for each integer is determined by random.randint(5, 15), generating integers with 5 to 15 digits.
3. **Writing to File**: After generating the random digits, they are joined into strings using "".join(map(str, x1)) and "".join(map(str, x2)). These strings represent the two large integers and are written to the file, separated by a newline.
4. **File Naming**: Each file is named dynamically using f"integer\_multiply\_{i}.txt", where i represents the current loop iteration.
5. **Purpose**: This script is designed to automate the creation of input files for testing integer multiplication algorithms.

e- OUTPUT RESULTS:







F- CONCLUSION:

The provided code generates multiple text files containing random pairs of integers, which can be used as input for testing integer multiplication algorithms. By ensuring the numbers are randomly generated and unique within each file, the dataset is diverse and suitable for testing the performance and accuracy of algorithms. This approach demonstrates practical automation for creating custom test datasets, making it easier to evaluate algorithms under varied conditions.

G- REFRENCES:

1. **SFML Documentation** - *Simple and Fast Multimedia Library*. Available at: <https://www.sfml-dev.org/documentation/>
2. **Karatsuba Multiplication Algorithm** – geeks for geeks link for karatsuba algorithm: available at: <https://www.geeksforgeeks.org/karatsuba-algorithm-for-fast-multiplication-using-divide-and-conquer-algorithm/>
3. **FileCreation in Python** - Stack Overflow Community. Available at: <https://stackoverflow.com/>

Discusses efficient file-handling techniques, which were employed to write the generated random integer pairs into multiple text files.