University of Engineering and Technology, Lahore

Department of Computer Science

Project Report

Team Members

Name	Roll Number	Email
Abdul Hannan	2025-CS-121	abdulhanan4156@gmail.com
M. Salahudin	2025-CS-163	msalahudin441@gmail.com

Session: Fall'25 Morning **Section:** C **Semester:** 1st

Course: CSC101 - Discrete Mathematics **Teacher:** Mr. Waqas Ali

January 1, 2026

Table of Contents

Project Report

Submission Deadline: January 6, 2026

1. Project Title

Sorting Algorithm Visualizer

2. Chosen Track

State which research track your project belongs to. This helps contextualize your work within the course curriculum and clarifies the primary discrete mathematics concepts you are applying.

- Track 1: Foundations of Logic
- Track 2: Discrete Structures
- Track 3: Algorithm Design & Analysis
- Track 4: Graph & Tree Algorithms
- Track 5: Number Theory Applications
- Track 6: Counting & Combinatorics

3. Overview

This Sorting Algorithm Visualizer simplifies complex data structures by providing real-time representations of sorting techniques using hash-sign bars. It effectively demonstrates time complexity and efficiency of algorithm, helping users to understand the sorting process. It also helps users to choose the best sorting algorithm

4. Summary

The Sorting Algorithm Visualizer addresses the issues faced by students to understand the working of sorting algorithm. Using a series of hash signs (#) to represent data values of varying magnitudes, the tool provides a clear visual of popular algorithms such as Bubble Sort, Quick Sort, Merge Sort, Insertion sort and Selection sort. This visual is paired with live data tracking, specifically highlighting the time complexity (O(n)) and the number of comparisons made during the process. This methodology allows users to see exactly why certain algorithms cannot provide better results when data volume is increased.

This project is a basic concept of Discrete **Mathematics**, especially in **algorithm analysis**. Sorting is a practical application of permutations and set theory. By visualizing the "steps" of an algorithm, the project provides a knowledge of **Big O notation** and the growth rates of functions. It turns the mathematical proof of an algorithm's efficiency into a visible work showing the importance of mathematical optimization in computer science. The book that we used as a reference is **Discrete Mathematics and Its Applications by Kenneth Rosen.** The concepts of this book helped us in completing this project and enhancing our knowledge of sorting algorithm.

5. Technology Stack & Libraries Utilized

Programming Languages: C++

Libraries: The libraries used are:

- Iostream
- Chrono
- thread

These libraries helped us to sort and visualize the data entered by the user to sort.

The IDE that we used is Visual Studio Code (VS CODE). The reason behind using VS CODE as IDE is that it is a well-structured and easy to use.

6. List of Features

The core feature of our project is to find the best Sorting Algorithm that can be used to solve the real-world problems. The list of Algorithm and features are as follows:

- User Friendly interface

The program displays a menu that helps the user to select among the sorting algorithms.

- Sorting

A sorting algorithm is **stable** if it preserves the relative order of elements.

If you sort a list of students by "Name" and then by "Grade," a stable sort ensures that students with the same grade remain in alphabetical order.

7. Mathematical Foundations

Key discrete mathematics concepts from **Rosen's textbook** that your project will utilize includes:

- Algorithm Design
- Complexity Analysis
- Comparison Logic

8. Examination of Key Functions/Methods

I. The Visualization Engine: visualizer

- **Function Name & Signature:** void visualizer(int arr[], int X)
- **Purpose:** This is the heart of the project. It provides the "visualizer" aspect by rendering the state of the array as a horizontal bar chart in the console after every swap or significant movement.
- **Algorithm Description:**
  1. Iterate through each element of the array.
  2. For each value, determine the number of # characters to print (capped at 30 for UI stability).
  3. Print the bars. If the value exceeds the cap, append ... to indicate a larger value.
  4. Print the actual integer value in parentheses for clarity.
- **Time/Space Complexity:** * **Time:** O (n \times m) where n is the array size and m is the value of the elements (capped at 30). In Big-O terms relative to input size, it is effectively $O(n)$.
  - **Space:** O(1) (In-place printing).
- **Implementation Details:** The use of a hard cap (bars = 30) is a crucial design choice. It prevents the console output from breaking onto new lines if the user inputs very large integers, maintaining the visual integrity of the chart.
- **Code Snippet:**

C++

```
void visualizar(int arr[], int X) {
   for (int i = 0; i < X; i++) {
      int bars = arr[i]; // Use value to determine bar length
      if (bars > 30) bars = 30; // Optimization: Cap length for display
      for (int j = 0; j < bars; j++)
         cout << "# "; // Print visual representation
      cout << " (" << arr[i] << ")" << endl; // Show literal value
   }
   cout << endl;
}
```

II. Divide and Conquer: quickSort & partition

- **Function Name & Signature:** int partition(int arr[], int n, int low, int high)
- **Purpose:** partition is the operational core of Quick Sort. It organizes elements around a pivot, which is essential for the efficiency of the Divide and Conquer strategy.
- **Algorithm Description:**

Plaintext

1. Select the last element as the pivot.
2. Maintain an index 'i' to track the boundary of elements smaller than the pivot.
3. Loop through the array from 'low' to 'high-1':

- If current element < pivot, increment 'i' and swap elements.
- Call visualizar () to show the swap.

4. Swap the pivot into its correct sorted position (i + 1).
5. Return the index of the pivot.

- **Time/Space Complexity:**
  - **Time:** O(n \log n) average case; $O(n^2) worst case.
  - **Space:** O(\log n) due to recursive stack frames.
- **Implementation Details:** The visualization call is placed inside the if block during swaps. This allows the user to see the "partitioning" process happen in real-time, which is often the hardest part of the algorithm to conceptualize.
- **Code Snippet:**

C++
```
int partition(int arr[], int n, int low, int high) {
  int pivot = arr[high], i = low - 1;
  for (int j = low; j < high; j++) {
    if (arr[j] < pivot) { // Compare against pivot
      i++;
      swap(arr[i], arr[j]);
      visualizar (arr, n); // Update UI after swap
    }
  }
  swap(arr[i + 1], arr[high]); // Place pivot in middle
  visualizar(arr, n);
  return i + 1;
}
```

III. Data Management & Performance: main (Driver Logic)

- **Function Name & Signature:** int main()
- **Purpose:** Manages the application lifecycle, handles user input/output, and performs high-precision timing of the algorithms.
- **Algorithm Description:**
  1. Initialize an infinite loop to allow multiple sorting runs.
  2. Prompt user for array size and individual elements.
  3. **Optimization:** Create an arrCopy to preserve the original user input, allowing for future extensions (like comparing different algorithms on the same data).
  4. Capture the start time using high_resolution_clock.
  5. Execute the chosen algorithm via a switch statement.
  6. Calculate the duration and display complexity results.
- **Time/Space Complexity:**
  - **Time:** O (1) overhead plus the complexity of the chosen sort.
  - **Space:** O(n) to store the array and its copy.
- **Implementation Details:** The use of std::chrono::high_resolution_clock ensures that even for small arrays, the performance difference between O(n^2) and O (n \log n) algorithms is measurable in milliseconds.
- **Code Snippet:**

C++
```
// Logic for performance tracking
auto start = high_resolution_clock::now(); // Start timer

switch (choice) {
  case 1: bubblesort(arrCopy, size); break;
  //
}
```

```
auto stop = high_resolution_clock::now(); // End timer
chrono::duration<double, milli> duration = stop - start;
display Complexity(choice, duration.count()); // Pass time to UI
```

9. Limitations

1. Scope Limitations

- **Data Type Restriction**: The project only supports integer arrays. It does not handle floating-point numbers, strings, or custom objects.
- **Visual Representation Cap**: The visualizer is designed for console output, which limits the graphical representation.
- **Single-Sort Focus**: The system cannot perform side-by-side comparisons of two algorithms simultaneously; users must run them one after another.

2. Performance Constraints

- **Input Size Bottleneck**: Because visualizar is called after nearly every swap or comparison, the overhead of console I/O becomes the dominant time consumer. For input sizes exceeding 100 elements, the program may appear to "hang" due to the volume of text being printed.
- **Console Buffer Limits**: Large arrays will exceed the vertical scroll back buffer of standard terminal windows, making it impossible for the user to see the initial steps of the sort.
- **System Commands**: The use of system("cls") is platform-dependent (Windows) and is a slow operation that adds significant latency between sorting runs.

10. User Interface

Our user interface is Command Line (CLI) based.

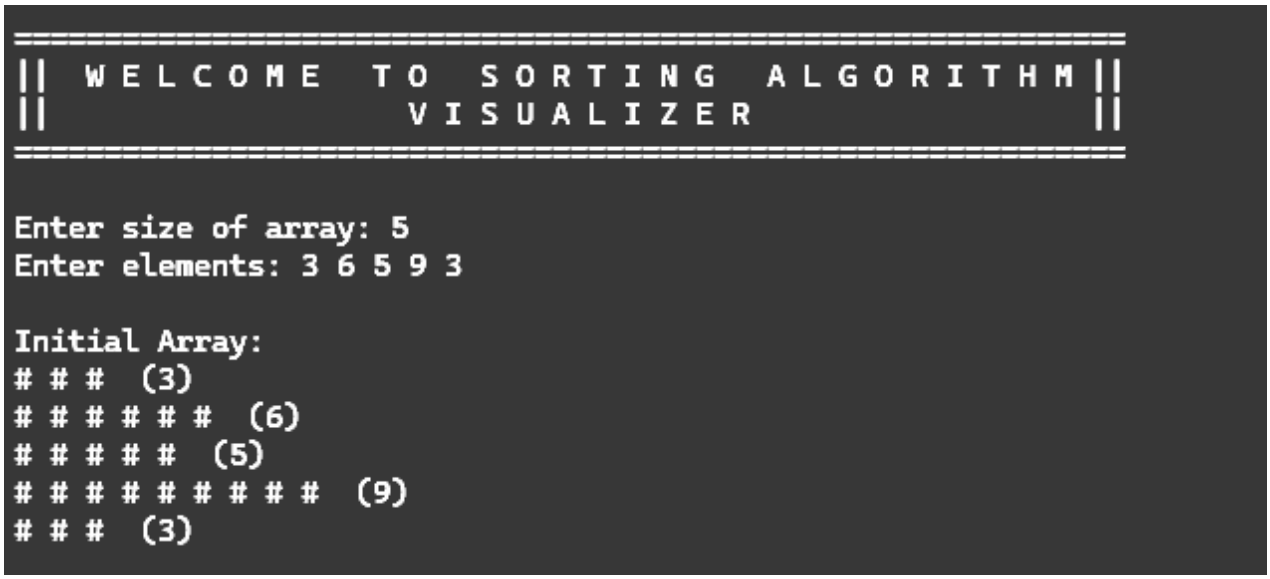
10.1 Interface Overview

The user interface consists of menu that is displayed to the user to choose among the different **sorting algorithms.** The menu asks for the size of array from user and then asks the user to enter array elements. After that it asks the user to choose among the sorting algorithms.

10.2 Screen-by-Screen Documentation

For each major screen or view in your interface:

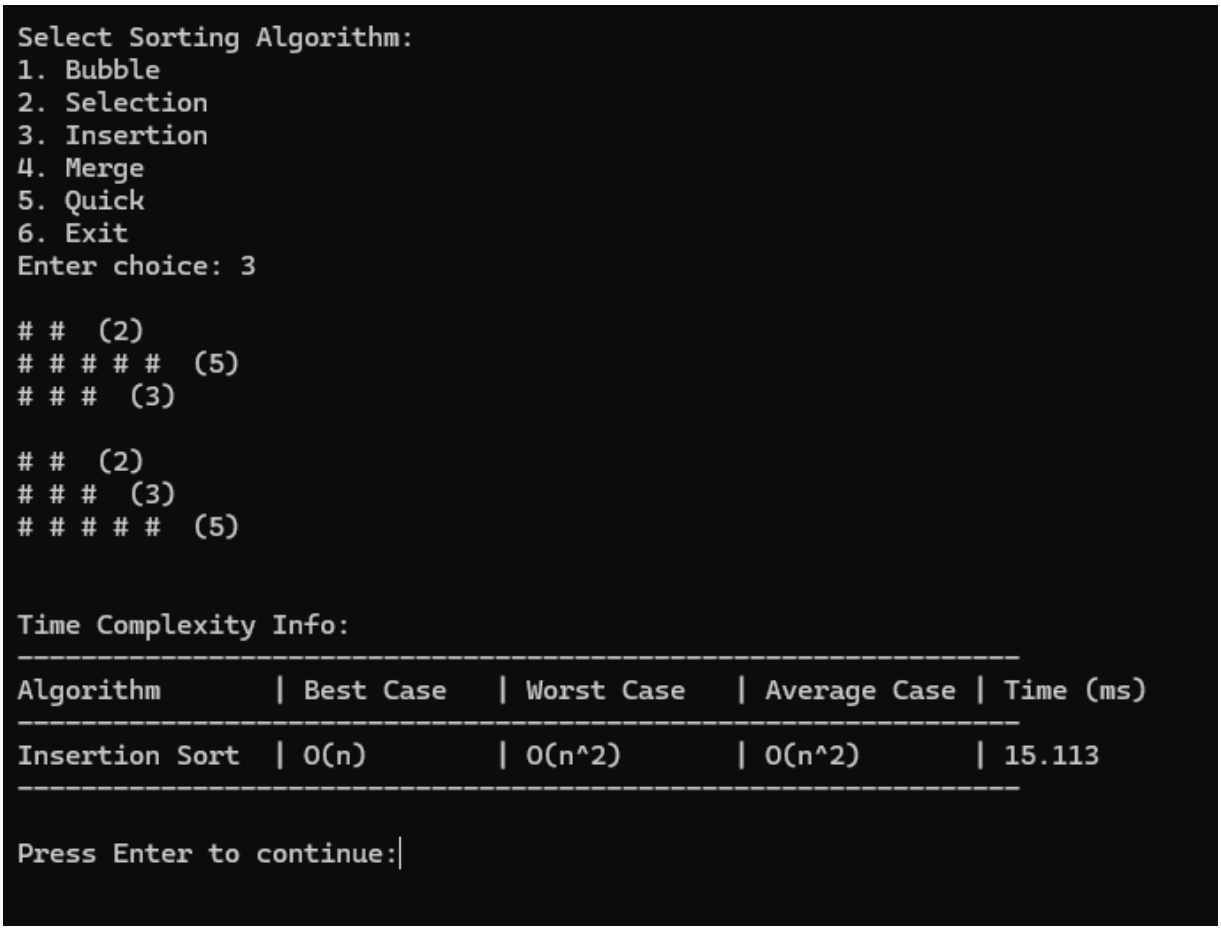
- **Screen Name:**
  MAIN MENU.
- **Screenshot:**



- **Purpose:** The user enters the size of array and array elements.
- **User Actions:** The options are:
  - ✓ Size of array
  - ✓ Elements of array
- **Input Description:** The user enters size of array of data type **i.e. int, float, double etc.**
- **Output Description:** The visualization of the entered data is shown as output.
- **Navigation:** As the user enters elements and output is shown the user is automatically redirected to another screen.

For each major screen or view in your interface:

- **Screen Name:**
  SUB MENY
- **Screenshot:**



- **Purpose:** The menu asks the user to choose among the different sorting algorithms.
- **User Actions:** The options are:
  - ✓ Bubble sort
  - ✓ Selection sort
  - ✓ Insertion Sort
  - ✓ Merge sort
  - ✓ Quick Sort
  - ✓ Exit
- **Input Description:** The user selects the option by entering integer value i.e. 1 to 5.
- **Output Description:** The visualization of that particular sort is shown to the user.
- **Navigation:** The user is asked to "Press any key to continue" and the screen is returned to the main menu.

10.3 Sample Test Cases

- **Test Case Name:** Unordered array of size 6.
- **Input:** Data entered is {7, 6 ,9, 10, 5, 3,}
  Choose sorting algorithm: Bubble sort
- **Expected Output:** The system responded {3, 5, 6, 7, 9, 10}.
- **Actual Output:** The actual was {3, 5, 6, 7, 9, 10}
- **Status:** Pass

11. Future Improvements

The improvements in future will be:

- **In-Place Merge Sort**
  The current merge function allocates new arrays (L and R) on the heap during every recursive call. Using an in-place merging algorithm or a single auxiliary array would drastically reduce memory fragmentation and allocation time.

- **Comparison Mode**
  Allow the user to select two algorithms and run them side-by-side on the same input data to visually compare their efficiency and movement patterns.

- **Hybrid Algorithms**
  Exploring hybrid approaches like Tim sort (used in Python and Java), which combines Insertion Sort and Merge Sort, would demonstrate how real-world libraries optimize for partially sorted data.

12. Conclusion

Key Achievements

The development of the **Sorting Algorithm Visualizer** has been a comprehensive exercise in bridging the gap between theoretical discrete mathematics and practical applications in Computer Science. By transforming the sorting logic into a real-time visual representation, this project serves as both a functional tool and a testament to the importance of algorithmic efficiency.

Learning Outcomes

This project helped us to understand the Chapter 3 of **Discrete Mathematics and its Applications by Kenneth Rosen,** which explains the algorithms and their time complexity.

Challenges Overcome:

- **Recursive Visualization**
  One of the primary hurdles was maintaining visual consistency during recursive calls in Merge Sort. Unlike iterative loops, recursion requires passing the original array size through multiple stack frames to ensure the visualizar function renders the full array correctly at every step.

Reflection

The significance of this project lies in its ability to make the "invisible" mechanics of computer science visible. It highlights that a developer's role is not just to produce a result, but to understand the path taken to reach that result

Final Remarks

This project has demonstrated that mastering discrete mathematics is essential for writing code that is not only functional but optimal. As I conclude this project, I carry forward a heightened competence in C++ and a profound respect for the mathematical foundations that power modern computing.

13. References & Learning Sources

GitHub: https://github.com/DM-Final-Project/Sorting-ALgorithm-Visualizer.git

Video Demo: By Abdul Hannan [https://www.linkedin.com/posts/abdul-hannan-5330093a4_uetlahore-computerscience-computerscience-activity-7413931949519634432-dPiV?utm_source=social_share_send&utm_medium=android_app&rcm=ACoAAGL0_7cBoMtUQGvHNLUJakxD_QktiB1oBlg&utm_campaign=copy_link]

BY Muhammad Salahudin [https://www.linkedin.com/posts/muhammad-salahudin-587275380_uetlahore-computerscience-computerscience-activity-7413931984629944320-Z8Tz?utm_source=share&utm_medium=member_android&rcm=ACoAAF4C8_cBV-MWHhCPIzq5loqssTmVAw2MnYQ]

Textbooks: **Rosen, Discrete Mathematics and Its Applications**, 8th edition

Online Documentation: Geeks for Geeks(https://www.geeksforgeeks.org/) & W3 Schools

(https://www.w3schools.com/)
Tutorials & Guides: No video Tutorials
Academic Papers: No academic papers were read.
Code References: We used **Geeks for Geeks** website for visualization help.
AI Assistance: The AI assistance was from Google's Gemini and GitHub Copilot.