|  |  |  |
| --- | --- | --- |
| REPUBLIQUE DU CAMEROUN  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  MINISTERE DE L’ENSEIGNEMENT SUPPERIEUR  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  UNIVERSITE SAINT JEAN |  | REPUBLIC OF CAMEROON  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  MINISTRY OF THE HIGHER EDUCATION  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  SAINT JEAN UNIVERSITY |



***THEME :***

**visualization tool that visualizes various sorting algorithms**

**ALGORITHM PROJECT**

UE Code : ING 3166

**GROUP 2:**

|  |  |
| --- | --- |
| KAMDEM GUY JORDAN | 2122I111 |
|  |  |

**SEMESTER 1**

**CLASS**: INGE 3 ISI ANGLOPHONE

**SUPERVISOR:** Engr. KOUGANG Guy Rostand

School Year

2023-2024

Table of Contents

[Introduction 3](#_Toc155464805)

[Problem Statement: 3](#_Toc155464806)

[Objectives 3](#_Toc155464807)

[Main Objectives: 3](#_Toc155464808)

[Specific Objectives: 3](#_Toc155464809)

[Methodology used to get the Solution 4](#_Toc155464810)

[Technical specification of our solution 5](#_Toc155464811)

# Introduction

Sorting algorithms are fundamental in computer science, crucial for arranging data in a systematic order. This report aims to analyze and visualize various sorting algorithms, exploring their efficiency, strengths, and weaknesses. Through this exploration, we seek to provide a comprehensive understanding and comparative evaluation of these algorithms.

# Problem Statement:

The complexity and diversity of sorting algorithms pose challenges in comprehending their functionality and performance variations. Traditional textual descriptions can often be convoluted, making it challenging for learners and professionals alike to grasp the nuances of these algorithms effectively. Thus, there is a pressing need for intuitive visualizations that can simplify the understanding of sorting algorithms.

# Objectives

The Main objective :

The main objective of this project is to develop an interactive visualization tool that provides a clear and engaging way to understand different sorting algorithms. Through animation and user interaction, the tool aims to:

* Demonstrate the step-by-step execution of various sorting algorithms.
* Highlight key comparisons, swaps, and data movements within the algorithms.
* Allow users to experiment with different algorithms and data sizes.
* Provide visual feedback on the performance and efficiency of each algorithm.

Specific Objectives:

* Implement visualizations for several common sorting algorithms (e.g., Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort).
* Enable user selection of algorithms and data sizes.
* Design an intuitive interface for easy interaction and control.
* Incorporate performance metrics and visuals to compare different algorithms.
* Provide informative descriptions and explanations for each algorithm.

**Domain of Application:**

This visualization tool can be used in various contexts, including:

* Computer Science education: As a learning tool for students to understand sorting algorithms in a more engaging and interactive way.
* Software development: As a reference tool for developers to compare and analyze different sorting algorithms for specific applications.
* General audience: As a fun and interactive way to learn about how computers work behind the scenes.

### Methodology:

#### **Selection of Sorting Algorithms:**

* Selection Criteria: We will select a range of sorting algorithms, including but not limited to Bubble Sort, Selection Sort, Insertion Sort, and Merge Sort. The selection will be based on their popularity, diversity in approach, and relevance in practical applications.
* Implementation: Algorithms will be implemented using commonly used programming languages Python,

#### **Visualization Tool Development:**

* Development Platform: Utilizing modern visualization libraries Matplotlib
* Design Considerations: The tool will aim for user-friendliness, enabling users to observe step-by-step execution, visualize data movements, and comprehend algorithmic behavior.

# 

# Technical specification of our solution

1. Necessary Libraries:

* NumPy: Provides essential tools for numerical computations and array manipulations.

2. Importing NumPy:

With : import numpy as np

3. Defining the System of Equations:

* Represent the coefficients of the variables in a NumPy matrix A.
* Store the constants on the right-hand side in a NumPy vector b.

# Example:

A = np.array([[2, 1, -1], [3, 2, 1], [1, 1, 1]])

b = np.array([5, 8, 3])

4. Applying RREF:

Option 1: Using numpy.linalg.solve:

* This function internally applies RREF to solve the system efficiently.

Python

x = np.linalg.solve(A, b)

print("Solution:", x)

Option 2: Manual RREF Implementation:

* Use NumPy's array manipulation capabilities to perform RREF steps:

Python

# Obtain RREF using QR decomposition

rref = np.linalg.qr(A)[0]

# Extract solutions from RREF

x = np.where(rref[:, 0] == 1)[0][0]

y = np.where(rref[:, 1] == 1)[0][0]

z = np.where(rref[:, 2] == 1)[0][0]

print("Solution:", np.array([b[x], b[y], b[z]]))

Key Considerations:

* Accuracy: Ensure numerical accuracy during computations, especially for large or ill-conditioned systems.
* Computational Efficiency: For large systems, consider optimized algorithms like LU decomposition or iterative methods for better performance.
* Error Handling: Incorporate error handling mechanisms to address potential issues like singular matrices or inconsistent systems.
* Alternative Methods: Explore alternative methods, such as Gaussian elimination with back-substitution, depending on specific problem requirements.