PROJECT PROPOSAL

NAME : NOORULAIN(14663)

DEPARTMENT :BSCS

SEMESTER : 3RD (A)

PROJECT NAME: SORTING ALGORITHM VISUALIZER

SUBJECT: DATA STRUCTURE AND ALGORITHM

SUBMITTED TO : SIR JAMAL ABDUL AHAD

YEAR : 2024 - 2025

PROPOSAL

INTRODUCTION:

Sorting algorithms are among the most fundamental concepts in computer science and software development. These algorithms are crucial for organizing data and ensuring optimal performance in various applications. However, understanding how these algorithms work, especially with different data sets, can be challenging without a visual aid. This project proposes to create an Algorithm Visualizer that focuses on three well-known sorting algorithms—Insertion Sort, Bubble Sort, and Selection Sort—to help users better understand their mechanics. The visualizer will be developed in Python using the Flet library, which allows for the creation of interactive web applications. This tool will allow users to input arrays of different sizes, generate random arrays, and visualize the sorting process. The user can choose to see the sorting process in either auto-sort mode or step-wise mode. Additionally, the visualizer will display a graph showing array values and their corresponding time complexity, providing a deeper insight into how each algorithm performs based on the array size.

ABSTRACT:

This project aims to develop an Algorithm Visualizer that demonstrates the inner workings of Insertion Sort, Bubble Sort, and Selection Sort using the Python programming language and the Flet library. The visualizer will allow users to input arrays manually, or generate random arrays of different sizes. Once an array is selected, users can choose from the three sorting algorithms to visualize the sorting process in real time. The tool will feature two modes for visualization: auto-sort mode, where the sorting happens automatically, and step-wise mode, where the user can proceed through each sorting step manually. The visualizer will also provide a graph displaying the array elements, and the time complexity of the sorting process will be displayed dynamically, allowing users to analyze how each algorithm performs with various input sizes. This project is aimed at helping students, developers, and educators understand sorting algorithms in a clear and interactive manner.

PROJECT OVERVIEW:

The goal of this project is to create an interactive Algorithm Visualizer that helps users understand and visualize the working of three popular sorting algorithms: Insertion Sort, Bubble Sort, and Selection Sort. This tool will allow users to input an array, choose any of the three sorting algorithms, and visualize the sorting process step-by-step. Additionally, the tool will display time complexity information and generate performance graphs for each sorting algorithm.

KEY FEATURES:

Array Size Input:

- The user will be able to specify the size of the array that needs to be sorted.
- Array sizes can range from small to large, depending on the user's requirement.

Input Array Options:

- **Manual Array Input:** The user can manually enter the elements of the array.
- **Random Array Generator:** The user can choose an option to generate a random array of integers.

SORTING ALGORITHM SELECTION:

The user will be able to choose from three sorting algorithms:

- Insertion Sort
- Bubble Sort
- Selection Sort

VISUALIZATION OPTIONS:

- **Auto Sorting:** The user can choose to let the algorithm run automatically, displaying each step of the sorting process.

- **Step-wise Sorting**: The user can step through the sorting process manually, observing each change in the array as the algorithm progresses.

GRAPHICAL VISUALIZATION:

- A bar graph will be displayed to visualize the array at each step of the sorting process. Each bar will represent an element in the array, and the height of the bar will correspond to the value of that element.
- The graph will dynamically update to show the state of the array after each step of the sorting algorithm.

TIME COMPLEXITY DISPLAY:

After the algorithm finishes sorting, the time complexity for that specific input array will be displayed.

The time complexity will vary based on the algorithm chosen:

- **Insertion Sort:** O(n^2) in the worst case, O(n) in the best case (when the array is already sorted).
- **Bubble Sort:** O(n^2) in the worst case and average case.
- **Selection Sort:** O(n^2) in the worst case and average case.

PERFORMANCE GRAPH:

- A graph will display the time taken to complete the sorting for different input sizes.
- The graph will show a comparison of the three algorithms' performance over various array sizes.

INTERACTIVITY:

- Users will be able to pause, resume, and restart the sorting process.
- The user can choose to run the sorting algorithms with different array sizes to see how each algorithm performs as the array size increases.

- The system will allow users to compare how different algorithms perform with various input arrays (random, sorted, or reverse sorted).

USER INTERFACE (UI) DESIGN:

- **Main Page:**
    - Options to select array size and input method (manual or random).
    - Dropdown to choose a sorting algorithm (Insertion, Bubble, or Selection).
    - Buttons to start sorting, pause, and step through sorting.
    - A large area to display the visual array as a bar graph.
- **Array Input Section:**
    - Text box for manual input of array elements or a button to generate a random array.
    - Display for the generated/random array.
- **Sorting Controls:**
    - Start button to run the sorting algorithm.
    - Pause button to stop the sorting process.
    - Step button to proceed through the sorting algorithm step-by-step.
- **Graph Section:**
    - Bar graph displaying the array's state at each step.
    - X-axis representing the array index, Y-axis representing the element value.
- **Complexity and Performance Section:**
    - Text area displaying the time complexity (for the specific algorithm and array input).
    - Performance graph showing time vs. array size for each algorithm.

ALGORITHM DETAILS:

1. **Insertion Sort:**
    - Insertion sort works by iterating over the array, picking one element at a time, and inserting it into the correct position among the already sorted elements to its left.
2. **Bubble Sort:**
    - Bubble sort works by repeatedly comparing adjacent elements and swapping them if they are in the wrong order. This process is repeated until the array is sorted.
3. **Selection Sort:**
    - Selection sort works by dividing the array into two parts: the sorted part and the unsorted part. It repeatedly selects the smallest element from the unsorted part and swaps it with the first unsorted element, thus growing the sorted part step-by-step.

TECHNICAL REQUIREMENTS:

Frontend:

- **Python :**
    1. For designing the user interface and implementing the interactivity.
    2. For visualizing the array as a bar graph.

Backend (Optional):

- Python (if needed for handling user input and computation behind the scenes).

Algorithm Implementation:

- Implement each sorting algorithm (Insertion Sort, Bubble Sort, Selection Sort) in JavaScript or Python (depending on the tech stack).

PROJECT TIMELINE:

- **Week 1-2:** Research and gather requirements. Design the user interface and backend structure (if needed).
- **Week 3-4:** Implement the sorting algorithms and visualization (bar graph and step-by-step view).
- **Week 5**: Implement time complexity display and performance graph.
- **Week 6**: Final testing, bug fixes, and documentation.
- **Week 7:** Project completion, user feedback, and final adjustments.

BENEFITS OF THE PROJECT:

1. **Improved Understanding of Sorting Algorithms:**
    - The visualizer will provide an intuitive and interactive way to observe how sorting algorithms work. By watching the sorting process step by step, users will gain a clearer understanding of how each algorithm organizes the array.
2. **Support for Multiple Sorting Algorithms:**
    - Users can compare Insertion Sort, Bubble Sort, and Selection Sort side by side and analyze how each algorithm behaves differently under various conditions.
3. **Customization Options:**
    - The tool allows users to customize the array input by providing options to manually enter array values or generate random arrays of different sizes. This flexibility will help users observe how different data sets impact the sorting process.
4. **Real-Time Visualization and Interaction:**
    - The option for step-wise sorting enables users to pause and proceed through each step of the algorithm, helping them learn in a more controlled, deliberate manner.

- The auto-sort mode provides a seamless experience for users who prefer to watch the sorting process unfold automatically.

5. **Performance Insights:**

    - By displaying the time complexity of each algorithm, users will be able to see how the performance of the sorting algorithms varies with the size of the array. This will provide insights into which algorithm is more efficient for a given task.

    - Graphical representation of the array will visually show how the values change during each sorting step, making the sorting process easy to follow.

6. **Educational Tool:**

    - The visualizer serves as an excellent educational tool for students learning algorithms. It can be used in classrooms, online courses, or by individuals seeking to understand the inner workings of sorting algorithms.

    - Teachers and instructors can use the visualizer to illustrate how different sorting algorithms work in real-time, facilitating better learning.

7. **Comparative Analysis:**

    - The tool allows for the comparison of different sorting algorithms in terms of their execution speed and step-by-step processes, helping users choose the right algorithm based on the problem's requirements.

8. **Efficiency in Software Design:**

    - By helping users understand how different sorting algorithms perform, developers can apply this knowledge to optimize their code, select more efficient algorithms, and improve system performance.

9. **Accessibility and Ease of Use:**

- Developed with Python and the Flet library, the visualizer provides a simple, web-based user interface that is easy to use, accessible, and intuitive for all users, regardless of technical expertise.

CONCLUSION:

The Algorithm Visualizer project will provide an interactive and visual way to understand the sorting algorithms and their performance. By allowing users to visualize the sorting process step-by-step, compare different algorithms, and analyze the time complexity, this project will greatly enhance the learning experience for anyone studying algorithms and their behaviors.
