**Abbottabad University of Science and Technology**

**Python Sorting Algorithm Compiler using Flet Framework**

**FOR**

**< Running App Analyzer >**

**By**

**Maryum Aftab 14658**

**Supervisor**

**Sir Jamal Abdul Ahad**

Table of Contents

[1. Abstract 2](#_Toc186714399)

[1.1. Objective 2](#_Toc186714400)

[1.2. Features 2](#_Toc186714401)

[Predefined Algorithm Library: 2](#_Toc186714402)

[User Interface Components: 3](#_Toc186714403)

[Performance Insights: 3](#_Toc186714404)

[Dynamic Interaction: 3](#_Toc186714405)

[Technical Specifications 3](#_Toc186714406)

[Technology Stack: 3](#_Toc186714407)

[Frontend: 3](#_Toc186714408)

[Backend: 3](#_Toc186714409)

[Key Functionalities: 3](#_Toc186714410)

[Dropdown Menu: 3](#_Toc186714411)

[Code Execution: 3](#_Toc186714412)

[Dynamic Output: 3](#_Toc186714413)

[Clear Button: 3](#_Toc186714414)

[Algorithms: 4](#_Toc186714415)

[Performance Metrics: 4](#_Toc186714416)

[Design and Implementation 4](#_Toc186714417)

[User Interface: 4](#_Toc186714418)

[Application Flow: 4](#_Toc186714419)

[Usage Workflow 4](#_Toc186714420)

[Code Structure 5](#_Toc186714421)

[Algorithm Dictionary: 5](#_Toc186714422)

[Main Components: 5](#_Toc186714423)

[Dropdown: 5](#_Toc186714424)

[Text Field: 5](#_Toc186714425)

[Button: 5](#_Toc186714426)

[Logic Highlights: 5](#_Toc186714427)

[Strengths and Limitations 5](#_Toc186714428)

[Strengths: 5](#_Toc186714429)

[Limitations: 6](#_Toc186714430)

[Future Enhancements 6](#_Toc186714431)

[Data flow diagram(DFD) 6](#_Toc186714432)

[Conclusion 7](#_Toc186714433)

# 1. Abstract

The "Python Sorting Algorithm Compiler" is an interactive desktop application developed using the Flet framework in Python. It serves as a learning and execution tool for studying sorting and searching algorithms. The application provides users with predefined algorithms, an editor for reviewing and modifying code, and a console to execute the code and view results, including runtime and Big-O complexity.

# 1.1. Objective

The primary objective of this project is to create an educational tool to help users:

1. Understand the implementation of common sorting and searching algorithms.
2. Experiment with Python code execution in real time.
3. Learn about Big-O complexities and execution times of algorithms.

# 1.2. Features

# Predefined Algorithm Library:

* + Merge Sort
  + Insertion Sort
  + Selection Sort
  + Bubble Sort
  + Quick Sort
  + Linear Search
  + Binary Search

## User Interface Components:

* + Dropdown for selecting algorithms.
  + Text field for displaying and editing Python code.
  + Output console for displaying execution results, runtime, and Big-O complexity.
  + Buttons for running code, clearing fields, and switching algorithms.

## Performance Insights:

* + Displays algorithm runtime in seconds.
  + Provides Big-O complexity analysis.

## Dynamic Interaction:

* + Code editor supports modifications and re-execution of algorithms.

# Technical Specifications

## Technology Stack:

Frontend: Flet framework (Python-based).

Backend: Python’s native libraries (time, io, sys).

## Key Functionalities:

Dropdown Menu: Provides a selection mechanism for algorithms.

Code Execution: Uses Python's exec () function for real-time code execution.

Dynamic Output: Captures and displays stdout for print statements.

Clear Button: Resets both code and output fields.

## Algorithms:

* + Sorting algorithms (Merge, Quick, Bubble, Selection, Insertion).
  + Searching algorithms (Linear, Binary).

## Performance Metrics:

* + Calculates and displays runtime for each execution.
  + Compares user-modified code to predefined templates to determine Big-O complexity.

# Design and Implementation

## User Interface:

* + Designed with responsiveness to support various screen sizes.
  + Components include dropdowns, text fields, buttons, and labeled sections.
  + Visual aesthetics:
    - Light theme with a background color #6A1E55.
    - Buttons styled with rounded corners and a gradient color scheme.

## Application Flow:

* + User selects an algorithm via the dropdown.
  + Corresponding Python code is loaded into the editor.
  + User executes the code to view the output and performance metrics.
  + Clear functionality allows resetting the workspace for new executions.

# Usage Workflow

1. Launch the application.
2. Select an algorithm from the dropdown menu.
3. View or edit the algorithm's code in the editor.
4. Click "Run" to execute the algorithm.
5. Observe the sorted/search results, Big-O complexity, and runtime in the output console.
6. Use "Clear" to reset for another session.

# Code Structure

## Algorithm Dictionary:

* + Stores predefined algorithms and their Big-O complexities.
  + Example: {"Merge Sort": {"code": "def merge\_sort...", "big\_o": "O(n log n)"}}.

## Main Components:

Dropdown: Allows algorithm selection.

Text Field: Code editor and output display.

Button: Executes code and clears fields.

## Logic Highlights:

* + Redirects stdout to capture execution logs.
  + Dynamically calculates runtime for each algorithm.
  + Validates user-modified code against predefined templates

# Strengths and Limitations

## Strengths:

* Interactive and educational tool for algorithm learning.
* Clear and minimalistic UI.
* Real-time insights into algorithm performance.

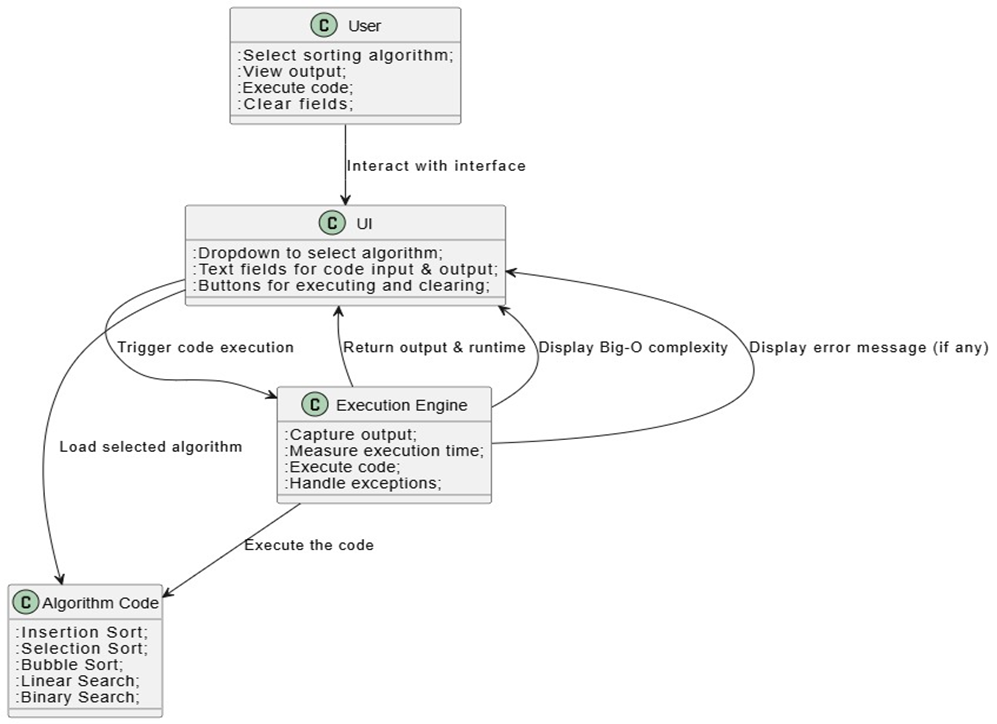
## Limitations:

* Execution restricted to algorithms defined in the application.
* No error-handling for advanced Python syntax.

# Future Enhancements

1. Add support for visualizing algorithm execution steps.
2. Include more algorithms (e.g., Heap Sort, Radix Sort).
3. Enable loading user-defined Python scripts.
4. Implement a dark mode for better usability.

# Data flow diagram(DFD)



# 

# Conclusion

The "Python Sorting Algorithm Compiler" effectively bridges the gap between theoretical understanding and practical implementation of algorithms. Its user-friendly interface and real-time feedback make it an excellent tool for students, educators, and Python enthusiasts. This project demonstrates the power of the Flet framework for developing educational applications with minimal overhead.

