OBJECT-ORIENTED PROGRAMMING

SORTING VISUALIZATION

June 26, 2024

Lecturer: Dr. Tran The Hung

Class ID: 147839

Group 3

Participants:
Bui Xuan Son - 20226065: GUI and Selection Sort
Pham Thanh Nam - 20225989: Merge Sort
Luan Quang Minh - 20225985: Quick Sort
Vu Duc Manh - 20226054: Shell Sort

Hanoi University of Science and Technology

Abstract

To complete this project, we have received invaluable assistance from everyone involved.

First and foremost, we would like to extend our heartfelt gratitude to Dr. Tran The Hung, our main lecturer for the Object-oriented Programming course. We have gained extensive knowledge, particularly in OOP techniques and the Java language. These skills are not only beneficial for the current project but will also be invaluable in our future endeavors.

Additionally, we sincerely appreciate our classmates who dedicated their time and effort to assist us. Their support and knowledge sharing have helped us tackle challenging questions and enhance our report, creating a positive learning environment that motivated us to overcome difficulties.

Thanks to everyone's contributions and support, the project has successfully overcome challenges and achieved favorable outcomes. We are proud of what we have accomplished and believe that this project will continue to deliver value and provide an excellent user experience.

Once again, we would like to express our heartfelt thanks and gratitude to all those who have contributed to the success of this project.

CONTENTS

1 INTRODUCTION

1.1 Project Overview

This project aims to visualize the sorting algorithms: Selection Sort, Merge Sort, Shell Sort, and Quick Sort. The goal is to provide a clear and interactive representation of these algorithms to enhance understanding of their operations.

1.2 Objectives

- Develop a graphical user interface (GUI) to visualize sorting algorithms.
- Implement and visualize Selection Sort, Merge Sort, Shell Sort, and Quick Sort algorithms.
- Provide a detailed explanation and step-by-step visualization of each sorting algorithm.

2 REQUIREMENT ANALYSIS

In this project, the following specific requirements are identified for developing a program to visualize four types of sorting algorithms:

1. Proper illustration for each sorting algorithms.
    - Algorithms work exactly
    - Understandable visualization
2. Fulfill all assigned tasks about GUI and workflow.
    - User must select a sort type in order to start the demonstration
    - Help menu show the basic usage and aim of the program
    - Quit exits the program. Remember to ask for confirmation
3. Adherence to principles in Object-Oriented Programming.
    - Encapsulation: Bundling data and methods that operate on the data into a single unit (class), and restricting access to some of the object's components.
    - Abstraction: Hiding complex implementation details and showing only the essential features of an object.

- Inheritance: Allowing a class (subclass or derived class) to inherit properties and behaviors from another class (superclass or base class).
- Polymorphism: The ability to use the same interface for different data types or classes, enabling flexibility and extensibility in code.

3 PROJECT DESIGN

3.1 File System Structure

```
SortingVisualizer/
|-- .vscode/
|-- bin/
|-- src/
|   |-- algorithms/
|   |   |-- MergeSort.java
|   |   |-- SelectionSort.java
|   |   |-- ShellSort.java
|   |   |-- SortAbstraction.java
|   |-- utils/
|   |   |-- ArrayGenerator.java
|   |-- views/
|       |-- Help.java
|       |-- Screen.java
|       |-- Visualizer.java
|-- Main.java
|-- README.md
|-- run.bat
|-- run.sh
```

3.2 General class diagram: Illustrate the classes and their relationships

- **Main Class:** Entry point initializing the program. Extends ‘javax.swing.JFrame‘ to create the application window and adds a confirmation dialog when closing the window.
- **views Package:** Includes components for the user interface.

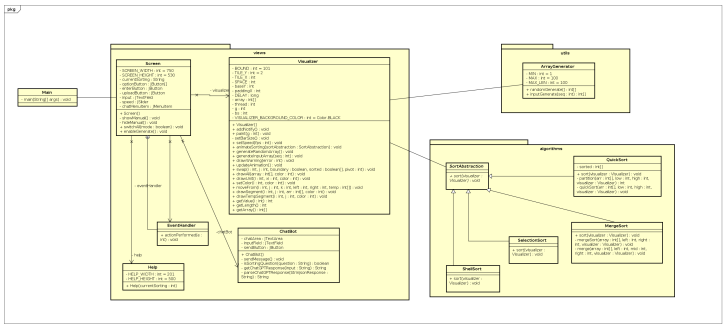


Figure 1: Class Diagram

  - **Help Class:** Extends ‘javax.swing.JLabel‘, displays guidelines for users in each sorting animation.
  - **Screen Class:** Extends ‘javax.swing.JPanel‘, serves as a wrapper including elements (controls) like ‘JButton‘, ‘JPanel‘, ‘JTextField‘, ‘JSlider‘, etc.
  - **Visualizer Class:** Extends ‘java.awt.Canvas‘, displays sorting animation.
  - **Chatbot Class:** Extends ’javax.swing.JPanel’, box chat between user and customized chatbot deployed by OpenAI API

- **utils Package:** Includes utility classes.
  - **ArrayGenerator Class:** Helper class that can generate random arrays or transform input sequences into arrays of numbers.

- **algorithms Package:** Includes classes performing sorting processes, using methods of the ‘Visualizer‘ class for visualizing tasks.
  - **SortAbstraction Class:** Abstract class, generalizes the sorting classes.
  - **SelectionSort, MergeSort, ShellSort Classes:** Extend ‘SortAbstraction‘ to implement the ‘sort()‘ method, setting up logic for animation in the ‘Visualizer‘ class.

3.3 General use-case diagram: Illustrate the use-cases and explain flows of program

The program mainly focuses on visualization tasks so that the complexity of workflow is unremarkable.

- Generating an array can do in 2 ways: randomly or input

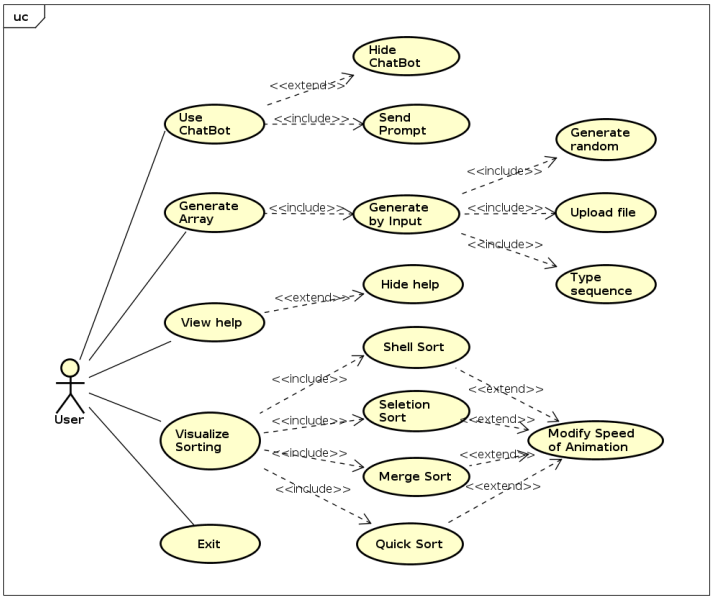


Figure 2: Use-case Diagram

- Visualizing algorithms: 4 choices, can modify speed of animations
- View help: display help for each sorting visualization
- Using custom chatbot: answer sorting-related questions

4 User Interface Accomplished

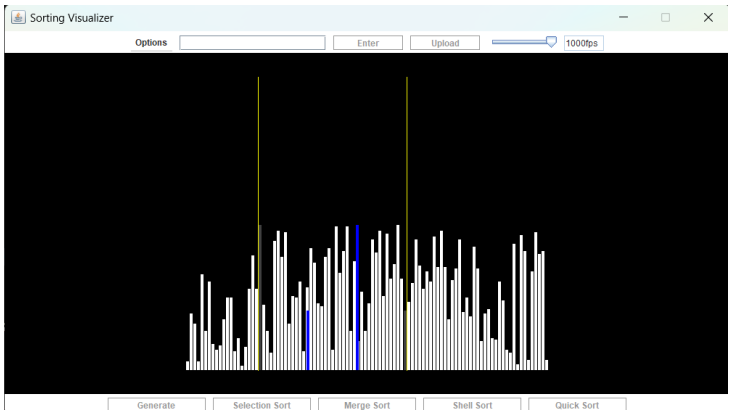
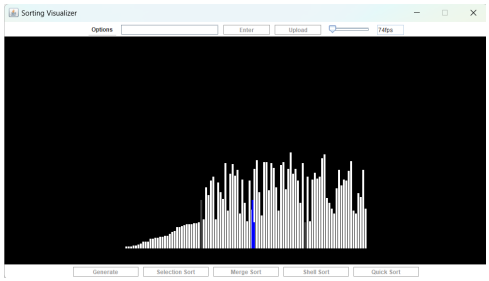
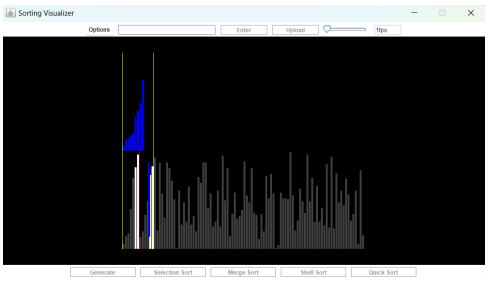


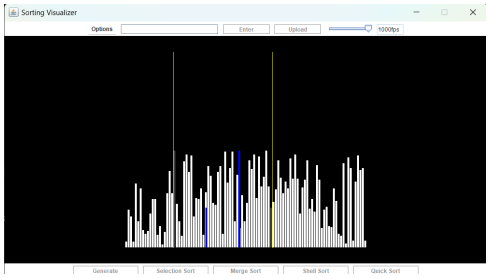
Figure 3: General UI



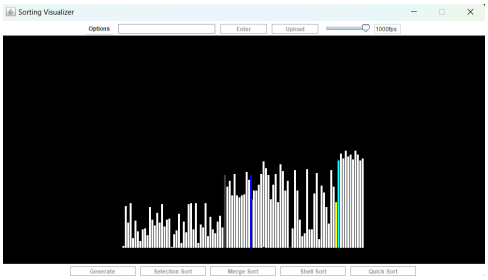
(a) Selection Sort



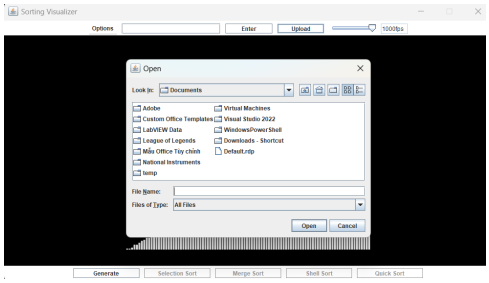
(b) Merge Sort



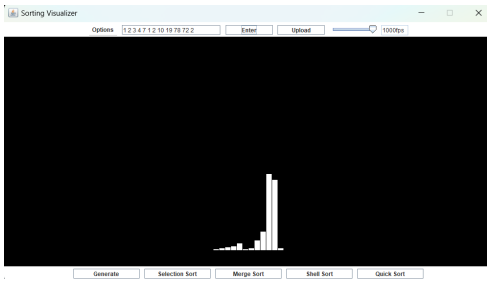
(c) Selection Sort



(d) Merge Sort



(e) Upload array from a text file



(f) Input array from Keyboard

5 ALGORITHMS

5.1 Selection Sort

5.1.1 Step-by-step

1. Initialize ‘i = 1‘.
2. Find the minimum ‘a[min]‘ in the array from ‘a[i]‘ to ‘a[n]‘.
3. Swap ‘a[min]‘ and ‘a[i]‘.
4. If $i \leq n-1$, increment ‘i‘ and repeat step 2. Otherwise, stop (n-1 elements sorted).

5.1.2 Time Complexity

- Best Case: $O(n^2)$

- Average Case: $O(n^2)$
- Worst Case: $O(n^2)$

5.2 Merge Sort

5.2.1 Step-by-step

1. Divide by finding the midpoint between the first and last elements.
2. Conquer by recursively sorting the subarrays.
3. Combine by merging the two sorted subarrays back into a single sorted subarray.

5.2.2 Time Complexity

- Best Case: $O(n \log n)$
- Average Case: $O(n \log n)$
- Worst Case: $O(n \log n)$

5.3 Shell Sort

5.3.1 Step-by-step

1. Initialize the gap size, say ‘h‘.
2. Divide the list into smaller sub-parts, each having intervals equal to ‘h‘.
3. Sort these sub-lists using insertion sort.
4. Repeat until the list is sorted.
5. Print the sorted list.

5.3.2 Time Complexity

- Best Case: $\Omega(n \log n)$
- Average Case: $O(n \log n)$ to $O(n^{1.25})$
- Worst Case: $O(n^2)$

5.4 Quick Sort

5.4.1 Step-by-step

1. Choose a pivot element from the array.
2. Partition the other elements into two sub-arrays based on the pivot.
3. Recursively apply quicksort to the left and right sub-arrays.
4. The recursion terminates when the sub-array has 0 or 1 elements.

5.4.2 Time Complexity

- Best Case: $\Omega(N \log N)$
- Average Case: $\Theta(N \log N)$
- Worst Case: $O(n^2)$

6 CONCLUSION

This project successfully visualized the Selection Sort, Merge Sort, Shell Sort, and Quick Sort algorithms, providing an educational tool to understand these sorting methods. The implementation, combined with an intuitive user interface, demonstrates the effectiveness of visual learning in comprehending complex algorithms.

REFERENCES

- "Introduction to Algorithms" by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein.
- "Algorithms, Part I" by Robert Sedgewick and Kevin Wayne.