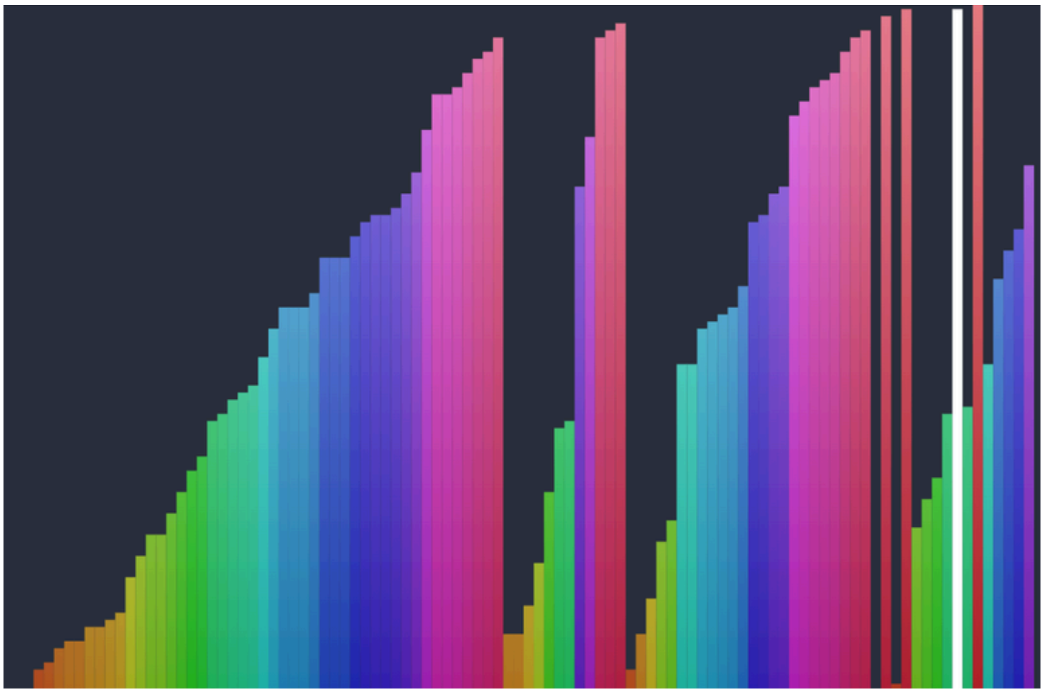
cs1004-OBJECT ORIENTED PROGRAMMING

FINAL PROJECT ALGO-VIZ



MUHAMMAD OBAIDULLAH
24L-0509

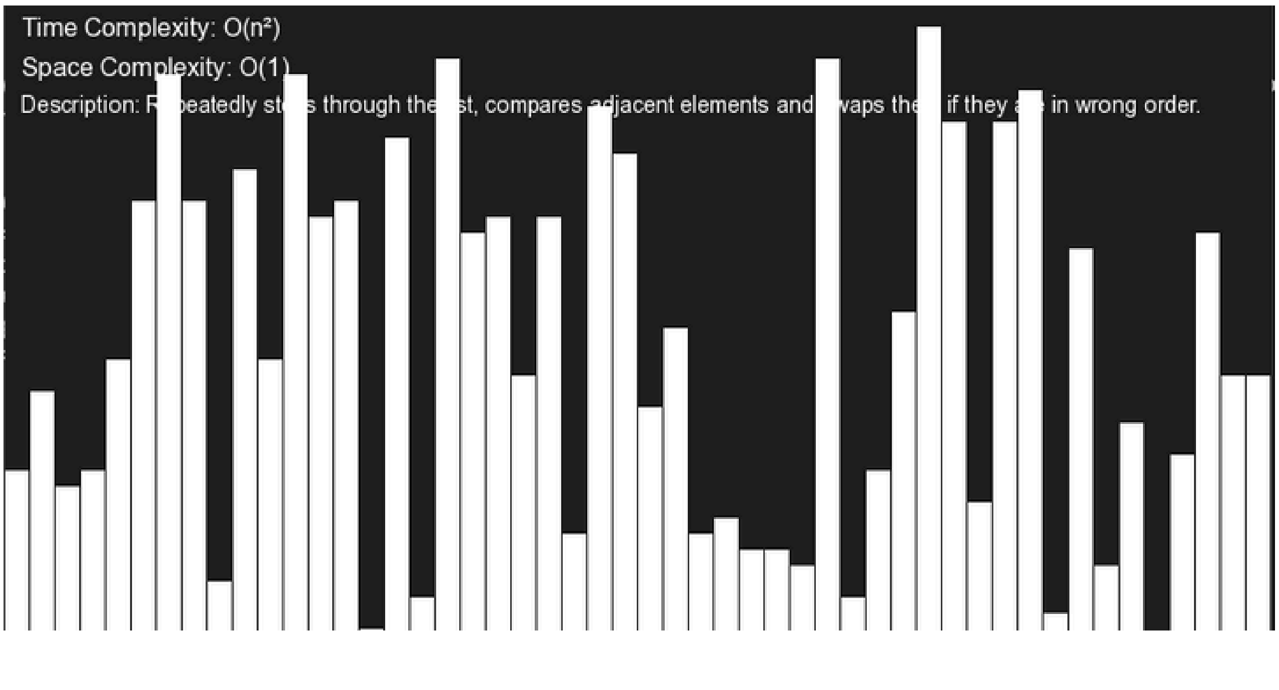
Table of Contents

Abstract

AlgoViz, the title name for my project Algorithm Visualizer is a desktop application created in C++ and SFML (Simple Fast Multimedia Library) that allows the user to visually see the the implementation of basic sorting algorithms which include but are not limited to Bubble Sort, Insertion Sort and selection sort.

The purpose of AlgoViz is to provide a visual and interactive representation of the fundamental algorithms, making it easier for students and beginners to understand how these algorithms work step by step. By transforming abstract logic into animated visuals, AlgoViz aims to support learning through visualization, especially for visual or kinesthetic learners and make algorithm analysis more intuitive, allowing users to observe performance characteristics such as comparisons, swaps and time complexities.

AlgoViz utilizes the Core Object Oriented principles to ensure a modular, extensible, and maintainable codebase. Encapsulation is used to ensure abstraction and hide the internal logic for the methods and pure virtual functions have been utilized to ensure run time polymorphism. Furthermore, inheritance is used to share the common characteristics of the base class within the derived class.



Time Complexity: O(n²)
Space Complexity: O(1)
Description: R peatedly st s through the st, compares adjacent elements and waps the if they a in wrong order.

Introduction

Problem Statement

Understanding the complexities of algorithms has been a challenging task for students, when the concepts are hard to visualize and abstract. Traditional teaching methods often fail to emphasize on the understanding necessary to grasp complex algorithmic behaviors, leading to lower self confidence and confusion. There is a clear need for an approach that is interactive and visual and bridges the gap between theoretical knowledge and practical understanding.

Objectives:

1) Visualize Algorithmic Processes:
Developing step by step animations for common algorithms like sorting to enhance conceptual clarity.

2) Support interactivity:
Allows user to interact with the program by choosing different forms of the flow of execution.

3) Promote Interactive Learning:
Engage users with hand-on-experimentation and observe real time changes in the behaviors.

4) Aid Educators and Institutions:
Provide a classroom-friendly tool that instructions can use to demonstrate algorithmic behaviors during lectures.

Motivations:

1) Improve Learning Outcomes:
This project aims to bridge the gap of the difficulty of understanding complex algorithms.

2) Make Algorithms Approachable:
Breaking down the algorithms visually can reduce the fear and the anxiety associated with the complex topics in computer science.

3) Encourage Curiosity and Exploration:
A visual platform can spark curiosity, encouraging students to dig deeper into how and why algorithms work.

4) Align with Model Pedagogical Trends:
This project aligns with the modern teaching methods and the ones that are taught at top tier academic institutions such as Harvard and Stanford.

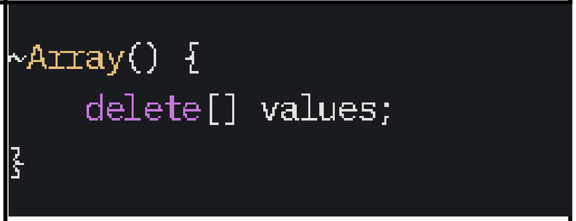
OOP CONCEPTS USED

The project effectively employs OOP principles to create a modular, reusable code that allows us to maintain the sorting visualizer. Encapsulation ensures data integrity in the array and Visualizer classes. Inheritance and polymorphism are used to enable the flexible use of multiple sorting algorithms through the sorter base class. Abstraction allows us to simplify the logic for interaction with complex sorting mechanisms. Composition structures the application by integrating Array, Sorter and Visualizer within the Controller. Therefore, these practices allow us to produce a well organized system allowing us to visualize the sorting algorithms and having the extensibility for adding new algorithms. Here is a detailed breakdown of the OOP concepts involved:

01. Encapsulation

Encapsulation involves binding data and methods that operate on that data within a single unit (class) while restricting direct access to some components to ensure data integrity.

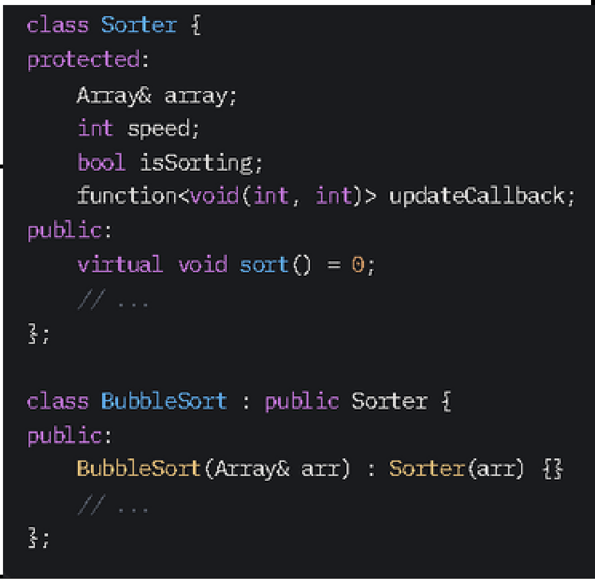
EXAMPLES IN THE CODE:

ARRAY CLASS	VISUALIZER CLASS	MEMORY MANAGEMENT
Dynamic array values and its size is encapsulated as private members, preventing direct external access	Encapsulates SFML window properties, the array object and sorting state variables as private members	The array class uses a destructor (~Array) to deallocate the dynamic memory is a key aspect of encapsulation
Public methods like randomize(), swap(), get_value(), and set provide controlled access to manipulate the array data	Public methods like startSort() and draw() provide controlled interaction with these resources	 `~Array() { delete[] values; }`

02. Inheritance

Inheritance allows the derived classes to inherit the methods from the base classes, prompting reusability of the code and ensuring that there is a hierarchical relationship.

Example in the Code:

Sorter Base Class	Derived Classes	Example Code
The sorter class serves as an abstract base class.	The derived classes are the Bubble Sort, Selection Sort, Insertion Sort.	
The sorter class defines the common attributes and the methods which are inherited by the derived classes.	Each derived class inherits the sorter constructor and methods, extending functionality by overriding virtual methods like sort() and getalgorithmname()	

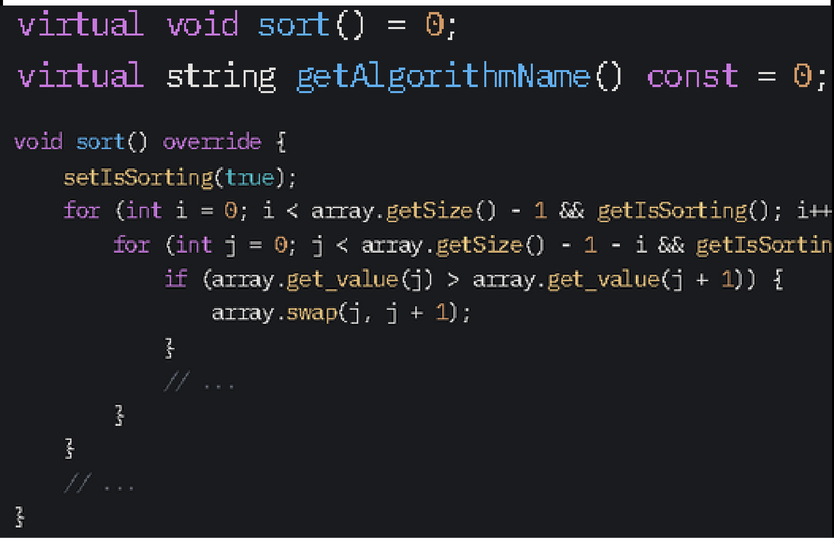
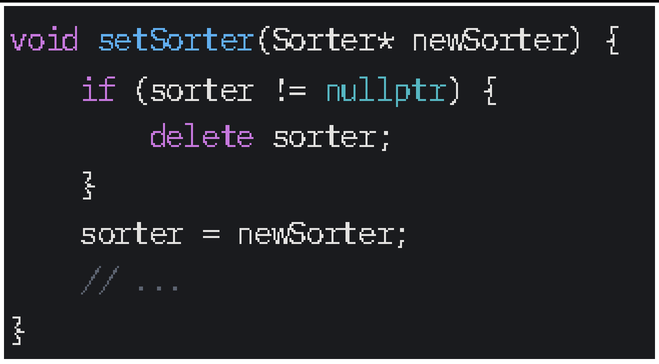
```cpp
class Sorter {
protected:
    Array& array;
    int speed;
    bool isSorting;
    function<void(int, int)> updateCallback;
public:
    virtual void sort() = 0;
    // ...
};

class BubbleSort : public Sorter {
public:
    BubbleSort(Array& arr) : Sorter(arr) {}
    // ...
};
```

03. Polymorphsim

Polymorphism allows objects of different classes to be treated as objects of a common base class, typically through virtual functions, enabling dynamic behavior based on the actual object type.

<u>Example in the Code:</u>

Virtual Functions in Sorter	Dynamic Polymorphism in Visualizer	
The sorter class declares the pure virtual functions that must be overridden by derived classes	The visualizer class uses a Sorter* pointer to reference any derived sorter class. The Set Sorter method assigns a specific sorter and calls the appropriate derived class implementation at runtime.	
		
		

```cpp
virtual void sort() = 0;
virtual string getAlgorithmName() const = 0;

void sort() override {
    setIsSorting(true);
    for (int i = 0; i < array.getSize() - 1 && getIsSorting(); i++
        for (int j = 0; j < array.getSize() - 1 - i && getIsSortin
            if (array.get_value(j) > array.get_value(j + 1)) {
                array.swap(j, j + 1);
            }
            // ...
        }
    }
    // ...
}
```

```cpp
void setSorter(Sorter* newSorter) {
    if (sorter != nullptr) {
        delete sorter;
    }
    sorter = newSorter;
    // ...
}
```

04. Abstraction

Abstraction hides the complex details, exposing only the necessary interface to the user, often achieved through abstract classes or interfaces

Example in the Code:

Sorter as an Abstract Class	Visualizer Interface
The Sorter class is abstract, with pure virtual functions, forcing derived classes to implement sorting algorithms	The visualizer class abstracts the complexity of rendering and managing the SFML window, array visualization and sorting process.
While hiding their details from the Visualizer and the Controller class. Users interact with the sorter interface without needing to understand the underlying algorithm	Methods like start sort , reset array, and choose sorting algorithm provide a high level interface for users, hiding details like thread management and SFML rendering.

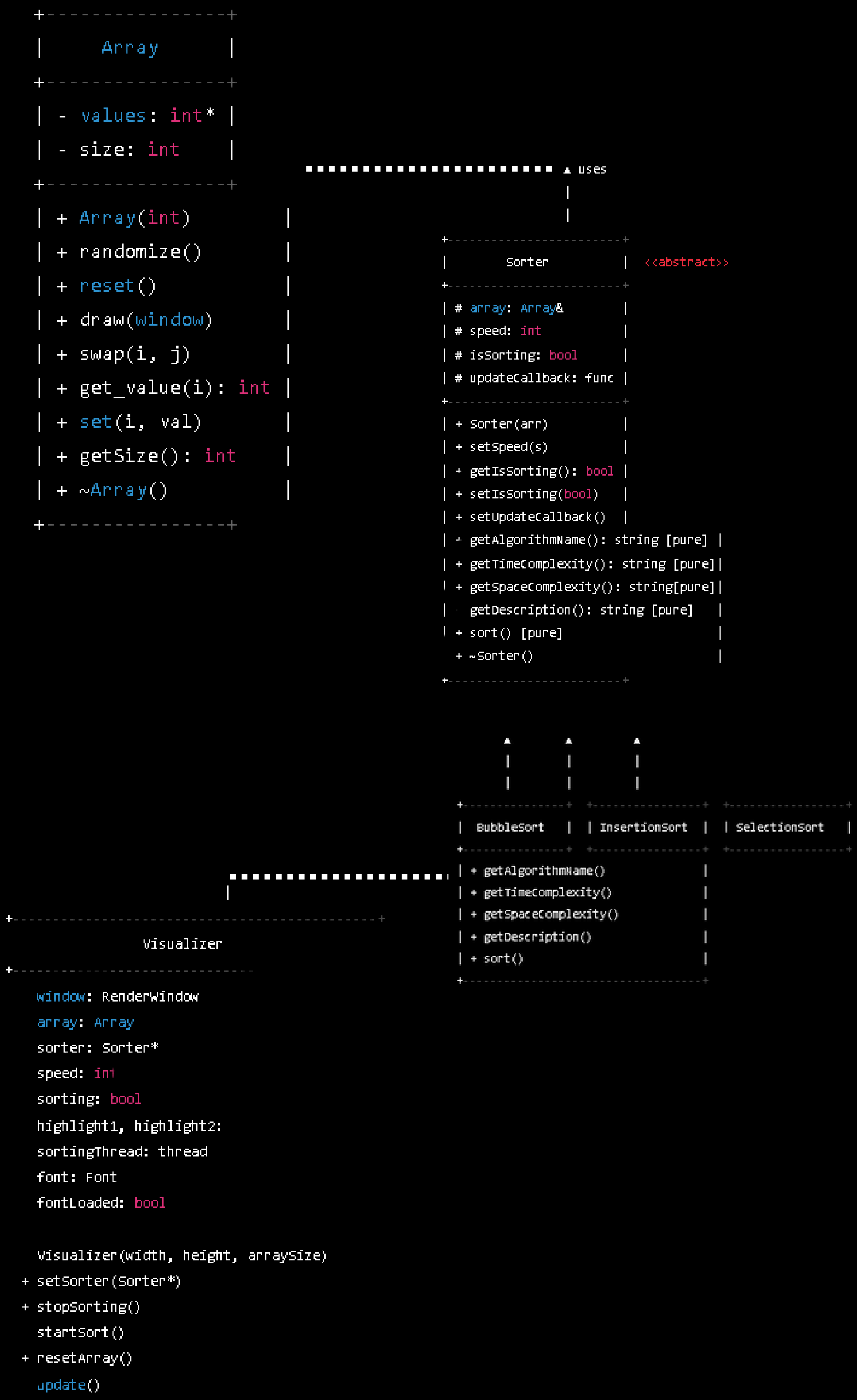
05. Composition

Composition involves building complex objects by combining simpler objects, where one class contains instances of other classes as members, creating “has-a” relationship.

Example in the Code:

Controller and Visualizer	Visualizer/Array Sorter
The controller class contains a visualizer object as a private member, demonstrating composition.	The visualizer class contains an Array object and a Sorter pointer, composing these objects to manage the array data and the sorting process.
The compiler uses the Visualizer to manage the application’s UI and logic delegating tasks like event handling and rendering	The Array object handles the data to be sorted, while the Sorter object manages the sorting algorithm, together enabling the visualization.

CLASS DIAGRAMS



```
+----------------+
|     Array      |
+----------------+
| - values: int* |
| - size: int    |
+----------------+
| + Array(int)         |
| + randomize()        |
| + reset()            |
| + draw(window)       |
| + swap(i, j)         |
| + get_value(i): int  |
| + set(i, val)        |
| + getSize(): int     |
| + ~Array()           |
+----------------+
```

```
▲ uses
|
|
+----------------------+
|        Sorter        |  <<abstract>>
+----------------------+
| # array: Array&      |
| # speed: int         |
| # isSorting: bool    |
| # updateCallback: func |
+----------------------+
| + Sorter(arr)        |
| + setSpeed(s)        |
| + getIsSorting(): bool |
| + setIsSorting(bool) |
| + setUpdateCallback() |
| + getAlgorithmName(): string [pure] |
| + getTimeComplexity(): string [pure]|
| + getSpaceComplexity(): string[pure]|
|   getDescription(): string [pure]   |
| + sort() [pure]                     |
  + ~Sorter()                         |
+----------------------+
```

```
   ▲          ▲           ▲
   |          |           |
   |          |           |
+-------------+ +---------------+ +---------------+
|  BubbleSort  | | InsertionSort | | SelectionSort |
+-------------+ +---------------+ +---------------+
| + getAlgorithmName()          |
| + getTimeComplexity()         |
| + getSpaceComplexity()        |
| + getDescription()            |
| + sort()                      |
+-------------------------------+
```

```
+------------------------------------------+
              Visualizer
+--------------------------------
   window: RenderWindow
   array: Array
   sorter: Sorter*
   speed: int
   sorting: bool
   highlight1, highlight2:
   sortingThread: thread
   font: Font
   fontLoaded: bool

   Visualizer(width, height, arraySize)
 + setSorter(Sorter*)
 + stopSorting()
   startSort()
 + resetArray()
   update()
```

Test Cases

The test cases developed for the sorting visualizer application comprehensively evaluate its functionality, covering core components such as array operations, sorting algorithms, visualization, and user interactions.

1. **Verify that the Array class initializes correctly and randomizes values within the specified range (10 to 50).**

2. **Verify that the Array::swap() method correctly swaps two elements in the array.**

3. **Verify that the BubbleSort class correctly sorts an array in ascending order.**

4. **Verify that the Insertion Sort class correctly sorts an array in ascending order.**

5. **Verify that the Selection Sort class correctly sorts an array in ascending order.**

6. **Verify that sorting algorithms handle an array of size 0 without crashing.**

7. **Verify that the sorting animation speed can be adjusted using Sorter::set Speed().**

8. **Verify that the Visualizer starts the sorting process when the user presses the 'S key.**

9. **Verify that the Visualizer resets the array when the user presses the 'R' key.**

10. **Verify that the Visualizer allows changing the sorting algorithm when the user presses the 'A' key.**

11. **Verify that the Visualizer handles font loading failure by falling back to consol output.**

Future Directions

As Algoviz continues to evolve, there are several promising directions in which the platform can grow to enhance its educational impact and technical capabilities

01. Expansion of Algorithm Library

Including a broader range of algorithms such as dynamic programming, backtracking, greedy algorithms, to cater advance learners

02. Adaptive Learning Features

Integrate user progress tracking, personalized learning paths, and quiz based assessments to tailor the individual experience

03. Cross Platform Availability

Develop mobile and desktop applications with offline capabilities to ensure learners can access AlgoViz anytime and anywhere

we envision Algoviz in becoming a go to companion for algorithm learning that empowers the students globally. Moving forward, educators, developers and learners are expected to contribute and co create a future where computer science education is intuitive and inspiring.

GitHub Project Link

Link:

https://github.com/thesocialobaid/ALGOVIZ

We thank you for your continued support in our efforts throughout the Course

Contact

MUHAMMAD OBAIDULLAH

24L-0509

l240509@lhr.nu.edu.pk