Hanoi University of Science and Technology

University of Information and Communication Technology

****

**DEMONSTRATION OF SORTING ALGORITHMS ON AN ARRAY**

| Le Duc Anh Tuan | 20204929 |
| --- | --- |
| Nguyen Anh Tuan | 20204930 |
| Bui Thanh Tung | 20204931 |
| Dao Van Tung | 20204932 |

**ACKNOWLEDGEMENTS**

We would like to express our deepest appreciation to Prof. Nguyen Thi Thu Trang who gave us a golden opportunity to work on this project. This work would not have been possible without your tremendous support and guidance. We would like to extend our sincere gratitude to Nguyen Thi Thu Giang and Vuong Dinh An, the teaching assistants of our class, for your invaluable advice and guidance throughout the whole semester.

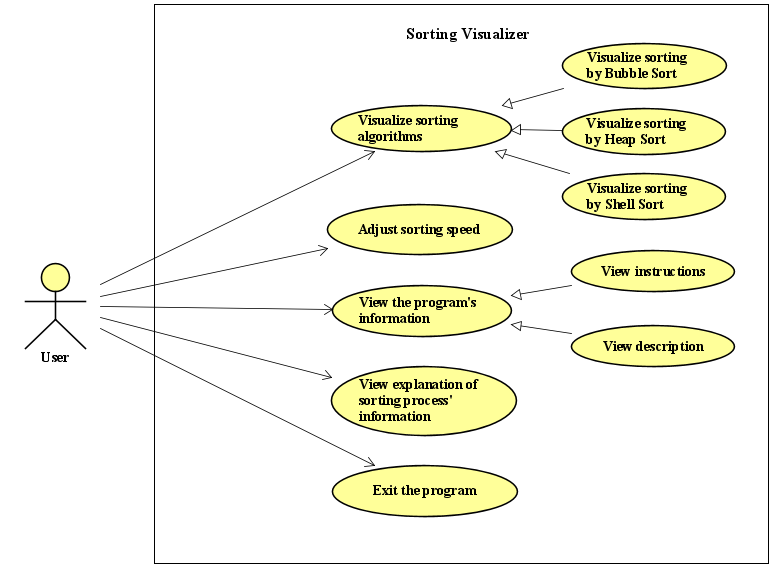
1. **DESCRIPTION ABOUT THE PROJECT**

A sorting algorithm is an algorithm used to rearrange the elements of an array into an order. From the beginning of computers, the sorting problems have attracted many researchers due to the difficulties of solving those simple, familiar problems efficiently. Bubble sort and shell sort are two famous, simple, yet rarely used in the real world because of their poor performance. Heapsort is a much more efficient and popular sorting algorithm. The reasoning behind this is that the use of a heap data structure to more quickly find the largest element in each step.

Our group will make an application to explain, demonstrate and provide some useful information with three aforementioned sorting algorithms. In this project, Java Swing would be our choice to design the graphical user interfaces combined with applying and implementing some object-oriented theory (Encapsulation, Abstraction, Polymorphism, Inheritance) into our program.

1. **REQUIREMENT**

Our goal is to provide the users a simple, comfortable, easy-to-navigate, and informative UI. With our program, the users are able to witness the whole sorting process of 3 sorting algorithms (Bubble sort, Shell sort, Heap sort).

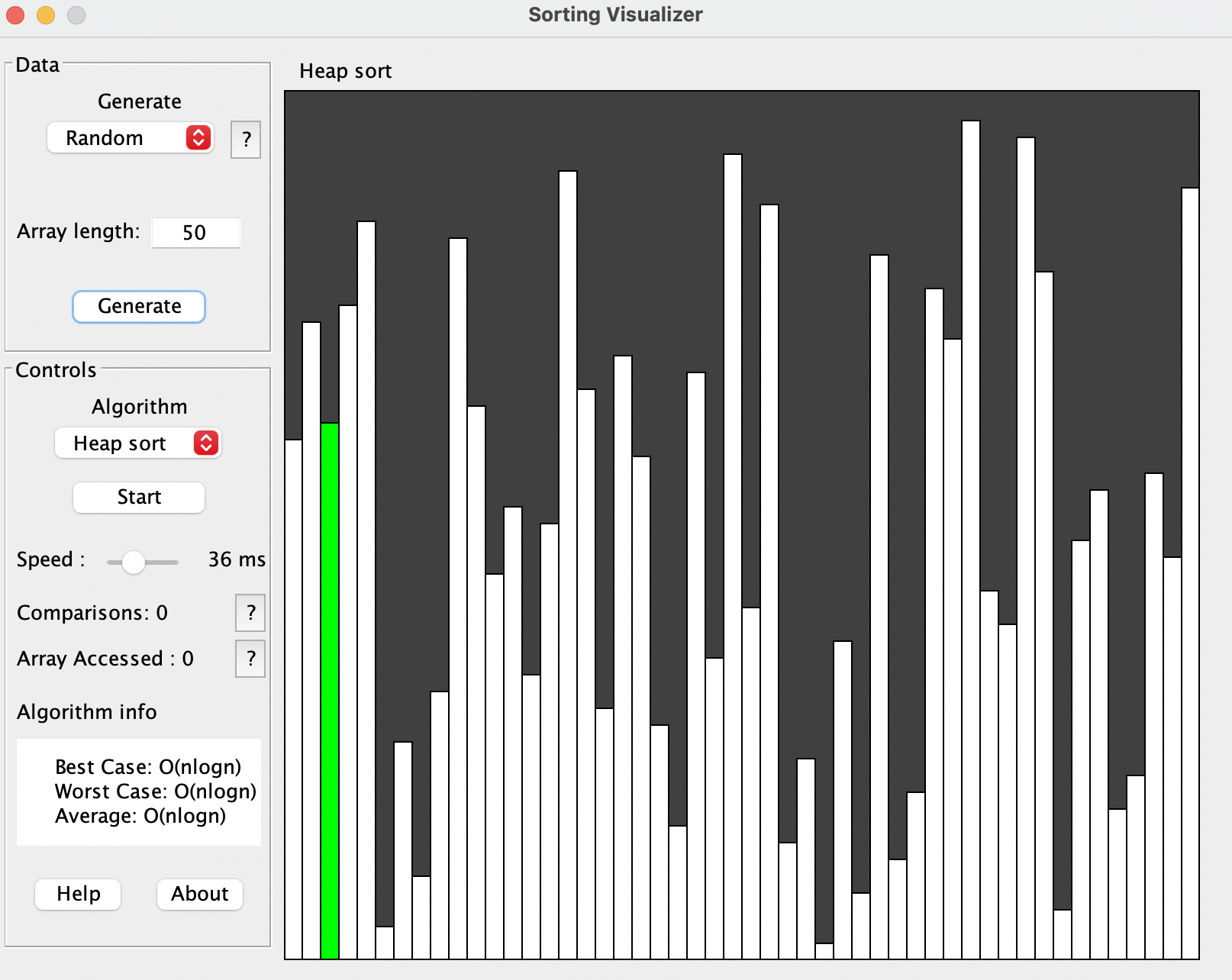


*Figure 1: Use case diagram*

Firstly, the users need to create the array to be sorted. We could select to do this randomly or manually by choosing the desired method in the *“Create data”* dropdown button. After selecting the way that our array’s elements would be generated, the users need to input the array length (which is initially set at 50). This is the number of elements that would be in our array. When everything in the data section is set, the array will respectively be visualized by the white rectangles from left to right in the main screen after the “Generate” button is clicked. The height of the rectangles represent how large or small the elements in the array are, i.e. the larger the elements, the taller the rectangles.

Secondly, you have the option to choose the desired algorithm for our program to visualize by using the *“Algorithm”* dropdown button. After that, the sort would immediately start when the *“Start”* button is chosen. While the sorting process is running, the users could stop the sort temporarily using the *“Pause”* button, after that the current sort could continue at any time, using the *“Resume”* button. Another useful action that could be used during the sort is the *“Stop”* button, which would completely stop the ongoing process.

The users are able to adjust the sorting process by making use of the *“Speed”* slider, the process would speed up when we pull the slider to the left and contrastly, slow down when we pull it to the right. We show some information about the sorting process, including the number of times two numbers in the array have been compared in the *“Comparisons”* part and the number of times the array has been accessed in the *“Array Accessed”* part. The time complexity of the chosen algorithm is also shown in the *“Algorithm info”* section.

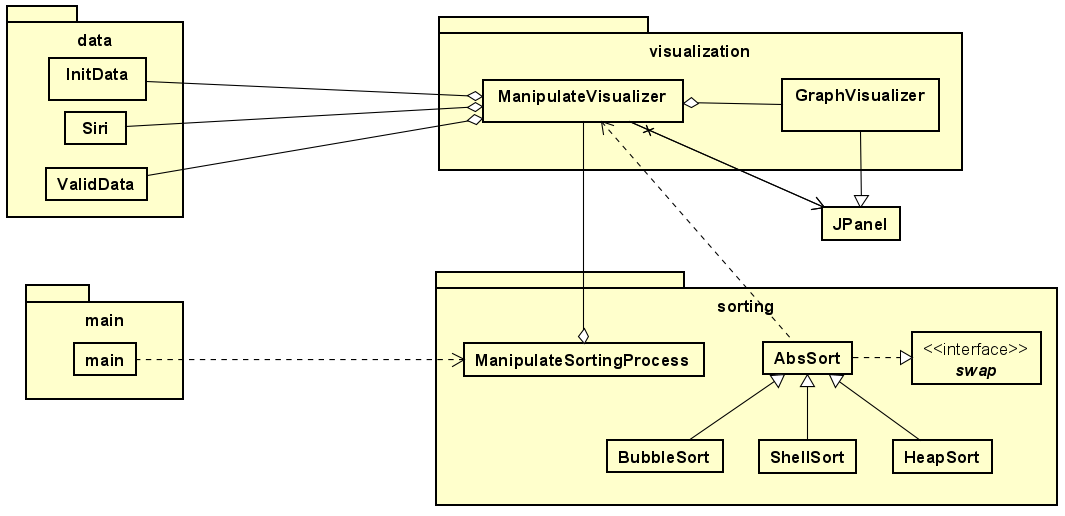
****

*Figure 2: Initial screen*

The *“Help”* button and the *“About”* button provide instructions and description for the program, respectively. The users are also provided with explanations and requirements to the according part using the *“?”* button.

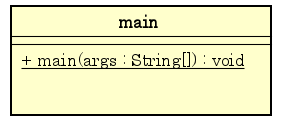
1. **DESIGN**

We structure the program into four packages (*main, data, sorting, visualization*). The *ManipulateVisualizer* class is an aggregation of 4 other classes, i.e. *InitData, Siri, ValidData, GraphVisualizer*; and it also. The abstract class AbsSort, which implements the *swap* interface, is the parent class of three classes, namely *BubbleSort, ShellSort, ShellSort, HeapSort*. The *ManipulateSortingProcess* class contains a reference to class *ManipulateVisualizer,* hence is an aggregation of *ManipulateVisualizer*. The program will be executed in the main class by creating an instance of *ManipulateSortingProcess* and calling the *init()* method.

****

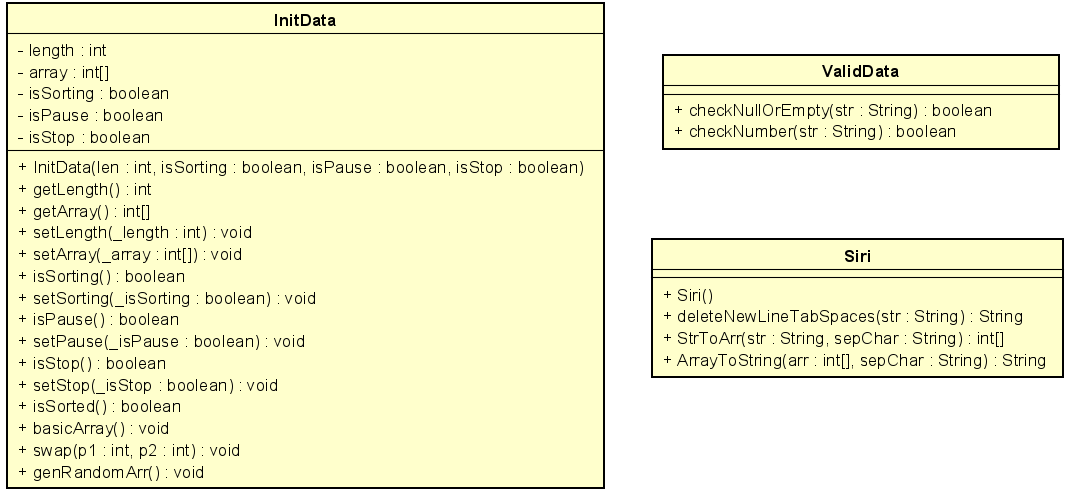
*Figure 3: General class diagram*

When we invoke the *main* function, the program will be executed and the GUI will appear on the screen.

****

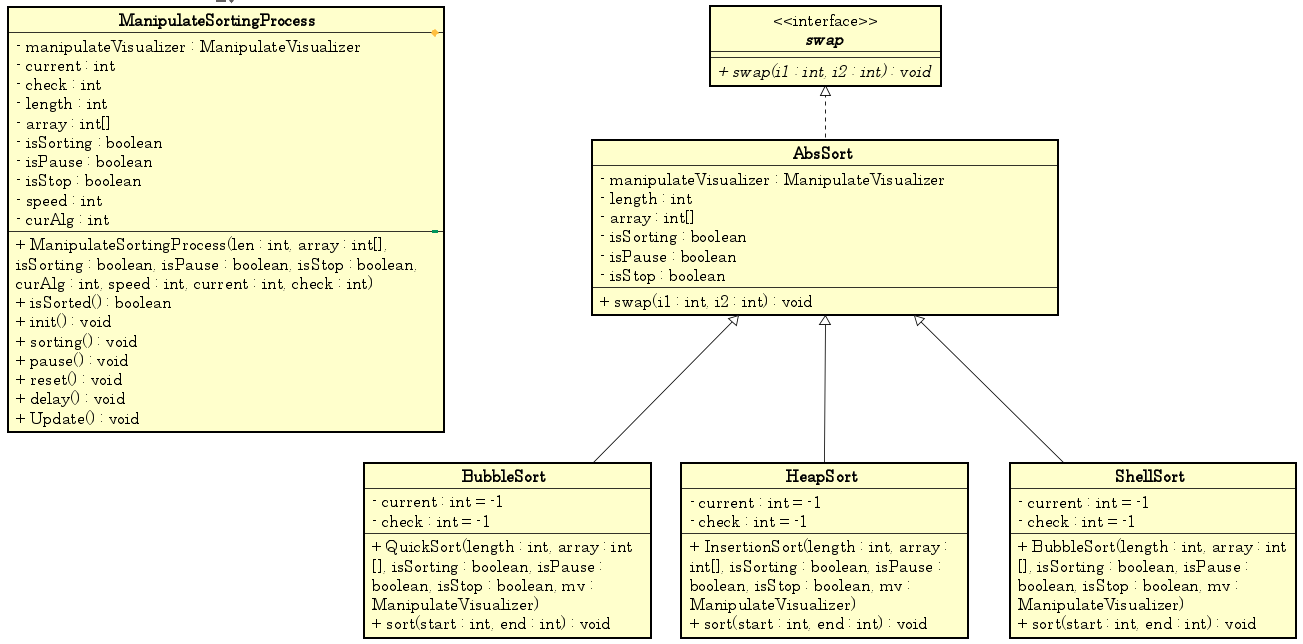
*Figure 4: Class diagram for package main*

The *data* package is used for data validation. The *InitData* class’s job is to initialize the random array if the users choose to create the array randomly. In terms of manual input, the *ValidData* class makes sure that the input elements in the array are in the correct form (all the elements are numerical and the array cannot be empty) while the *Siri* class transforms the input data into a pure array for us to work with in the sorting stage.

**

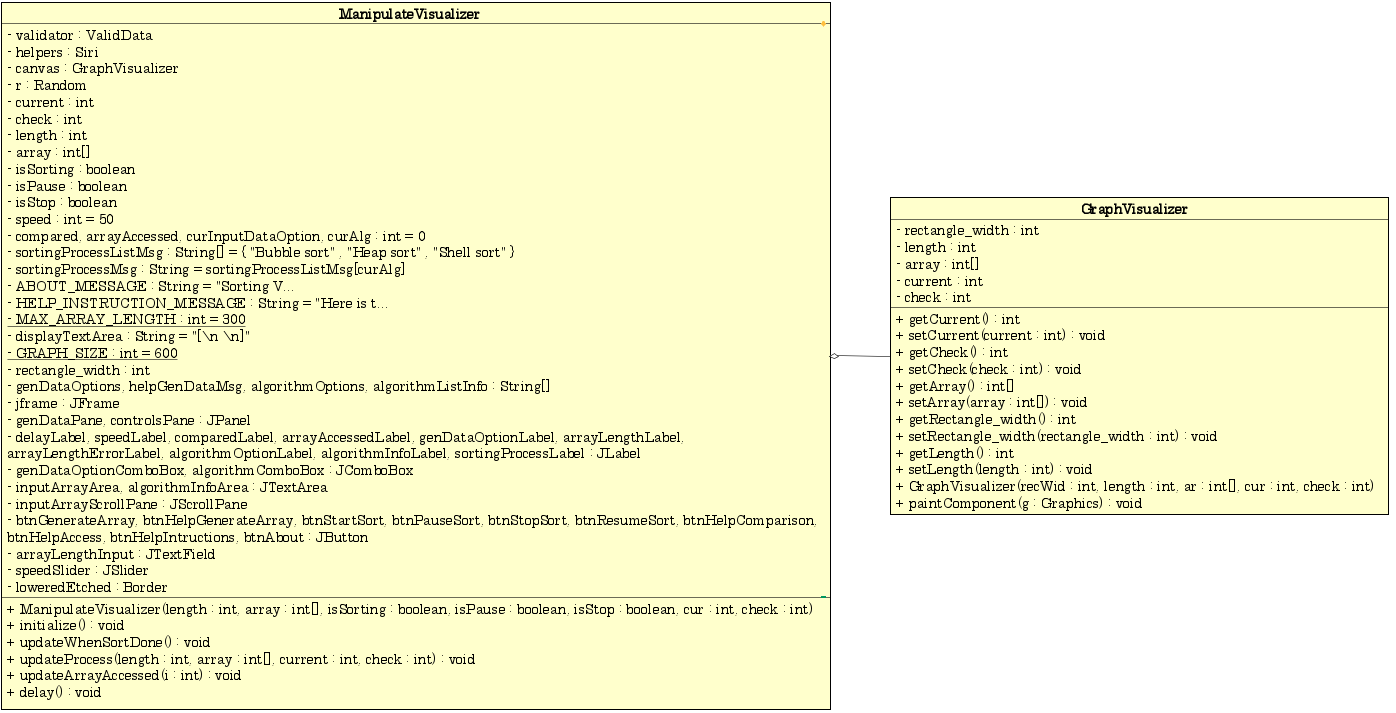
*Figure 5: Class diagram for package data*

The *sorting* package is where all the sorting of the three algorithms happen. The abstract class *AbsSort* is used as a template that holds all the common functionality for 3 sorting classes. This class implements the *swap* interface in order to help us swap the position of two elements in the array. The *BubbleSort* class, the *HeapSort* class, the *ShellSort* class extend *AbsSort* and implement the corresponding algorithm to sort the array. The *ManipulateSortingProcess* class is in charge of the whole sorting process by managing some variables, i.e. the current, the check, the length of the array, the array itself, whether the sorting process is running, being paused or has been stopped, the sorting speed and the chosen algorithm.

**

*Figure 6: Class diagram for package sorting*

The visualization package is in charge of displaying the GUI and visualizing the sorting process. There are two classes in this packages, i.e. *ManipulateVisualizer* and *GraphVisualizer.* The *GraphVisualizer* class which inherits from *JPanel* is responsible for handling the rectangles that represent the elements of the array. This class overrides the *paintComponent()* method to repaint the GUI when needed. The *ManipulateVisualizer* class is used to set up all the GUI components and event handling. This class also includes methods for updating all the components when the sorting process is happening.

*****Figure 7: Class diagram for package visualization*

1. **IMPLEMENTATION**

**4.1 Encapsulation**

All of our instance variables are declared as private, which encapsulates each attribute of its class. The object’s attributes are now only accessible within that particular class, through the use of setters and getters. This prevents other classes from manipulating the attributes, provides a safety net so that they are not eventually abused and helps to change the behavior, regulate and perform validation on our variables when needed.

**4.2 Abstraction**

We implement abstraction in two ways:

* Using Abstract Class: The abstract class *AbsSort* creates a template for all three sorting algorithms classes and requires its subclasses to provide implementations for the abstract methods *sort(int start, int end).*
* Using Interface: The *swap* interface specifies the *swap* method (which is used to swap two elements in an array) for the *AbsSort* class to implement.

**4.3 Inheritance**

* *BubbleSort, HeapSort, ShellSort* inherits from the abstract *AbsSort* class.
* *GraphVisualizer* inherits from *JPanel.*

**4.4 Polymorphism**

The *GraphVisualizer* class, which is a subclass of *JPanel*, overrides the *paintComponent()* method so that when we call the *repaint()* method defined by JComponent (the parent class of *JPanel*) in the *GraphVisualizer* class, the graphic on the panel is automatically cleared and the *paintComponent()* method will be automatically executed to redraw the graphics on this panel.

All three classes *BubbleSort, HeapSort, ShellSort* provide an implementation of the *sort(int start, int end)* method. In other words, these three classes override the *sort(int start, int end)* method from their superclass, enabling each of the subclasses to customize the behavior of that method (which is to operate the sorting process of the array with the according algorithms). Both methods implemented by the child and the parent classes share the same name and parameters but have different functionality.

The *AbsSort* class also implements the swap interface and therefore, must override the *swap(int i1, int i2)* method.

1. **CONCLUSIONS AND FUTURE WORK**

In conclusion, we have successfully solved our problems of visualizing three popular sorting algorithms using four pillars of object-oriented programming (Encapsulation, Abstraction, Polymorphism, Inheritance) and implement them with a complete graphical user interface. After finishing this project, we have gained a lot of experience applying different OOP concepts into a particular task. In addition, all the time working and discussing with each other as a group has undoubtedly enhanced our understanding about this subject as well as helping us learn how to think about the structure of the program and planning at the beginning of coding. In the future, we would like to add more algorithms into this program and provide the users with a complete visualization of all popular sorting algorithms right now.

1. **LIST OF TASKS**

| **Le Duc Anh Tuan** | * Data: Init, ValidData & Siri * Soring: AbsSort, Bubble Sort, ManipulateSort, swap * Visualization: GraphVisualizer & ManipulateVisualizer * Refactor code |
| --- | --- |
| **Dao Van Tung** | * Sorting: Heap Sort, Shell Sort & Manipulate Sort * Fix bug * Refactor code |
| **Bui Thanh Tung** | * Visualization: GraphVisualizer * Fix Bug * Refactor code * Use case & Class Diagram * Report |
| **Nguyen Anh Tuan** | * Main * Data: Siri * Slide * Report |