PROJECT REPORT
GROUP 15

Object-oriented Programming - 131678

Topic: Demonstration of sorting algorithms on an array (1)

Lecturer: PhD. Nguyen Thi Thu Trang

Group member:

Hoàng Trần Nhật Minh – 20204883

Nguyễn Quang Minh – 20204884

Nguyễn Trần Xuân Mạnh – 20200385

Dương Vũ Tuấn Minh – 20209705

1. Introduction of the project and requirements:

1.1 Introduction:

In this project, we will be working with array – one of the most basic data structures of computer science. We will demonstrate 3 sorting algorithms: bubble sort, heapsort and shellsort by creating an application using Java Swing for the users to interact and understand the concept and mechanisms of each algorithm.

1.2 Requirements:

On the main menu: title of the application, 3 types of sorting algorithms for a user to choose, help menu and quit.

- User must select a sort type to start the demonstration

- Help menu to show the basic usage and aim of the program

In the demonstration:

- A button to for the user to input an array for the demonstration

- A button to randomly create an array for the demonstration

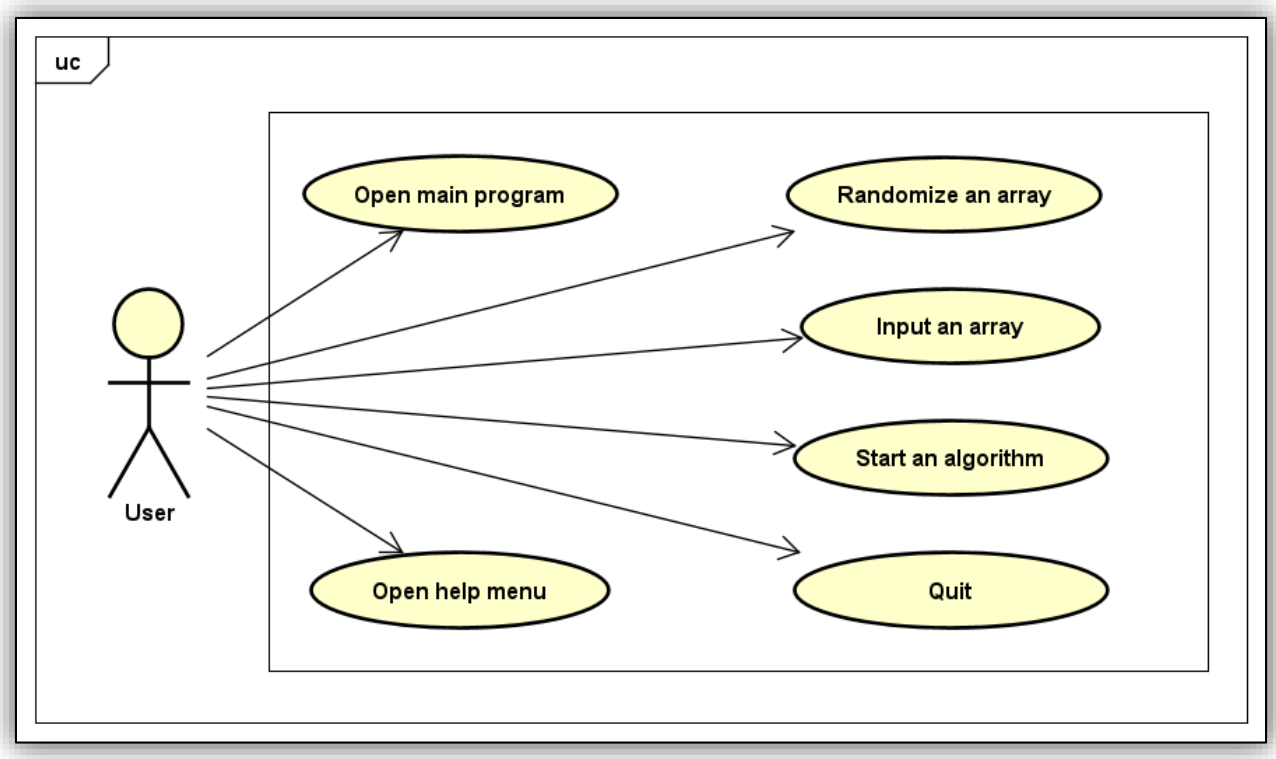
- 3 buttons representing each sorting algorithms for the users to choose

- A textpane to demonstrate each step of sorting and the final result

- A textfield for the user to input the number of instances for randomizing array

2. Project analysis:

2.1. Use case analysis:



In this application, the user can open the main program, choose an algorithm out of the 3 being listed in this project to demonstrate. The user must input an array or randomize an array for the algorithms to try to sort. Once the array has been inputed, the user can choose to start the algorithm. The array will be sorted step by step and returned completely sorted. In addition, the user can get guidance through the help menu and quit the application whenever they want.

2.2. Sorting algorithms:

2.2.1. Bubble sort:

Consider an array of size n: a1, a2, a3, ..., an.

Starting from left to right, the algorithm will loop through the array until it reaches an. During the loop, it will compare two adjacent elements and swap them if the element on the left is bigger than the one on the right. It

will continue to loop again until there are no swaps throughout the whole loop. This means the array has been completely sorted.

Example: Consider the array [5, 1, 4, 2, 8]

First loop: - [**5, 1**, 4, 2, 8] -> [**1, 5**, 4, 2, 8] (elements in bold are compared), swapping 5 and 1 since 5 > 1.

- [1, **5, 4**, 2, 8] -> [1, **4, 5**, 2, 8], swapping 5 and 4 since 5 > 4.
- [1, 4, **5, 2**, 8] -> [1, 4, **2, 5**, 8], swapping 5 and 2 since 5 > 2.
- [1, 4, 2, **5, 8**] -> [1, 4, 2, **5, 8**], no swap since 5 < 8.

Second loop: - [**1, 4**, 2, 5, 8] -> [**1, 4**, 2, 5, 8], no swap since 1 < 4.

- [1, **4, 2**, 5, 8] -> [1, **2, 4**, 5, 8], swapping 4 and 2 since 4 > 2.
- [1, 2, **4, 5**, 8] -> [1, 2, **4, 5**, 8], no swap since 4 < 5.
- [1, 2, 4, **5, 8**] -> [1, 2, 4, **5, 8**], no swap since 5 < 8.

Third loop: - [**1, 2,** 4, 5, 8] -> [**1, 2,** 4, 5, 8], no swap since 1 < 2.

- [1**, 2, 4**, 5, 8] -> [1, **2, 4**, 5, 8], no swap since 2 < 4.
- [1, 2, **4, 5**, 8] -> [1, 2, **4, 5**, 8], no swap since 4 < 5.
- [1, 2**,** 4, **5, 8**] -> [1, 2**,** 4, **5, 8**], no swap since 5 < 8.

There are no swaps throughout this loop so our array has been sorted completely.

2.2.2. Heapsort:

In the first step, a heap is built out of the data. The heap is often placed in an array with the layout of a complete binary tree. The complete binary tree maps the binary tree structure into the array indices; each array index represents a node; the index of the node's parent, left child branch, or right child branch are simple expressions. For a zero-based array, the root node is stored at index 0; if `i` is the index of the current node, then

```
iParent(i)  = floor((i-1) / 2) where floor functions map a real number to the largest leading integer.
```

```
iLeftChild(i)  = 2*i + 1
iRightChild(i) = 2*i + 2
```

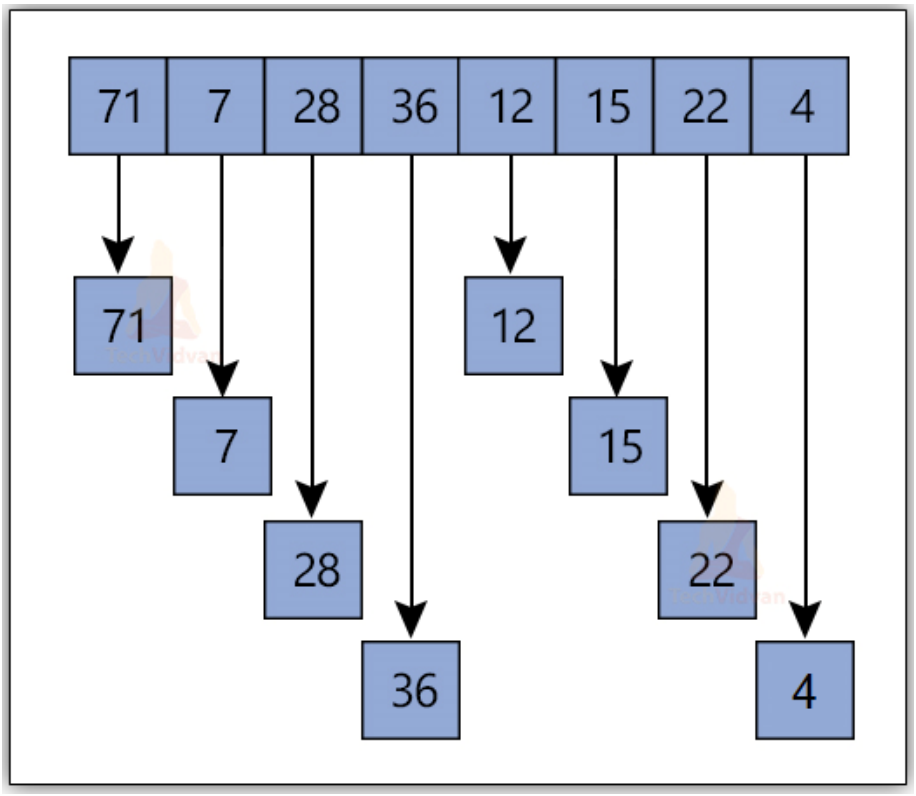
In the second step, a sorted array is created by repeatedly removing the smallest element from the heap (the root of the heap), and inserting it into the array. The heap is updated after each removal to maintain the heap property. Once all objects have been removed from the heap, the result is a sorted array.

2.2.3. Shellsort:

Shellsort is an optimization of insertion sort that allows the exchange of items that are far apart. The idea is to arrange the list of elements so that, starting anywhere, taking every *h*th element produces a sorted list. Such a list is said to be *h*-sorted. It can also be thought of as *h* interleaved lists, each individually sorted. Beginning with large values of *h* allows elements to move long distances in the original list, reducing large amounts of disorder quickly, and leaving less work for smaller *h*-sort steps to do. If the list is then *k-sorted* for some smaller integer *k*, then the list remains *h*-sorted. Following this idea for a decreasing sequence of *h* values ending in 1 is guaranteed to leave a sorted list in the end.

Example: Consider the array [71, 7, 28, 36, 12, 15, 22, 4] of size N = 8

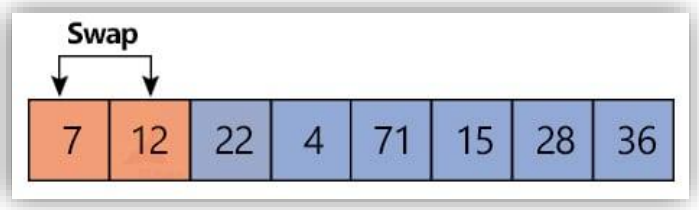
The first step in shell sort is to decide the gap/interval. Let us take interval = N/2 = 4. Thus, we will select pairs of elements present at an interval 4 from each other and swap them if they are out of order.



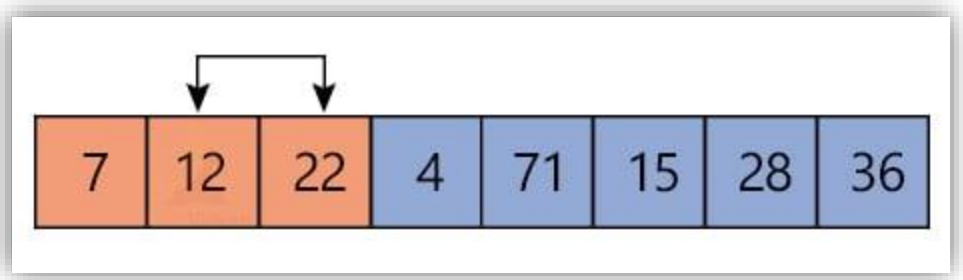
On swapping these pairs, the array will be:

After performing this step, we will simply apply insertion sort on this array.

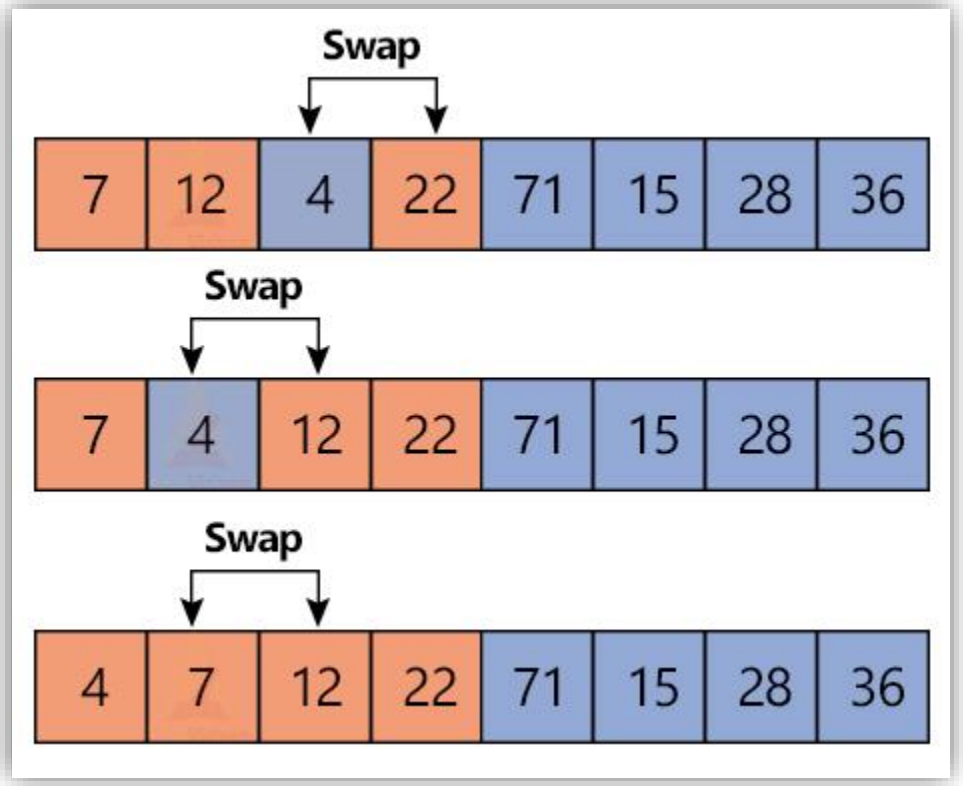
Iteration 1:



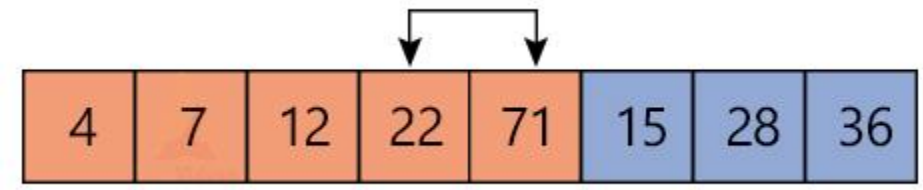
Iteration 2:



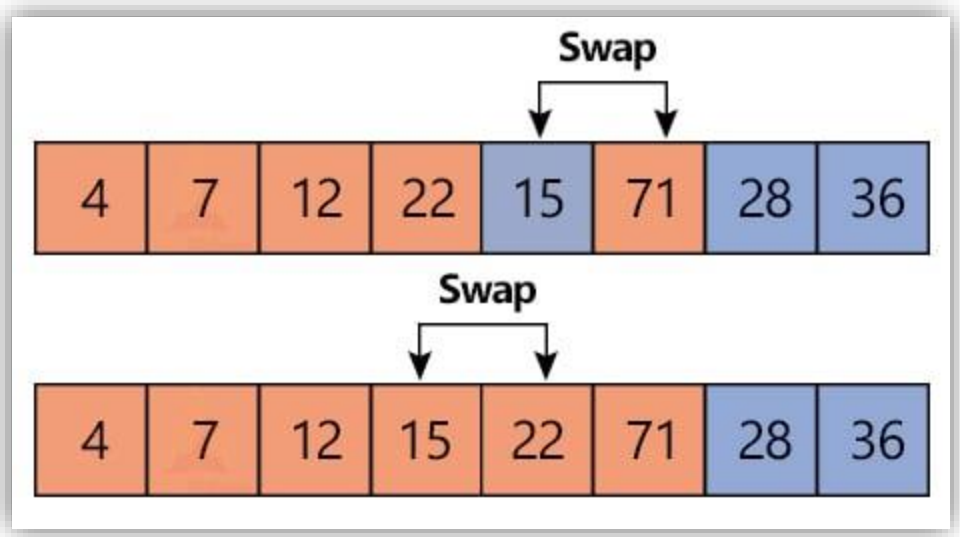
Iteration 3:



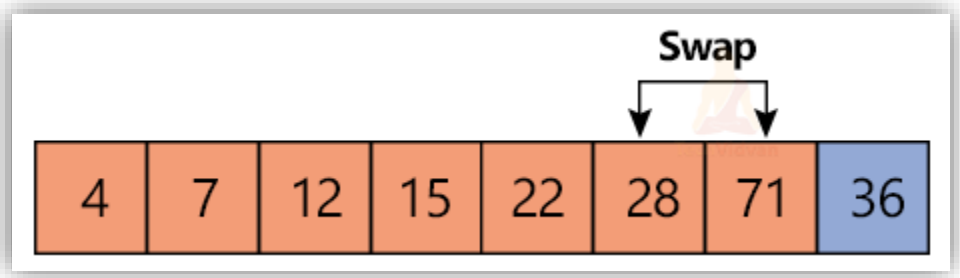
Iteration 4:



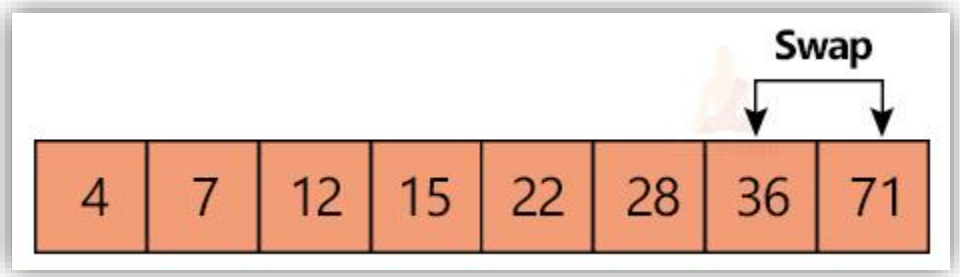
Iteration 5:



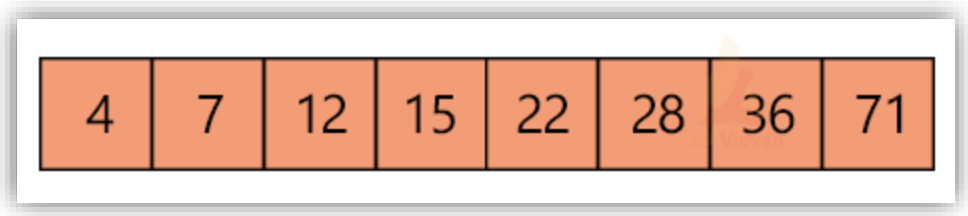
Iteration 6:



Iteration 7:

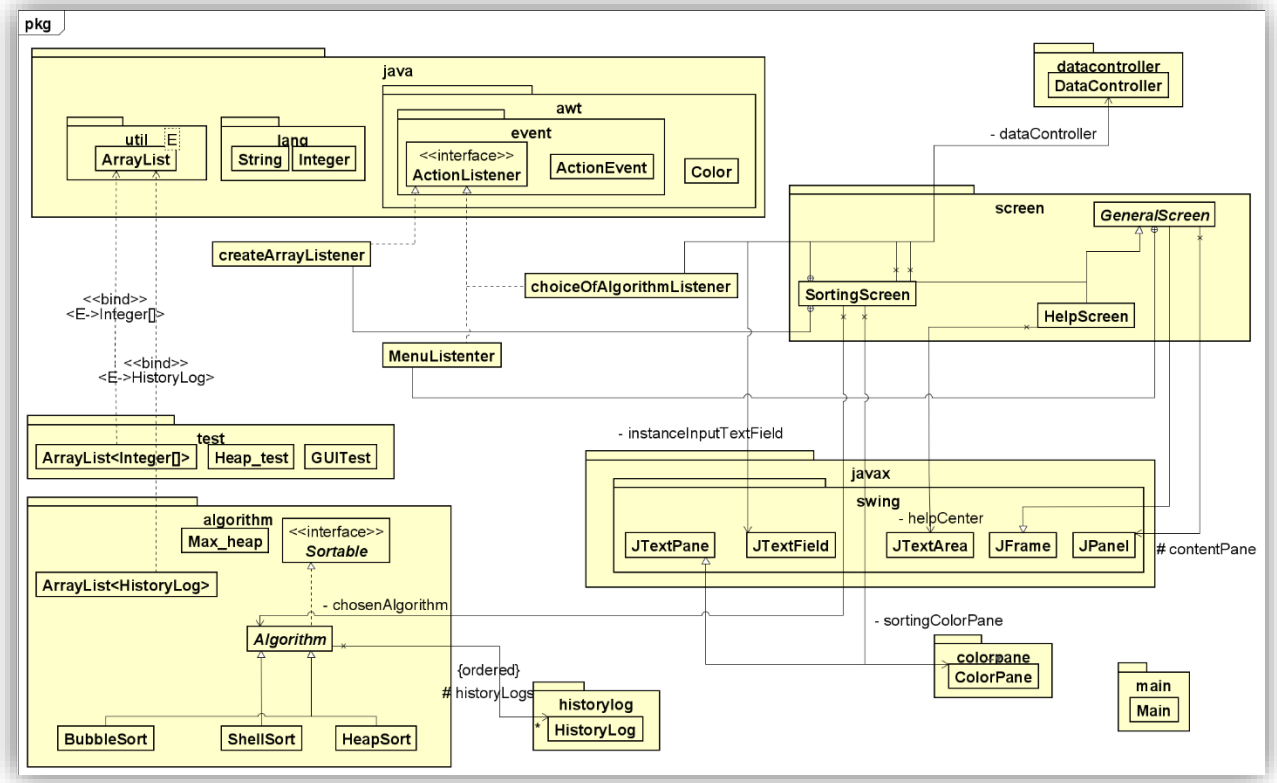


Thus, the final sorted array is:



3. Project design:

3.1. General class diagram:



Explanation of each package:

- algorithm: Include all the sorting algorithms for the system to use and for the users to demonstrate.

- colorpane: Include a JTextPane with a method to append colored text.

- datacontroller: Include a class that manipulate and control our array.

- historylog: Include a class that will record changes inside Algorithm.sort() method

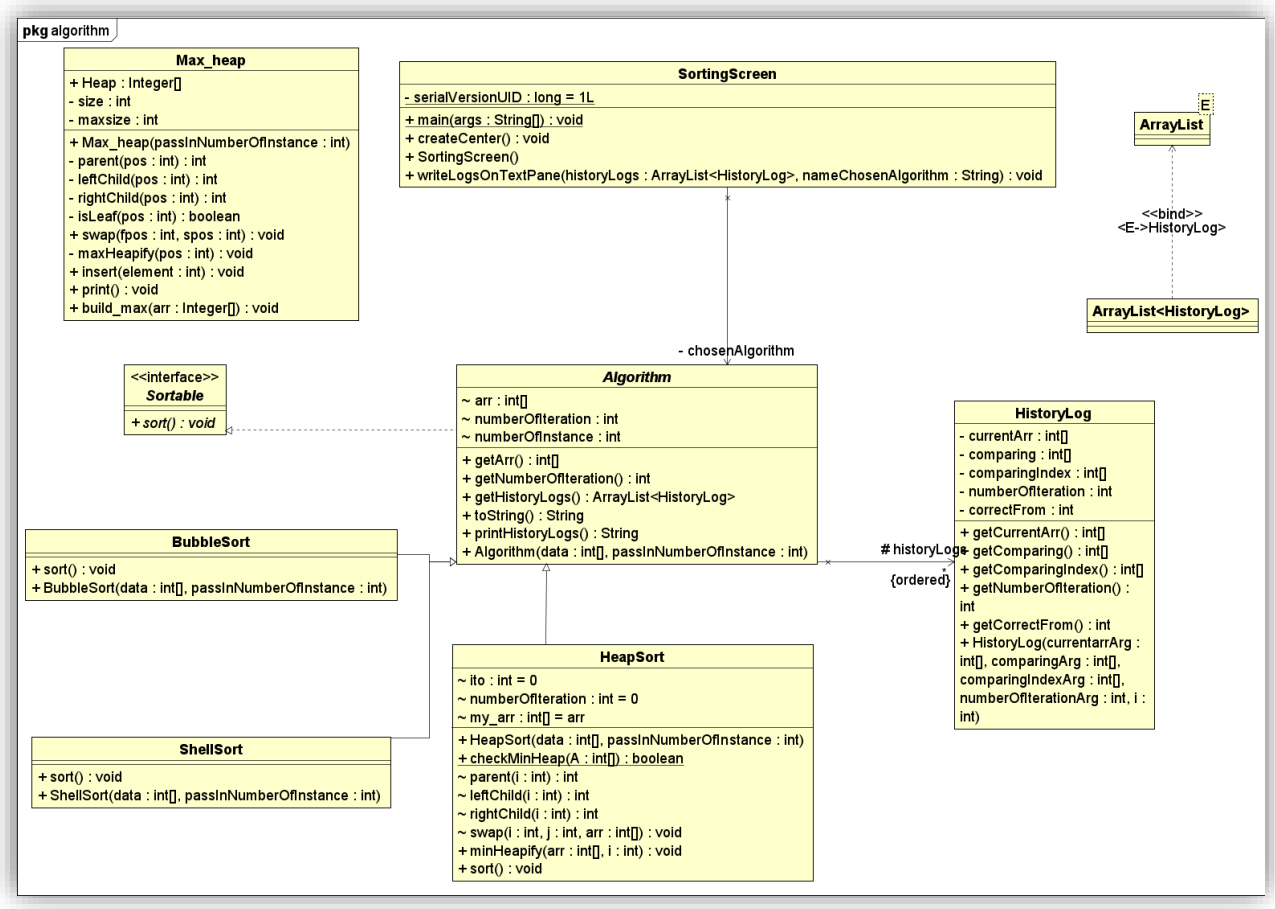
- screen: Include all the screens in the application for the user to interact.

- test: test the GUI and heapsort

- main: Include the main() method for the program to run.

3.2. Class diagrams for each package:

3.2.1. Algorithm:



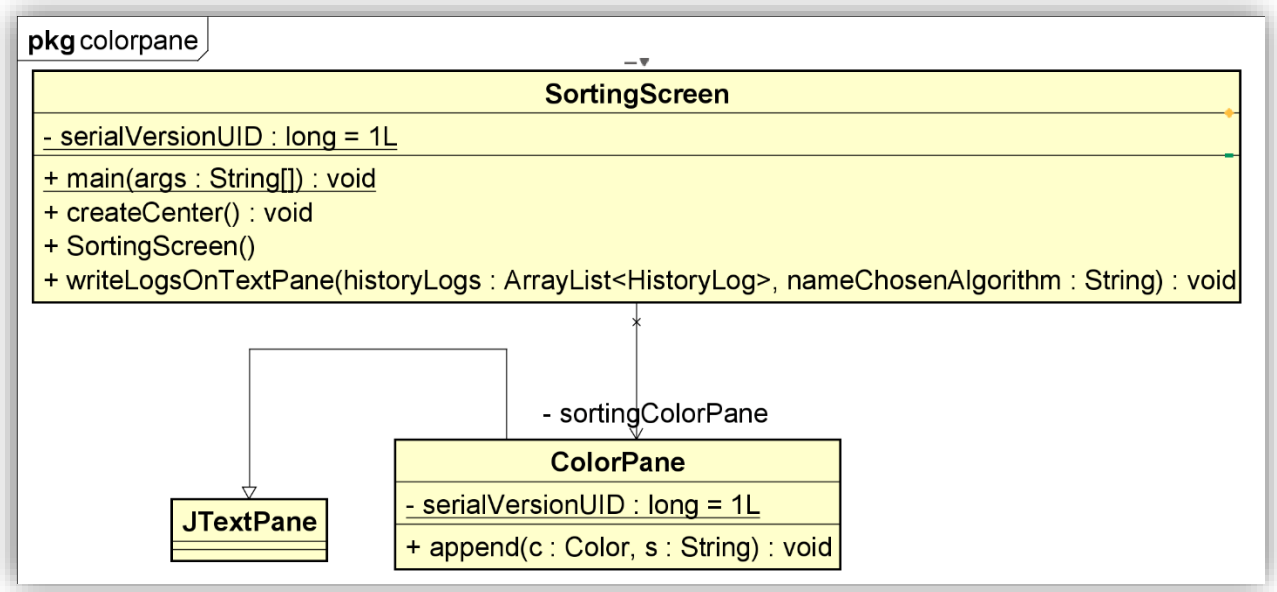
Class/interface name	Attribute	Operation

Algorithm	- **arr**: the array for the algorithms to sort - **numberOfIteration**: the number of iterations in the sorting process - **numberOfInstance**: the number of elements in the array - **historyLogs**: an arraylist to keep tracks of changing states of a specify algorithm	- **getArr()**: return the array being sorted - **getNumberOfIteration()**: return the number of iterations - **toString()**: return a String of the current array - **printHistoryLogs()**: return the historyLog in the process of sorting - **Algorithm(int[] data, int passInNumberOfInstance)**: constructor method of class Algorithm.
BubbleSort		- **sort()**: sorting the array using Bubble Sort algorithm - **BubbleSort(int[] data, int passInNumberOfInstance)**: constructor method of class BubbleSort
HeapSort	- **ito**: number of elements in correct order - **my_arr**: the array for the algorithm to sort	- **sort()**: sorting the array using Heapsort algorithm - **checkMinHeap(int[] A)**: check an array A in heap or not - **parent(int i)**: return the parent index of index i - **leftChild(int i)**: return the left child index of index i - **rightChild(int i)**: return the right child index of index i

		- **swap(int i, int j, int[] arr)**: swapping element index i and index j in the array arr - **minHeapify(int[] arr, int i)**: build min heap arr from index i - **HeapSort(int[] data, int passInNumberOfInstance):** constructor method of class HeapSort
ShellSort		- **sort():** sorting the array using Shell Sort algorithm - **ShellSort(int[] data, int passInNumberOfInstance)**: constructor method of class ShellSort
Sortable **<interface>**		- **sort():** sorting the array

All 3 classes: BubbleSort, HeapSort, ShellSort inherit from class Algorithm (Inheritance) and implement Sortable interface. Each sort() method of all 3 algorithms has its own sorting process (Polymorphism). Attribute arr and numberOfIteration can be accessed through get method (Encapsulation).

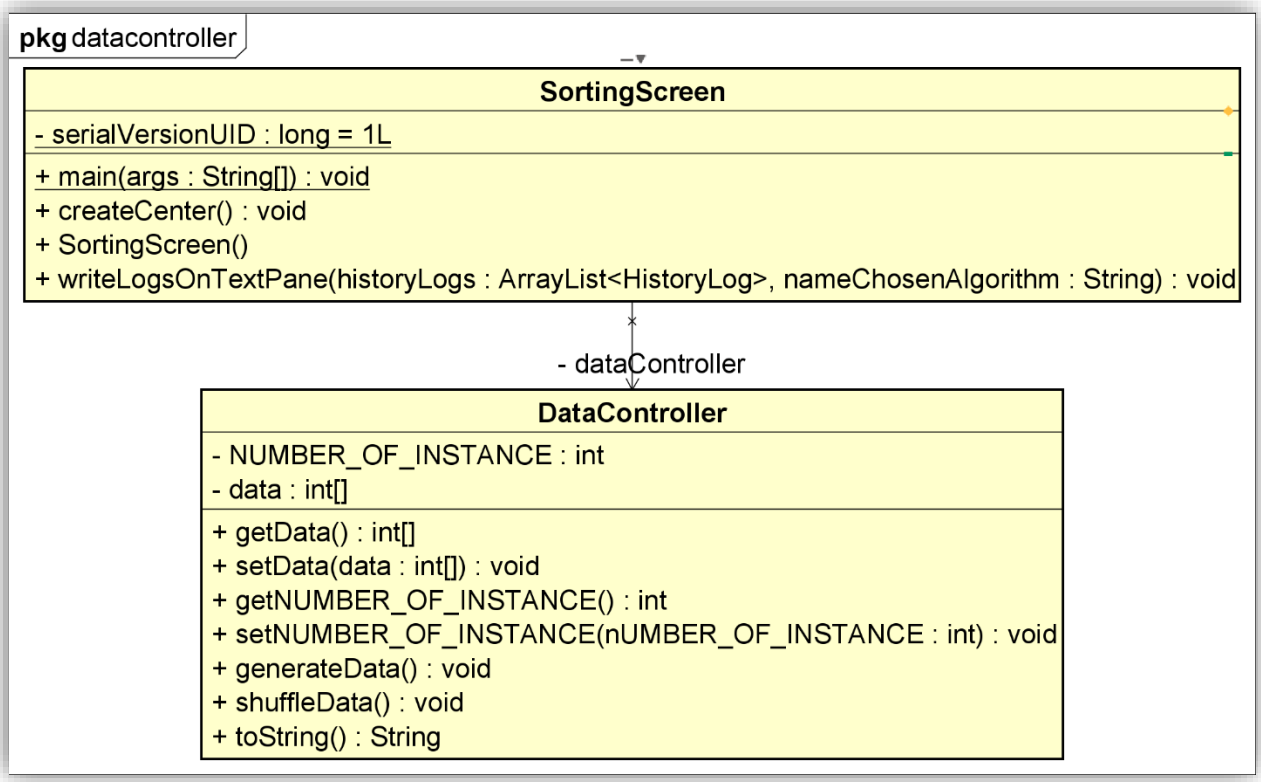
3.2.2. Colorpane:



Class/attribute name	Attribute	Operation
ColorPane	**- serialVersionUID:** help JVM identify the state of the object when it reads the state of the object from a file	- **append():** append colored text

ColorPane class inherits JTextPane class from Java Swing (Inheritance).

3.2.3. Datacontroller:

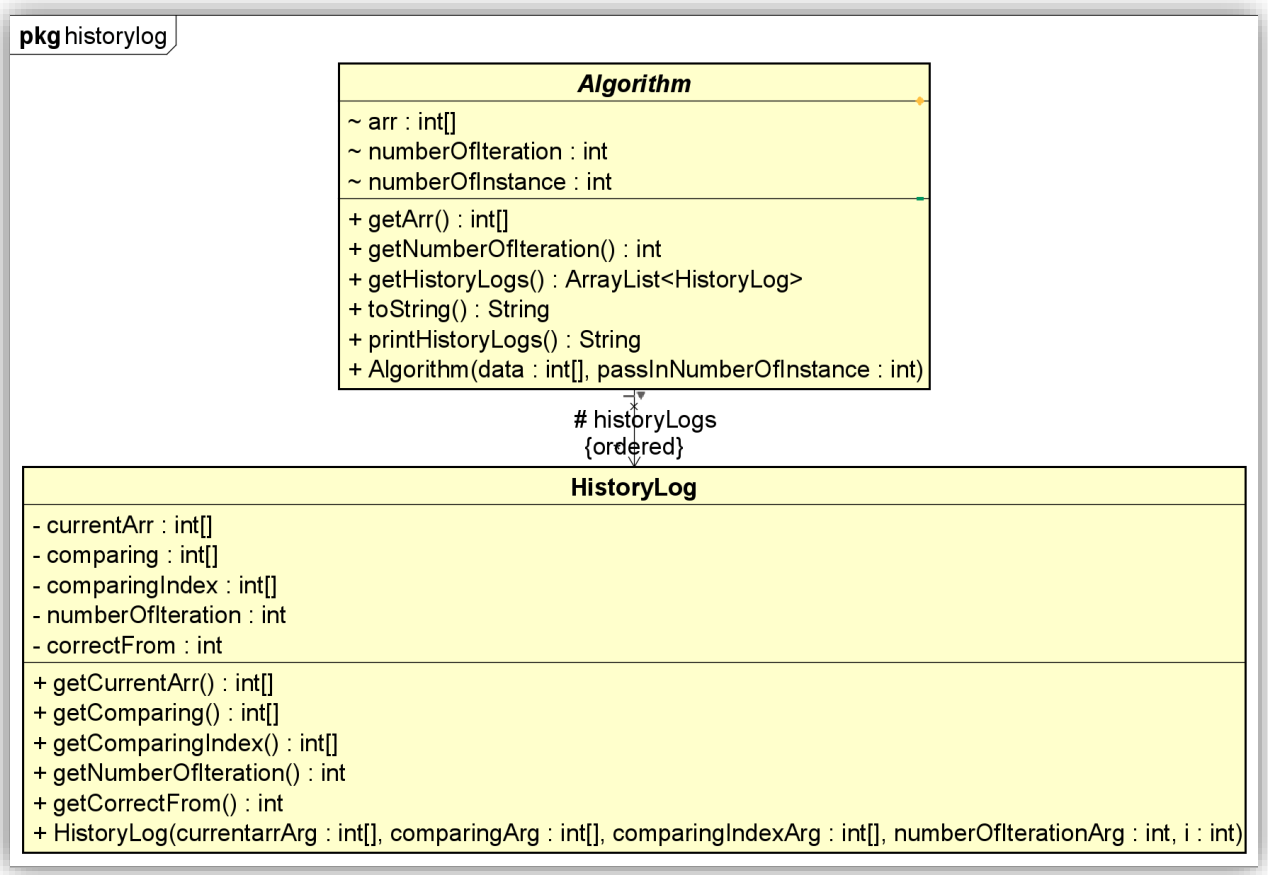


Class name	Attribute	Operation
DataController	- **data**: the array that our algorithm is working with - **NUMBER_OF_INSTANCE**: number of elements in our array.	- **getData():** return the array at the current state - **setData(int[] data):** change the array with the array in parameter - **getNUMBER_OF_INSTANCE():** return the number of elements in our array - **setNUMBER_OF_INSTANCE(int nUMBER_OF_INSTANCE):** change the number of elements in our array - **generateData():** generate an array based on the number of elements

		- **shuffleData():** randomly shuffle the elements in the array - **toString():** return the current state of the array

Attribute data and NUMBER_OF_INSTANCE can be accessed and modified through get and set method (Encapsulation).

3.2.4. Historylog:

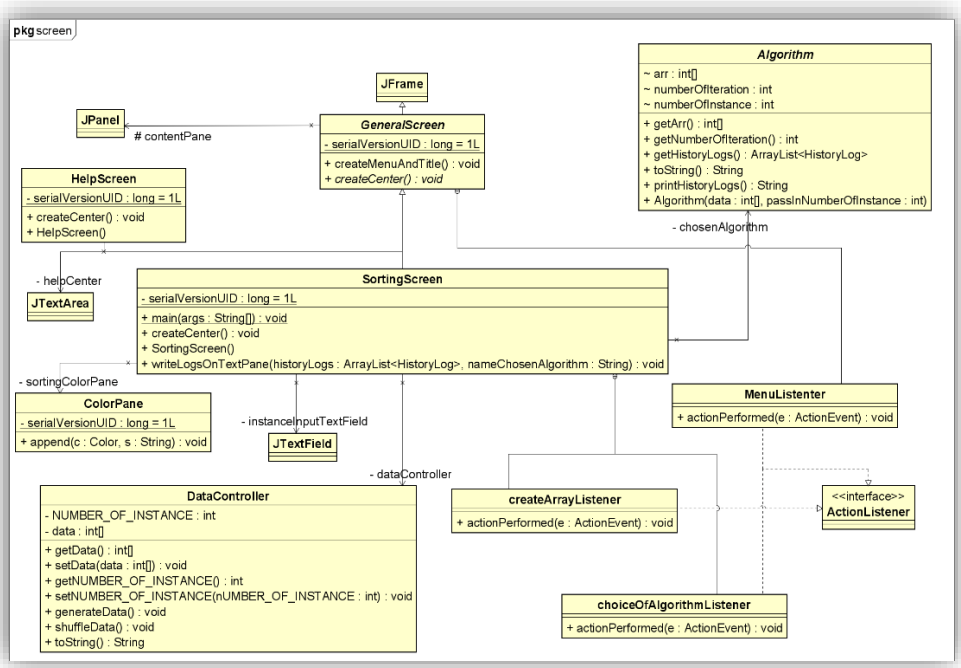


Class name	Attribute	Operation
HistoryLog	- **currentArr**: the state of the array at the time of recorded (when a HistoryLog is created)	- **getCurrentArr():** return currentArr attribute - **getComparing():** return comparing attribute

	- **comparing**: item inside int arr which are in comparison - **comparingIndex**: index of the item in comparison - **numberOfIteration**: number of iteration passed at the time recorded - **correctFrom**: used to append text of green color to the screen, indicates from which index that the arr is correctly sorted	- **getComparingIndex():** return comparingIndex attribute - **getNumberOfIteration():** return numberOfIteration attribute - **getCorrectFrom():** return correctFrom attribute - **HistoryLog(int[] currentarrArg, int[] comparingIndexArg, int numberOfIterationArg, int i):** constructor method of HistoryLog class

All attributes in this class are private attributes and can only be accessed through get method (Encapsulation).

3.2.5. Screen:

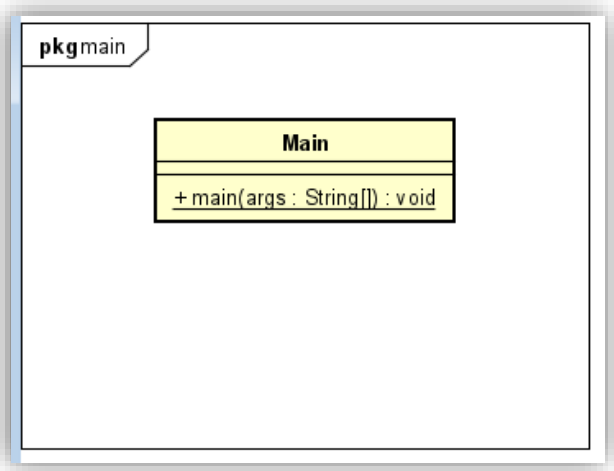


Class name	Attribute	Operation/inner classes
GeneralScreen	- **serialVersionUID**: help JVM identify the state of the object when it reads the state of the object from a file - **contentPane():** a JPanel that contains all the main objects of the screen	- **createMenuAndTitle():** create the menu and title for our application - **createCenter():** create center component for the screen - **MenuListener**: a subclass that defines listeners for menu events
SortingScreen	- **dataController**: an object for controlling and manipulating our array - **chosenAlgorithm**: the sorting algorithm being demonstrated - **instanceInputTextField**:	- **main():** run the sorting screen - **createCenter():** create center component for SortingScreen - **writeLogsOnTextPane():** change the text in ColorPane

	a JTextField for the user to input the number of elements - **sortingColorPane**: a ColorPane used for demonstrating sorting algorithm	with a history of algorithm's changing states - **createArrayListener:** an inner class for handling the create array command - **choiceOfAlgorithmListener:** an innerclass for choosing a sorting algorithm - **SortingScreen():** constructor method of SortingScreen class
HelpScreen	- **helpCenter**: a JTextArea to show the user the basic usage and aim of the program	- **createCenter()**: create center component for HelpScreen - **HelpScreen():** constructor method of HelpScreen class

GeneralScreen is an abstract class (Abstraction) that inherits JFrame class from Java Swing (Inheritance). In this class, there is an abstract method createCenter() (Abstraction) for the child classes to use and an inner class MenuListener that implements ActionListener interface. Both HelpScreen and SortingScreen inherit GeneralScreen (Inheritance) and has its own process of createCenter() method (Polymorphism).

3.2.6. Main:



Class name	Attribute	Operation
Main		- **main(String[] args):** run the program

4. Member task:

- Hoàng Trần Nhật Minh – 20204883:

+ Use case diagram

+ Shellsort class

+ Package arrangement

- Dương Vũ Tuấn Minh – 20209705:

+ Class diagrams

+ Screen package

+ Bubble sort class

- Nguyễn Trần Xuân Mạnh – 20200385:

+ Heapsort class

+ Slide presentation

- Nguyễn Quang Minh – 20204884:

+ Report writing

+ Sort algorithms research

5. References and external links:

Wikipedia contributors. (23 June 2022). *Heapsort.* Wikipedia.

https://en.wikipedia.org/wiki/Heapsort

Wikipedia contributors. (2 June 2022). *Shellsort.* Wikipedia.

https://en.wikipedia.org/wiki/Shellsort

Anonymous. (n.d.). *Check if an array represents a min-heap or not.* Techie Delight.

https://www.techiedelight.com/check-given-array-represents-min-heap-not

Anonymous. (n.d.). *Extension of JTextPane that allows the user to easily append colored text to the document : TextPane.* Java2s.

http://www.java2s.com/Code/Java/Swing-JFC/ExtensionofJTextPanethatallowstheusertoeasilyappendcoloredtexttothedocument.htm?

Anonymous. (n.d.). *Shell Sort in Data Structure.* TechVidvan.

https://techvidvan.com/tutorials/shell-sort/