Project 4: Sample Runs

Example 1: Displaying a Graph

Once you have the Graph class working, you can run the main program to test and debug. The main program is **`SimGUI`**, which can be compiled and run with the following commands if you are on Windows:

```
javac -cp .;../310libs.jar *.java
java -cp .;../310libs.jar SimGUI
```

or the following commands if you are on Linux/MacOS:

```
javac -cp .:../310libs.jar *.java
java -cp .:../310libs.jar SimGUI
```

Why is there extra stuff? The **`-cp`** is short for **`-classpath`** (meaning "where the class files can be found). The **`.;../310libs.jar`** or **`.:../310libs.jar`** has the following components: **`.`** the current directory, **`;`** or **`:`** the separator for Windows or Linux/MacOS respectively, **`310libs.jar`** the provided jar file which contains the library code for JUNG. By default, the **`.jar`** file is placed right above the folder with all .java files of P4. If you have moved it, you will need to use a different classpath to run **`SimGUI`**.

If you run the simulator with the above command, you will get a six-node graph with some random edges. Each time you hit "reset" you get another graph, but the same sequence of graphs is always generated (for your testing). However, the simulator can also be run with some additional optional parameters to get some more interesting results: The number of nodes, the likelihood that two nodes have an edge between them, the random seed for the graph generator. The next few tables give examples of what you can do.

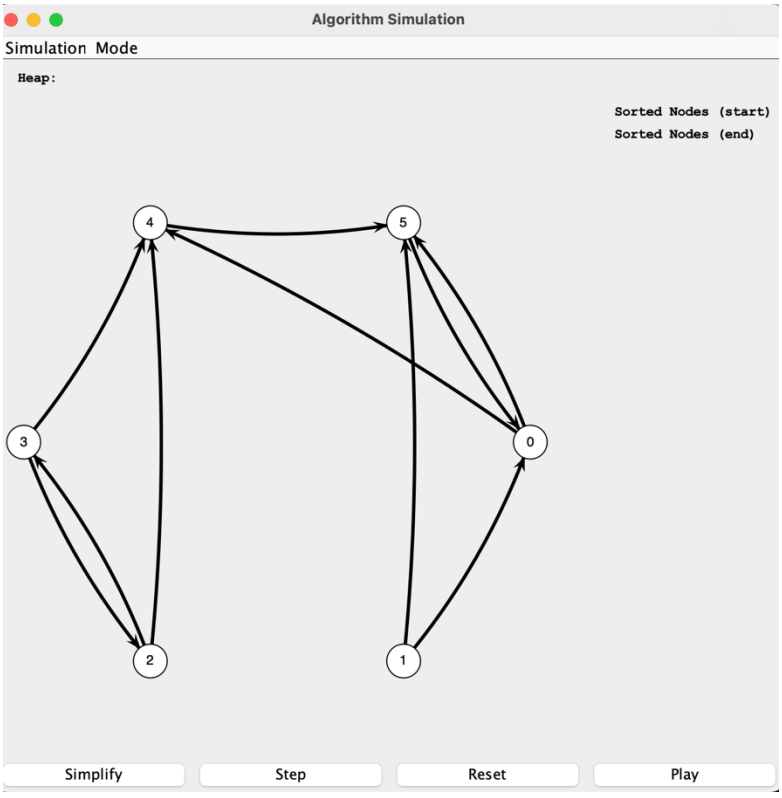
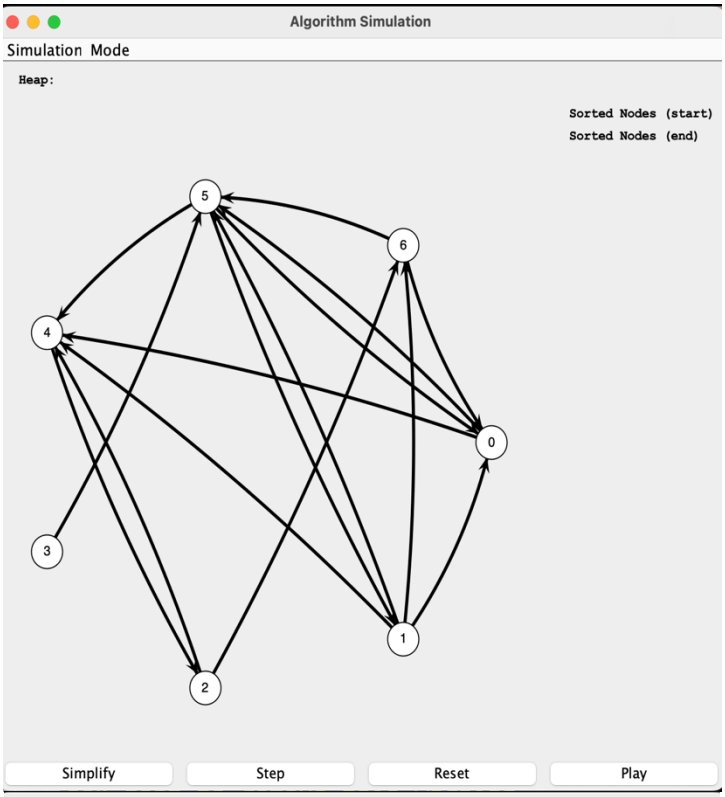
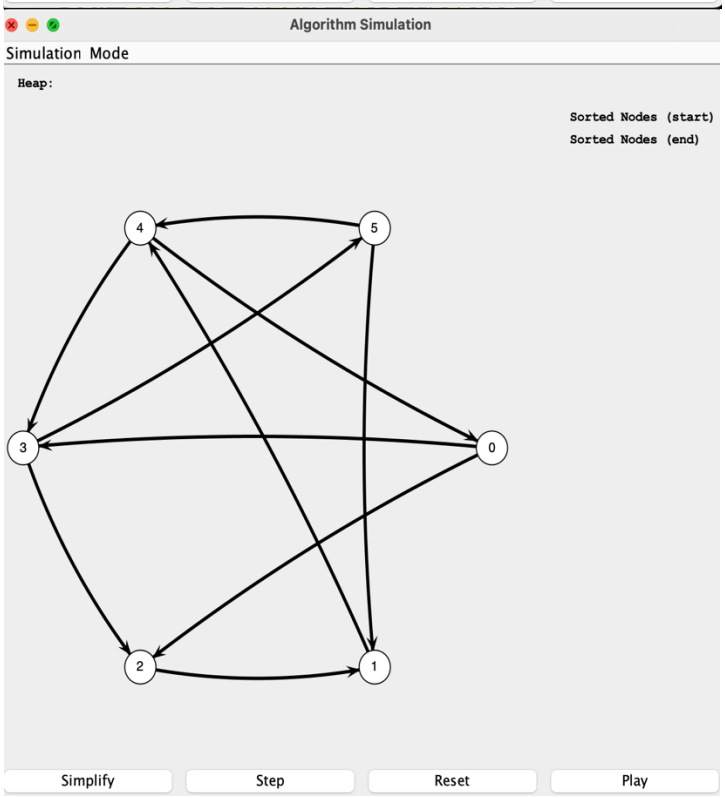
Image	Command +Explanation
	**`java -cp .;../310libs.jar SimGUI`** Generate a six-node graph, with connection probability of 0.4, and seed 0.

Image	Command +Explanation
	`java -cp .;../310libs.jar SimGUI 7 0.5` Generate a seven-node graph where nodes have a 50% chance of being connected.
	`java -cp .;../310libs.jar SimGUI 6 0.4 23` Generate a different sequence of graphs using seed 23.

Example 2: Adding nodes and edges to a graph

You'll want to test out adding multiple nodes and edges to make sure you've gotten out all the bugs.

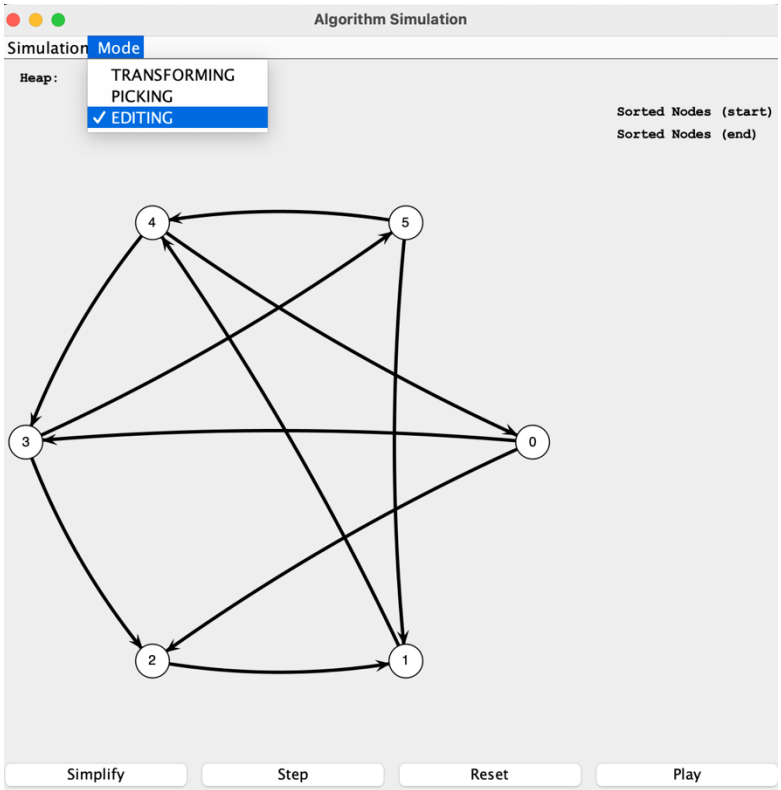
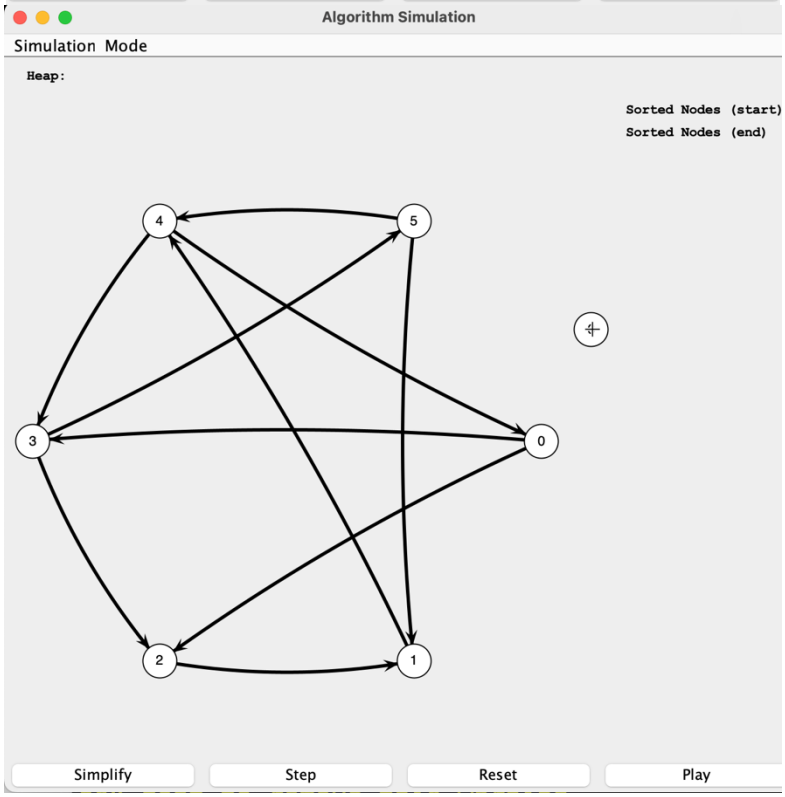
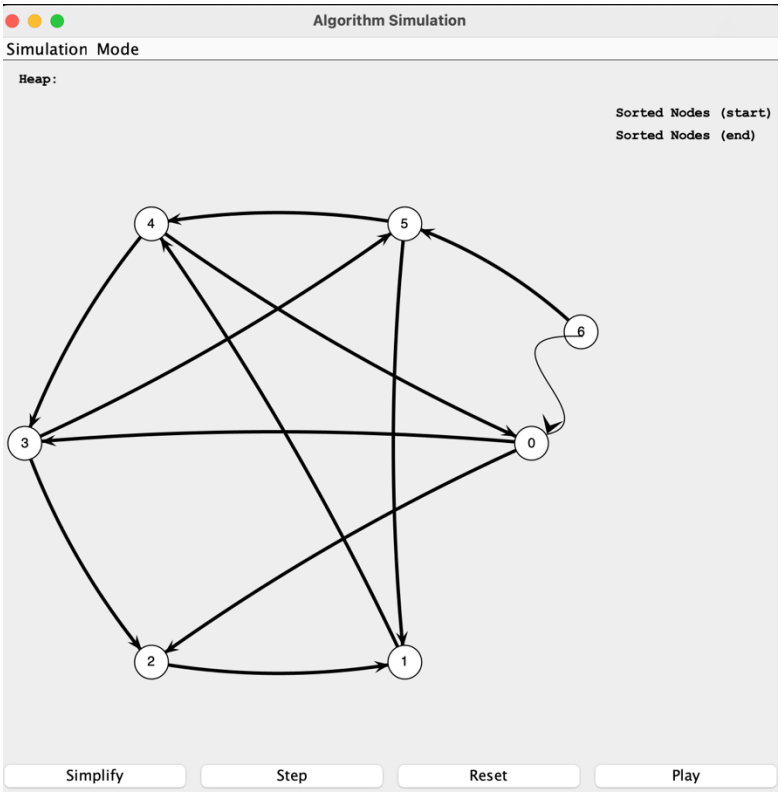
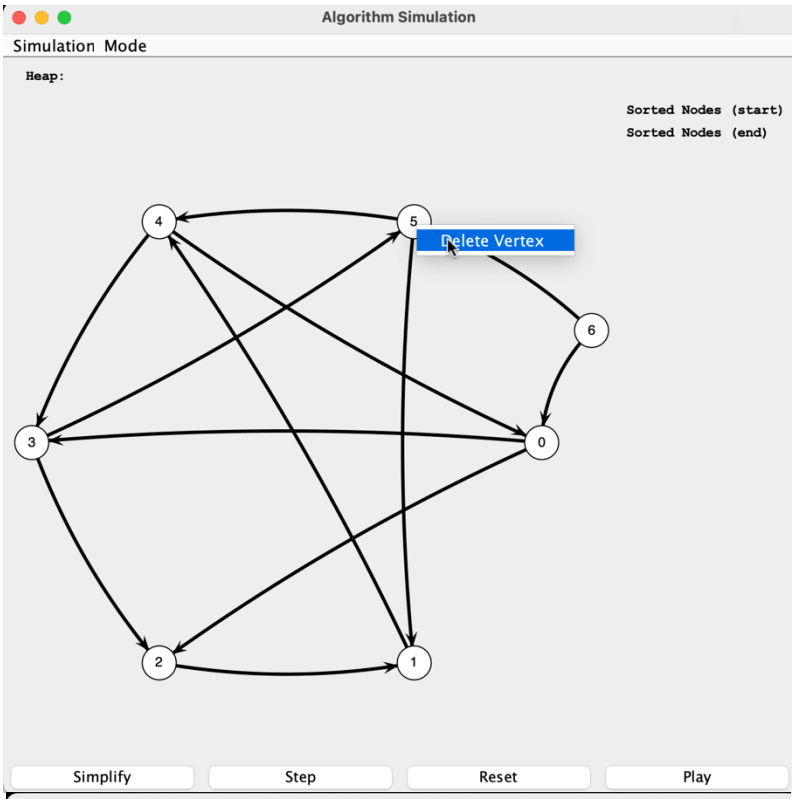
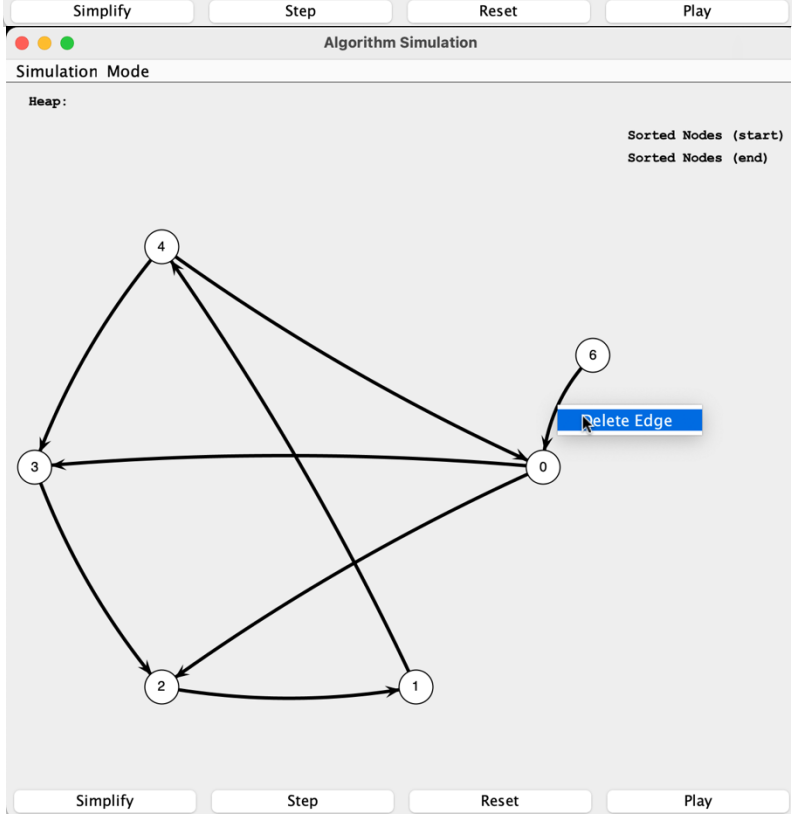
Image	Command +Explanation
	Select "Mode", then "Editing".
	Click anywhere on the graph surface to add a node.

Image	Command +Explanation
	Drag from one node to another node to add an edge. (No self-loops nor parallel edges allowed.)

Example 3: Removing a nodes and edges from a graph

Image	Command +Explanation
	Right click a node and select "Delete Vertex".
	Right click an edge and select "Delete Edge".

Example 4: Topological Sorting in Simulator

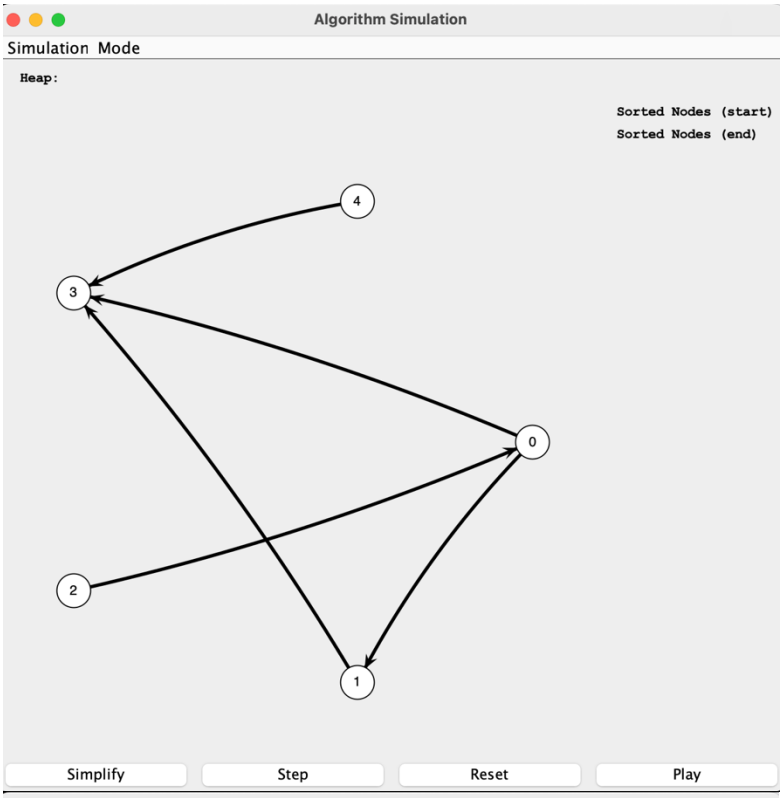
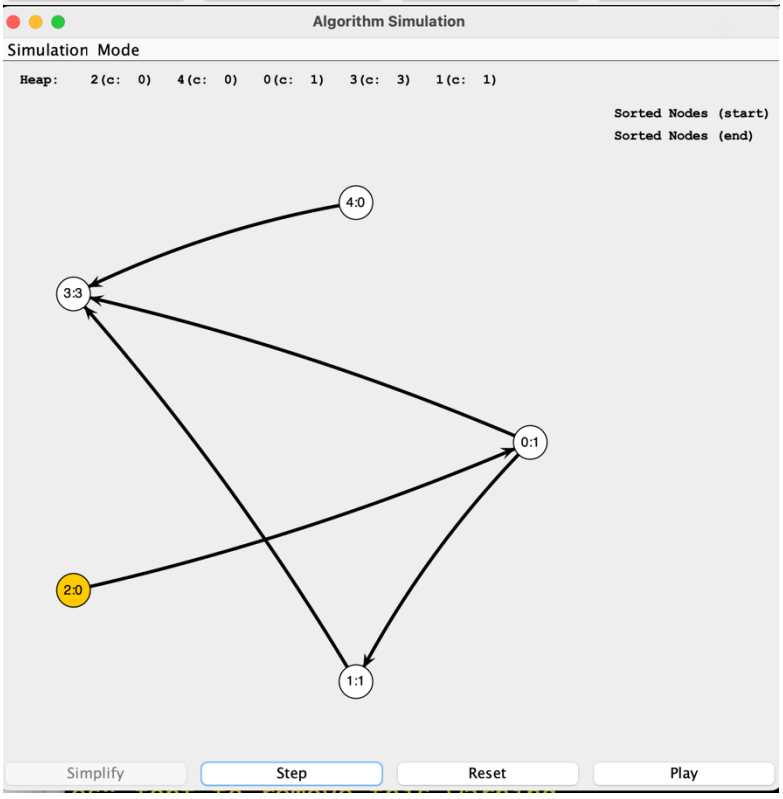
Image	Command +Explanation
	Set up your graph (empty heap and empty sorted list). Graph generated by command: `java -cp .:../310libs.jar SimGUI 5 0.4 15`
	Hit "Step": nodes should display `ID:cost`; all ndoes added to heap; node of lowest indegree highlighted; node ID used to break the tie.

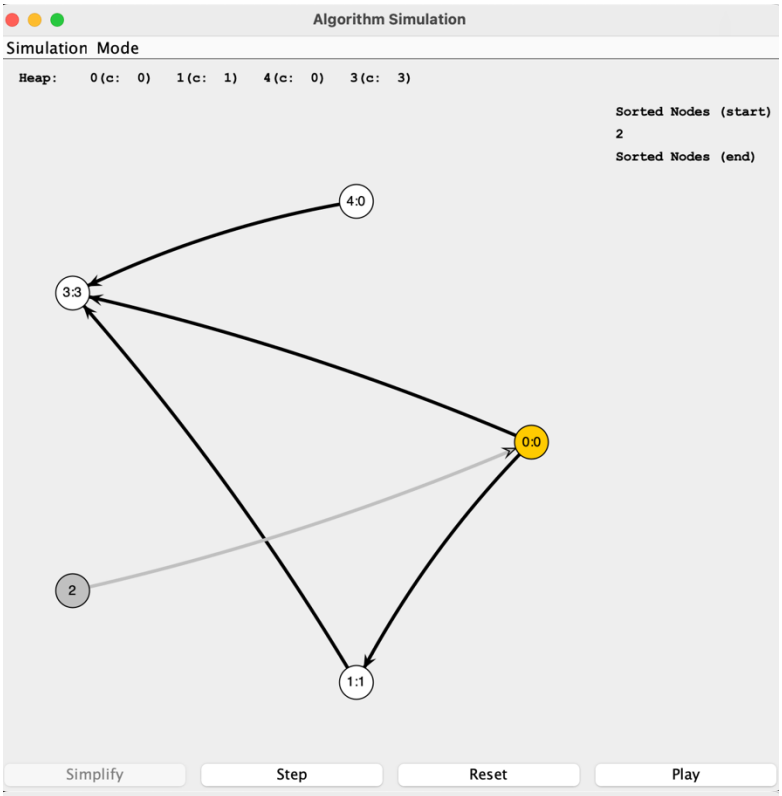
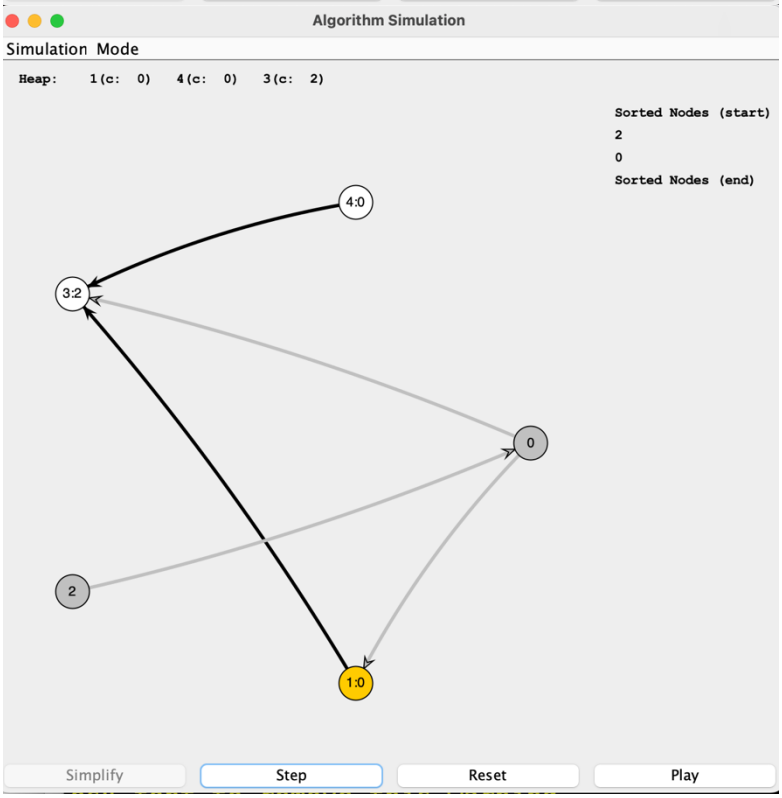
Image	Command +Explanation
	Hit "Step": min node added to sorted list with its edges inactive; successors show their updated cost; next min node highlighted, repeat ...
	Hit "Step": same as above, repeat …

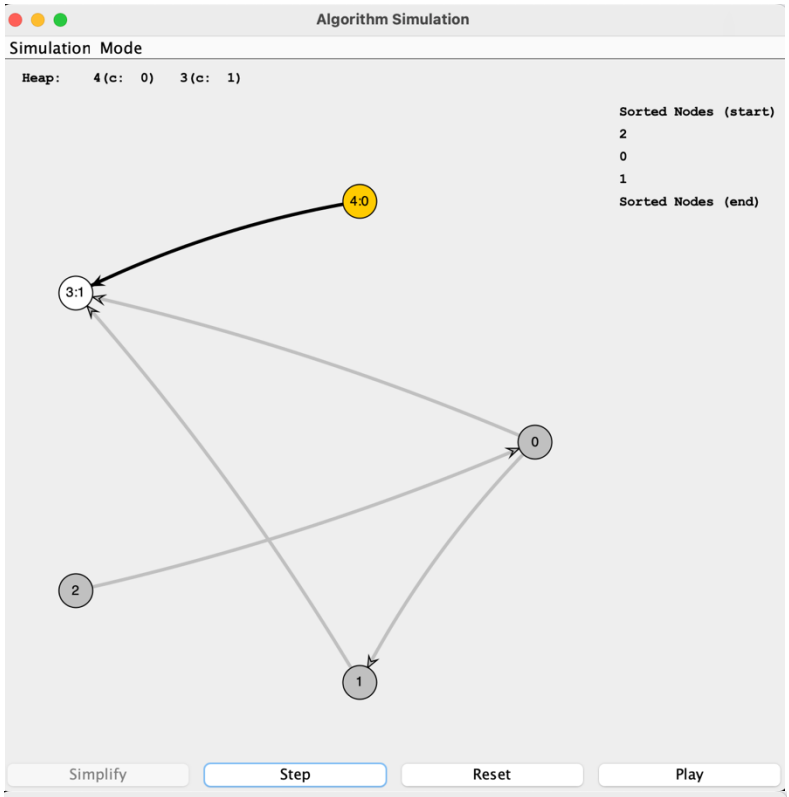
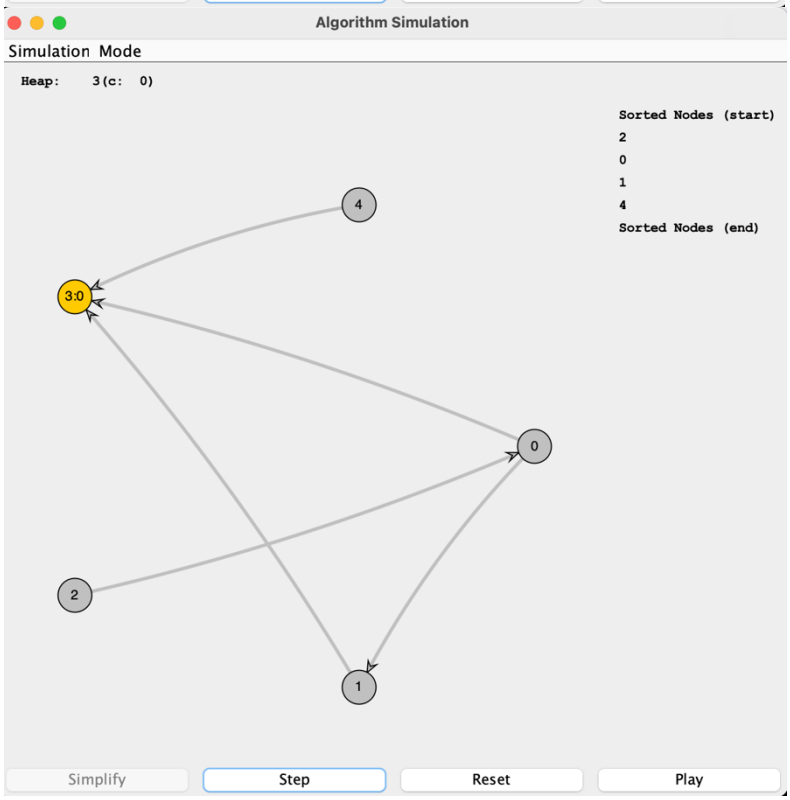
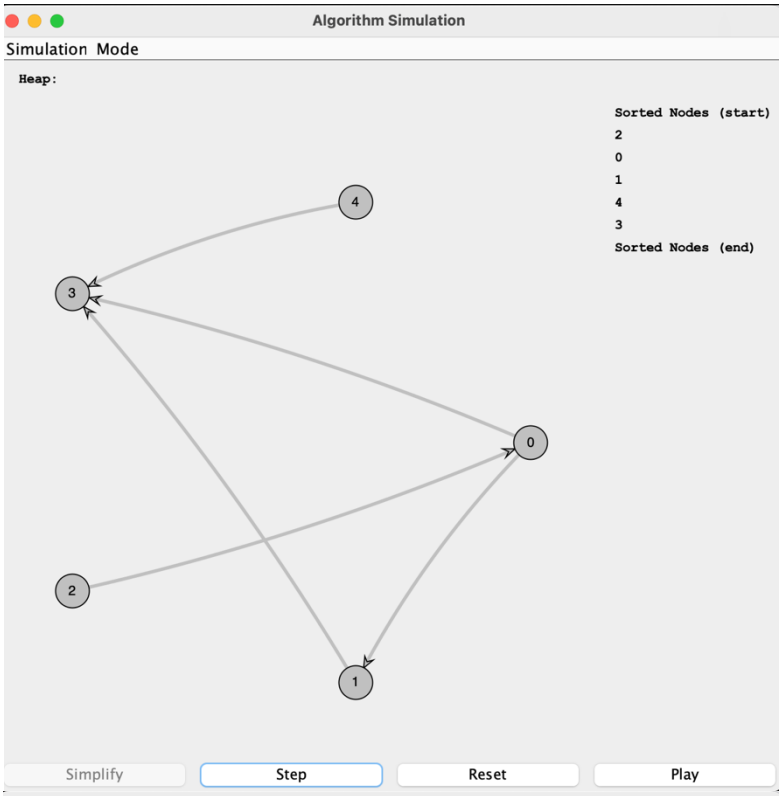
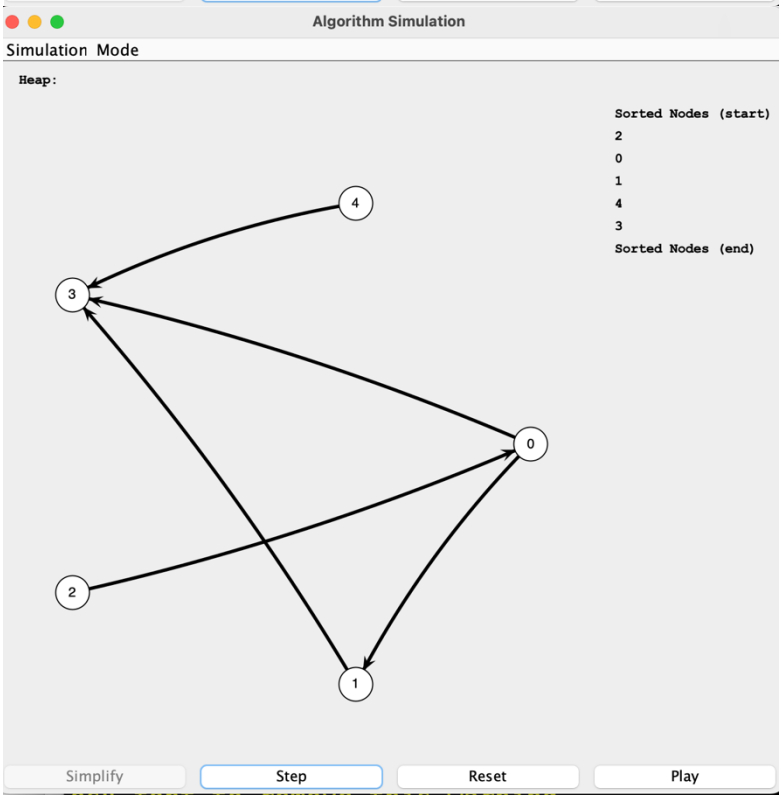
Image	Command +Explanation
	Hit "Step": same as above, repeat …
	Hit "Step": node 3 can finally be scheduled after all three predecessors {0, 1, 4} are done. This is the last node to sort.

Image	Command +Explanation
	Hit "Step": All nodes are topologically sorted. Heap is empty.
	Hit "Step": All nodes recover original state after sorting completes. Sorted list show the final result.

Example 5: Graph with a cycle: topological sorting not applicable to all nodes

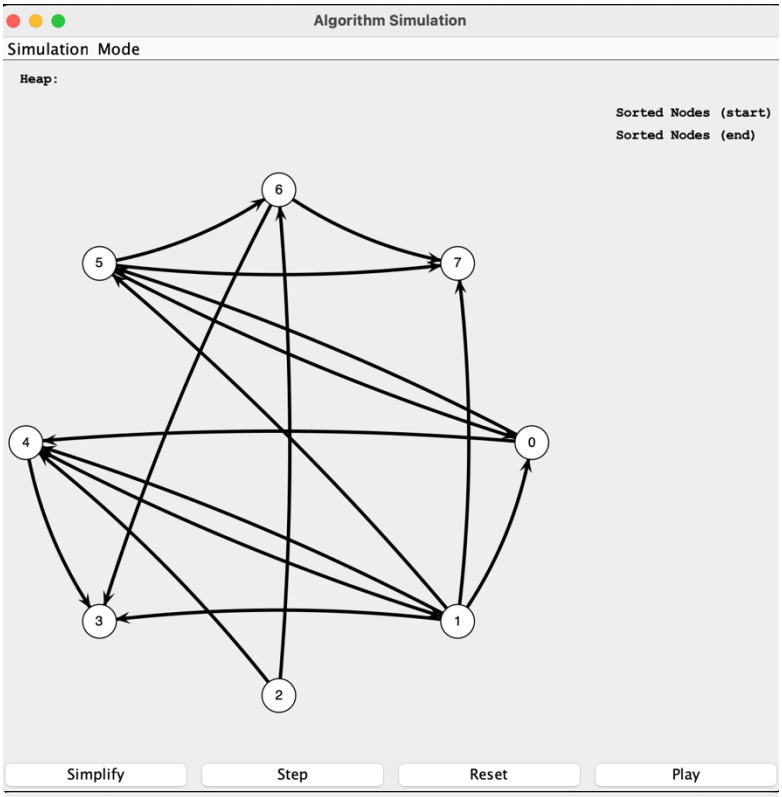
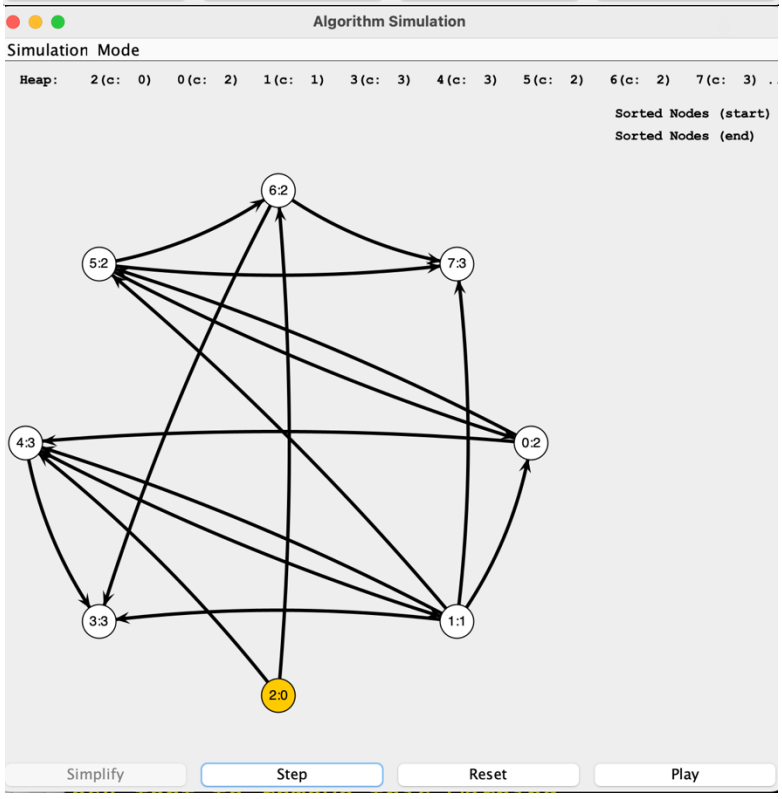
Image	Command +Explanation
	Set up your graph (empty heap and empty sorted list). Graph generated by command: `java -cp .:../310libs.jar SimGUI 8 0.4 0`
	Hit "Step": nodes should display `ID:cost`; all ndoes added to heap; node of lowest indegree highlighted.

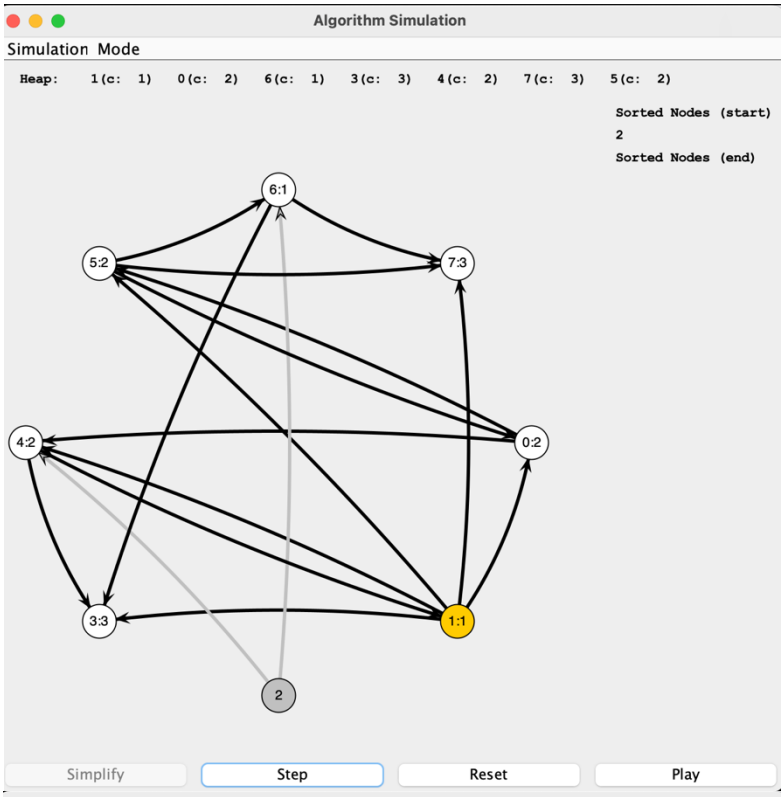
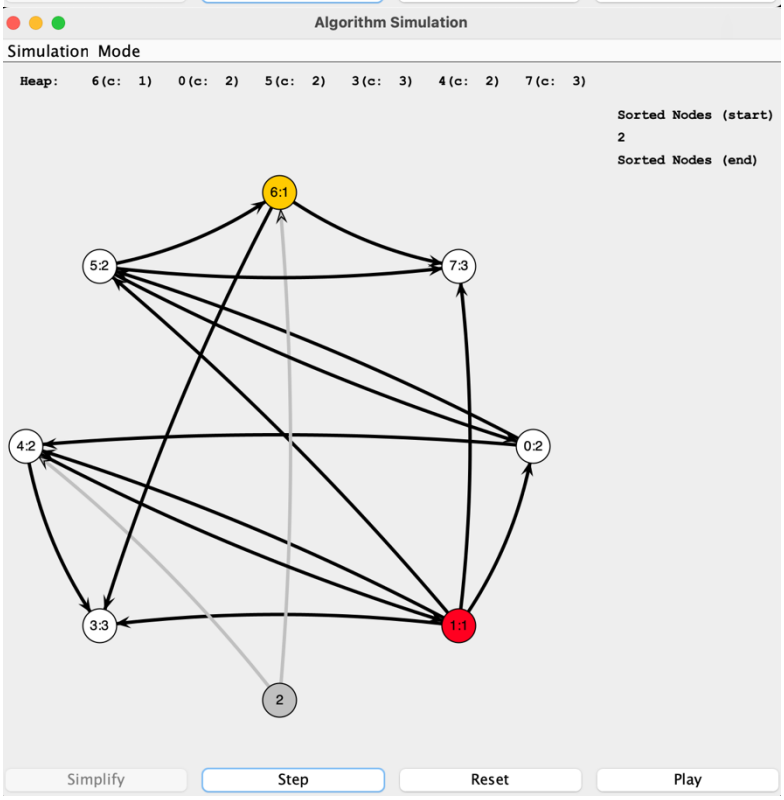
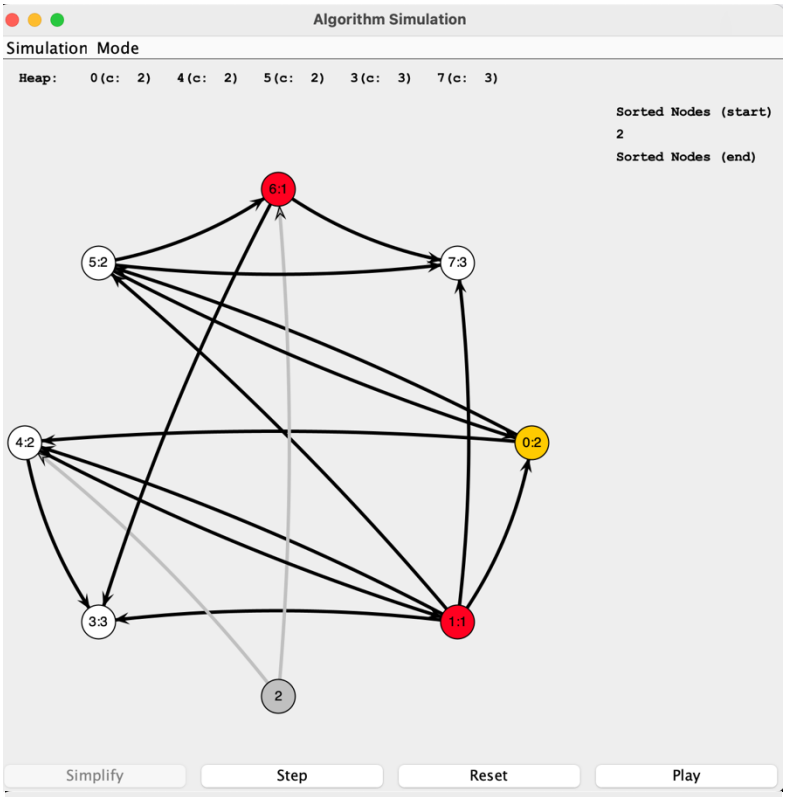
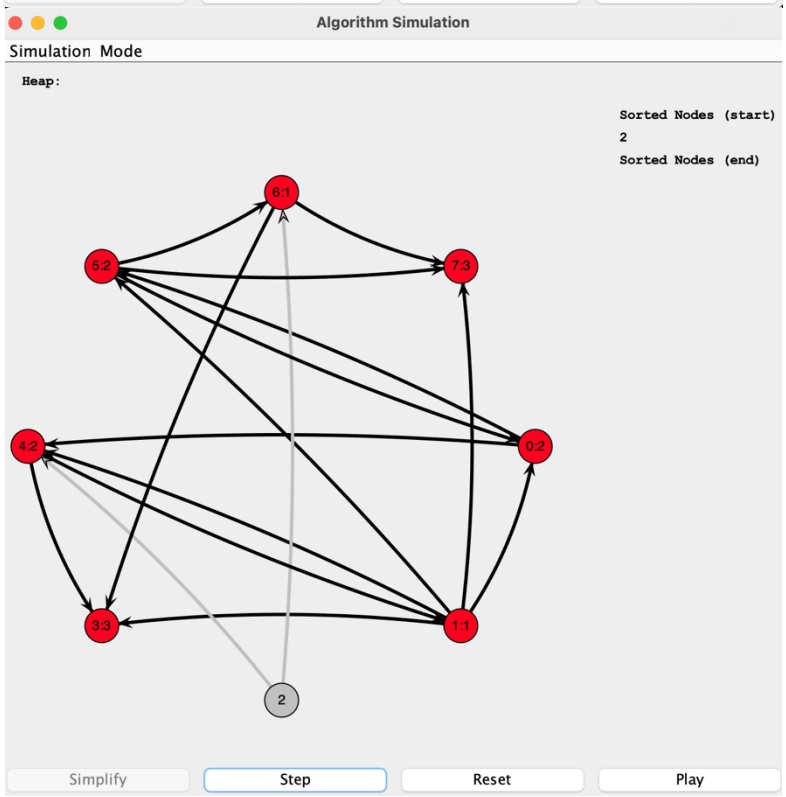
Image	Command +Explanation
	Hit "Step": min node added to sorted list with its edges inactive; neighbors show their updated cost; next min node highlighted. Now the lowest indegree of remaining nodes is >0, hence they cannot be sorted.
	Hit "Step": We still process every node one by one in their heap order. But since their remaining indegree is >0, the node is marked with the red warning color (`ThreeTenColor.COLOR_WARNING`). Node not added into the sorted list. Edges of the node not greyed out; cost of its successors not updated.

Image	Command +Explanation
	Hit "Step": node with a cost >0, same as above, repeat …
	Hit "Step" multiple times until all nodes processed. Heap empty.

Example 6: Simplify a Graph before Topological Sorting

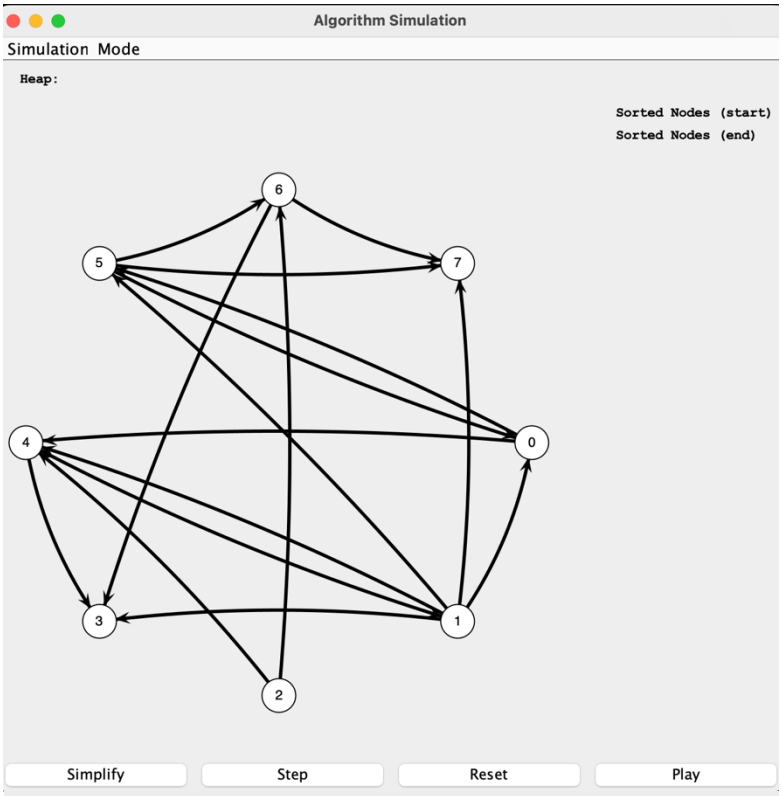
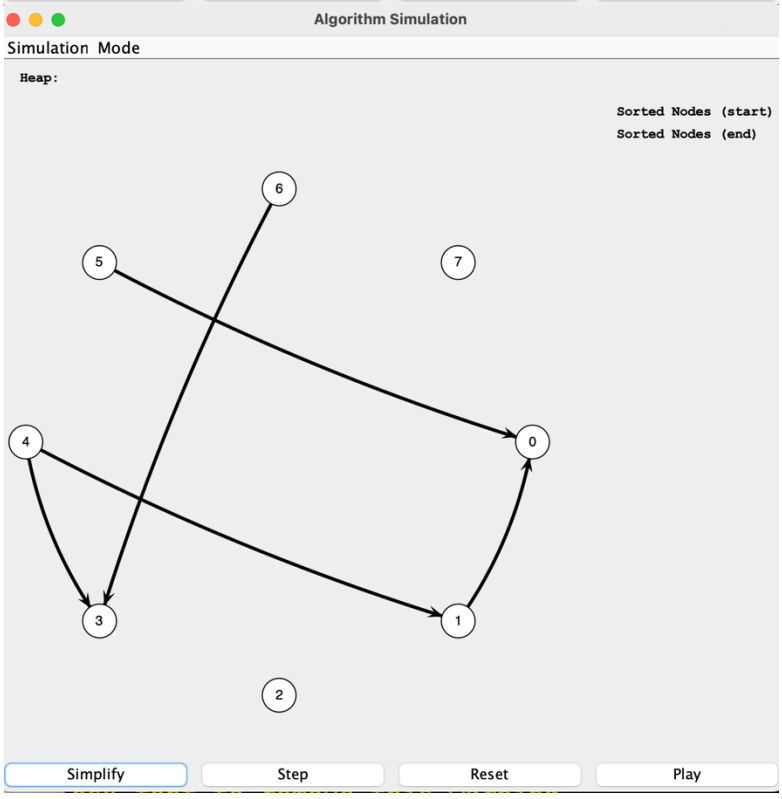
Image	Command +Explanation
	Set up your graph (empty heap and empty sorted list). Graph generated by command: `java -cp .:../310libs.jar SimGUI 8 0.4 0`
	Hit "Simplify": all edges that point from a lower ID node to a higher ID node (e.g. 2→4, 2→6) are deleted. This essentially ensure there is no cycle in the graph.

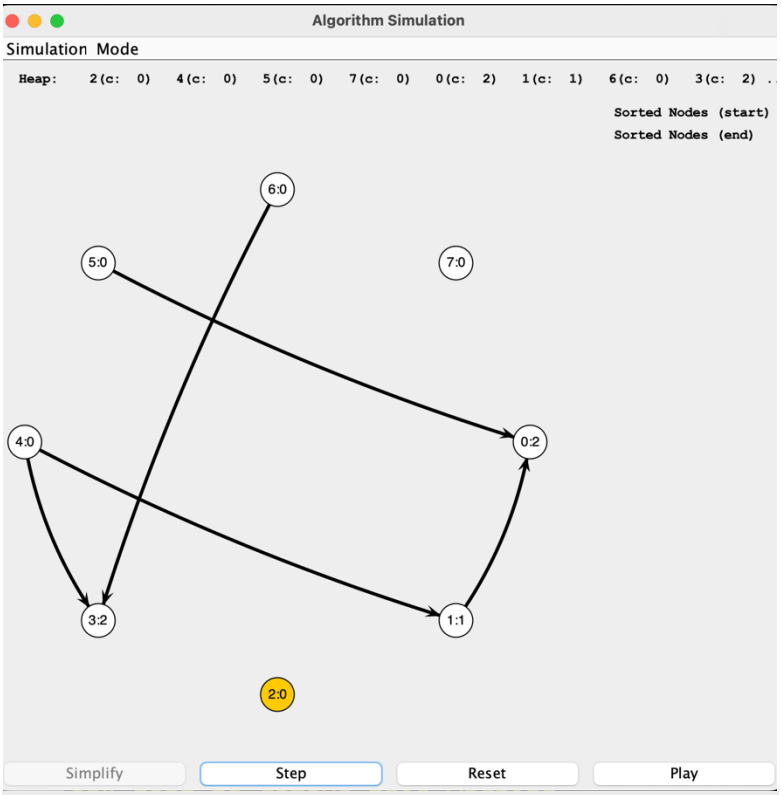
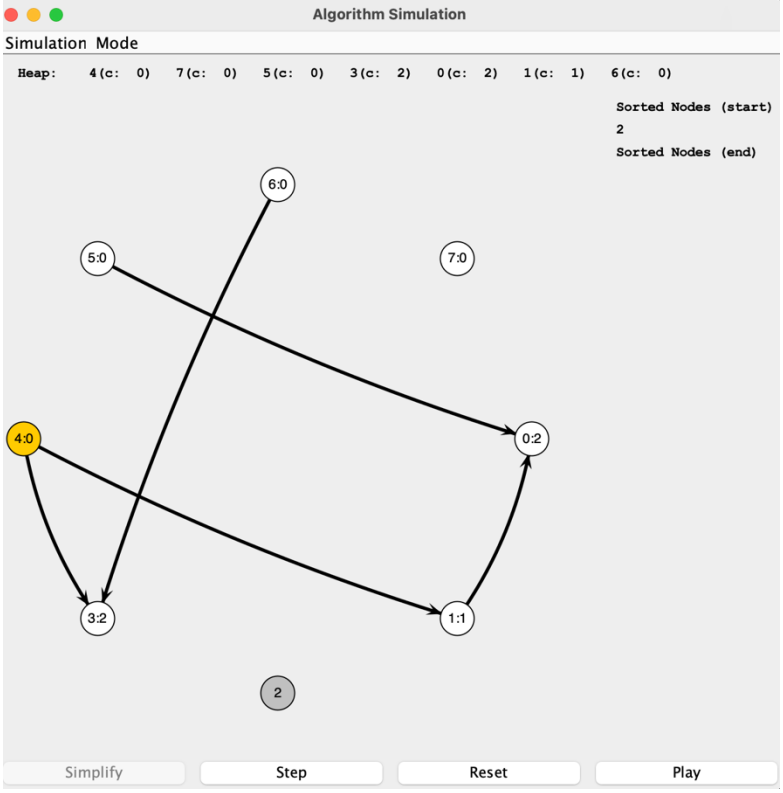
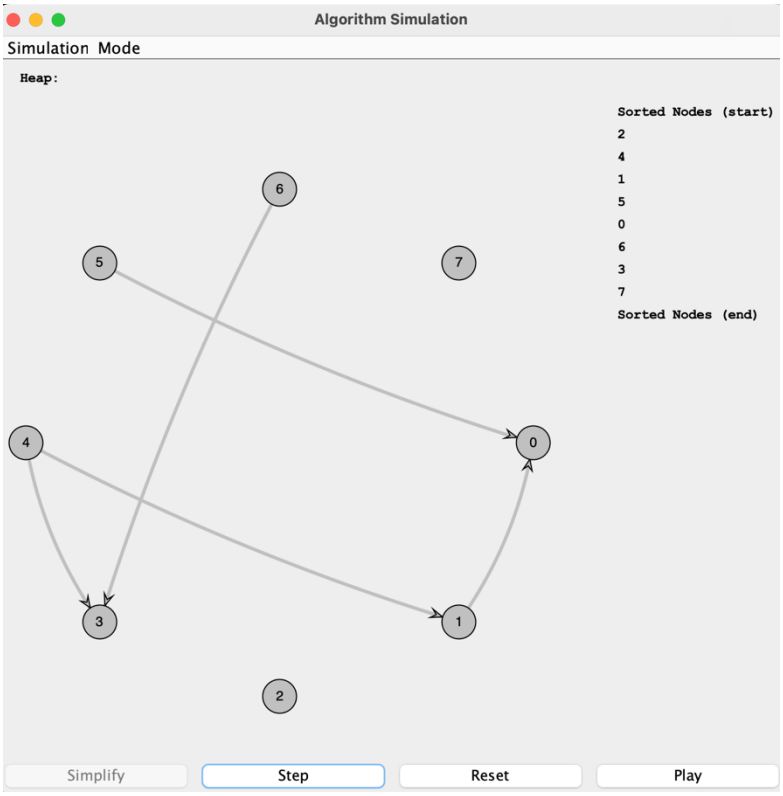
Image	Command +Explanation
	Hit "Step": now the min node has no incoming edges (cost == 0). We will be able to apply topological sorting on the simplified graph.
	Hit "Step": min node added to sorted list; heap updated; next min node highlighted.

Image	Command +Explanation
	Hit "Step" multiple times until all nodes are sorted. Heap empty.