Programming Assignment #1
Airport Control Tower Simulation

By: Ayush Gaggar
CS 146, Section 4

Table Of Contents

Design and Implementation 2

Description of Classes and Methods 4

Demonstration of Meeting Functional Requirements 8

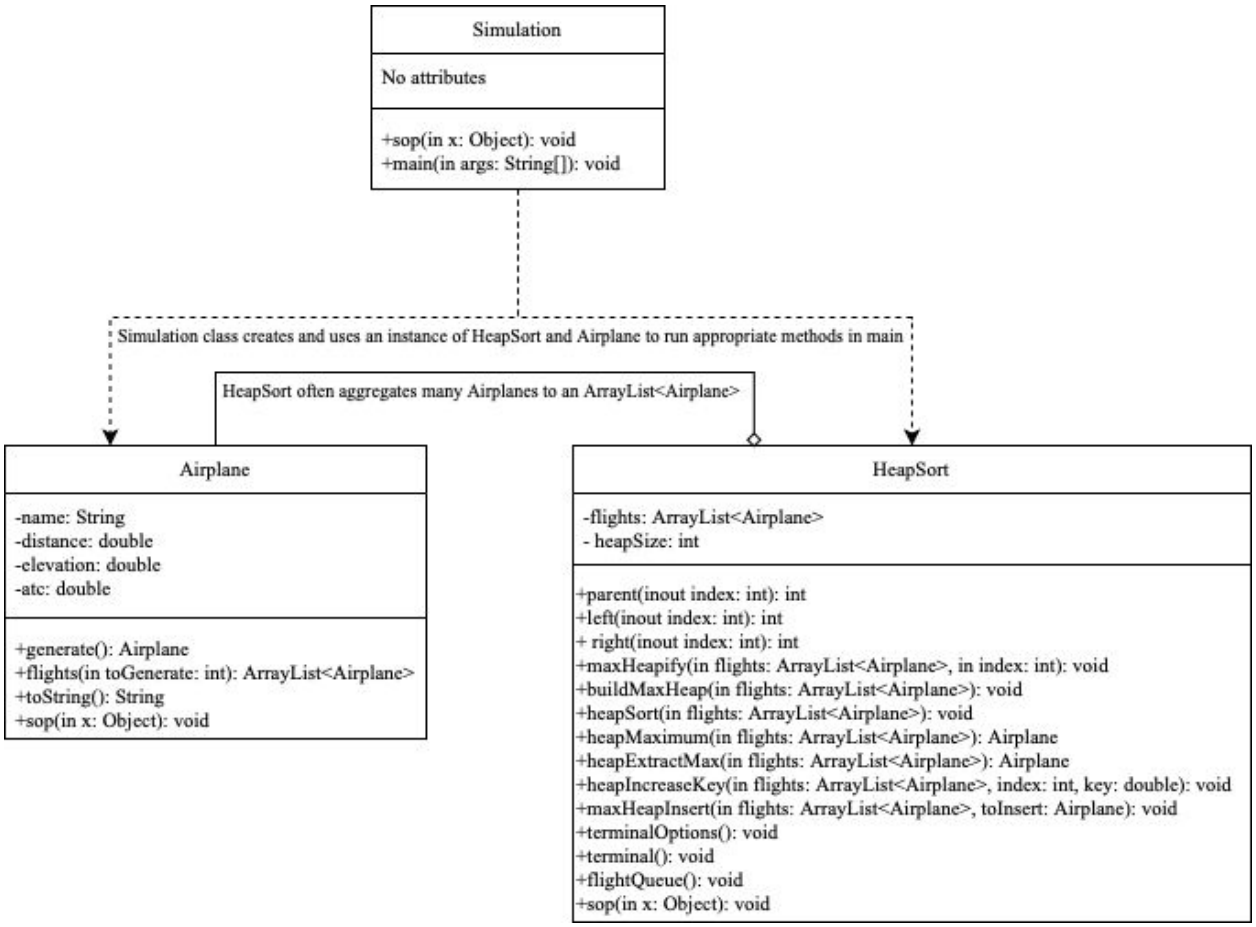
Installation Procedure 21

Problems Encountered During Implementation 22

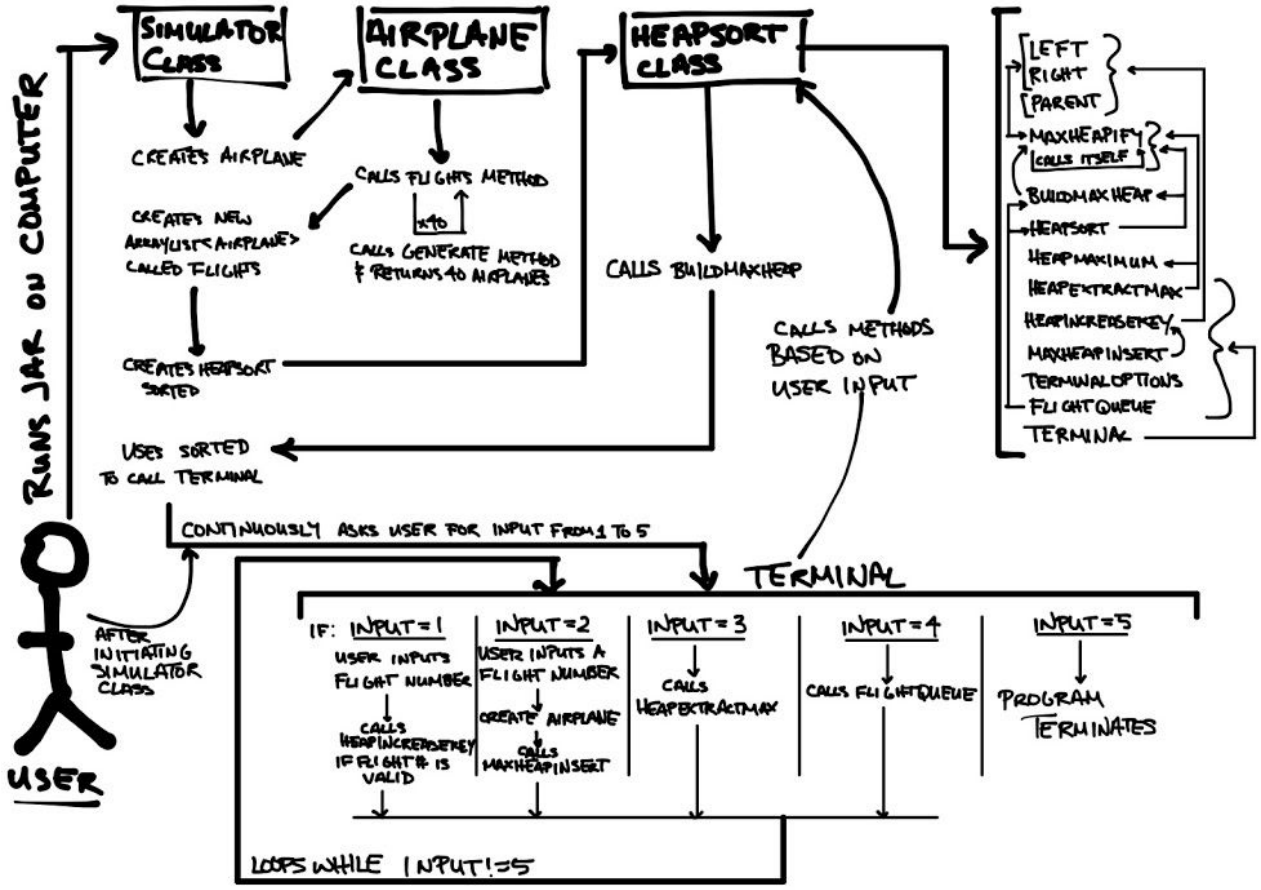
Lesson Learned 28

Design and Implementation

Below is a UML diagram detailing the 3 classes, as well as the variables and methods contained within each class. It's important to note that although Simulator is a class, its only purpose is to create, initialize, and run methods so that the user can interact with a max-heap of Airplanes.



A modified Sequence Diagram is shown on the next page, detailing how the user interacts with methods, and how methods interact with other methods before program termination. Follow the arrow leading from the User to the Simulator Class, and continue to follow the arrows until program termination. The arrows represent the order of execution, from tail to head; if multiple arrows are located at a specific box, read in order of left to right; within a specific box, read from top to bottom unless otherwise stated.



The methods within the HeapSort class are the most crucial parts to the entire programming assignment, so it's critical that an appropriate data structure is implemented and called upon by these methods. I chose to use an ArrayList aggregating objects of type Airplane (ArrayList<Airplane>) as the data structure the HeapSort methods will use. An ArrayList provides the flexibility to dynamically add and remove Airplanes to the end of the ArrayList, and also allows for a greater ease of functionality when swapping, setting, and getting Airplanes located at a specific index. A dynamic data structure should be used when building a max-heap; because heapSize is a dynamic variable, changing within methods, the length of the data structure should be equally dynamic and flexible. A further benefit of using an ArrayList is that it is much easier to iterate through the entire List, using the iterator functionality built into it. Further analysis as to why the use of ArrayLists was chosen to implement the max-heap and priority queue methods is discussed in the Problems Encountered section of this report.

Description of Classes and Methods

Class 1: Airplane

- 4 private variables: String name, double distance, double elevation, double atc
- Constructors:
    - Takes in arguments: String name, double distance, double elevation
        - Initializes each variable: sets “flight number” to name, “direct distance to runway” to distance, and “elevation of plane” to elevation.
        - Sets atc = 15000 - (distance + elevation)/2
    - Takes in argument: Airplane
        - Sets all class variables equal to Airplane’s variables
- Methods: all methods are public unless otherwise specified
    - Setters and getters for each private variable (8 total)
    - generate: Returns Airplane
        - Used to create a randomly generated Airplane, i.e. name, distance, and elevation are all created using a random number generator (import java.util.Random, and calls random.nextInt(range)).
    - flights: Returns ArrayList<Airplane>
        - Calls generate method toGenerate times and adds each randomly generated airplane to an ArrayList<Airplane> called incomingFlights, which it then returns.
    - toString: Returns String
        - Calls get methods for all variables, and returns a String displaying information for a given Airplane cleanly and legibly.
    - sop: void, no return type
        - A helper method which simply calls System.out.println() in order to print the input object

Class 2: HeapSort

- 2 private variables: ArrayList<Airplane> flights, int heapSize
- Constructors:
    - Takes in arguments: ArrayList<Airplane> flights, int heapSize
    - Sets class variable flights and heapSize to the input arguments
- Methods: all methods are public unless otherwise specified
    - Setters and getters for each private variable (4 total)
    - parent: Returns int
        - For the input parameter index, returns the location of where the index's parent is located in the ArrayList.
    - left: Returns int

    - For the input parameter index, returns the location of where the index's left child is located in the ArrayList.
- right: Returns int
    - For the input parameter index, returns the location of where the index's right child is located in the ArrayList.
- maxHeapify: void, no return type
    - Takes the input parameter of an ArrayList<Airplane> flights and an int index in order to rearrange the ArrayList flights such that the Airplane located at the input index would satisfy the max-heap property.
    - In other words, this method ensures the max-heap property is met for the given input "node" located at the specified index.
        - The Max-Heap Property states that the Airplane's key at the given node must be less than or equal to the key of that Airplane's parent node. Each Airplane's "key" is specified by the value of its ATC variable.
- buildMaxHeap: void, no return type
    - At the start of this method, set the heapSize to the number of Airplanes (nodes) in flights.
    - Calls maxHeapify method for all PARENT (root) nodes for the input ArrayList<Airplane> flights. By the end of this method, all Airplanes in flights satisfy the max-heap property.
- heapSort: void, no return type
    - Sorts the input ArrayList<Airplane> called flights by calling buildMaxHeap. (flights already satisfies the max-heap property, and so can easily be reversed in order to sort ArrayList flights, based on their key, from low to high.
- heapMaximum: Returns Airplane
    - Returns the Airplane with the highest key, i.e. the root node. Assumes that the ArrayList passed in already satisfies the max-heap property.
- heapExtractMax: Returns Airplane
    - Extracts (removes) and returns the Airplane with the highest key (i.e. the root.
    - Sets the new root node as the last node in the ArrayList, and calls maxHeapify on the new root node.
    - Sets heapSize and removes the last node from the input ArrayList.
    - Assumes the input satisfies the max-heap property.
- heapIncreaseKey: void, no return type
    - "Force changes" the key of an Airplane at the specified index to the new specified key.

        - If the new key is smaller than the current key, it prints an error, and does not increase the current key to the new key.
        - Rearranges the ArrayList such that the Airplane at the specified index, with a new key, will satisfy the max-heap property at its new location.
    - maxHeapInsert: void, no return type
        - Inserts the input Airplane into the input ArrayList<Airplane> called flights, given that all nodes in flights satisfy the max-heap property.
        - After adding the airplane to the end of flights, calls heapIncreaseKey to move the Airplane to its appropriate location in the max-heap.
    - terminalOptions: void, no return type
        - Calls method sop to print the options the user can select from
    - terminal: void, no return type
        - Directs the user to choose one of the options from terminalOptions, and responds according to the option by calling the correct methods.
        - The print statements embedded within terminalOptions and within this method use method sop, and allow for a seamless and easy interface for the user to interact with when testing heapSort and Airplane methods.
    - flightQueue: void, no return type
        - A helper method, called by terminal, to print out all the airplanes in the heapSort's current “heap” (the flights ArrayList) using toString for each Airplane.
    - sop: void, no return type
        - A helper method used to make printing information easier and faster; a shorthand method which has the same functionality as System.out.println

Class 3: Simulator

- The Simulator class is used as a runner class to test and run the necessary methods from both Airplane and HeapSort classes, and simulates the Air Traffic Controller Terminal this assignment is meant to replicate.
- No variables, and no explicit constructor calls
- Methods:
    - sop: void, no return type
        - A helper method used to make printing information easier and faster; a shorthand method which has the same functionality as System.out.println
    - main: void, no return type

- The main method is where all objects are created and where these objects call the necessary methods.
- First, main creates an empty plane, which calls the flights method from Airplane class, and stores the randomly created Airplanes in an ArrayList<Airplane> called flights.
- It then creates a an object heapSort, called sorted, from flights and flights.size(). heapSort calls buildMaxHeap to turn flights from an ordinary ArrayList to a max-heap.
- Main then "boots up" the terminal, and allows users to interact with the
- simulator, by calling the terminal method in the HeapSort class.
- This class terminates whenever the terminal method terminates, based on user interaction with the Air Tower Controller Simulator.

Demonstration of Meeting Functional Requirements

NOTE: For the purpose of conserving space and testing efficiently, examples of screenshot outputs often use a smaller sample size to demonstrate meeting requirements. Every method also works on larger sample sizes and heaps, such as those with more than 30 Airplanes.

Functional Requirements as stated in Programming Assignment #1:

1. Calculate and store Approach Code (AC) for each airplane:
   In the Air Traffic Control Simulator, the landing queue contains objects of type Airplane. In order to be an Airplane in the landing queue, the Airplane must be constructed using the following code (Constructor 1 from Airplane class):

```
public Airplane(String name, double distance, double elevation) {
    this.setName(name);
    this.setDistance(distance);
    this.setElevation(elevation);
    this.setATC(15000 - .5*(distance + elevation));
}
```

   The AC is represented by the variable atc in this code, and is calculated from the following equation: atc = 15000 - (distance + elevation)/2. The constructor stores the atc to its correct number, dependent on the input parameters of distance and elevation. The atc of any Airplane, and thus its Approach Code, can be retrieved by calling the getATC() method on the given Airplane. Further implementation of calculating the atc for an Airplane is demonstrated in sub-point number 2

2. Generate at least 30 random airplanes:
   In order to create random airplanes, a random integer generator is used by importing java.util.Random. A variable called “random” of type Random is created, as well as an Array called “alpha”, which contains all 26 letters.
   The generate method then randomly creates the following: a new String composed of 2 letters and a 4 digit integer, a random double between 3000 and 20000 to represent the distance to the runway in meters, and another random double between 1000 and 3000 to represent the elevation in meters. The generate method then assigns the randomly created variables as the input parameters for a new Airplane. This method then returns the new, randomly generated Airplane.
   Finally, the method “flights” creates a new ArrayList, with nodes of type Airplane called incomingFlights. The flights method then calls the generate method toGenerate times, each time adding the generated flight to incomingFlights.
   The code is shown below:

```
Random random = new Random();
String[] alpha = {"A", "B", "C", "D", "E",
                  "F", "G", "H", "I", "J",
                  "K", "L", "M", "N", "O",
                  "P", "Q", "R", "S", "T",
                  "U", "V", "W", "X", "Y", "Z"};
```

```java
public Airplane generate() {
    name = new String(
            alpha[random.nextInt(26)] +
            alpha[random.nextInt(26)] +
            Integer.toString(random.nextInt(9999) + 1));
    distance = random.nextDouble()*17000 + 3000;
    elevation = random.nextDouble()*2000 + 1000;
    return new Airplane(name, distance, elevation);
}

public ArrayList<Airplane> flights(int toGenerate) {
    ArrayList<Airplane> incomingFlights = new ArrayList<Airplane>();
    for (int counter = 0; counter < toGenerate; counter ++)
        incomingFlights.add(counter, this.generate());
    return incomingFlights;
}
```

This chunk of code was tested 40 times; i.e. flights method was called once with the input parameter toGenerate = 40, which in turn called the generate method 40 times. The output for this code is shown below, and demonstrates that at least 30 airplanes were generated, each with a Flight number, distance, and elevation, and that the AC for each was stored as well.

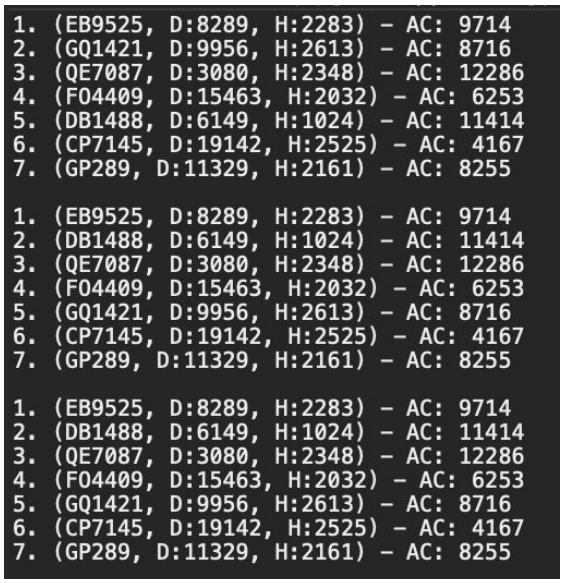
```
1. (ZW3432, D:9544, H:2399) - AC: 9028
2. (ST5734, D:6393, H:2897) - AC: 10355
3. (LM659, D:17701, H:1233) - AC: 5533
4. (TS4311, D:16804, H:1348) - AC: 5924
5. (WK806, D:3385, H:1922) - AC: 12346
6. (DX2665, D:19274, H:1023) - AC: 4852
7. (AF1991, D:12983, H:2383) - AC: 7317
8. (CM8729, D:5289, H:2424) - AC: 11144
9. (YP9738, D:17935, H:2556) - AC: 4755
10. (JH4520, D:10628, H:1182) - AC: 9095
11. (RS1804, D:11720, H:2034) - AC: 8123
12. (YS7846, D:13385, H:1567) - AC: 7524
13. (NC9906, D:12897, H:2149) - AC: 7477
14. (UU6637, D:16532, H:2243) - AC: 5612
15. (DE8391, D:6849, H:2554) - AC: 10299
16. (UF8603, D:6869, H:1117) - AC: 11007
17. (UC8083, D:9008, H:1630) - AC: 9681
18. (XP7772, D:13948, H:2369) - AC: 6842
19. (PE2319, D:17583, H:1994) - AC: 5212
20. (GY2365, D:12073, H:2326) - AC: 7800
```

```
21. (WU7442, D:9332, H:1873) - AC: 9397
22. (TQ1151, D:3197, H:2626) - AC: 12089
23. (TU4054, D:5796, H:2177) - AC: 11014
24. (XT3448, D:15016, H:1725) - AC: 6629
25. (JJ3351, D:9235, H:1727) - AC: 9519
26. (LD3062, D:12865, H:2119) - AC: 7508
27. (KJ8807, D:17099, H:2092) - AC: 5405
28. (JN2470, D:18999, H:2981) - AC: 4010
29. (CQ6470, D:5257, H:2064) - AC: 11340
30. (FW4345, D:17934, H:1946) - AC: 5060
31. (YB673, D:19444, H:1297) - AC: 4630
32. (AR255, D:9923, H:2033) - AC: 9022
33. (GO6694, D:15887, H:2619) - AC: 5747
34. (NF4916, D:14387, H:2285) - AC: 6664
35. (LG945, D:9789, H:1470) - AC: 9371
36. (AF1537, D:15044, H:2923) - AC: 6016
37. (RZ6974, D:19448, H:1895) - AC: 4329
38. (BJ5242, D:17176, H:2543) - AC: 5140
39. (YQ7881, D:17389, H:2747) - AC: 4932
40. (ME4198, D:10755, H:1195) - AC: 9025
```

3. Max-Heapify()

   Max-Heapify takes in the arguments ArrayList<Airplane> and an int index. The method then compares the key of the Airplane at index with the keys of its children (if the Airplane has any children). If the Airplane at index is not a leaf node and has a child with a key larger than itself, the Airplane at index switches place with the AIrplane at its child, and calls Max-Heapify at its new location.

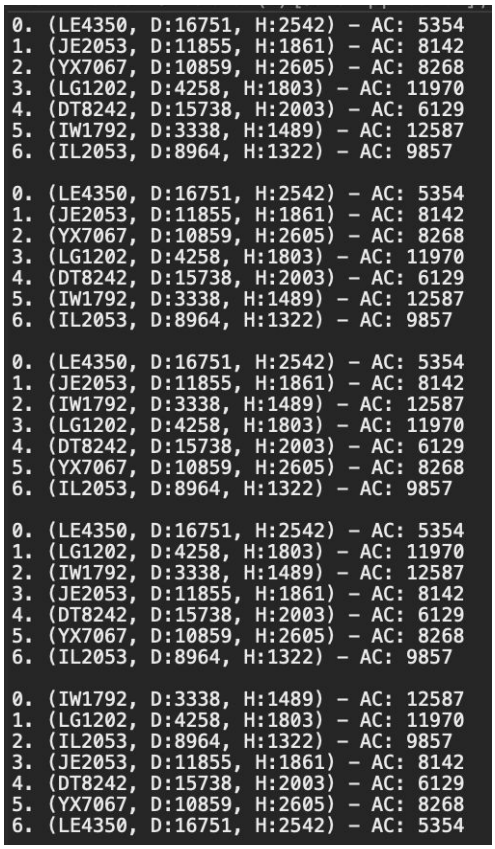
   The below output shows the method at work for the Airplane at index 1 for a randomly generated ArrayList with 7 nodes. In between Max-Heapify calls, there is a blank line printed. The Flight Number of the Airplane at index 1 is GQ1421, and watch how the Airplane moves to its correct location in the Max-Heap.



The screenshot to the left shows the 3 steps in the iteration of Max-Heapify to get flight number GQ1421 to its correct position. Watch as GQ switches position between the first and second step. Initially, GQ1421 has 2 children (FO4409 and DB1488). The AC of flight GQ is smaller than that of FO, and so they switch positions in the second step. In the third step, Max-Heapify is called on GQ at its new index (3); since GQ is now a leaf node, Max-Heapify terminates, and the method call is complete. Max-Heapify is called repeatedly by Build-Max-Heap(), so further output screenshots will continue to show the result of Max-Heapify.

4. Build-Max-Heap()

   Calls Max-Heapify in a bottom-up manner, i.e. on all parent nodes, in order to convert the input ArrayList<Airplane> into a max-heap. The leaves of a max-heap are all on the right half of the ArrayList, and so Build-Max-Heap only calls Max-Heapify on all parent nodes.



The output to the left converts an ArrayList<Airplane> with 7 randomly generated Airplanes to a Max-Heap based off their AC: The first step shows the original ArrayList. The second step compares the last parent node located in the left half of the ArrayList (YX7067) with its children (IW1792 and IL2053). Seeing that the key of IW is larger than that of YX, the locations of the two are switched in the 3rd step. Since YX is now a leaf node, it no longer switches positions. In the 3rd step, Build-Max-Heap now calls Max-Heapify on Node 2 (JE2053) and compares its key with that of its children (LG1202 and DT8242). Since the key of LG is larger than that of JE, the two switch positions between the third and fourth step. Again, JE is now a leaf node and so Max-Heapify terminates for that Airplane.

Finally, Build-Max-Heap calls Max-Heapify on the root node (LE4350), which first switches with its child IW1792, and then switches further with its new child IL2053. Upon analyzing the max heap, for any given node, the node's key is less than or equal to the key of its parent, and so Build-Max-Heap's purpose is met. The above screenshot also demonstrates 4 successful calls to Max-Heapify, where a node's position is switched with that of its child.

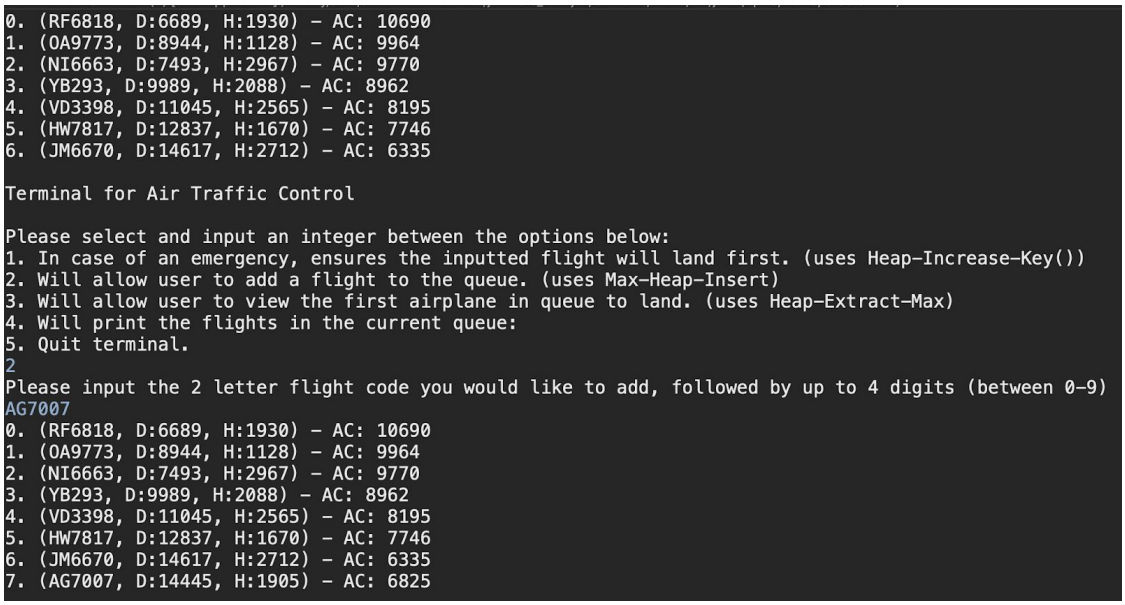
5. HeapSort()
   This method takes an ArrayList<Airplane>, and first converts it to a MaxHeap with heapSize equal to the total number of indices in the ArrayList. This method then switches the positions of the root and the last node in the heap, and decreases the heapSize by 1. It then calls Max-Heapify on the new root node to ensure that the new root is maximum among the remaining nodes in the heap. HeapSort continues to switch, decrease, repeat, until the ArrayList is sorted in ascending order of the key.

```
0. (YI3351, D:4324, H:1830) - AC: 11923
1. (ZX4572, D:9846, H:1634) - AC: 9260
2. (YB1485, D:9000, H:2822) - AC: 9089
3. (HP347, D:18968, H:2897) - AC: 4068
4. (JN4809, D:15178, H:2301) - AC: 6260
5. (PW5123, D:12752, H:1057) - AC: 8096
6. (QG1337, D:18766, H:1913) - AC: 4661

0. (ZX4572, D:9846, H:1634) - AC: 9260
1. (JN4809, D:15178, H:2301) - AC: 6260
2. (YB1485, D:9000, H:2822) - AC: 9089
3. (HP347, D:18968, H:2897) - AC: 4068
4. (QG1337, D:18766, H:1913) - AC: 4661
5. (PW5123, D:12752, H:1057) - AC: 8096
6. (YI3351, D:4324, H:1830) - AC: 11923

0. (YB1485, D:9000, H:2822) - AC: 9089
1. (JN4809, D:15178, H:2301) - AC: 6260
2. (PW5123, D:12752, H:1057) - AC: 8096
3. (HP347, D:18968, H:2897) - AC: 4068
4. (QG1337, D:18766, H:1913) - AC: 4661
5. (ZX4572, D:9846, H:1634) - AC: 9260
6. (YI3351, D:4324, H:1830) - AC: 11923

0. (PW5123, D:12752, H:1057) - AC: 8096
1. (JN4809, D:15178, H:2301) - AC: 6260
2. (QG1337, D:18766, H:1913) - AC: 4661
3. (HP347, D:18968, H:2897) - AC: 4068
4. (YB1485, D:9000, H:2822) - AC: 9089
5. (ZX4572, D:9846, H:1634) - AC: 9260
6. (YI3351, D:4324, H:1830) - AC: 11923
```

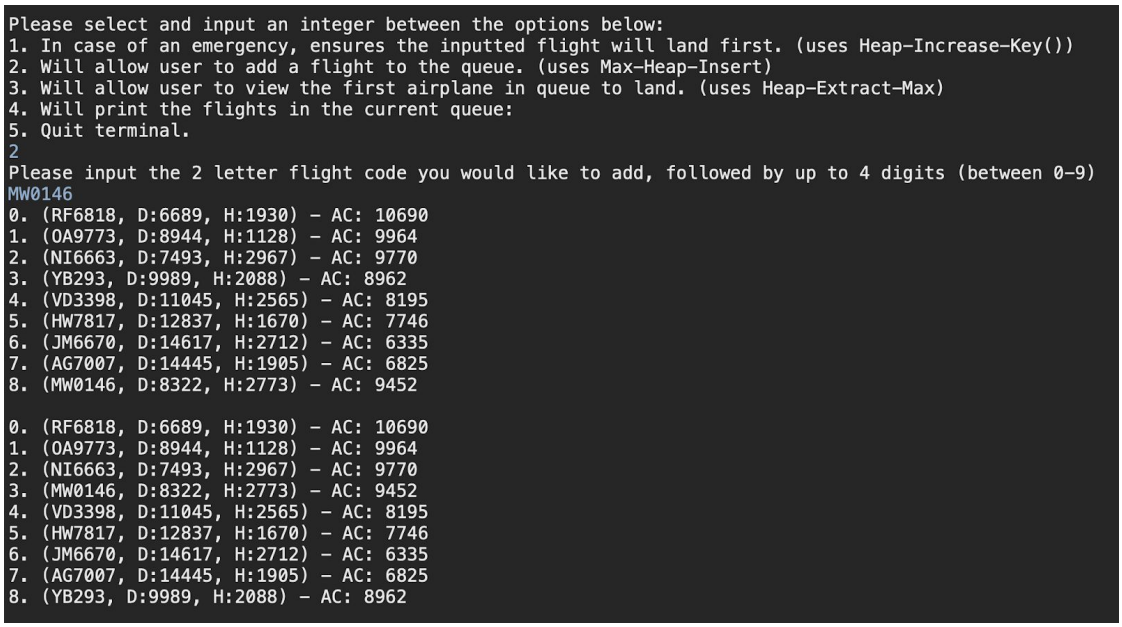
```
0. (JN4809, D:15178, H:2301) - AC: 6260
1. (HP347, D:18968, H:2897) - AC: 4068
2. (QG1337, D:18766, H:1913) - AC: 4661
3. (PW5123, D:12752, H:1057) - AC: 8096
4. (YB1485, D:9000, H:2822) - AC: 9089
5. (ZX4572, D:9846, H:1634) - AC: 9260
6. (YI3351, D:4324, H:1830) - AC: 11923

0. (QG1337, D:18766, H:1913) - AC: 4661
1. (HP347, D:18968, H:2897) - AC: 4068
2. (JN4809, D:15178, H:2301) - AC: 6260
3. (PW5123, D:12752, H:1057) - AC: 8096
4. (YB1485, D:9000, H:2822) - AC: 9089
5. (ZX4572, D:9846, H:1634) - AC: 9260
6. (YI3351, D:4324, H:1830) - AC: 11923

0. (HP347, D:18968, H:2897) - AC: 4068
1. (QG1337, D:18766, H:1913) - AC: 4661
2. (JN4809, D:15178, H:2301) - AC: 6260
3. (PW5123, D:12752, H:1057) - AC: 8096
4. (YB1485, D:9000, H:2822) - AC: 9089
5. (ZX4572, D:9846, H:1634) - AC: 9260
6. (YI3351, D:4324, H:1830) - AC: 11923
```

   The above code starts off with a Max-Heap with the given 7 Airplanes. HeapSort then swaps the root node with the node at heapSize, decreases heapSize by 1, and calls Max-Heapify on the new root. HeapSort performs this action for all Airplanes in the given ArrayList and, as you can see, stacks the largest values at the "bottom" (end) of the ArrayList such that all the Airplanes are now sorted in ascending order by their respective AC.

6. Max-Heap-Insert()
   Max-Heap-Insert allows users to insert an Airplane to the end of the queue. This method then calls Heap-Increase-Key in order to move the flight to the correct spot.



The first step shows a set of 7 randomly generated Airplanes, sorted into a Max-Heap. The second step is terminal information, where the user selects the second option (adding a flight to queue) and inputs an Airplane with Flight Number “AG7007”. The distance and elevation are randomly calculated, and this airplane is added to the end of the landing queue in Step 3. In Step 4, Max-Heap-Insert calls Heap-Increase-Key in order to “float” the inputted Airplane to the correct location. More on this in sub-bullet Number 7. The below screenshot is an example of adding another flight “MW0146” to the queue, after which it “floats” to its correct position.



7. Heap-Increase-Key()

   This method can be used to both add a new Airplane to the queue, or to change and increase the key of an existing Airplane. This method does ***not*** call Max-Heapify, but rather switches the existing Airplane's location with that of its parent if the key of the existing Airplane is less than that of its parent. This method is called in "terminal" to allow the user to change and increase an airplane's AC, in the case of an emergency.

```
0. (OF2516, D:4823, H:2710) - AC: 11233
1. (UW3732, D:14025, H:1925) - AC: 7025
2. (KT9531, D:9331, H:1982) - AC: 9344
3. (SU9880, D:15555, H:1806) - AC: 6319
4. (GO3834, D:16203, H:2138) - AC: 5830
5. (DX2302, D:13587, H:2405) - AC: 7004
6. (TD3277, D:10596, H:2946) - AC: 8229

Terminal for Air Traffic Control

Please select and input an integer between the options below:
1. In case of an emergency, ensures the inputted flight will land first. (uses Heap-Increase-Key())
2. Will allow user to add a flight to the queue. (uses Max-Heap-Insert)
3. Will allow user to view the first airplane in queue to land. (uses Heap-Extract-Max)
4. Will print the flights in the current queue:
5. Quit terminal.
1
For the emergency landing flight, please input the FLIGHT you'd like to change
i.e. 'MW0146' is a valid format for flight numbers.
GO3834
(GO3834, D:16203, H:2138) - AC: 5830 has been cleared for emergency landing

Please input the Approach Code to change GO3834 to:
15000
0. (OF2516, D:4823, H:2710) - AC: 11233
1. (GO3834, D:16203, H:2138) - AC: 15000
2. (KT9531, D:9331, H:1982) - AC: 9344
3. (SU9880, D:15555, H:1806) - AC: 6319
4. (UW3732, D:14025, H:1925) - AC: 7025
5. (DX2302, D:13587, H:2405) - AC: 7004
6. (TD3277, D:10596, H:2946) - AC: 8229

0. (GO3834, D:16203, H:2138) - AC: 15000
1. (OF2516, D:4823, H:2710) - AC: 11233
2. (KT9531, D:9331, H:1982) - AC: 9344
3. (SU9880, D:15555, H:1806) - AC: 6319
4. (UW3732, D:14025, H:1925) - AC: 7025
5. (DX2302, D:13587, H:2405) - AC: 7004
6. (TD3277, D:10596, H:2946) - AC: 8229
```

In the above example, the user selects flight GO3834 and changes its AC to 15000. In the second step, GO (originally at node 4) compares its AC with its parent (UW3732 at node 1) and switches positions. GO again compares its AC with its parent (OF2516 at node 0) and again switches positions. The heap-property is again satisfied.

Below is another example of Heap-Increase-Key, this time on Flight UW3732.

```
Please select and input an integer between the options below:
1. In case of an emergency, ensures the inputted flight will land first. (uses Heap-Increase-Key())
2. Will allow user to add a flight to the queue. (uses Max-Heap-Insert)
3. Will allow user to view the first airplane in queue to land. (uses Heap-Extract-Max)
4. Will print the flights in the current queue:
5. Quit terminal.
1
For the emergency landing flight, please input the FLIGHT you'd like to change
i.e. 'MW0146' is a valid format for flight numbers.
UW3732
(UW3732, D:14025, H:1925) - AC: 7025 has been cleared for emergency landing

Please input the Approach Code to change UW3732 to:
12000
0. (GO3834, D:16203, H:2138) - AC: 15000
1. (UW3732, D:14025, H:1925) - AC: 12000
2. (KT9531, D:9331, H:1982) - AC: 9344
3. (SU9880, D:15555, H:1806) - AC: 6319
4. (OF2516, D:4823, H:2710) - AC: 11233
5. (DX2302, D:13587, H:2405) - AC: 7004
6. (TD3277, D:10596, H:2946) - AC: 8229
```

8. Heap-Maximum()
   This method returns the Airplane at the root node for an input Max-Heap ArrayList. The root node will always be the object with the largest key in a Max-Heap. This method is called by Heap-Extract-Max.

```
0. (HL6933, D:6250, H:1053) – AC: 11349
1. (YJ2336, D:4995, H:2602) – AC: 11202
2. (VB1045, D:8509, H:1190) – AC: 10150
3. (NV5850, D:11971, H:1122) – AC: 8453
4. (TC2299, D:6335, H:2074) – AC: 10795
5. (XB525, D:16706, H:1519) – AC: 5887
6. (ZP2235, D:16535, H:1480) – AC: 5992

(HL6933, D:6250, H:1053) – AC: 11349
After Heap-Maximum
0. (HL6933, D:6250, H:1053) – AC: 11349
1. (YJ2336, D:4995, H:2602) – AC: 11202
2. (VB1045, D:8509, H:1190) – AC: 10150
3. (NV5850, D:11971, H:1122) – AC: 8453
4. (TC2299, D:6335, H:2074) – AC: 10795
5. (XB525, D:16706, H:1519) – AC: 5887
6. (ZP2235, D:16535, H:1480) – AC: 5992
```

   ← This screenshot shows a Max-Heap of 7 Airplanes in an ArrayList. The node with the max AC is HL6933, and the Heap-Maximum method returns that Airplane. Notice how the heap is unchanged after this method is called. That is not the case if Heap-Extract-Max was called instead. See sub-bullet Number 9.

9. Heap-Extract-Max() - used to allow users to view (and remove) the first ranked airplane (the first airplane in queue to land)
   This method calls and assigns the Airplane returned by Heap-Maximum() to a new Airplane called max. Heap-Extract-Max sets the root node to the last leaf node, removes the last leaf node from the ArrayList, decreases the HeapSize by 1, calls Max-Heapify on the new root node to maintain max-heap property, and returns Airplane max. Notice how the root is *extracted*, i.e. removed, from the heap.

```
0. (HL6933, D:6250, H:1053) – AC: 11349
1. (YJ2336, D:4995, H:2602) – AC: 11202
2. (VB1045, D:8509, H:1190) – AC: 10150
3. (NV5850, D:11971, H:1122) – AC: 8453
4. (TC2299, D:6335, H:2074) – AC: 10795
5. (XB525, D:16706, H:1519) – AC: 5887
6. (ZP2235, D:16535, H:1480) – AC: 5992
After Heap-Extract-Max
(HL6933, D:6250, H:1053) – AC: 11349
0. (YJ2336, D:4995, H:2602) – AC: 11202
1. (TC2299, D:6335, H:2074) – AC: 10795
2. (VB1045, D:8509, H:1190) – AC: 10150
3. (NV5850, D:11971, H:1122) – AC: 8453
4. (XB525, D:16706, H:1519) – AC: 5887
5. (XB525, D:16706, H:1519) – AC: 5887
```

   This screenshot shows a simple example of what happens to a heap after Heap-Extract-Max. Flight HL6993 is still the max flight, but it is now removed from the heap. In the “terminal”, if the user chooses to view the first ranked AC, the Heap-Extract-Max method is called, returning and removing the root node, as shown below.

```
0. (WN7458, D:3129, H:2277) - AC: 12297
1. (DP1574, D:8330, H:1577) - AC: 10046
2. (UF7427, D:11379, H:2025) - AC: 8298
3. (WQ1326, D:14085, H:1079) - AC: 7418
4. (CV8511, D:12423, H:1772) - AC: 7903
5. (NG4121, D:14452, H:1588) - AC: 6980
6. (FM2552, D:18999, H:1123) - AC: 4939

Terminal for Air Traffic Control

Please select and input an integer between the options below:
1. In case of an emergency, ensures the inputted flight will land first. (uses Heap-Increase-Key())
2. Will allow user to add a flight to the queue. (uses Max-Heap-Insert)
3. Will allow user to view the first airplane in queue to land. (uses Heap-Extract-Max)
4. Will print the flights in the current queue:
5. Quit terminal.
3
(WN7458, D:3129, H:2277) - AC: 12297
0. (DP1574, D:8330, H:1577) - AC: 10046
1. (CV8511, D:12423, H:1772) - AC: 7903
2. (UF7427, D:11379, H:2025) - AC: 8298
3. (WQ1326, D:14085, H:1079) - AC: 7418
4. (NG4121, D:14452, H:1588) - AC: 6980
5. (NG4121, D:14452, H:1588) - AC: 6980
```

The above screenshot shows the process a user interacting with the terminal would follow in order to view and extract the Airplane with the highest AC. The max-heap at the top is the original heap, and the max-heap at the bottom of the screenshot is the max-heap after Heap-Extract-Max is called.

Overall Design Implementation, demonstrating all functional requirements for 40 Airplanes: (Note that screenshots are broken up to fit on the page, yet are still in order; the terminal output is one continuous, smooth output that the user can interact with. The screenshots continue until page 20.)

```
Terminal for Air Traffic Control

Please select and input an integer between the options below:
1. In case of an emergency, ensures the inputted flight will land first. (uses Heap-Increase-Key())
2. Will allow user to add a flight to the queue. (uses Max-Heap-Insert)
3. Will allow user to view the first airplane in queue to land. (uses Heap-Extract-Max)
4. Will print the flights in the current queue. (uses heapSort)
5. Quit terminal.
14
Please select a valid choice.
4
Current Flight Queue (largest AC to smallest AC):
1. (CE5326, D:3358, H:1733) - AC: 12454
2. (OQ3710, D:3396, H:2607) - AC: 11998
3. (VD5931, D:4276, H:2244) - AC: 11740
4. (RY8684, D:4126, H:2977) - AC: 11449
5. (PB2842, D:5286, H:2775) - AC: 10970
6. (RL4784, D:5330, H:2844) - AC: 10913
7. (LX9620, D:6056, H:2542) - AC: 10701
8. (DC789, D:7198, H:2067) - AC: 10367
9. (ZZ5529, D:6413, H:2858) - AC: 10364
10. (SQ4842, D:8328, H:1194) - AC: 10239
11. (YV6380, D:6836, H:2747) - AC: 10208
12. (QP8776, D:8449, H:1153) - AC: 10199
13. (WC8975, D:7493, H:2428) - AC: 10039
14. (EO3938, D:7772, H:2368) - AC: 9930
15. (FR8268, D:7634, H:2529) - AC: 9918
16. (FU9546, D:9013, H:1542) - AC: 9722
17. (UP770, D:8842, H:1798) - AC: 9680
18. (TN18, D:9515, H:1366) - AC: 9559
19. (MI6875, D:8518, H:2367) - AC: 9557
20. (CM6654, D:10220, H:1248) - AC: 9266
21. (JV2494, D:9405, H:2381) - AC: 9107
22. (MW3187, D:10283, H:1666) - AC: 9026
23. (JH585, D:10746, H:1552) - AC: 8851
24. (WR7227, D:12444, H:1138) - AC: 8209
25. (JR3163, D:11778, H:2019) - AC: 8101
26. (MA709, D:13637, H:1134) - AC: 7614
27. (IS2292, D:13347, H:1470) - AC: 7591
28. (MM8950, D:14137, H:1039) - AC: 7412
29. (NO9850, D:13743, H:1458) - AC: 7400
30. (CD3260, D:12778, H:2745) - AC: 7239
31. (GI9408, D:13225, H:2310) - AC: 7233
32. (VT7952, D:13093, H:2914) - AC: 6997
33. (LF2750, D:14690, H:2261) - AC: 6524
34. (ZN1885, D:15635, H:1410) - AC: 6478
35. (MG8281, D:16117, H:2262) - AC: 5811
36. (QN9911, D:17465, H:1014) - AC: 5760
37. (GK6960, D:17185, H:2119) - AC: 5348
38. (FT9898, D:17061, H:2798) - AC: 5071
39. (GF932, D:18517, H:1966) - AC: 4758
40. (RY1613, D:19865, H:1556) - AC: 4289
```

```
Please select and input an integer between the options below:
1. In case of an emergency, ensures the inputted flight will land first. (uses Heap-Increase-Key())
2. Will allow user to add a flight to the queue. (uses Max-Heap-Insert)
3. Will allow user to view the first airplane in queue to land. (uses Heap-Extract-Max)
4. Will print the flights in the current queue. (uses heapSort)
5. Quit terminal.
3
(CE5326, D:3358, H:1733) - AC: 12454
```

```
Please select and input an integer between the options below:
1. In case of an emergency, ensures the inputted flight will land first. (uses Heap-Increase-Key())
2. Will allow user to add a flight to the queue. (uses Max-Heap-Insert)
3. Will allow user to view the first airplane in queue to land. (uses Heap-Extract-Max)
4. Will print the flights in the current queue. (uses heapSort)
5. Quit terminal.
3
(OQ3710, D:3396, H:2607) - AC: 11998

Please select and input an integer between the options below:
1. In case of an emergency, ensures the inputted flight will land first. (uses Heap-Increase-Key())
2. Will allow user to add a flight to the queue. (uses Max-Heap-Insert)
3. Will allow user to view the first airplane in queue to land. (uses Heap-Extract-Max)
4. Will print the flights in the current queue. (uses heapSort)
5. Quit terminal.
4
Current Flight Queue (largest AC to smallest AC):
1. (VD5931, D:4276, H:2244) - AC: 11740
2. (RY8684, D:4126, H:2977) - AC: 11449
3. (PB2842, D:5286, H:2775) - AC: 10970
4. (RL4784, D:5330, H:2844) - AC: 10913
5. (LX9620, D:6056, H:2542) - AC: 10701
6. (DC789, D:7198, H:2067) - AC: 10367
7. (ZZ5529, D:6413, H:2858) - AC: 10364
8. (SQ4842, D:8328, H:1194) - AC: 10239
9. (YV6380, D:6836, H:2747) - AC: 10208
10. (QP8776, D:8449, H:1153) - AC: 10199
11. (WC8975, D:7493, H:2428) - AC: 10039
12. (EO3938, D:7772, H:2368) - AC: 9930
13. (FR8268, D:7634, H:2529) - AC: 9918
14. (FU9546, D:9013, H:1542) - AC: 9722
15. (UP770, D:8842, H:1798) - AC: 9680
16. (TN18, D:9515, H:1366) - AC: 9559
17. (MI6875, D:8518, H:2367) - AC: 9557
18. (CM6654, D:10220, H:1248) - AC: 9266
19. (JV2494, D:9405, H:2381) - AC: 9107
20. (MW3187, D:10283, H:1666) - AC: 9026
21. (JH585, D:10746, H:1552) - AC: 8851
22. (WR7227, D:12444, H:1138) - AC: 8209
23. (JR3163, D:11778, H:2019) - AC: 8101
24. (MA709, D:13637, H:1134) - AC: 7614
25. (IS2292, D:13347, H:1470) - AC: 7591
```

```
26. (MM8950, D:14137, H:1039) - AC: 7412
27. (NO9850, D:13743, H:1458) - AC: 7400
28. (CD3260, D:12778, H:2745) - AC: 7239
29. (GI9408, D:13225, H:2310) - AC: 7233
30. (VT7952, D:13093, H:2914) - AC: 6997
31. (LF2750, D:14690, H:2261) - AC: 6524
32. (ZN1885, D:15635, H:1410) - AC: 6478
33. (MG8281, D:16117, H:2262) - AC: 5811
34. (QN9911, D:17465, H:1014) - AC: 5760
35. (GK6960, D:17185, H:2119) - AC: 5348
36. (FT9898, D:17061, H:2798) - AC: 5071
37. (GF932, D:18517, H:1966) - AC: 4758
38. (RY1613, D:19865, H:1556) - AC: 4289

Please select and input an integer between the options below:
1. In case of an emergency, ensures the inputted flight will land first. (uses Heap-Increase-Key())
2. Will allow user to add a flight to the queue. (uses Max-Heap-Insert)
3. Will allow user to view the first airplane in queue to land. (uses Heap-Extract-Max)
4. Will print the flights in the current queue. (uses heapSort)
5. Quit terminal.
2
Please input the 2 letter flight code you would like to add, followed by up to 4 digits (between 0-9)
AG007
```

```
Please select and input an integer between the options below:
1. In case of an emergency, ensures the inputted flight will land first. (uses Heap-Increase-Key())
2. Will allow user to add a flight to the queue. (uses Max-Heap-Insert)
3. Will allow user to view the first airplane in queue to land. (uses Heap-Extract-Max)
4. Will print the flights in the current queue. (uses heapSort)
5. Quit terminal.
2
Please input the 2 letter flight code you would like to add, followed by up to 4 digits (between 0-9)
KL1234

Please select and input an integer between the options below:
1. In case of an emergency, ensures the inputted flight will land first. (uses Heap-Increase-Key())
2. Will allow user to add a flight to the queue. (uses Max-Heap-Insert)
3. Will allow user to view the first airplane in queue to land. (uses Heap-Extract-Max)
4. Will print the flights in the current queue. (uses heapSort)
5. Quit terminal.
4
Current Flight Queue (largest AC to smallest AC):
1. (VD5931, D:4276, H:2244) - AC: 11740
2. (RY8684, D:4126, H:2977) - AC: 11449
3. (PB2842, D:5286, H:2775) - AC: 10970
4. (RL4784, D:5330, H:2844) - AC: 10913
5. (LX9620, D:6056, H:2542) - AC: 10701
6. (DC789, D:7198, H:2067) - AC: 10367
7. (ZZ5529, D:6413, H:2858) - AC: 10364
8. (SQ4842, D:8328, H:1194) - AC: 10239
9. (YV6380, D:6836, H:2747) - AC: 10208
10. (QP8776, D:8449, H:1153) - AC: 10199
11. (KL1234, D:8000, H:1910) - AC: 10045
12. (WC8975, D:7493, H:2428) - AC: 10039
13. (EO3938, D:7772, H:2368) - AC: 9930
14. (FR8268, D:7634, H:2529) - AC: 9918
15. (FU9546, D:9013, H:1542) - AC: 9722
16. (UP770, D:8842, H:1798) - AC: 9680
17. (TN18, D:9515, H:1366) - AC: 9559
18. (MI6875, D:8518, H:2367) - AC: 9557
19. (CM6654, D:10220, H:1248) - AC: 9266
20. (JV2494, D:9405, H:2381) - AC: 9107
21. (MW3187, D:10283, H:1666) - AC: 9026
22. (JH585, D:10746, H:1552) - AC: 8851
23. (AG007, D:11762, H:1634) - AC: 8302
24. (WR7227, D:12444, H:1138) - AC: 8209
25. (JR3163, D:11778, H:2019) - AC: 8101
```

```
26. (MA709, D:13637, H:1134) - AC: 7614
27. (IS2292, D:13347, H:1470) - AC: 7591
28. (MM8950, D:14137, H:1039) - AC: 7412
29. (NO9850, D:13743, H:1458) - AC: 7400
30. (CD3260, D:12778, H:2745) - AC: 7239
31. (GI9408, D:13225, H:2310) - AC: 7233
32. (VT7952, D:13093, H:2914) - AC: 6997
33. (LF2750, D:14690, H:2261) - AC: 6524
34. (ZN1885, D:15635, H:1410) - AC: 6478
35. (MG8281, D:16117, H:2262) - AC: 5811
36. (QN9911, D:17465, H:1014) - AC: 5760
37. (GK6960, D:17185, H:2119) - AC: 5348
38. (FT9898, D:17061, H:2798) - AC: 5071
39. (GF932, D:18517, H:1966) - AC: 4758
40. (RY1613, D:19865, H:1556) - AC: 4289

Please select and input an integer between the options below:
1. In case of an emergency, ensures the inputted flight will land first. (uses Heap-Increase-Key())
2. Will allow user to add a flight to the queue. (uses Max-Heap-Insert)
3. Will allow user to view the first airplane in queue to land. (uses Heap-Extract-Max)
4. Will print the flights in the current queue. (uses heapSort)
5. Quit terminal.
```

```
1
For the emergency landing flight, please input the FLIGHT you'd like to change
i.e. 'MW0146' is a valid format for flight numbers.
AG007
(AG007, D:11762, H:1634) - AC: 8302 has been cleared for emergency landing
Please input the Approach Code to change AG007 to:
15000

Please select and input an integer between the options below:
1. In case of an emergency, ensures the inputted flight will land first. (uses Heap-Increase-Key())
2. Will allow user to add a flight to the queue. (uses Max-Heap-Insert)
3. Will allow user to view the first airplane in queue to land. (uses Heap-Extract-Max)
4. Will print the flights in the current queue. (uses heapSort)
5. Quit terminal.
1
For the emergency landing flight, please input the FLIGHT you'd like to change
i.e. 'MW0146' is a valid format for flight numbers.
notValidFlightNumber
notValidFlightNumber not found in the current flight queue.


Please select and input an integer between the options below:
1. In case of an emergency, ensures the inputted flight will land first. (uses Heap-Increase-Key())
2. Will allow user to add a flight to the queue. (uses Max-Heap-Insert)
3. Will allow user to view the first airplane in queue to land. (uses Heap-Extract-Max)
4. Will print the flights in the current queue. (uses heapSort)
5. Quit terminal.
1
For the emergency landing flight, please input the FLIGHT you'd like to change
i.e. 'MW0146' is a valid format for flight numbers.
FT9898
(FT9898, D:17061, H:2798) - AC: 5071 has been cleared for emergency landing
Please input the Approach Code to change FT9898 to:
2000
new key is smaller than current key

Please select and input an integer between the options below:
1. In case of an emergency, ensures the inputted flight will land first. (uses Heap-Increase-Key())
2. Will allow user to add a flight to the queue. (uses Max-Heap-Insert)
3. Will allow user to view the first airplane in queue to land. (uses Heap-Extract-Max)
4. Will print the flights in the current queue. (uses heapSort)
5. Quit terminal.
1
For the emergency landing flight, please input the FLIGHT you'd like to change
i.e. 'MW0146' is a valid format for flight numbers.
FT9898
(FT9898, D:17061, H:2798) - AC: 5071 has been cleared for emergency landing
Please input the Approach Code to change FT9898 to:
12950
```

```
Please select and input an integer between the options below:
1. In case of an emergency, ensures the inputted flight will land first. (uses Heap-Increase-Key())
2. Will allow user to add a flight to the queue. (uses Max-Heap-Insert)
3. Will allow user to view the first airplane in queue to land. (uses Heap-Extract-Max)
4. Will print the flights in the current queue. (uses heapSort)
5. Quit terminal.
4
Current Flight Queue (largest AC to smallest AC):
1. (AG007, D:11762, H:1634) - AC: 15000
2. (FT9898, D:17061, H:2798) - AC: 12950
3. (VD5931, D:4276, H:2244) - AC: 11740
4. (RY8684, D:4126, H:2977) - AC: 11449
5. (PB2842, D:5286, H:2775) - AC: 10970
6. (RL4784, D:5330, H:2844) - AC: 10913
7. (LX9620, D:6056, H:2542) - AC: 10701
8. (DC789, D:7198, H:2067) - AC: 10367
9. (ZZ5529, D:6413, H:2858) - AC: 10364
10. (SQ4842, D:8328, H:1194) - AC: 10239
11. (YV6380, D:6836, H:2747) - AC: 10208
12. (QP8776, D:8449, H:1153) - AC: 10199
13. (KL1234, D:8000, H:1910) - AC: 10045
14. (WC8975, D:7493, H:2428) - AC: 10039
15. (EO3938, D:7772, H:2368) - AC: 9930
16. (FR8268, D:7634, H:2529) - AC: 9918
17. (FU9546, D:9013, H:1542) - AC: 9722
18. (UP770, D:8842, H:1798) - AC: 9680
19. (TN18, D:9515, H:1366) - AC: 9559
20. (MI6875, D:8518, H:2367) - AC: 9557
21. (CM6654, D:10220, H:1248) - AC: 9266
22. (JV2494, D:9405, H:2381) - AC: 9107
23. (MW3187, D:10283, H:1666) - AC: 9026
24. (JH585, D:10746, H:1552) - AC: 8851
25. (WR7227, D:12444, H:1138) - AC: 8209
26. (JR3163, D:11778, H:2019) - AC: 8101
27. (MA709, D:13637, H:1134) - AC: 7614
28. (IS2292, D:13347, H:1470) - AC: 7591
29. (MM8950, D:14137, H:1039) - AC: 7412
30. (NO9850, D:13743, H:1458) - AC: 7400
31. (CD3260, D:12778, H:2745) - AC: 7239
32. (GI9408, D:13225, H:2310) - AC: 7233
33. (VT7952, D:13093, H:2914) - AC: 6997
34. (LF2750, D:14690, H:2261) - AC: 6524
35. (ZN1885, D:15635, H:1410) - AC: 6478
36. (MG8281, D:16117, H:2262) - AC: 5811
37. (QN9911, D:17465, H:1014) - AC: 5760
38. (GK6960, D:17185, H:2119) - AC: 5348
39. (GF932, D:18517, H:1966) - AC: 4758
40. (RY1613, D:19865, H:1556) - AC: 4289
```

```
Please select and input an integer between the options below:
1. In case of an emergency, ensures the inputted flight will land first. (uses Heap-Increase-Key())
2. Will allow user to add a flight to the queue. (uses Max-Heap-Insert)
3. Will allow user to view the first airplane in queue to land. (uses Heap-Extract-Max)
4. Will print the flights in the current queue. (uses heapSort)
5. Quit terminal.
5
```

Installation Procedure

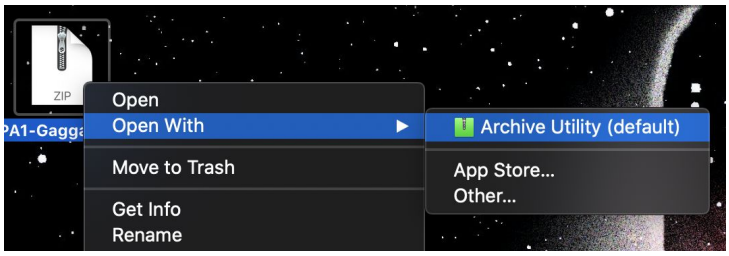
The following installation procedure is for Mac OSX. The procedure is probably very similar in Windows or Linux, but each step might be a little different. Note: A JDK must be pre-installed on the computer.

Step 1: Download the file PA1-Gaggar.zip

The process will be easier if you save the zip file to a convenient file location, i.e. Desktop, Downloads, etc. In this procedure, I downloaded and saved the zip file on my Desktop.

Step 2: Unzip PA1-Gaggar.zip

Double click to unzip the file. If that doesn't work, right click, click open with, and click the default Archive Utility Opener. Once the folder is unzipped, the zip file can be deleted.



Step 3: Navigating the folder PA1-Gaggar

Within the folder will be multiple documents, including this formal report, a folder called pa1_CS146 which contains the java classes themselves, and a jar executable file called pa1.jar

Step 4: Running the jar file on your computer

On Mac, open up Terminal. (If you cannot find it, go to the top right of your screen and click the search icon (the magnifying glass). Type Terminal and open the application).

Note: The below commands might be different if you saved the file at a different location.

Type the following into the terminal screen, and hit enter in between commands:

cd Desktop

(OPTIONAL Command to see what files are on your Desktop: ls)

cd PA1-Gaggar

java -jar pa1.jar

Upon hitting enter on that last command, the following should begin printing out:

```
[ayushs-mbp:~ ayush$ cd Desktop                                                 ]
[ayushs-mbp:Desktop ayush$ cd PA1-Gaggar                                        ]
[ayushs-mbp:PA1-Gaggar ayush$ java -jar pa1.jar                                 ]
Terminal for Air Traffic Control

Please select and input an integer between the options below:
1. In case of an emergency, ensures the inputted flight will land first. (uses H
eap-Increase-Key())
2. Will allow user to add a flight to the queue. (uses Max-Heap-Insert)
3. Will allow user to view the first airplane in queue to land. (uses Heap-Extra
ct-Max)
4. Will print the flights in the current queue:
5. Quit terminal.
```

At this point, the user is free to interact with the software as needed, and the software is fully installed. To re-run the program, simply input "java -jar pa1.jar" into Terminal again!

Problems Encountered During Implementation

- The Difference Between HeapSize and ArrayList.size()

  The largest, most prominent problem I continuously encountered was understanding the distinction between HeapSize and the length of the ArrayList (ArrayList.size()) being converted to a max-heap. After simply following the pseudocode provided in the textbook, it seemed as if the algorithm was correctly transforming the input ArrayList to a max-heap. At first glance, it seems like the correct implementation; however, upon closer inspection, this process only utilized the maxHeapify and buildMaxHeap functions, neither of which uses the heap's heapSize too often. Overlooking this fact led to further errors when implementing the heapExtractMax and heapSort methods.

  Upon testing the heapExtractMax() method, the algorithm correctly found and extracted the root node, and decreased heapSize by 1. However, the output ArrayList still contained the same number nodes, when the expected ArrayList would have one less node, as the root was extracted. It was frustrating to consistently see the last leaf node be copied multiple times whenever this method was called! So what was the mistake? After testing numerous times, I noticed that it was only the last node in the max-heap that was being copied. Additionally, if the heapExtractMax method was called multiple times, the size of the ArrayList would stay the same even if heapSize was decreased, and the last node would be copied the amount of times heapExtractMax was called. Thus, I needed a way to relate the length of the ArrayList with the heapSize. My initial solution for this was to add the following for-loop to an otherwise generic setHeapSize method:

```
public void setHeapSize(int heapSize) {
    this.heapSize = heapSize;
    for (int i = (getFlights().size() - 1); i > heapSize; i--) {
        getFlights().remove(i);
    }
}
```

  Since heap-size cannot be larger than the length of the ArrayList, the only time heapSize decremented was in heapExtractMax and heapSort. So when calling heapExtractMax, the first node is removed, a new root is set, and maxHeapify is called on the new root. This solution is simple: remove the nodes at indices larger than heapSize, as they are copies of other nodes. And voila, heapExtractMax worked!

  Not so fast... upon testing the heapSort method, it became apparent that something was wrong once again, as the output was never the entire sorted max-heap. In this case, the problem lay in that decreasing heapSize should not always remove the ends of the ArrayList. In heapExtractMax, the last nodes were copies of current nodes and

could be removed; in heapSort however, every node contains an important Airplane object, and so removing nodes from the ends of the ArrayList would be omitting information. So what was the answer?

The key to this entire problem was understanding that heapSize is a subset of the length of the ArrayList. heapSize represented the length of the max-heap represented by the ArrayList. Whenever a node is removed from the ArrayList, both the length of the ArrayList and the heapSize should decrease by 1. However, if a node is removed from the max-heap, only the *heapSize* should decrement. Removing a node was the purpose of the heapExtractMax method, whereas ordering an ArrayList was the purpose of the heapSort method; the first removed a node, whereas the second decreases the number of nodes which satisfy the max-heap property when sorting and rearranging an ArrayList. The solution is not to *always* chop off the extra leaf nodes, but rather to only remove the last node when calling HeapExtractMax.

I reached this conclusion after hours of testing, tweaking, and examining each of the heapSort methods and deliberating the possibilities of what could be causing this error. Understanding this fundamental distinction between the two was crucial in correctly implementing the functional requirements of this assignment. The highlighted code in heapExtractMax below was the key realization in linking heapSize and the length of an ArrayList, and the setter method for heapSize was reverted back to a generic setter method.

```
public Airplane heapExtractMax(ArrayList<Airplane> flights) {
    if (this.getHeapSize() < 0)
        throw new IndexOutOfBoundsException("heap underflow");
    Airplane max = heapMaximum(getFlights());
    flights.set(0, flights.get(this.getHeapSize() - 1));
    setHeapSize(this.getHeapSize() - 1);
    flights.remove(flights.size() - 1);
    maxHeapify(this.getFlights(), 0);
    return max;
}
```

- Max-Heap ArrayList vs HeapSorted ArrayLists
  One of the errors I noticed when testing user interaction with the terminal software was that I always tested the program by going through the selection choices in order, 1 through 5. However, when switching up the order, my program started to break down. Upon inspection, I realized that the bugs started after viewing the current flight queue; calls to heapExtractMax, heapIncreaseKey, and maxHeapInsert were silently overlooking certain nodes at times, and often it was the root or last leaf node being overlooked when these methods were called. Additionally, I realized that the

ordering of the ArrayList no longer satisfied the max-heap property. Why was this error occuring?

In calling heapMaximum, the program should be grabbing the correct node (the root) from an ArrayList built as a max-heap. (This error also tied in closely with the heapSize error discussed above.) Upon closer inspection, I realized that the call to print the current flight queue used the heapSort method, after which methods were free to interact with the ArrayList. Although the heapSort method sorted the nodes in ascending order, the nodes no longer satisfied the max-heap; even reverse ordering the nodes did not build them back to a max-heap. Many of these methods relied on the input ArrayList to be a max-heap, and so the program printed out false results when the input was instead a sorted ArrayList.

The ArrayList is arranged vastly differently after calling buildMaxHeap and after calling heapSort, and failing to recognize the importance of this difference was the cause of this error. Additionally, each method ordered the indices in the ArrayList differently, so the program should be built such that the flight number the user selects is the airplane who's AC is increasing. In order to solve this problem, I isolated where and when exactly the heapSort method was called (printing the current flight queue in descending order of key, reversing heapSort method), and I called buildMaxHeap as soon as the method printed the Airplanes. In this way, the flights are presented in an organized manner, and yet the user can continue to interact with them with the same ease and functionality of a max-heap ArrayList.

```
public void flightQueue() {
    this.heapSort(getFlights());
    int counter = 1;
    for (int i = (this.getFlights().size() - 1); i >= 0; i--) {
        sop(counter + ". " + this.getFlights().get(i).toString());
        counter++;
    }
    buildMaxHeap(getFlights());
}
```

- Troubles with "this" keyword
  When creating the heapSort class, each heapSort object contains an ArrayList variable. The difficulty in this case was figuring out what parameters to pass in to the heapSort methods: should I pass in an ArrayList, should I pass in a heapSort and call heapSort.getFlights(), or should there be no input parameter and instead use just the keyword "this" to call getFlights(). I often used the last two interchangeably, where I would pass in a heapSort into the method, and instead of calling getFlights() on the variable passed in, the code worked perfectly when using the "this" keyword instead. The logic as to why the code worked confounded me for the longest time, until I

realized that each of the above three options essentially did the same thing. Why was this the case? And what would best practice be otherwise?

In the Simulator class, a heapSort object "sorted" is created with an ArrayList<Airplane> called flights, with the heapSize equal to the size of flights. Sorted then called the buildMaxHeap method on its own flights; buildMaxHeap then called maxHeapify on the same ArrayList flights, which had the same result as calling the method on sorted.getFlights(). The other methods were all called by the user when interacting with the terminal interface, and the input parameter was always this.getFlights(), which again, had the same result as calling the method on sorted.getFlights(). At the end of the day, all of these methods are calling the methods on the same ArrayList (flights), and so it does not matter *how* flights is called. So why then does my code use the input parameters of ArrayList?

I believe that the best practice would be to code the methods to take in the input parameters of an ArrayList. Although this specific programming assignment calls these methods on the same ArrayList, this practice would allow for future flexibility, in case the user only wants to interact with a portion of the ArrayList, or a portion of the heap. This design implementation allows for the greatest flexibility when considering future uses for this class, and would be more usable in a wider number of cases if the heapSort methods had the input parameter as ArrayLists rather than heapSort objects.

- Writing the terminal method to have user interact with methods
  Programming a clean user interface was also a challenge to consider when designing this simulation. Initially, I designed the terminal to type in words such as "emergency" to change AC, "view" to extract the highest priority flight from queue, "add" to add in a flight, or "done" to exit terminal. However, the while loops, checks, and room for error in having users input words was turning the terminal method into a convoluted, complex method. Using a Scanner class to read input was also difficult, as I continuously ran into errors when the user typed in one too many words or entered too many things in a row.

Initial terminal code:

```
If this is an emergency, please type EMERGENCY (case sensitive):
OR Please input the 2 letter flight code followed by up to 4 digits (between 0-9)
OR Type 'view' to view the first airplane in queue to land
OR Type 'done' if you do not wish to add any flights into the queue:
view
(LG600, D:4677, H:1089) - AC: 12117
EMERGENCY
For the emergency landing flight, please input the index of the flight number (from above)
1
(ZG2584, D:5365, H:1375) - AC: 11630 has been cleared for emergency landing
If the emergency remains, please type EMERGENCY (case sensitive):
done
```

As you can see, the above code has too many variables and too many options, that is confusing both for the user, and for a programmer to account for. Instead of using the scanner class, I instead switched to using InputStreamReader and BufferedReader, which had a stronger structure to react to user selections. I also changed the user interface to a much cleaner "menu" layout, and had the user select an integer from the option choices. The resulting terminal was cleaner and more robust:

```
Please select and input an integer between the options below:
1. In case of an emergency, ensures the inputted flight will land first. (uses Heap-Increase-Key())
2. Will allow user to add a flight to the queue. (uses Max-Heap-Insert)
3. Will allow user to view the first airplane in queue to land. (uses Heap-Extract-Max)
4. Will print the flights in the current queue:
5. Quit terminal.
```

- Using ArrayList, indexes, etc

  Another error I encountered about halfway into the programming assignment was that the pseudocode called for inputs of arrays. However, I quickly realized that to dynamically add and remove Airplanes from my max-heap, I needed a data structure with a flexible length, and so I changed all my code to take in the parameters of an ArrayList instead. Although this doesn't seem like a big change, it was challenging to review through lines of code and ensure the consistency was there between methods called by and called on an Array versus an ArrayList.

  Consistency was a huge factor in this assignment as well. A lot of the code was originally written for an array with indices from [1, ... n], but programming the parent/children methods as well as the maxHeapify called a variety of indices constantly. Even methods such as heapSize had to be adjusted back one space to account for the implementation of the ArrayList. Much of my frustration from the initial testing was directly linked to flawed logic in correctly locating the desired node in the ArrayList. Ideally, perfect logic only needs to be tested and implemented once; obviously, that's not the case, and running smaller samples and testing corner cases was critical in trying to solve a basic logical fallacy. The solution to this problem was

drawing the input ArrayList and desired max-heap output side by side, and then tracing the path of the desired test node through the ArrayList to where it should end up in the heap. These sketches and diagrams helped me analyze and cross-examine my code to ensure there were no IndexOutOfBoundsExceptions being thrown.

With such a large program to run, even small problems are exacerbated; for example, when verifying the consistency of my code, I noticed that the maxHeapify function was only working correctly about 50% of the time. In checking the comparisons of the index node's key, I realized that the method was comparing the current key with the *right* instead of comparing the key with the *largest* key. Even small, harmless, and simple-to-fix errors are easily overlooked when debugging a larger program.

Lesson Learned

This programming assignment has been the largest computer science project I have ever worked on. I only began my coding journey less than a year ago, and invested much of my time in really understanding the fundamentals of Java and object oriented programming, and developing the logic required to tackle a simulation such as this one. Thus far, the largest projects I had worked on were from the Intro to Data Structures class offered at SJSU, CS 46B, many of which were basic homework assignments or lab work which could be completed in a matter of hours. This project took a couple weeks to develop and test fully, and required a thorough understanding of the HeapSort Algorithm in order to truly be successful. Furthermore, it expanded and strengthened the programming skills I had, and gave me exposure to what a sample project in the workplace might look like.

By developing and debugging each of the methods for the heapSort algorithm, I was truly able to grasp the inner workings of what makes each line of code in the algorithm function as it does. An understanding of each line of code results in an understanding of the code; it is not possible to have one without the other. Although the code was a direct translation of the pseudocode from the textbook, actually coding it for myself and debugging it with Airplane objects helped me understand what exactly was occurring behind the scenes at every line of code. As mentioned in the Problems Encountered section, the biggest learning point was between heapSize and ArrayList.size. It makes sense when reading the definitions between the two, but in actual implementation of the code, confusing heapSize with the size of the ArrayList could lead to a whole *heap* of problems. Through the process of debugging, I was able to concretely understand the purpose of buildMaxHeap versus heapSort, watch a node "float" to the correct position in heapIncreaseKey versus calling maxHeapify on that node, and overall, just fundamentally learn heapSort to its fullest extent.

Some of the other errors I encountered helped strengthen my understanding of the basics of Java, and take my programming skill from a novice, to one ready for the workforce. For example, I'd always known the definition of and the result of using the "this" keyword in programming, but through this assignment, I became more confident in deciding when and where would it be appropriate to use the "this" keyword versus making static methods. In deciding to use a dynamic and adjustable data structure of ArrayLists, I was able to aggregate Airplane objects into the ArrayList, and aptly use the data structure to swap, sort, and remove Airplanes from the max-heap. Before this project, the only Java Input object I was comfortable with was the Scanner class, and I knew about a couple Stream and File Readers as well. I challenged myself to learn something new and use a more robust Input Reader, and so I programmed the terminal method using IO streams I've never really used before.

Programming the terminal method was my first introduction to User Interface (UI) design, and challenged me to think of the system as one that an average user would not be familiar with. Although this was a more basic interaction with the user, using simple System.out.print statements, it still helped me think from a developer's perspective. How do I guide the user to select the appropriate choice, and thus call the appropriate method to achieve the desired goal? How do I handle exceptions if the user inputs invalid arguments? How do I display selection choices in a clear, organized, and legible manner? I slowly built the terminal method while trying to keep these questions in mind; the terminal method evolved from accepting strings like "EMERGENCY" to invoke heapIncreaseKey, to a menu numbered 1-5 that would display every time the user made a selection. After making a selection, the terminal often printed a follow-up statement to further assist the user in inputting a value compatible with the program. Programming the terminal using InputStreamReader and BufferedReader, readers I wasn't too comfortable with, helped me strengthen my knowledge of Java IO streams, and familiarize myself with UI design in general.

This programming assignment has really inspired me to learn more coding languages and further the depth of my knowledge in programming. I truly love logical problem solving, which is at the heart of Computer Science in general; logically approaching this assignment and tackling it in what I see as the best way possible really appealed to my logical nature. Developing this simulation inspired me to learn new skills and expand on my programming knowledge, in ways not possible through a textbook, classroom format; I believe that with more time, I would learn how to create graphical user interfaces (GUIs) and further enhance the experience from a user's perspective. Further functionality could easily be added into this program, such as removing a specific flight rather than just the root node, sorting flights by either ascending or descending keys, decreasing the key of a given node, and more.

In the future, I'd love to work in the industry with the data structures knowledge I have right now. Obviously, over the next couple years at SJSU, I want to continue learning in-depth coding practices related to Bioinformatics and Artificial Intelligence along with mastering data structures; I hope to be able to combine my knowledge from these three fields in Computer Science with the pursuit of my Mechanical Engineering degree as well to work with robotics in the future, ideally in the aerospace or healthcare industries. The first stepping stone to that would be landing an internship to truly test and grow my CS knowledge and establish an educational foundation in robotics. I hope to someday take my passion and expertise towards founding a startup focused on improving current healthcare practices using advanced, efficient algorithms controlling robotics. I'm extremely fortunate that this programming assignment, in creating and simulating an Airport Control Tower, was able to strengthen my grasp of data structures and algorithms while challenging me in an exciting and unique way!