# Write-up (Sean):

## Class Descriptions:

### Message:

The Message class is fairly simple, all it has is three integer fields which store the Priority of the Message, the time of its instantiation (or "arrival" in terms of this assignment) and its number, which is somewhat unnecessary, but I used it for debugging so that I could tell what Messages were being created and where they were being sent.

### MessagePriorityQueue:

#### Overview:

The class is predominantly static to make for easier testing. The method

#### Fields:

The messages field is an ArrayList of PriorityQueues of Messages, where the index of the priority queue indicates its priority. There are two static ArrayLists of Integers, waitingTimes and numElements, which keep track of the total waiting time and number of elements in each PriorityQueue. The index of each integer in these array lists corresponds to the priority of the queue that they match. The last two fields curArrivalTime and curMsg which serve mostly as numbers to keep track of the current arrival time and the current message number. Technically the current Message is unnecessary, but I found them useful for testing so that I could see how Messages were distributed into the correct queues.

The three final constants are processTime, waitTime and numQueues, and there simply so that numeric literals are not used, and the code can be changed when necessary for testing. The waitTime constant tells the program how long to wait once a Message is added before it can be processed. The numQueues can be changed to increase or decrease the number of potential priorities while processTime is simply the number of messages which are created.

#### Methods:

The main just calls the most important methods, namely makeQueues, runSystem, and showResults.

* The makeQueues method provides the setup for the actual running of the program, and it instantiates the three ArrayList fields, including messages, waitingTimes, and numElements, and sets their size based on the numQueues constant.
* The runSystem method is simply calls the appropriate messages, creating messages every second until the curArrivalTime reaches the constant that is set to be the limit, in this case 100,000. It instantiates these messages using the addMessage method.
  + The addMessage method creates a Message with the parameter as the Message's number, a random priority within the bounds of numQueues constant, and the curArrivalTime as the Message's arrival time. It then adds this Message to the appropriate queue based on its priority field.
  + Once the Messages are created, runSystem calls processMessages, which removes the next Message in the priority queue as long as that message has been there at least the number of minutes equal to the constant set at the start, in this case 4.
  + The removed message has the method processMessage called on it. Note the difference between processMessage**s** and processMessage; one refers to the queues as a whole and the other processes the Message that is passed in a parameter.
  + The processMessage method simply prints the details of the message and adjusts the appropriate information in the fields, mainly the waitingTimes and numElements fields.
  + Finally, after all the requisite Messages are created, the runSystem method processes all the remaining elements in the queues just in case any were missed.
* At last, the showResults method prints the average waiting time for the messages for each priorityQueue.

#### Results:

With a sample size of 10000000, the waiting times of the queues were as follows:

* The average waiting time of the Queue with Priority 0 was: 4.96
* The average waiting time of the Queue with Priority 1 was: 5.13
* The average waiting time of the Queue with Priority 2 was: 5.53
* The average waiting time of the Queue with Priority 3 was: 6.97
* The average waiting time of the Queue with Priority 4 was: 16.04

Therefore, the waiting time is about O(n) except for the last queue which has a notation of O(n2). This distribution is relatively low and efficient but a heap implements it in O(log n) time, meaning that a heap is more efficient. This is because a heap uses binary search trees.