Gray Houston
ghousto

Project 1 Analysis Questions

Part 1

1.)

a.) The worst case running time/computation for the enqueue operation when no resizing of the array occurs is **O(c)** where "c" is an arbitrary constant value. This operation occurs in constant time because the array is never traversed. The only operations that occur in this function (without resizing) are placing the patient into the array at a known index, modularly incrementing the tail index, and incrementing the counter for the patients. All of these operations run in O(c) time. Therefore, the enqueue operation runs a worst case of O(c) time.

b.) The worst case running time for the dequeue operation without resizing the array is **O(c)** where "c" is an arbitrary constant. Similar to the enqueue operation, dequeue runs in constant time because the array is never traversed and the internal computations/assignments occur in O(c) time (see above for examples of internal operations running in constant time).

c.) The worst case running time for the size() operation regardless of the array resizing or not is **O(c)** where "c" is an arbitrary constant. First, the array is never traversed. Second, this function only returns the value of a counter variable, which runs in O(c) time. Third, the increment/decrement of this counter runs in O(c) time and occurs during the enqueue/dequeue operation. Therefore, the worst case running time of the size operation is O(c).

Furthermore, because only a variable is returned during a call to the size() function and the counter variable is updated before any potential array resizing in the enqueue and dequeue operations (in fact, the counter variable is used as a loop bound in array resizing), the running time of the size operation/function is independent of array resizing. Therefore, the worst case running time in all situations for the size operation is O(c).

2.)

a.) The worst case running time for the enqueue operation with array resizing is **O(n)** where "n" is the number of elements in the queue. During array resizing, all of the elements in the old array must be copied to the new array, resulting in "n" number of assignment operations. Because the memory allocation and the computations in the array resizing run in O(c) and the other computations of the enqueue method run in O(c) time (as shown in part "a" above), the "n" assignments used to copy the elements becomes the determining factor. Because the worst case running time for copying the array runs in O(n), the overall worst case running time for the enqueue operation with array resizing is O(n).

b.) Similar to the enqueue operation, the dequeue operation with array resizing is **O(n)** where "n" is the number of elements in the queue. With the same reasoning as with enqueue, during array resizing, copying the elements into the new array will cost "n" assignments and therefore run in O(n) time while all other computations will cost O(c) time. Therefore, copying the array elements becomes the determining factor of running time, resulting in the dequeue operation with array resizing to run in O(n) time.

c.) As proven above in question 1.c, the size operation will run in O(c) time, where “c” is an arbitrary constant, in the worst case even with array resizing. Because the size function's only action is to return a counter variable and the counter variable is updated before array resizing occurs, the size operation is independent of array resizing and its O(n) running time. Therefore, the O(n) time of array resizing is not included when calculating running time for the size operation. For this reason, the answer is the same as in question 1.c: the worst case running time of the size operation is O(c), even with array resizing.

3.)

The length of the queue can mathematically be calculated from the array length and head & tail indices using modulus arithmetic to accounting for wrapping around the end of the array. The equation for the queue length using Java modulus logic is

qLength = ((tail – head) % arrayLength + arrayLength) % arrayLength

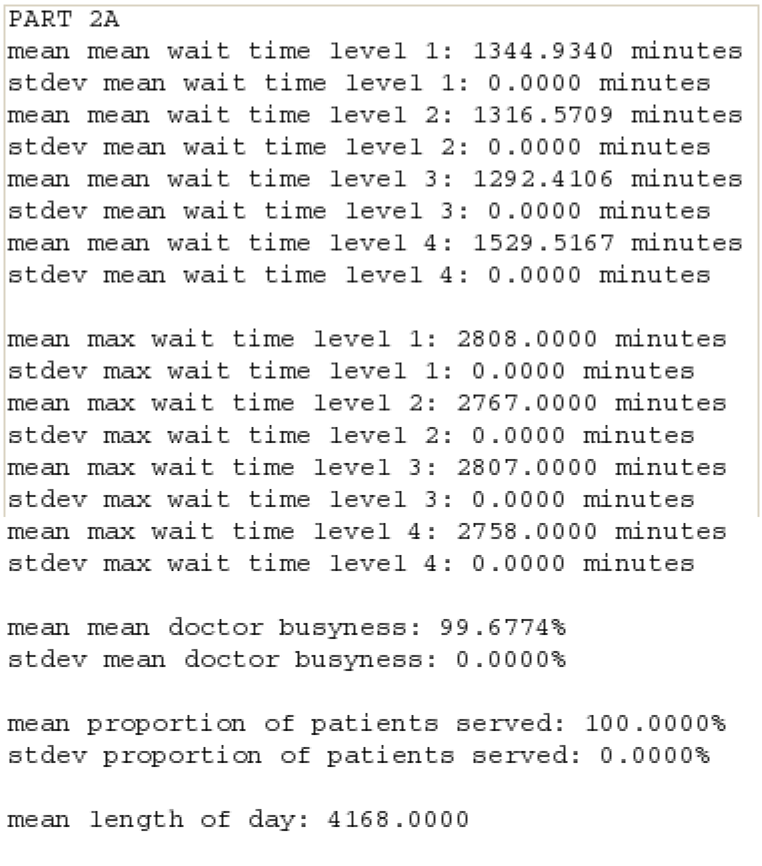
where “qLength” is the length of the queue, “tail” is the tail index (i.e. index to insert a new element during the enqueue operation), “head” is the head index (i.e. index of element at the front of the queue), and “arrayLength” is the length of the physical array.

In regards to running time, this calculation takes O(c) time. Because retrieving the length of an array occurs in O(1) or O(c) time due to the Java implementation and basic math operations run in O(c) time, the computation shown above will run in O(c) time. Because this calculation runs in constant time, this cost is negligible.

Part 2A

1.)

The output for Part 2A when the simulation runs with default parameters is



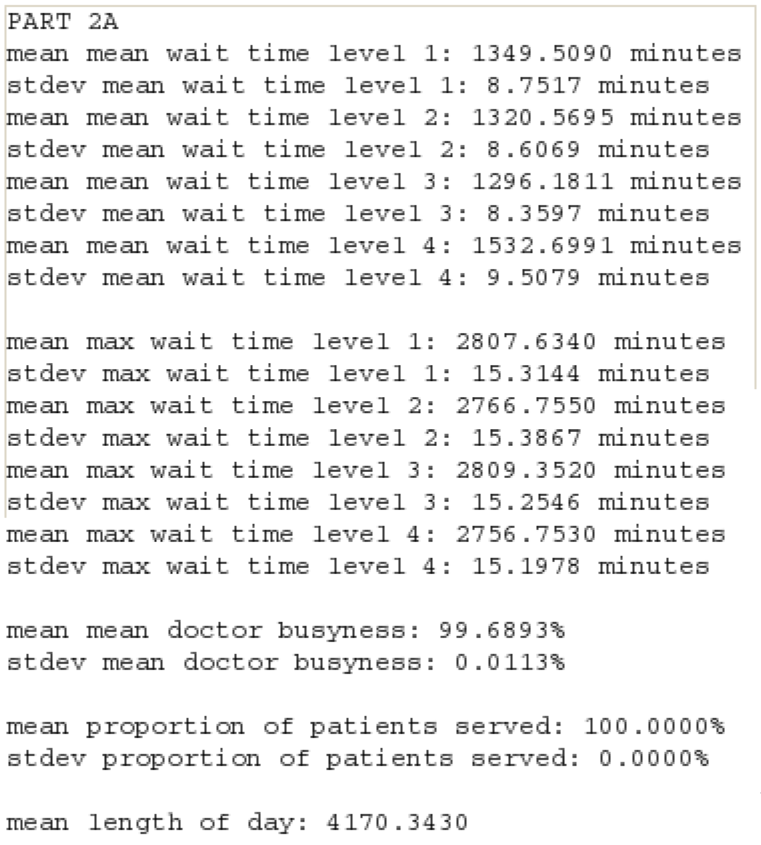
```
PART 2A
mean mean wait time level 1: 1344.9340 minutes
stdev mean wait time level 1: 0.0000 minutes
mean mean wait time level 2: 1316.5709 minutes
stdev mean wait time level 2: 0.0000 minutes
mean mean wait time level 3: 1292.4106 minutes
stdev mean wait time level 3: 0.0000 minutes
mean mean wait time level 4: 1529.5167 minutes
stdev mean wait time level 4: 0.0000 minutes

mean max wait time level 1: 2808.0000 minutes
stdev max wait time level 1: 0.0000 minutes
mean max wait time level 2: 2767.0000 minutes
stdev max wait time level 2: 0.0000 minutes
mean max wait time level 3: 2807.0000 minutes
stdev max wait time level 3: 0.0000 minutes
mean max wait time level 4: 2758.0000 minutes
stdev max wait time level 4: 0.0000 minutes

mean mean doctor busyness: 99.6774%
stdev mean doctor busyness: 0.0000%

mean proportion of patients served: 100.0000%
stdev proportion of patients served: 0.0000%

mean length of day: 4168.0000
```



```
PART 2A
mean mean wait time level 1: 1349.5090 minutes
stdev mean wait time level 1: 8.7517 minutes
mean mean wait time level 2: 1320.5695 minutes
stdev mean wait time level 2: 8.6069 minutes
mean mean wait time level 3: 1296.1811 minutes
stdev mean wait time level 3: 8.3597 minutes
mean mean wait time level 4: 1532.6991 minutes
stdev mean wait time level 4: 9.5079 minutes

mean max wait time level 1: 2807.6340 minutes
stdev max wait time level 1: 15.3144 minutes
mean max wait time level 2: 2766.7550 minutes
stdev max wait time level 2: 15.3867 minutes
mean max wait time level 3: 2809.3520 minutes
stdev max wait time level 3: 15.2546 minutes
mean max wait time level 4: 2756.7530 minutes
stdev max wait time level 4: 15.1978 minutes

mean mean doctor busyness: 99.6893%
stdev mean doctor busyness: 0.0113%

mean proportion of patients served: 100.0000%
stdev proportion of patients served: 0.0000%

mean length of day: 4170.3430
```

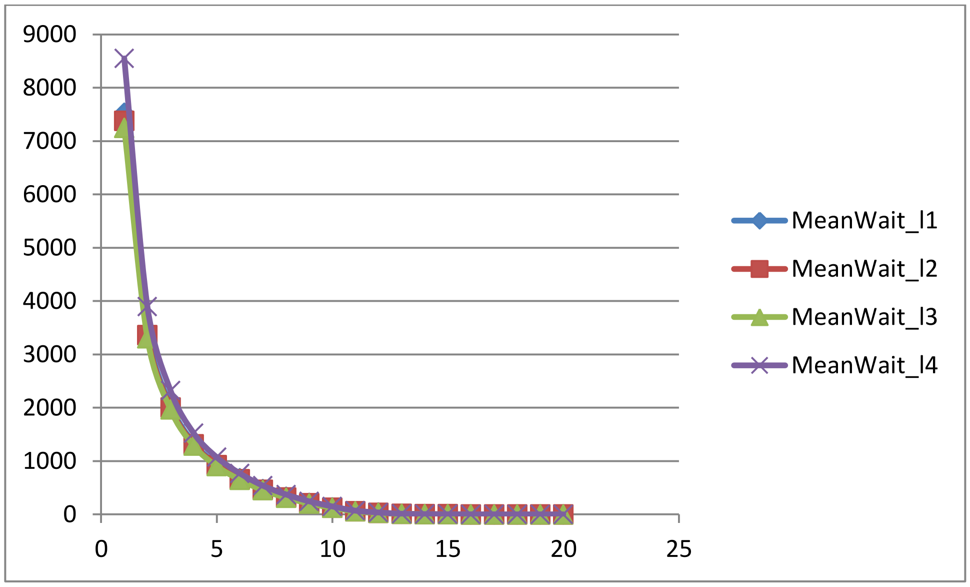
a.) The mean and max wait times for each urgency level is
Level 1: mean = 1344.9 minutes, max = 2808.0 minutes
Level 2: mean = 1316.6 minutes, max = 2767.0 minutes
Level 3: mean = 1292.4 minutes, max = 2807.0 minutes
Level 4: mean = 1529.5 minutes, max = 2758.0 minutes

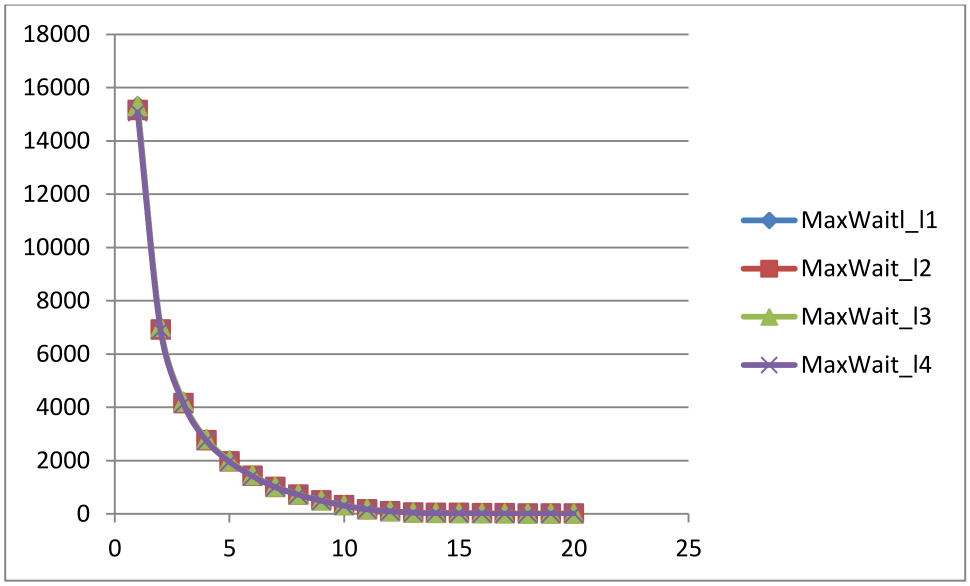
b.) The doctors are very busy, as the mean doctor busyness is 99.7%.

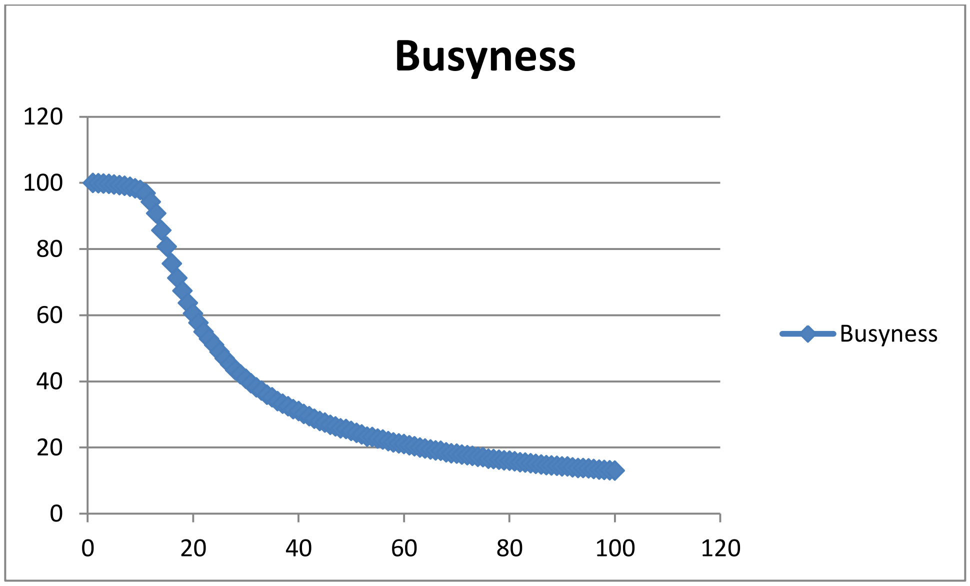
c.) This distribution of mean wait times is likely due to the relationship between each urgency level's proportion of patients and its treatment time. First, the proportion of patients for an urgency level is negatively associated with mean wait time. Essentially, increasing the percentage of patients seen for an urgency level decreases the wait time. This relationship can be explained by the fact that because patients of all urgency are chosen from a single queue, urgency levels with more patients are statistically more likely to appear at the front of the queue and be seen by a doctor. More patients of a single urgency level seen by a doctor indicates that their mean wait time will be lower. This reasoning helps explains why urgency levels 2 and 3, who have significantly the largest percentages of the population, have the lowest mean wait times.

Second, the results show that the relationship between each urgency level's average treatment time and mean wait time is a function of the urgency level's percentage of population and doctor busyness. Urgency levels with low treatment times (e.g. levels 1 & 2) use less of the doctor's time and therefore decrease doctor busyness while urgency levels with high treatment times (e.g. levels 3 & 4) increase doctor busyness. When doctor busyness decreases, it means that the patients waiting in the queue will be seen faster. This fact is advantageous for urgency levels with high population percentages, as they will statistically be seen more and their mean wait times will decrease.

2.







a. It can be inferred from the data and testing that the minimum number of doctors required to bring the all the maximum wait times down below 3 hours is 12 doctors. At this stage, the maximum wait time reported was 96 minutes by a level 3 patient.
b. Same as the answer above, 12 doctors is the minimum number of doctors required to bring the mean wait times all below 1 hour. The largest mean wait time is 33.55 minutes for the level 4 patients.
c. It can be inferred from the data and testing that the minimum number of doctors required to drop the doctor busyness below 75% is 17 doctors (the percentage was 75.56% at 16 doctors). When there are 17 doctors present, the doctor busyness is 71.2%. Overall, doctor busyness is close to a logistic function.

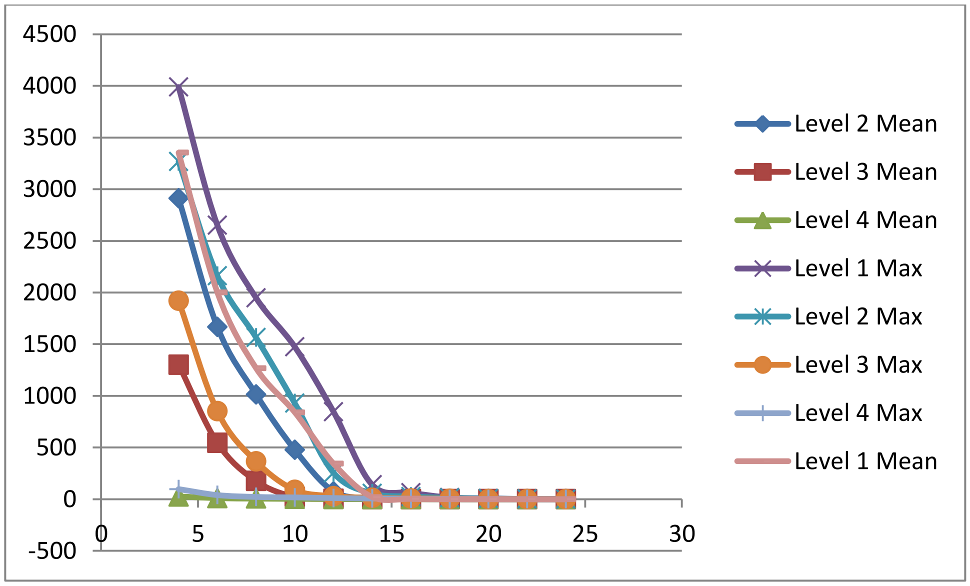
Part 2B

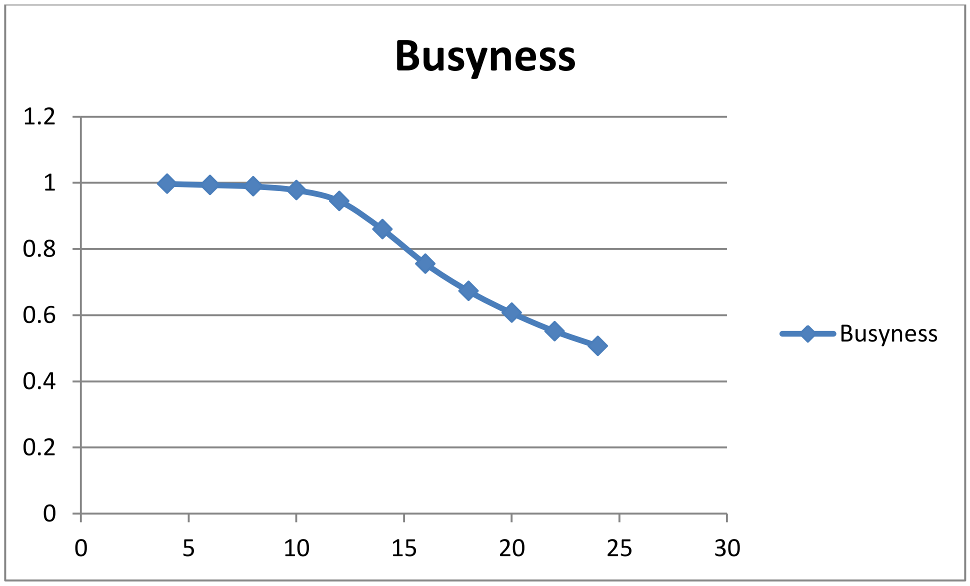
1.
    a. The effect of prioritization on the mean wait times with default values is a drastic increase in wait times for levels 1 and 2, relatively no difference in wait time for level 3, and a significant drop in wait time for level 4. Similarly, the max wait times for levels 1 and 2 have greatly increased, level 3 decreased somewhat, and the level 4 max wait time extremely decreased.
    b. The mean doctor busyness in part 2B was almost exactly the same as the busyness in part 2A. There was only a slight decrease of 0.1%.
2.
    a. The effect of emergencies is to bump lower urgency level patients to higher levels of urgency. Patients that are bumped up to higher urgency levels decrease the added benefits of treating patients that come in with higher urgency levels. Therefore, as the probability of emergency increases, the distribution of mean and max wait times

becomes closer and closer to the distribution of wait times in part 2A. In fact, when the emergency probability is 1 (every patient has an emergency), the mean wait times are almost exactly the same as in Part2A and the distribution of the max wait times is the same, they are all just greater.

b. The effect of walkouts is that fewer level 1 patients will be served and the percentage of patients served will decrease. In practice (i.e. non-extreme walkout probabilities and walkout times), because the mean wait time of level 1 patients is so large (3348 minutes) and if there is no promotion policy, all the level 1 patients will walkout. The wait times for the other emergency levels were completely unchanged.

3.





a. To reduce the max wait times below 3 hours with the prioritization system, 14 doctors are needed.
b. Similar to part a, to reduce the mean wait times to below 1 hour, 14 doctors are required.
c. To reduce mean doctor busyness below 75%, 17 doctors are required.

4. It is better to use a 45 minute promotion policy than a 90 minute promotion policy because the

mean wait times for levels 1 and 2 are each 100 minutes better than when the promotion is at 90 minutes. Furthermore, the mean wait times for levels 3 and 4 are the same and all the max wait times are the same. Therefore, it is better to use the 45 minute promotion policy.

5. In the case where several patients are promoted in the same timestep, each patient being promoted is placed into a queue, one queue for level 1 and a queue for level 2. Therefore, not only will the order of patient promotion be preserved, but it is easy to remember which level to promote the patient to.
6.
   a. Checking for patients that have emergencies runs in O(n). I the implementation, there are 2 for loops. However, because both of these loops will run a maximum of 4 times, their running time is O(c). Furthermore, the array is never traversed. However, the dequeue(int) method is called. Because this function must shift elements over to dequeue the selected index, the worst case running time is O(n). Therefore, the worst case running time for checking emergencies is O(n).
   b. The worst case running time for checking for walk outs is O(n). Because one of the for loops iterates over the size of the queue, this loop's running time is O(n). Also, the dequeue(int) function is called, whose worst case running time is O(n). However, because these two operations are independent of each other, the overall worst case running time for checking for walkouts is O(n).
   c. Because checking for promotions uses 2 for loops that independently iterate over the length of 2 queues, the running time for checking for promotions is O(n) time. Also, the order in which promotions are checked for is important. That is to say, you need to check level 2 for promotions before checking level 1. This is because it is theoretically possible that some patients are promoted from level 1 to level 2 first, and then are chosen to be promoted to level 3 because their wait time exceeded the time for the promotion policy and their wait time is never rest. It is possible that there could be patients promoted straight from level 1 to level 3. Therefore, it is required to check level 2 patients for promotions before checking level 1 patients.