**CS 232: DATA STRUCTURE – FINAL PROJECT**

**QUESTIONS FOR A LEVEL**

By Kieu My Vo

1. **How did you decide how to count the amount of “work” each function is doing? What did you change in the code?**

I decided to count the amount of "work" each function is doing by incrementing a "work" counter variable for each operation or step that is performed in the function, and also added a self.currentSize variable to keep track of the size of the queue.

Specifically, for the \_\_str\_\_ method, I initialized the work variable to 0 and incremented it by 1 for each operation that was performed in order to count the operation of getting the length of the items list.

For the is\_empty method, I incremented the work variable by 1 to count the comparison operation and also used a self.currentSize variable to keep track of the size of the queue.

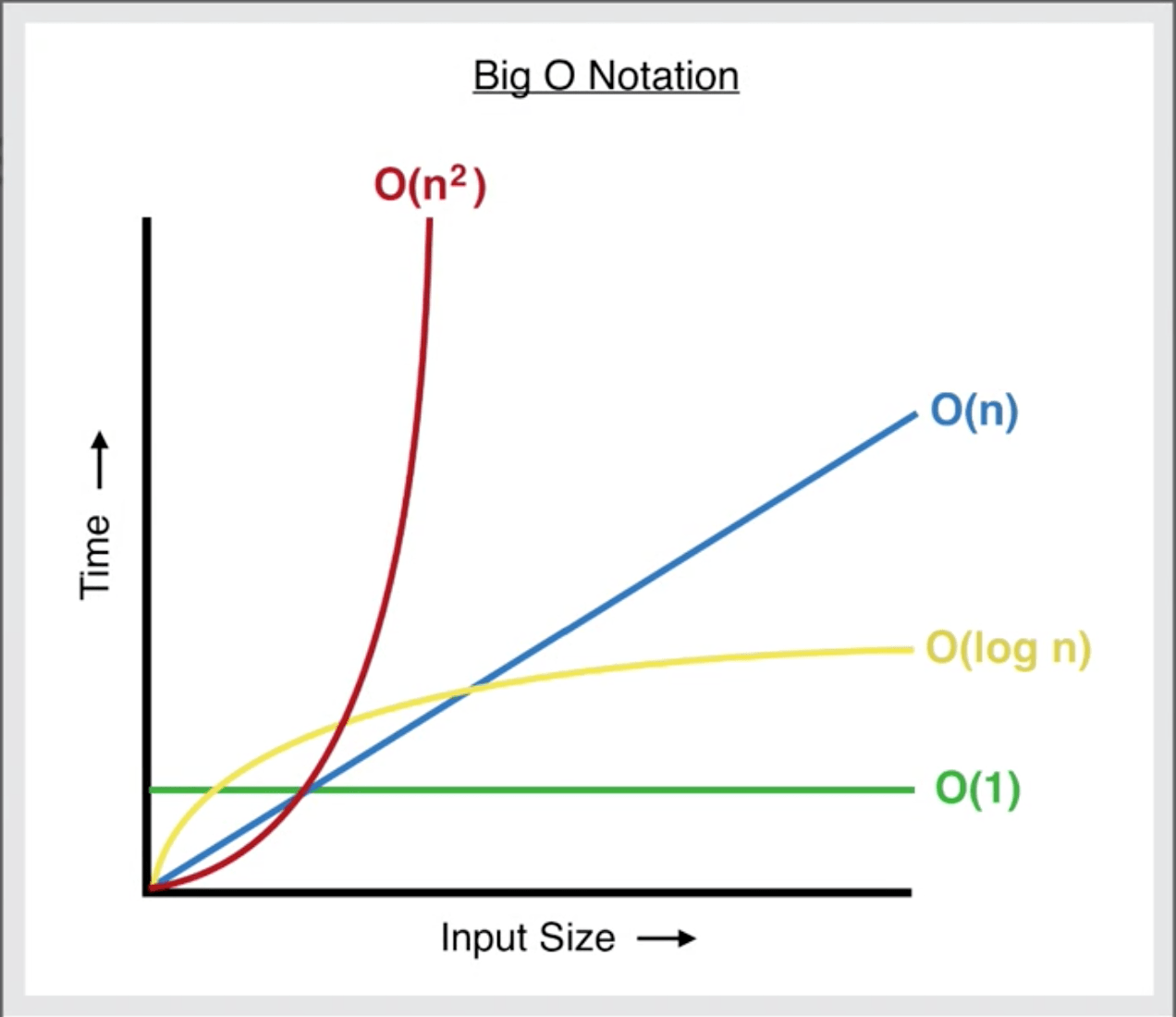
Similarly, for the enqueue method, I incremented the work variable by 1 to count the operation of appending the item to the items list and also updated the self.currentSize variable to keep track of the size of the queue.

For the dequeue method, I initialized the work variable to 0 and incremented it by 1 for each operation that was performed, such as iterating through the items list and updating the self. I also returned the work variable along with the size of the queue to indicate the amount of work done and the resulting size of the queue.

1. **Are the functions running in the expected Big O, more or less? Explain your answer and back it up with data for each function.**

The functions in both PQList\_test and PQNode\_test files are running the expected Big O.

* PQList\_test.py
* The enqueue function adds an item to the end of the list, which takes constant time on average O(1), so it can be clearly seen in my PQListDataTest\_enqueue.csv that column "work" returns the same value of 1.
* The is\_empty function simply checks if the list is empty, which takes constant time O(1), regardless of the input size. Specifically, in the PQListDataTest\_isEmpty.csv, column "work" returns the same value of 1.
* The size function simply returns the length of the list, which takes constant time O(1), regardless of the input size. Therefore, it can be clearly seen in my PQListDataTest\_size.csv that column "work" returns the same value of 1.
* The dequeue function searches for the highest-priority item in the list, which takes linear time O(n), where n is the length of the list. It also can be clearly seen in PQListDataTest\_dequeue.csv file that the “work” being done isn’t exactly O(n), but it’s pretty much the same with O(n+23).
* The \_\_str\_\_ function is similar to dequeue, it prints a string representation of all the items in the list, which takes linear time O(n), where n is the length of the list. Therefore, in PQListDataTest\_str\_.csv file, that the “work” being done isn’t exactly O(n), but it’s pretty much the same with O(n+8).
* All in all, the graphs for all functions are linear. However, the graphs for enqueue, isEmpty, and size are the linear O(1) which is parallel to the horizontal axis of size (n). Graphs for dequeue and \_\_str\_\_ functions are the linear O(n), which exactly the same with the picture below.



* PQNode\_test.py:
* Similarly with PQList\_test, the enqueue, is­­\_empty, and size are take constant time O(1), regardless of the input size. Therefore, the graphs for enqueue, isEmpty, and size are the linear O(1) which is parallel to the horizontal axis of size (n).
* The dequeue function searches for the highest-priority item in the list, which takes linear time O(n), where n is the length of the list. It also can be clearly seen in PQNodeDataTest\_dequeue.csv file that the “work” being done isn’t exactly O(n), but it’s pretty much the same with O(3\*n).
* The \_\_str\_\_ function is similar to dequeue, it prints a string representation of all the items in the list, which takes linear time O(n), where n is the length of the list. Therefore, in PQNodeDataTest\_str\_.csv file, the “work” being done isn’t exactly O(n), but it’s pretty much the same with O(2\*n).
* Graphs for dequeue and \_\_str\_\_ functions are the linear O(n), which exactly the same with the picture above.

1. **How does the “work” required to run the different operations change as the size of n changes? If you were to chart the times and work compared to the number of n, what would the graphs look like? Were you surprised by any of the outputs? If so, what?**

The “work” required to run the different operations change as the size of n changes because as the size of n increases, the number of items that need to be processed by the functions also increases, which in turn increases the amount of work that needs to be done.

As I have mentioned in the previous question and with the picture above, the graphs would likely show an increase in both as n increases. However, while graphs for enqueue, isEmpty, and size are the linear O(1) which is parallel to the horizontal axis of size (n), graphs for dequeue and \_\_str\_\_ functions are the linear O(n) which show an upward trend.

Overall, these results are not surprising, as they align with what I would expect based on the Big O analysis of the functions.