**REPORT**

**Thesis**

Array queue push: O(1) amortized

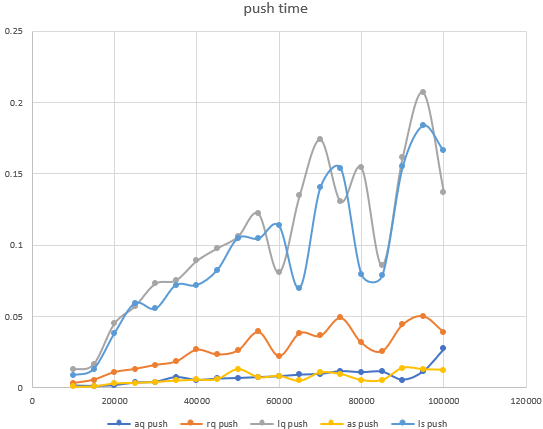
Ring queue push: O(1)

Array stack push:O(1) amortized

Linked queue push: O(1)

Linked stack push: O(1)

**Data**



**Analysis**

Based on the thesis, it looks like all the data structures we implemented seem to have a close runtime compared to each other. However, it turns out array queue and array stack had the shortest runtime due to their amortizing algorithm. I was surprised by how much longer linked queue and linked stack took to run. In regard to efficiency in runtime and complexity, array stack and array queues are the best data structures.

**Thesis**

Array queue pop: O(n)

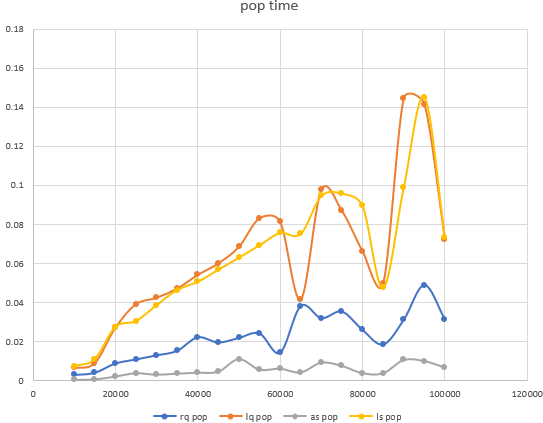
Ring queue pop: O(1)

Array stack pop: O(1)

Linked stack pop: O(1)

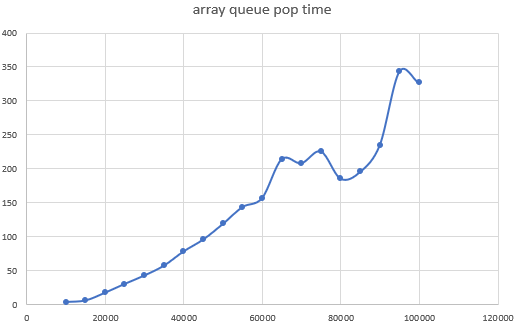
Linked queue pop: O(1)

**Data**



**Analysis**

Based on the thesis, array queue seems to take the most time when removing an element at the end, having to go through n elements and move their position. The rest of the structures have the same O(1) time complexity. In real time results, array stack and ring queue were most efficient in time. Unlike push results, linked queue and linked stack did not have as good runtimes, but they were still much more efficient than array queue.



Array queue was isolated in its own graph because its data was much more significant compared to the rest of the data structures. Array queue should not be used when one simply wants to remove an element at the end of a list.

**Conclusion**

In conclusion, array stack is the most efficient in time complexity when running push() and pop().