1. Follow along with the in-class exercise on this, do your best to get it working, and turn in what you come up with here!
2. Be sure to include at least one test for each function or piece of functionality that should verify that your code is working!  No slacking , you should start writing some tests *before* you write your implementations (just spend a few minutes thinking about the design and then write a few tests using natural language (English is preferred for me to be able to read it  )) A screenshot of a computer

   Description automatically generated with medium confidenceA screen shot of a computer

   Description automatically generated with medium confidence
3. Create an array-based list or a linked-list (**and a bonus for attempting both**) that:

(I tried both but only got the array based one working fully)

* 1. automatically inserts values in the correct position based on some order of sorting (perhaps ascending integers or lexicographical sorting of words) A screenshot of a computer program

     Description automatically generated with low confidence
  2. efficiently searches for elements (likely binary search for the array list, but what about the linked-list?) A computer code on a black background

     Description automatically generated with low confidence

1. Make a chart to compare the algorithmic complexity (use Big-O notation) of your insert, remove, and search algorithms you used for your structures

|  |  |  |  |
| --- | --- | --- | --- |
| functions | queue | Linked list | Autosorted list |
| Insert | O(1) | O(N) | O(logN) |
| Remove | O(1) | O(N) | n/a |
| search | O(1) only peeks ends | O(N) | O(logN) |

1. Once you have implemented and tested your code, add to the README file what line(s) of code or inputs and outputs show your work meeting each of the above requirements (or better, include a small screen snip of where it meets the requirement!).