Homework #2

Megan Ku
Data Structures and Algorithms

February 7, 2020

1

Implement a doubly linked list
See mku.py.

2

Analyze operation runtime

2.a

Index operation runtime

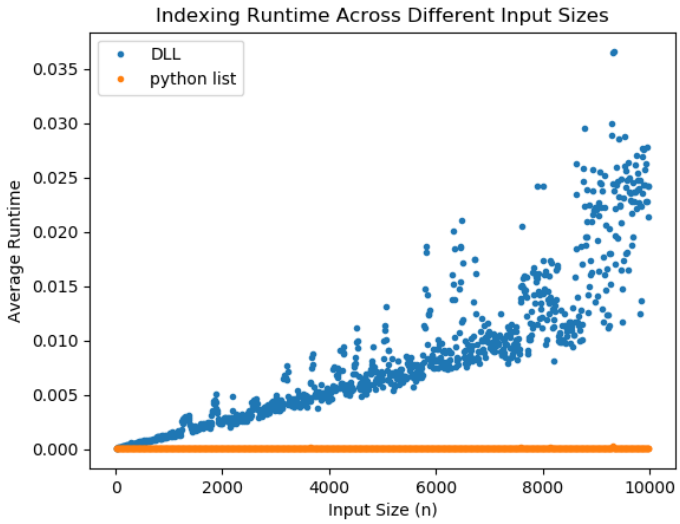

Figure 1: Runtime of doubly-linked list as n, the length of the DLL, increases in size from 10 to 10,000.

Let n be the length of the list or doubly linked list. There is hardly a change in runtime for indexing a python list, whether the length be 10 elements or 10,000. In comparison, the runtime for indexing a doubly-linked list (DLL) increases in a linear fashion as n increases. I noticed that there are a few outliers that I don't quite know how to interpret yet, but from the majority of the points, I can confirm that indexing in a python list is O(n), or constant time, while indexing in a doubly linked list occurs in O(n) time.

2.b

Multiply pairs operation runtime

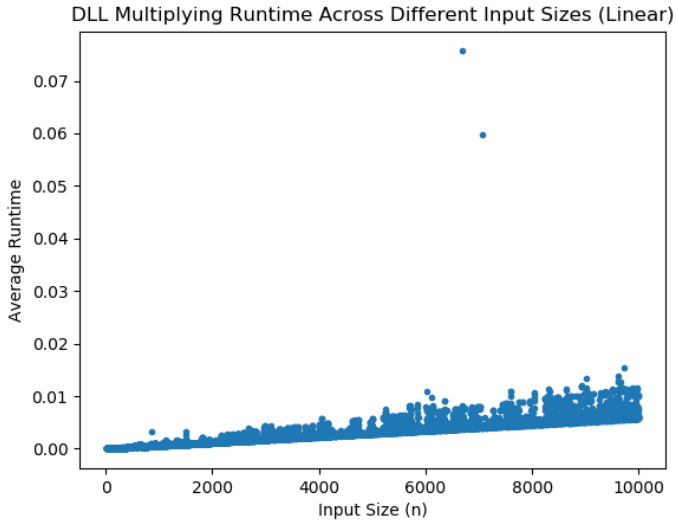

Figure 2: Runtime of multiplying pairs in a doubly-linked list as n, the length of the DLL, increases. This operation was written to have a runtime of O(n).

I had anticipated this multiplying operation to take O(n) time, where n is the number of elements of the doubly linked list. The data in the graph verifies this analysis; as n increases, the runtime for the operation also increases in a linear fashion.
Like with the runtime analysis for the indexing operation, there are a few outliers that I can't quite interpret in the graph for multiplying pairs.
I wanted to compare the runtime graphs of the linear implementation to the quadratic implementation, and the difference in the trajectory of the points as n increases is quite obvious. As n increases, the runtime to multiply all pairs increases quadratically, confirming that this quadratic implementation has a runtime of $O(n^2)$.

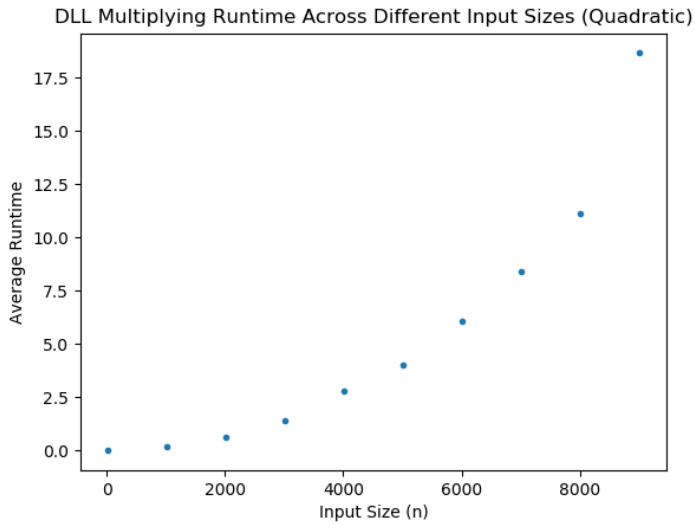

Figure 3: Runtime of multiplying pairs in a doubly-linked list as n, the length of the DLL increases. This operation has a runtime of O(n).