Ryan Ellis
COSC 220-003
Project #2: Linked Lists
Section 3.2.3, 3.2.4 – Timing Insertions
11/5/2024

Recorded Output Findings

Push Front Orientation Insertion

Upon reviewing the data recorded from the main program output tests, it can be noted that the timing for each front insertion structure is comparable in value with the exception of the **Vector** structure. Also it can be noted from the data table and figure below of the front insertion method that there are similarities in shape of the graph(s) with respect to the behavior of every structure except the **Vector**, wherein the **Vector** graph has the form of a parabola and every other structure has a linear form. This is indicative that the **Vector** structure has a hefty computational cost vs. the other structures in this specific instance. The linked list structures all seem to behave in the same manner, with a slight difference in the **Vector** and **List** (STL) structures.

Push Back Orientation Insertion

Looking now at the data recorded from the push back timing for each insertion method, every structure is comparable in value with the exception of the **Linked List** structure (including the textbook version). These data values are slightly different Also it can be noted from the data table and figure below of the push back insertion method that there are similarities in shape of the graph(s) with respect to the behavior of every structure except the **Linked List**, wherein the **Linked List** graph has the form of a parabola and every other structure has a linear form. This is indicative that the **Linked List** structure has a hefty computational cost vs. the other structures in this specific instance. The linked list structures all seem to behave in the same manner, with a slight difference in the **Linked List** structure.

Conclusion

Looking at the design for both the **Linked List** and **Vector** structures, we know that the **Linked List** structure is composed of nodes that contain a value and pointer to the next node in the sequence whereas a **Vector** structure is built with a dynamic array and automatic memory allocation. Given this information, it can be surmised that the data correctly reflects the data structure, where a **Linked List** would excel in insertion from either end point since it is just appending nodes on to the existing structure, whereas a **Vector** structure would take a bit longer to insert with shifting addresses of values within the array. Conversely, appending to the end of the array will result in the best outcome of data computation efficiency with a **Vector** structure, as it doesn't have to shift any significant amount of values to accommodate this insertion, whereas in a **Linked List** structure, a traversal is needed to go through the list to insert the new node at the end of the list.

List Sizes -Front	Linked (s)	Double (s)	Text (s)	Vector (s)	STL (s)
10000	0.000595	0.000585	0.000608	0.008546	0.001373
15000	0.000891	0.000882	0.000952	0.022285	0.002046
25000	0.002398	0.0015	0.001601	0.06847	0.002833
50000	0.002986	0.002979	0.003184	0.19396	0.004496
75000	0.004412	0.004434	0.004705	0.428883	0.006861
100000	0.003978	0.005796	0.006138	0.801019	0.008906
150000	0.009521	0.008664	0.00928	1.93954	0.015977
200000	0.013916	0.014127	0.0112	3.60058	0.020349

Table 1: Push Front Insertion

List Sizes -Back	Linked (s)	Double (s)	Text (s)	Vector (s)	STL (s)
10000	0.246793	0.00018	0.23663	0.000787	0.000757
15000	0.555305	0.000326	0.531957	0.001113	0.001081
25000	1.47846	0.00046	1.47413	0.001812	0.001793
50000	5.90486	0.000886	6.05216	0.003647	0.003809
75000	13.3651	0.002588	14.3434	0.005692	0.00702
100000	24.3308	0.003448	24.1679	0.007142	0.008775
150000	63.3582	0.0053	61.2326	0.011225	0.015316
200000	108.32	0.00699	108.116	0.015492	0.021214

Table 2: Push Back Insertion

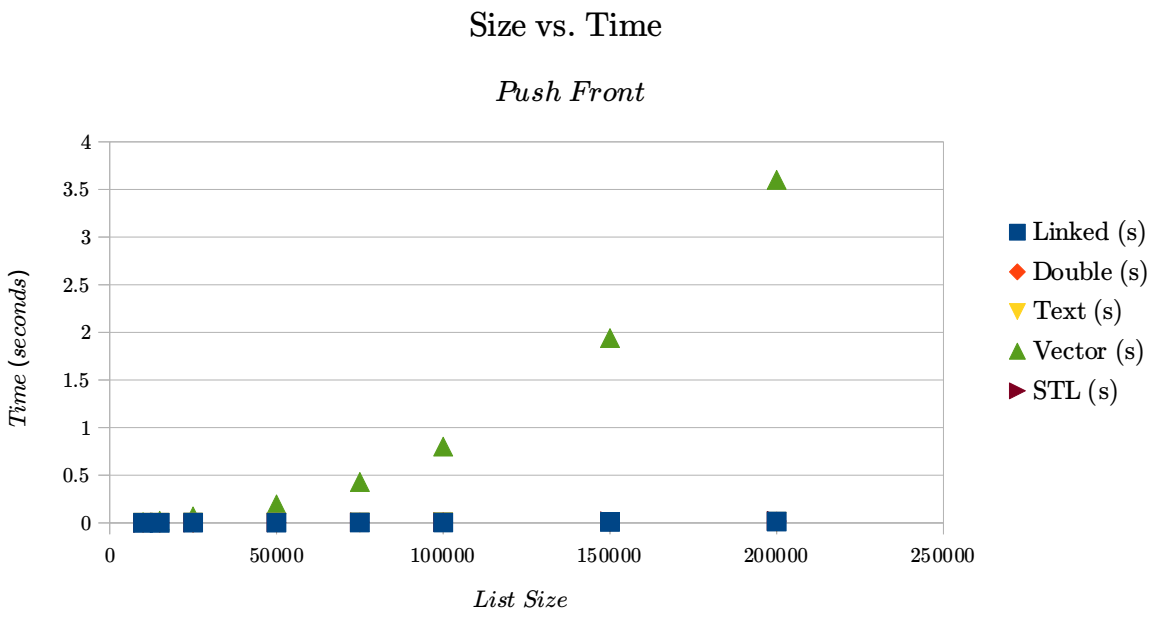

Figure 1: Push Front Insertion

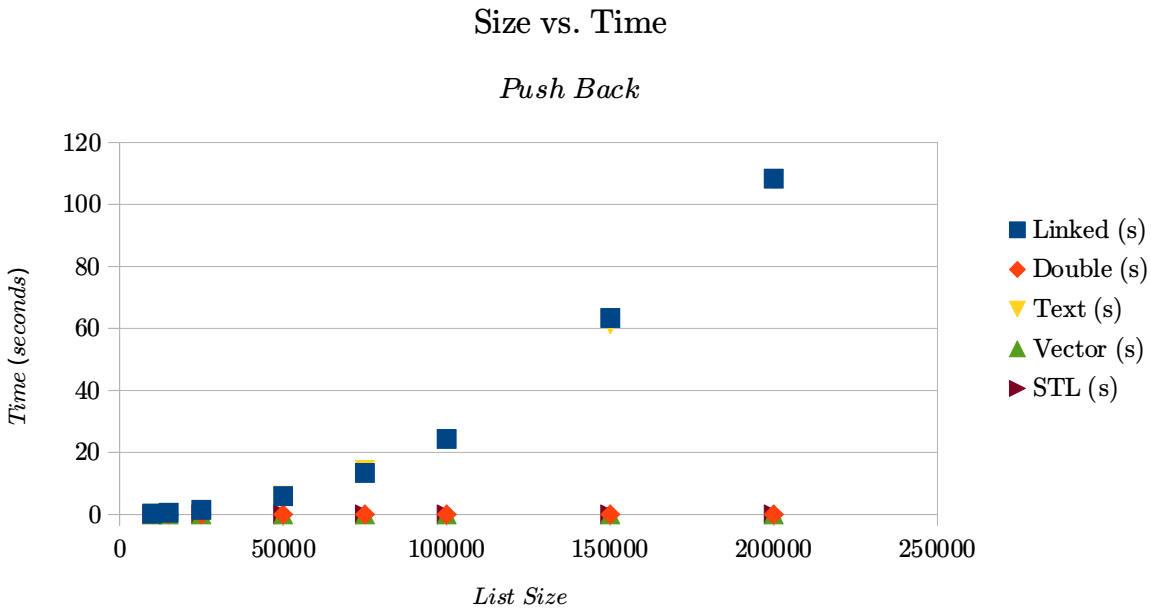

Figure 2: Push Back Insertion