Data		Array		Pointer		std Library		Time difference (array)	Time difference (ptr)
		Stack	List	Stack	List	Stack	List		
Iterated Insertion	Front	-	98.04	-	0.317	-	0.861	97.179	-0.544
	Back	-	0.133	-	138.263	-	0.78	-0.647	137.483
Traversal	-	-	0.114	-	275.19	-	0.157	-0.043	275.033
Iterated Deletion	Front	-	109.508	-	0.1	-	0.549	108.959	-0.449
	Back	-	0.068	-	254.163	-	0.552	-0.484	253.611
Iterated Push	-	0.125	-	0.214	-	0.282	-	-0.157	-0.068
Iterated Pop	-	0.124	-	0.205	-	0.185	-	-0.061	0.02

Based on data:

* **Array implementation of List** test data clearly shows the short comings described in the book
____Both delete and insert front requires the rest elements iteratively.

* **Array implementation of Stack** test data, however, is faster that Pointer implementation
____Stack structure here operates on the last element instead of the first, so array is much lighter and runs significantly faster.

* **Pointer implementation of Stack** data
____Both push and pop operation run time is really close to std library data

* Special notice on **Pointer implementaion of List**, as Back insertion, Traversal and Back deletion
____All 3 operations involve calling end();
____END() traverses through the whole tree with text book implementation, the spike of time required is huge
____The traversal has highest duration, but in fact it only calls to end() once.
____The iterated deletion calls to end() twice, but the size of the list continuously decrease, thus it takes less time.

* **Which is closer to std library?**
____The data shows that **pointer implementation is** much closer to std library of both stack and list, if points above are taken into consideration.
____Front push and Front deletion time difference are really low
____It is really hard to improve the performance of an array as elements have to be shifted no matter what
____A way to significantly reduce duration, which the std library use, is to have a pointer to previous node, which will eliminate calls to end()