Timetest.cpp Write up

File	ADT#	Time#1	Time#2	Time#3	Average Time
File 1	LinkedList	0.084551	0.057769	0.057926	0.066749
File 2	LinkedList	72.9625	72.9268	72.8575	72.9156
File 3	LinkedList	0.070696	0.045259	0.041231	0.052395
File 4	LinkedList	47.2658	47.3874	47.4504	47.3679
File 1	CursorList	0.070517	0.050961	0.050126	0.057201
File 2	CursorList	342.78	0.05228	0.05037	114.29422
File 3	CursorList	0.069583	0.052986	0.052278	0.058282
File 4	CursorList	171.643	0.054835	0.052595	57.250143
File 1	StackAr	0.037751	0.033452	0.033074	0.034759
File 2	StackAr	0.03533	0.031998	0.031756	0.033028
File 3	StackAr	0.034998	0.032124	0.031969	0.033030
File 4	StackAr	0.039232	0.034423	0.034092	0.035916
File 1	StackLi	0.054004	0.048695	0.048628	0.050442
File 2	StackLi	0.043899	0.038386	0.038457	0.040247
File 3	StackLi	0.042069	0.038449	0.038421	0.039646
File 4	StackLi	0.044025	0.040666	0.040663	0.041785
File 1	QueueAr	0.03764	0.034758	0.034249	0.035549
File 2	QueueAr	0.039264	0.034776	0.034676	0.036239
File 3	QueueAr	0.037412	0.033947	0.033619	0.034993
File 4	QueueAr	0.039911	0.036839	0.036146	0.037632
File 1	SkipList	0.155413	0.157386	0.16141	0.158070
File 2	SkipList	0.121777	0.115652	0.120241	0.119223
File 3	SkipList	0.135931	0.135716	0.132386	0.134678
File 4	SkipList	0.33081	0.348026	0.330587	0.336474

File	ADT#	Individual Insertion	Individual Deletion	Series of Insertions	Series of Deletions	Entire File
File 1	Linked List	O(1)	n/a	O(N)	n/a	O(N)
File 2	Linked List	O(1)	O(N)	O(N)	O(N^2)	O(N^2)
File 3	Linked List	O(1)	O(1)	O(N)	O(N)	O(N)
File 4	Linked List	O(1)	O(N)	O(N)	O(N^2)	O(N^2)
File 1	CursorList	O(1)	n/a	O(N)	n/a	O(N)
File 2	CursorList	O(1)	O(N)	O(N)	O(N^2)	O(N^2)
File 3	CursorList	O(1)	O(1)	O(N)	O(N)	O(N)
File 4	CursorList	O(1)	O(N)	O(N)	O(N^2)	O(N^2)
File 1	StackAr	O(1)	n/a	O(N)	n/a	O(N)
File 2	StackAr	O(1)	O(1)	O(N)	O(N)	O(N)
File 3	StackAr	O(1)	O(1)	O(N)	O(N)	O(N)

File 4	StackAr	O(1)	O(1)	O(N)	O(N)	O(N)
File 1	StackLi	O(1)	n/a	O(N)	n/a	O(N)
File 2	StackLi	O(1)	O(1)	O(N)	O(N)	O(N)
File 3	StackLi	O(1)	O(1)	O(N)	O(N)	O(N)
File 4	StackLi	O(1)	O(1)	O(N)	O(N)	O(N)
File 1	QueueAr	O(1)	n/a	O(N)	n/a	O(N)
File 2	QueueAr	O(1)	O(1)	O(N)	O(N)	O(N)
File 3	QueueAr	O(1)	O(1)	O(N)	O(N)	O(N)
File 4	QueueAr	O(1)	O(1)	O(N)	O(N)	O(N)
File 1	SkipList	O(log (N))	n/a	O(Nlog(N))	n/a	O(Nlog(N))
File 2	SkipList	O(log (N))	O(Nlog(N))	O(Nlog(N))	O(Nlog(N))	O(Nlog(N))
File 3	SkipList	O(log (N))	O(Nlog(N))	O(Nlog(N))	O(Nlog(N))	O(Nlog(N))
File 4	SkipList	O(log (N))	O(Nlog(N))	O(Nlog(N))	O(Nlog(N))	O(Nlog(N))

LinkedList performs same on insertions, since the cost to unlink two adjacent nodes and insert a new node is always constant. There is no shifting required after inserting a new node. For File 2 and File 4, individual deletion is O(N) because the algorithm must go through N list nodes to find the value to be deleted. File 3 is O(N) for LinkedList because nodes are being inserted into the front of the list, and the algorithm can iterate through the list and delete items in order.

CursorList performs same on insertions, since the cost to unlink two adjacent nodes and insert a new node is always constant. There is no shifting required after inserting a new cursor node. For File 2 and File 4, individual deletion is O(N) because the algorithm must go through N list nodes to find the value to be deleted. File 3 is O(N) for CursorList because elements are being inserted into the front of the list, and the algorithm can simply iterate through the list and delete items in order.

StackAr insertions have constant time since values are simply inserted into the next index of the vector (the top of the stack). The deletions for files 2, 3, and 4 also have constant time as the actual value of the element to be deleted can be ignored. These elements are removed from the end of the vector representing the stack, depending on order of insertion rather than value. Hence, all operations are O(N).

StackLi insertions have constant time since values are inserted at the beginning of the list (the top of the stack) using a “head” pointer. The deletions for files 2, 3, and 4 also have constant time as the actual value of the element to be deleted can be ignored. These elements are removed from the top of the list representing the stack, depending on order of insertion rather than value. Hence, all operations are O(N).

QueueAr insertions have constant time since values are inserted at the end of the vector representing the queue. The deletions for files 2, 3, and 4 also have constant time as the actual value of the element to be deleted can be ignored. These elements are removed from the front of the vector representing the queue, depending on order of insertion rather than value. Hence, all operations are O(N).

For SkipList, both insertions and deletions require a search starting from the highest link in the head node and traversing through higher levels to lower levels until the value is found. Since the maximum height of the SkipList is O(log2(n)), the maximum possible number of moves that can be made down the skip list for, all files, is O(log2(n)). Traversing forward on a level is an O(1) operation. So the total cost of traversing down levels and traversing forward levels to find an element is O(log2(n)) + O(1) = O(log2(n)) for all files.

StackAr, StackLi and QueueAr are nearly identical in terms of insertion and deletion. Elements are added to the front for all three data structures, regardless of data values, which is an O(1) operation. Similarly, deletions are O(1) for all three types because a search is not required. As a result, all three of these data types require O(N) amount of time to run and have very similar time complexities for all the data files. Although linked lists and CursorLists have the same time complexity, CursorLists take significantly longer times on average for files 2 and 4. This is explained by the fact that CursorList needs to move and keep track of its cursor for both insertions and deletions, using Freelist, which causes some overhead. Comparing LinkedList, CursorList, and skipList, skipList is more ideal for unpredictable behavior as demonstrated with File 4 ; the randomized insertions and deletions in File4 take less time for SkipList than the other two lists. On the other hand, if only insertions are performed as in File1, or data is removed in the opposite order of insertion, as in File3, both LinkedList and CursorList perform faster than

SkipList, since LinkedList and CursorList add nodes to the front of list while SkipList performs a search prior to insertion.