Logan Rockmore 18 January 2006 ECS 110 Program 1 Write Up

Linked List	File1.dat	File2.dat	File3.dat	File4.dat
Run 1	0.69	57.65	0.69	36.3
Run 2	0.69	57.67	0.66	36.52
Run 3	0.71	57.51	0.68	36.62
Average Run	0.697	57.61	0.677	36.48

Cursor List	File1.dat	File2.dat	File3.dat	File4.dat
Run 1	0.63	210.79	0.65	110.91
Run 2	0.64	211.2	0.66	208.43
Run 3	0.63	210.82	0.66	216.92
Average Run	0.633	210.937	0.657	178.753

Stack Array	File1.dat	File2.dat	File3.dat	File4.dat
Run 1	0.6	0.64	0.64	0.65
Run 2	0.63	0.63	0.63	0.66
Run 3	0.74	0.65	0.64	0.66
Average Run	0.657	0.64	0.637	0.657

Stack List	File1.dat	File2.dat	File3.dat	File4.dat
Run 1	0.66	0.66	0.65	0.68
Run 2	0.66	0.67	0.67	0.68
Run 3	0.66	0.69	0.66	0.69
Average Run	0.66	0.673	0.66	0.687

Queue Array	File1.dat	File2.dat	File3.dat	File4.dat
Run 1	0.6	0.64	0.67	0.67
Run 2	0.63	0.66	0.68	0.68
Run 3	0.62	0.66	0.67	0.67
Average Run	0.617	0.653	0.673	0.673

Logan Rockmore 18 January 2006 ECS 110 Program 1 Write Up

Skip List	File1.dat	File2.dat	File3.dat	File4.dat
Run 1	7.5	2.44	4.1	2.99
Run 2	7.25	2.4	3.98	2.92
Run 3	7.33	2.39	4.0	2.9
Average Run	7.36	2.41	4.027	2.937

Linked List	File1.dat	File2.dat	File3.dat	File4.dat
Individual Insertion	O(1)	O(1)	O(1)	O(1)
Individual Deletion		O(N)	O(1)	O(N)
Entire Series of Insertions	O(N)	O(N)	O(N)	O(N)
Entire Series of Deletions		$O(N^2)$	O(N)	$O(N^2)$
Entire FIle	O(N)	$O(N^2)$	O(N)	$O(N^2)$

Cursor List	File1.dat	File2.dat	File3.dat	File4.dat
Individual Insertion	O(1)	O(1)	O(1)	O(1)
Individual Deletion		O(N)	O(1)	O(N)
Entire Series of Insertions	O(N)	O(N)	O(N)	O(N)
Entire Series of Deletions		$O(N^2)$	O(N)	$O(N^2)$
Entire FIle	O(N)	$O(N^2)$	O(N)	$O(N^2)$

Stack Array	File1.dat	File2.dat	File3.dat	File4.dat
Individual Insertion	O(1)	O(1)	O(1)	O(1)

Stack Array	File1.dat	File2.dat	File3.dat	File4.dat
Individual Deletion		O(1)	O(1)	O(1)
Entire Series of Insertions	O(N)	O(N)	O(N)	O(N)
Entire Series of Deletions		O(N)	O(N)	O(N)
Entire FIle	O(N)	O(N)	O(N)	O(N)

Stack List	File1.dat	File2.dat	File3.dat	File4.dat
Individual Insertion	O(1)	O(1)	O(1)	O(1)
Individual Deletion		O(1)	O(1)	O(1)
Entire Series of Insertions	O(N)	O(N)	O(N)	O(N)
Entire Series of Deletions		O(N)	O(N)	O(N)
Entire FIle	O(N)	O(N)	O(N)	O(N)

Queue Array	File1.dat	File2.dat	File3.dat	File4.dat
Individual Insertion	O(1)	O(1)	O(1)	O(1)
Individual Deletion		O(1)	O(1)	O(1)
Entire Series of Insertions	O(N)	O(N)	O(N)	O(N)
Entire Series of Deletions		O(N)	O(N)	O(N)
Entire FIle	O(N)	O(N)	O(N)	O(N)

Skip List	File1.dat	File2.dat	File3.dat	File4.dat
Individual Insertion	O(log N)	O(log N)	O(log N)	O(log N)
Individual Deletion		O(log N)	O(log N)	O(log N)
Entire Series of Insertions	O(N log N)	O(N log N)	O(N log N)	O(N log N)
Entire Series of Deletions		O(N log N)	O(N log N)	O(N log N)
Entire FIle	O(N log N)	O(N log N)	O(N log N)	O(N log N)

The linked list and cursor list data structures are very similar in their time complexities. My code, as it were, has both lists inserting data to the front of the list, rather than at the back. As a result, the insertion code for both of these data structures is simply O(1), since the insertion routine does not matter with respect to the data value. The deletion routines are also very simple to understand. Since, for both lists, the worst possible case is that the item being deleted is at the very end, the routines are O(N). For some of the data files, because of the way the data is formatted, the deletion routine is much faster. For example, File3.dat is much faster than File2.dat in both lists because deleting backwards down the list will result in O(1) routine time, whereas deleting forwards up the list will require searching until the end of the list for every single deletion. As a result, File2.dat and File4.dat (being completely random) provide the worst possible time complexities for both the linked list and the cursor list.

One interesting note about both of the list data types is how they handle File4.dat. The files randomly insert values, which isn't a problem since these lists aren't sorted. However, the true test for the data structures comes with the deletion, where they must run a search for the specific value before deleting it. Now, if we were trying to look at the worst possible

scenario, we would have the situation that exists in File2.dat. We can see that this is very possible since some of the running times in File4.dat are very close to those in File2.day, especially in the cursor list. However, the data structures can be quicker, as is shown by the case of the linked list. In this situation, the program runs about 50% quicker than the worse case scenario, overall.

The stack array, stack list and the queue array all act in a similar way because insertion and deletion are virtually identical. For all implementations, the insertions are simple O(1), since, no matter the data value, we are adding the value to the front of the data structure. For both stack types, the deletion routines are O(1) because there is no need to search through the data structure for the value to delete. With the simplicity of the format of a stack, we cannot remove a specific value, but simple pop the top of the stack. Since the function is so simple, it merely requires O(1) amount of time. The same stands true for the queue array, except that it is the item at the end of the queue that is being removed, not at the beginning. As a result, all three of these data types require O(N) amount of time to run and create very similar time complexities over the board for all data files.

Finally, the skip list presents an interesting situation. This is the only data structure of the six that we tested in which both the inserts and the deletions required an amount of searching by the structure. Both inserts and deletions of a single value are O(log N), since the way the structure is formatted provides for faster clock times. The entire series accounts for N numbers, so we have times of O(N log N). This data structure, as opposed to all the others, provides for a very good average calculation time, especially when compared to the lists. While the lists took around 57 and 210 CPU cycles on average to complete the instructions in File2.dat, the skip list took only 2.4 on average. The insertions into the structure were slower than those of the other lists, but deletion is much faster, providing an overall time advantage over the lists.

Finally, the time trials provide a very clear diagnostic as to which data structures to use in which situations. The stack array, stack list and queue array are all fast at what they do, and none of them are really faster than the other. However, there's no possible way to do select deletions, so it's not a great data structure for all situations. Much more interesting is the case of the linked list, cursor list and skip list. If we get lucky, and we know that the data is in a specific organized way, we know which data structures are better. If we are only using inserts, like File1.dat, or we know that the data will be removed in the opposite order of how it will be inserted, like File 3.dat, both the linked list and the cursor list run faster than the skip list. The cursor list will be slower than the linked list, since it needs to move its cursor for both inserts and deletions, but they're comparable. However, if we can't predict how our data will work, the skip list is a better solution. For situations that are very difficult for the lists, like File2.dat, and random situations that are uncontrollable, like File 4.dat, the skip list works much, much faster than either of the other lists. Also, for the situations in which the cursor and linked lists are more ideal, the skip list is not that much slower. The skip list is 8 to 14 times slower than the other lists in certain situations (like File1.dat and File3.dat), but they can be between 28 and 105 times slower than the skip list (in File2.dat), so, overall, the skip list is a better solution for unpredictable data.