COMP20003 - Algorithms and Data Structures

Lucas Berezy (588 236)

Data Structure Analysis and Results

The singly linked list and skip list data structures were compared in practice to determine their qualities for insertion and search in their role in implementing a simple dictionary.
To compare these two data structures, the number of key comparisons during respective insertion operations and individual key lookups were counted.
Both data structures were kept in sorted order during insertion.

For the linked list based dictionary to insert into a sorted list requires a linear search through the nodes [O(n)] and a single insertion operation of [O(1)] (handling insertion at head), in terms of key comparisons. Here, the O(n) term dominates and the experimental results should reflect this. Search is also a linear operation in terms of key comparisons, on average requiring half the list to be traversed before finding the correct node, thus O(n) behaviour is expect for searching. As there are n items to insert, this creates an O(n^2) operation (actually $\frac{n(n-1)}{2}$) operations growing a sorted list from sorted input data and O(n) when sorted in reverse.

For the skip list based dictionary, insertion happens in a sorted manner by virtue of the data structure. The list contains a maximum number of levels possible for each node, and the probability of each node containing m levels using a value of p = 0.5 is $P_{(m)} = \frac{1}{(m+1)p}$. Thus 50% ($\frac{1}{p}$) of all nodes will have a maximum level of 0 (similar to regular linked list), 25% ($\frac{1}{2p}$) have 1 level and so on. A level cap of 6 was used for testing, thus ~28.5% of nodes had the maximum level. The insertion process shares much in common with the search process, requiring a search to find the correct position to insert the new node (stopping just before a matching node). As the data structure is traversed from the highest, most sparse level first, the probability of finding a matching node at a level is inversely proportional to the number of key comparisons required. Each level traversed towards the fully connected linked list at the bottom layer improved the probability of finding the desired node by $\frac{1}{p}$, the number of times this can occur is $log_{p^{-1}}(n)$. Thus the expected number of key comparisons is $\frac{1}{p}log_{p^{-1}}(n)$, which is O(logn). There exists an extra key comparison when checking if node found is null or greater than the search value, but this is absorbed by big-O notation. These expected key costs are only probabilistic owing to how a node is given a random level between 0 and max_level following the above distribution, but on average the insertion (and search) should still be logarithmic with n.

Files containing 10, 20, 50, 100, 200, 500 and 1000 key-value pairs were created for the following testing with keys between 10 and 100000. The test data is available below.
Sorted data output using unix ‘sort’ command.
e.g. $~ cat test_10.txt | sort -nr > test_10_sorted_reversed.txt

Insertion.

Insertion testing for n items, random ordering.

Random ordering.	Key comparisons:	
Data size (n)	Linked List	Skip List
10	24	13
20	111	35
50	696	183
100	2860	427
200	10363	1071
500	62313	3475
1000	257443	8874

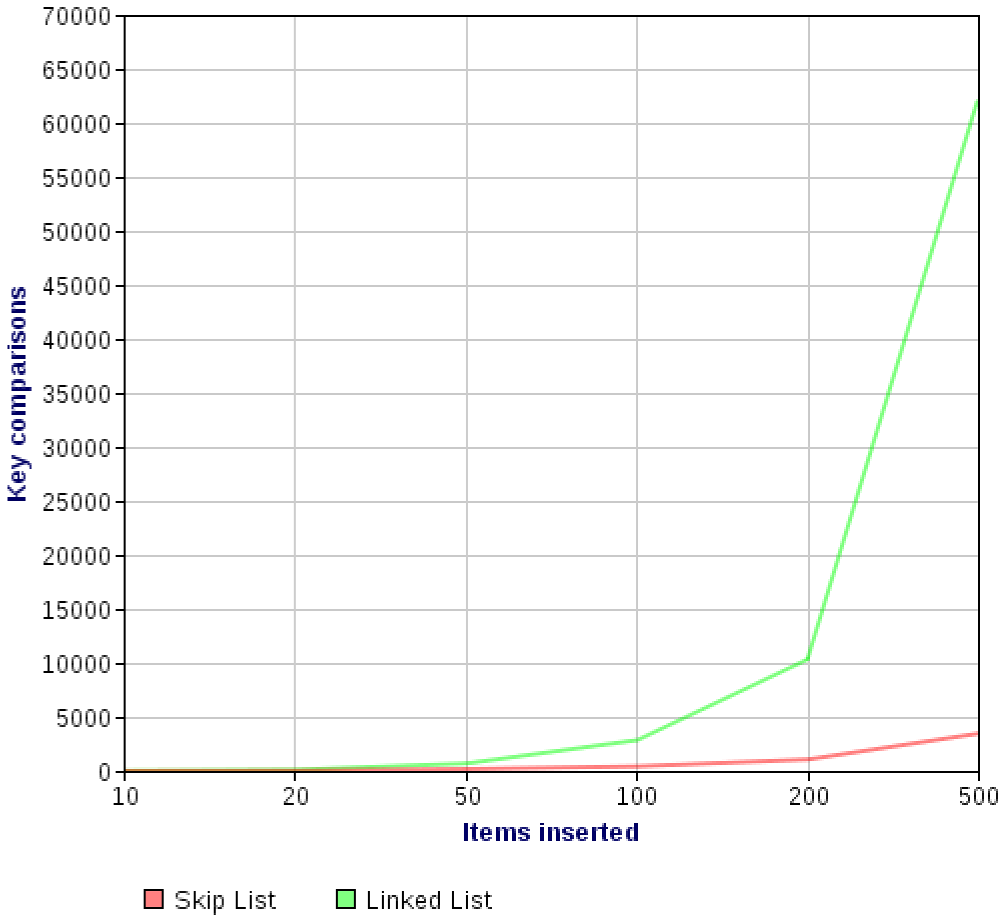

Figure 1

The skip list's O(logn) cost and the linked list O(n^2) cost show in the above graph.

Insertion testing for n items, reverse sorted.

Reverse ordering.	Key comparisons:	
Data size (n)	Linked List	Skip List
10	9	10
20	19	20
50	49	50
100	99	100
200	199	200
500	499	500
1000	999	1005

From the above table, it can be seen that the skip lists' performance behaves like a linked list. Each new node is being inserted at the head of the list for both data structures, requiring only one key comparison each time before finishing for n items, this O(n). It is worth noting that the skip list will have other overheads, namely the re-distribution of random list pointers.

Notes: Some small effort was made during the implementation of the skip dictionary to handle the case of a duplicate key being inserted. In this case, the value of the key is simply updated.

Search.

Search testing.
The search performance was tested using the unsorted on each data structure.
The “key not found” responses for each dictionary required n+1 operations to exhaustively search the entire list for the linked list and, on average, 2 to 3 comparisons to determine the existence of a key in a skip list.

Each dictionary was tested with the key at the head of the sorted data file, the largest key.

Search testing in n items, random order insertion, maximum key.

Reverse ordering.	Key comparisons:	
Data size (n)	Linked List	Skip List
10	10	7
20	20	4
50	50	8
100	100	11
200	200	11
500	500	19
1000	1000	23

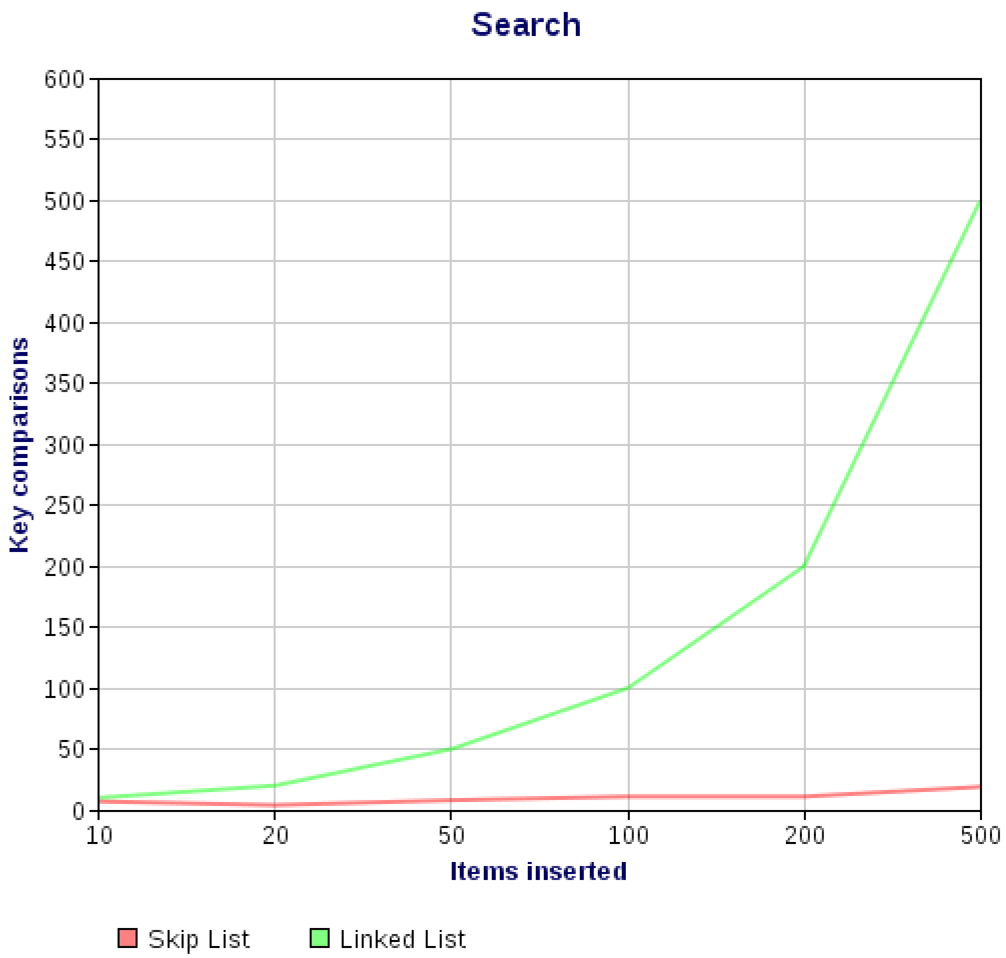

Figure 2

As shown by the graph, the data structures stay true to their asymptotic costs as n grows, as discussed in the introduction.

The randomness of the probabilistic logarithmic growth can start to be seen in the above figure and was noted during testing and (much) debugging of the programs.