Programming Assignment 5 Report
CSCE 221

Oneal Abdulrahim

1. In this assignment, we were tasked with implementing a Skip List. This is really the first true random data structure we have covered. In order to run my program, a makefile has been included. However, I have precompiled and left the object code ready to run with the command `./SkipList`.

   The standard and required functions of insertion, deletion, and search are included as well. The insert and delete functions make use of the search algorithm in order to get to the correct node. The search was used to find not only where a node belonged (for insertion) but also to assist in finding what nodes needed pointer reallocation during deletion.

2. The skip list is thought of as layers of linked lists or ordered nodes. The data structure may be implemented probabilistically or deterministically, and in this case we used the pseudo randomness of a coinflip in C++ to integrate the "skips". Each layer acts as an express lane which allows for quick traversal and fast search sequences. In general, the code includes a node structure, which holds an integer for data and a vector of pointers to its next nodes. The vector's indices represent the current level, and each hold a pointer to the corresponding next node on that level. These nodes were then represented holistically in a SkipList struct. This had properties including the pointer to the first node and one to the last. This assisted in the traversal of this.

   Then, levels of the skip list are traversed from size - 1 of the vector of the first to the last node in the first index of the last node. Essentially, we traversed top to bottom, left to right.

3. We expect the cost of insertion to be, on average & with high probability, $O(logn)$ since the cost of the "drop downs" is dependent on the height. In the worst case, we roll no extra heads and end up with one-level skip list, which behaves in the same manner of a linked list. Then the search cost would be $O(n)$
   The average cost of searching is the same as inserting with high probability $O(logn)$, because the same search operations must be done in order to insert. Again, the worst case is linear running time, for when we have only one level with all elements.
   The average cost of deletion, in fact, is expected to be the same: $O(logn)$ simply because in order to find the item to delete, a search must be done to find its location. The worst case for deletion is again, when we have only one level and must iterate on average half of n to get to the desired element, so $O(n)$.

4. Run your program several times with the same set of data. Is the number of comparisons for the search operation the same each time?
   No, they are not. Though they do not fluctuate greatly in magnitude, the values change each time due to the randomness (pseudo randomness) of the coin flips. The largest level for any insertion I achieved was actually eleven!

5. How likely is it that an item will be inserted into the nth level of the skip list?
   Since there is a 50/50 chance of inserting on each level, meaning you must keep flipping heads, the probability that an item gets inserted at a level n is $P(n) = (\frac{1}{2})^n$

   How does height affect the number of comparisons for the skip list operations?
   In general, since the search complexities are logarithmic, if the amount of levels (or height) is a power of two, similar to binary trees, as the height increases by 1, the search cost of any particular node also increases by 1. This is different from linear data structures where a single increase in node count causes a single increase in search complexity.

Does the order of the data (sorted, reverse sorted, random) affect the number of comparisons? How?
The order of the data does not affect insertion greatly. Since insertion happens by chance, and is in order by level, there is no difference the order of the data being inserted. Thus, we can say that the number of comparisons is affected most by the probabilistic qualities of insertion. Which, in general, can be said will create a logarithmic running time with high probability.

How does the runtime compare to a Binary Search Tree for the insert, search, and delete operations?
The runtime of skip lists is the same as a binary search tree for all three operations. The difference is that there is small probability that skip list ends up being worse than BST.

In what cases might a Binary Search Tree be more efficient than a skiplist? In what cases might it be less efficient?
Binary Search Tree performance and running time is guaranteed even in the worst cases, which allows them to be more reliable in general. However, BST often needs to rebalance and stay in-tact with its structure, which can be costly. On the other hand, Skip Lists scale incredibly and offer (with high probability) extremely fast access.