Programming Lab 1

Linear Search

Searching the `LinkedList` is a linear search. It must iterate across the list sequentially until the correct item is found. The list is sorted by data values, but searched by keys; I generated the keys as the numbers were inserted into the list in the order they were generated. This leaves the search times as random as the generated numbers.

As written in my code, each search has 1 setup operation, 2 operations to get and compare the current node to the supplied key, 2(N-1) operations to get and assign the next node as it iterates, and 1 return operation at the end. totaling to $2(n-1)+2n+2$ operations per search (these numbers gloss over the true number of operations, but are sufficient for estimation). In the best possible case, this is 4 operations when the sought item is first in the list. Worst case, it is as written and approaching infinity. On average, it will take about $\frac{n}{2}$ iterations.

Any given item will be found in the range of:

- $O(2(n-1)+2n+2) \sim O(4n) \sim O(n)$
- $\Omega(1)$
- $\Theta(\frac{2(n-1)+2n}{2}+2) \sim \Theta(2n+1) \sim \Theta(n)$

Binary Search

Because arrays are random-access, that is that any element can be accessed independently of the others, searching the sorted array can be accomplished with a binary search. Because the binary search is a divide and conquer algorithm, its worst case time complexity is far better than that of Linear Search.

Pulling from my code again, the binary search pattern I wrote requires 6 operations to start, 4 every time it generates a new guess, and 3 to reset the limits after each incorrect guess. Each guess eliminates $\frac{n}{\frac{\#\text{ guesses}}{2}}$, so the final result is $4log_2(n)+3log_2(n-1)+4$. The worst case is as written, and will be $\lceil log_2(n) \rceil$ iterations; the best case is one iteration if the number sought is dead center of the array, and the average will be half that.

- $O(4log_2(n)+3log_2(n-1)+4) \sim O(log_2 n)$
- $\Omega(1)$
- $\Theta(\frac{4log_2(n)+3log_2(n-1)+4}{2}) \sim \Theta(log_2 n)$

Comparison

Putting the differences in plain English, each iteration of the Linear Search eliminates 1 possible answer; each iteration of Binary Search eliminates $\frac{n}{2}$ possible answers. Using 100 items as an arbitrary example, in an *average* case a Linear Search will take $\frac{n}{2}=50$ iterations; in the *worst* case, a Binary Search will take $log_2(100) \approx 7$. Both have the possibility of finding the answer on the first round, but over a larger sample set the time taken by a Linear Search will grow dramatically faster than a Binary Search.