Christian Henning
CS 3310 – 100
04 Dec 20203

Assignment 4

Searching in Arrays & Linked Lists

Problem Specification

The project involves the implementation and analysis of various sorting and searching algorithms within the context of a linked list data structure. The primary objective is to evaluate the performance of these algorithms both theoretically and empirically.

Algorithm Descriptions

1. Insertion Sort: Customized for sorting objects in a linked list based on specific attributes. It inserts each element into its correct position in a sorted sublist.
2. Quick Sort: Implemented for the linked list, utilizing a best-of-three pivot selection strategy.
3. Linear Search: Searches each element sequentially to find the target value.
4. Binary Search: Conducted on both a linked list and a primitive Python list. The linked list version is for empirical comparison, while the list version is to observe standard binary search performance.

Theoretical Complexities

1. Insertion Sort: Worst case time complexity of $O(n^2)$ and best case of $\Omega(n)$
2. Quick Sort: Worst case time complexity of $O(n^2)$ and best case of $\Omega(n \log n)$
3. Linear Search: Average time complexity $\Theta(n)$
4. Binary Search: Average time complexity $\Theta(\log n)$

Observed Time Complexities

The execution times were measured in nanoseconds (ns) for various operations during runtime. The following table summarizes the observed times across 10 runs:

Algorithm	Average Time (ns)	Best Time (ns)	Worst Time (ns)
Insertion Sort	25,554,084,230	24,903,073,300	26,456,626,200
Quick Sort	473,884,150	457,242,000	515,187,000
Linear Search	450,568	272,030	688,440
Binary Search	419,281	337,880	561,540

Observations and Conclusion

The empirical observations likely deviated from theoretical complexities, particularly for binary search due to the face that the algorithm is not well suited for linked lists. Such a deviation highlights the importance of selecting appropriate data structures for specific algorithms and the impact of practical factors like data structure overhead and hardware limitations on algorithm performance.

More generally though, we can observe that insertion sort is a relatively slow algorithm in our implementation. Similarly, in our implementation linear search outperforms binary search.

Binary search would normally perform at *O*(log n) but when applied to a linked list it still must traverse the list to reach each element resulting in an observed *O*(n log n). When applied to a primitive python list we do see the expected time improvements ad the program is dramatically faster, averaging an execution time across 10 runs of 7,422 ns and breakeven point at 1195 searches.