ENSF 593 Lab 10

Exercise A

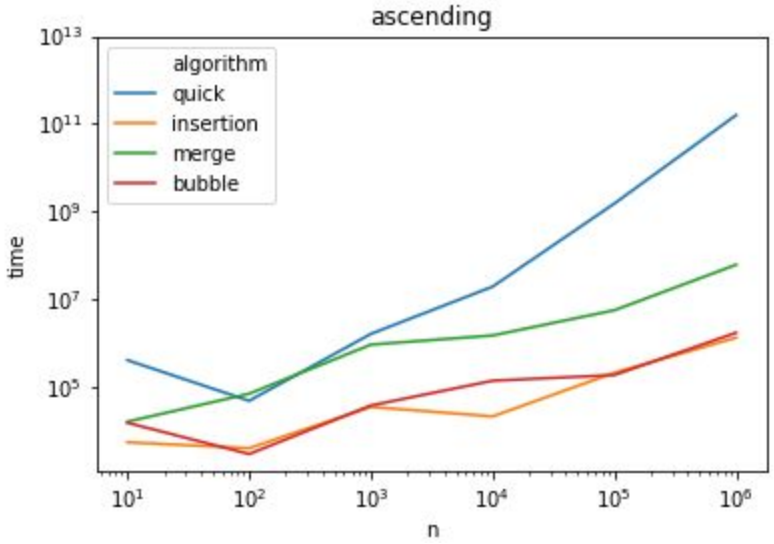
Experimental Method

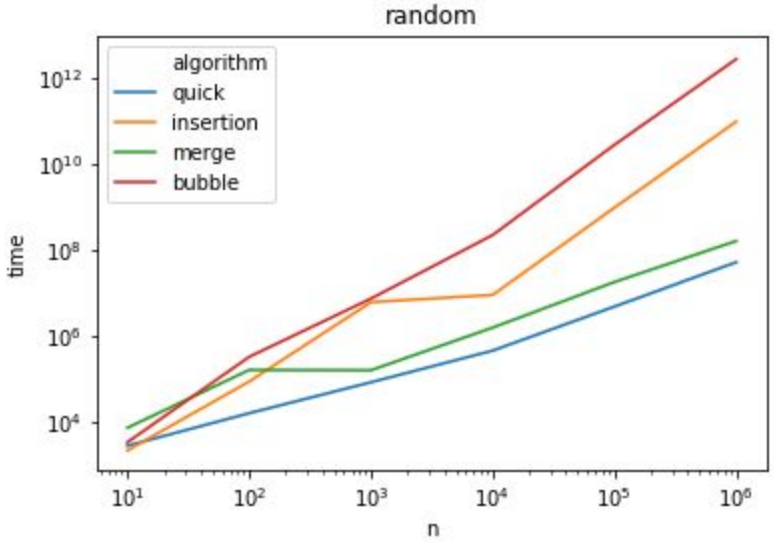
A test file was created which generates integer arrays in either ascending, descending, or random order, to a specified length n. These arrays were sorted using bubble, insertion, merge, and quick sort. The resulting process was timed and the time taken was logged.

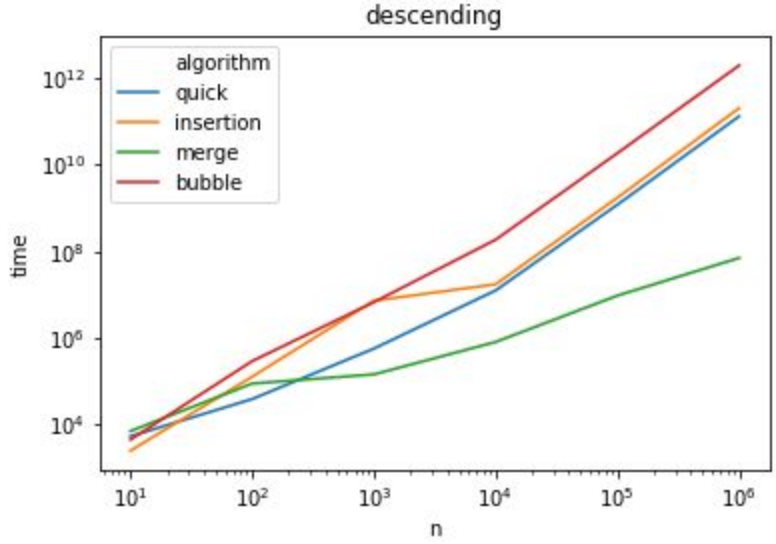
Data Collected

Timing was done to record the run times of merge, quick, insertion, and bubble sorts on ascending, descending, and random data sets with n at 10, 100, 1000, 10 000, 100 000, 1 000 000.

It was necessary to increase the stack size to allow the program to complete for arrays greater than 1000.


ascending
algorithm
quick
insertion
merge
bubble
time
n


random
algorithm
quick
insertion
merge
bubble
time
n


descending
algorithm
quick
insertion
merge
bubble
time
n

Discussion

For a random data set, quick sort was found to be significantly better than all other methods, which is as expected. However, for ascending and descending implementations of quicksort, the algorithm fares very badly. In some reading of literature, the reason for this is that I used the last element as a pivot, rather than the middle element or a random element. In future implementations of this I will keep this in mind as this is a more critical algorithm design consideration than I would have expected.

Comparing the other sort methods, bubble sort is the worst performer for unsorted lists. One surprise in this is that even in the case of an ascending sort, it is neck and neck with insertion sort, and on average loses out across the trials. It might have been expected that this method would have fared best for recognizing an ordered sort but this is not the case.

Merge sort was a reliable performer, I would expect it to come second to a middle pivot quick sort, but that will be a future exercise to verify.

In all cases, the data would indicate that all algorithms tested are approximately O(n^2), given that they are all linear on a log-log plot. More data points would be needed to verify this.

Conclusions

- A quick sort can perform much worse than expected if taking the pivot as the last element.
- Bubble sort was the worst option and should only be used for cases of very small data sets, in which case the ease of implementation might justify its use.
- Attention should be paid to the expected order of the data. If data will typically be nearly sorted, an insertion sort will perform best, and bubble sort will also be quite good. For other sorts, merge and quick sort are probably better.

Exercise B

Anagram Checking complexity

I used an XOR operator to check whether two words are anagrams of each other. The strings were converted to char arrays, and an XOR operation was applied at each position. Given that x^x = 0, x^0=x.. Only in the case of an anagram will the result of this test be zero.

For testing anagrams of length k:

- There would be 2 assignment operations to create 2 arrays of characters from the strings… this would be either O(k) or O(1) depending on how the compiler functions
- O(K) operations to perform XOR tests on all elements
- Approximately 4 + k*4 operations to perform this test.

This check will be O(n) in all cases where the strings are equal length, and O(1) in the case that the strings are not equal length.

Total Program Complexity

I wasn't successful in making my program run, so total complexity is undetermined.

- Anagram check for word length k -> O(k)
- Sorting list of length L with quick sort average case O(L log(L))

Summation of these suggests… O(k + L * log(L)).

References

- https://www.geeksforgeeks.org/insertion-sort/
- https://www.geeksforgeeks.org/quick-sort/
- https://www.geeksforgeeks.org/merge-sort/
- https://www.bigocheatsheet.com/