Christopher Barclay
48254426 - crbarcla
CS 165
4/29/19

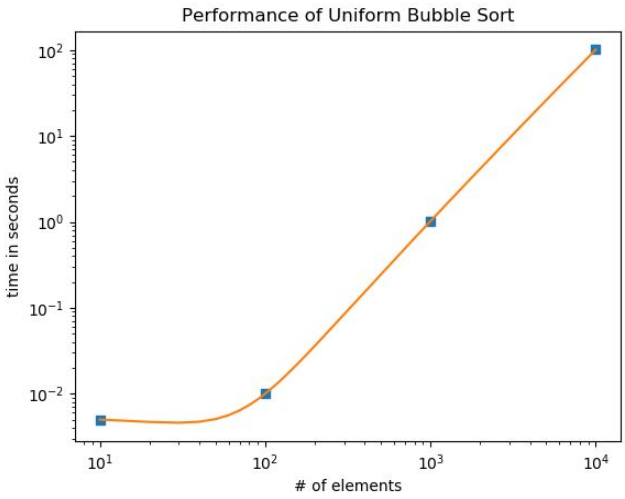
Project 1 Sorting Algorithms

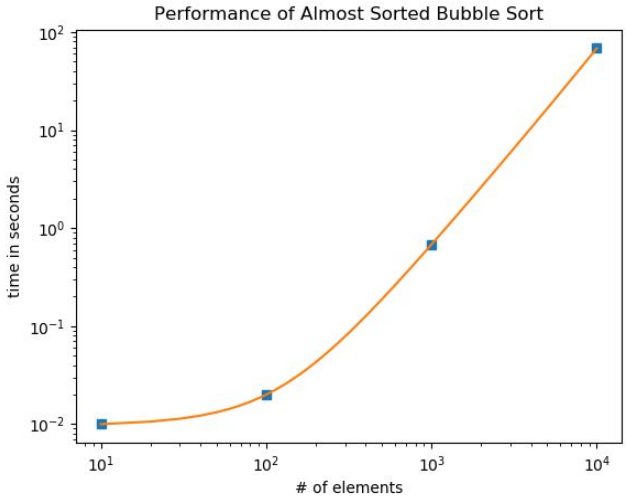
Introduction

This project was focused on implementing sorting algorithms and analyzing their runtimes through timings and graphs representing those timings. The algorithms we implemented and tested were bubble sort, insertion sort, spin-the-bottle sort, shell sort, and annealing sort. All of these were tested on two different distributions: uniform random and almost sorted. The elements were strictly integers held in various sized vectors. Each algorithm was tested on vectors sized 10, 100, 1000, and 10000. To get accurate results the timings were average over multiple repetitions. The language used for this project was C++, specifically the 2011 standard.

Bubble Sort

Bubble sort is a worst case $O(n^2)$ runtime algorithm. It operates by comparing consecutive elements in the list and swapping them if out of order. It makes multiple passes until the list is sorted. Comparing the two distributions, bubble sort performed better on almost sorted as expected.

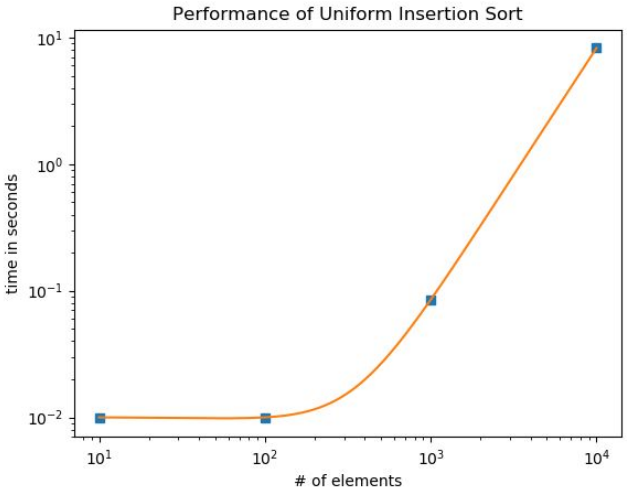


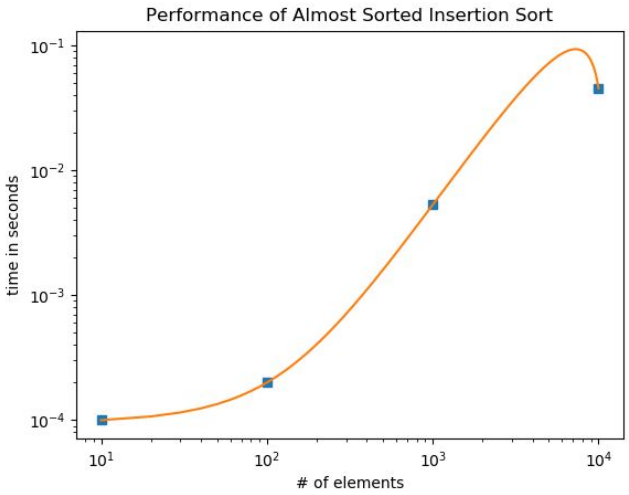


n	Uniform	Almost Sorted
10	0.005	0
100	0.01	0
1000	1.02	0.68
10000	101.14	68.48

Insertion Sort

Insertion sort is a worst case $O(n^2)$ runtime algorithm. It operates by iterating through the list and comparing elements to the previous element. If an element is out of place it backtracks through the list until it finds the correct index to place it. Comparing the two distributions, insertion sort performed better on almost sorted permutations as expected.

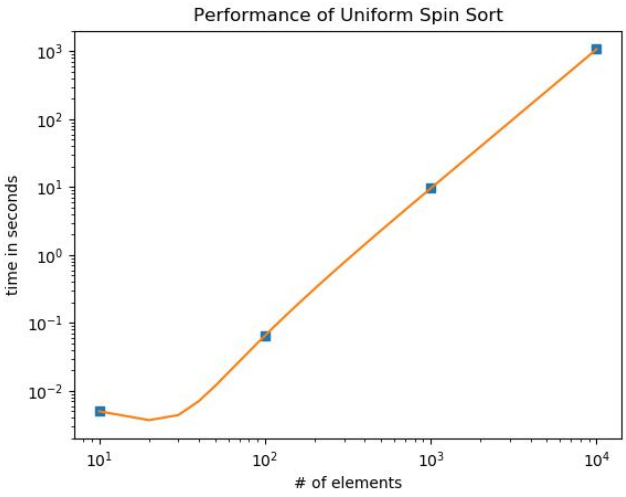


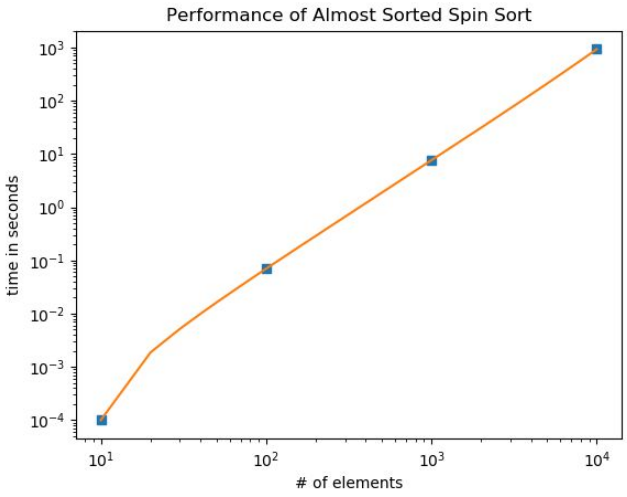


n	Uniform	Almost Sorted
10	0.0	0
100	0.0	0
1000	0.085	0.005
10000	8.275	0.045

Spin-The-Bottle Sort

Spin-the-bottle sort is a worst case $O(n^2\log(n))$ with high probability runtime algorithm that sort. It operates by iterating through the list and choosing a uniform and independently random index to compare to and swaps if necessary. This continues until the list is sorted. Comparing the two permutation types, spin-the-bottle sort performed better on almost sorted permutations.

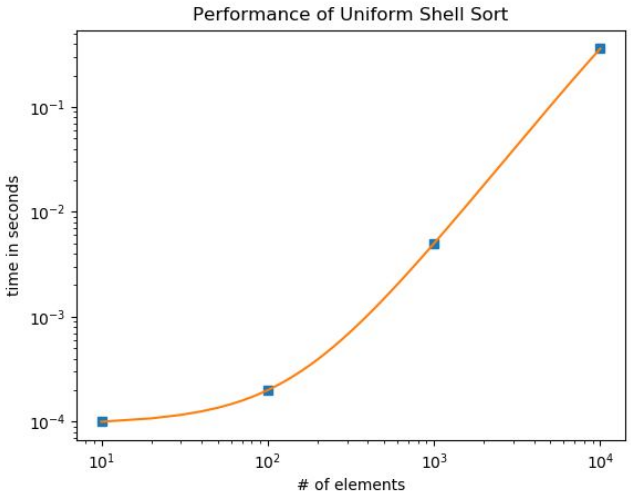


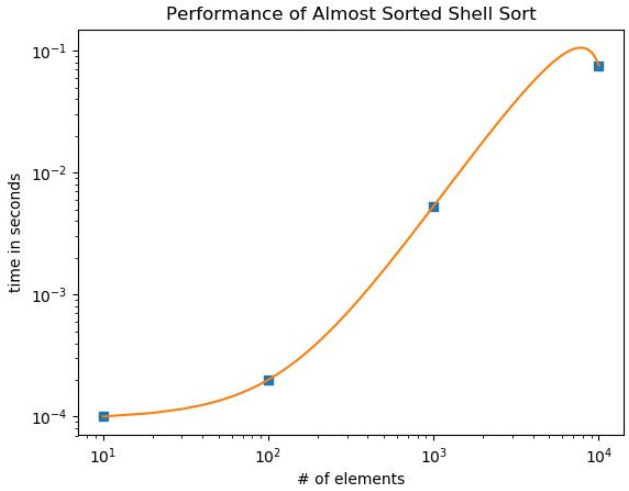


n	Uniform	Almost Sorted
10	0.005	0
100	0.065	0.07
1000	9.615	7.605
10000	1080.16	931.92

Shell Sort

Shell sort has a runtime based on a gap sequence used. It operates by making multiple passes with each pass comparing elements at a certain gap and swapping if necessary. This continues through the gap sequence until the last gap which is just an iteration through the list. With a basic gap sequence a worst case $O(n^2)$ runtime is achieved. The results below are based on Ciura 2001 gap sequence which was derived empirically. Comparing the two permutation types, shell sort performed better on almost sorted as expected.

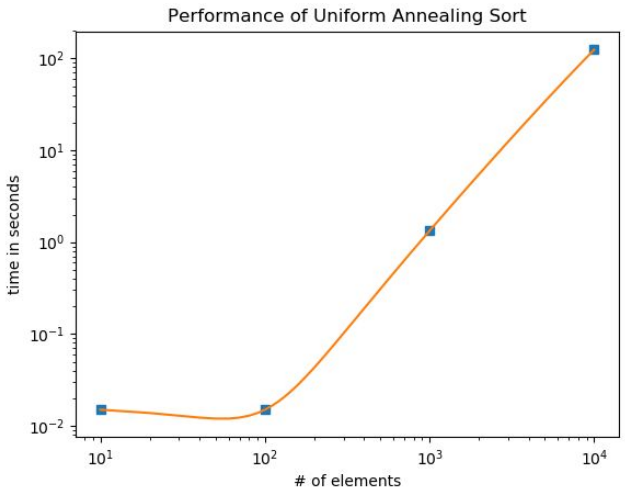


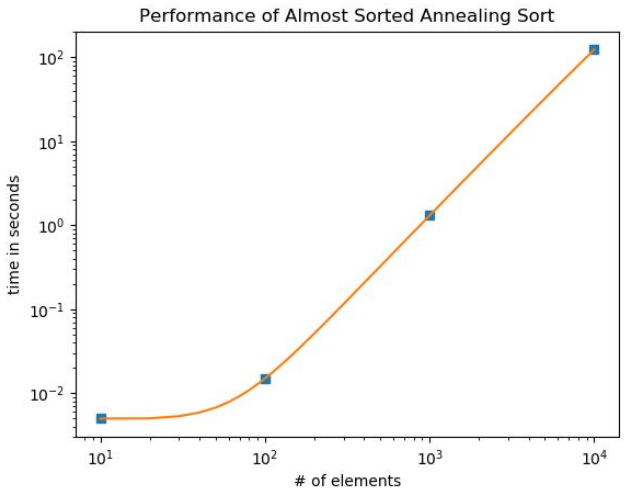


n	Uniform	Almost Sorted
10	0.0	0
100	0.0	0.00
1000	0.005	0.005
10000	0.36	0.075

Annealing Sort

Annealing sort has a runtime of O(nlog(n)) with high probability depending on the temperature sequence and repetition sequence. It operates by using the principles of simulated annealing to sort a list. It makes choices from neighbor states with choice from the repetition sequence and the states are confined by the temperature sequence. The temperature decreases until 0 which is called the annealing schedule. It is similar to spin-the-bottle sort but each pass is limited by the temperature and repetition sequences. Comparing the two permutation types, annealing sort performed slightly better on almost sorted.





n	Uniform	Almost Sorted
10	0.015	0.005
100	0.015	0.015
1000	1.33	1.31
10000	124.465	123.785

Problems Encountered

I had trouble getting reliable timings on sizes of n 10000 or higher. The process took a lot longer to run than expected and would sometimes freeze to an unresponsive state. The results I have for 10000 sized sorts seem to be higher than expected.

Conclusion

Comparing the results from the algorithms. Shell sort had the fastest run times followed by insertion sort. Shell sort can have faster run times compared to insertion and bubble because of its room for optimization due to the many possible gap sequences. I would expect annealing sort to be the fastest, but I assume it was due to bad temperature sequences. Annealing sort can be optimized for different size lists with empirically derived temperature and repetition sequences. Spin-the-bottle sort had the slowest run times as expected due to the amount of randomness in its sorting.