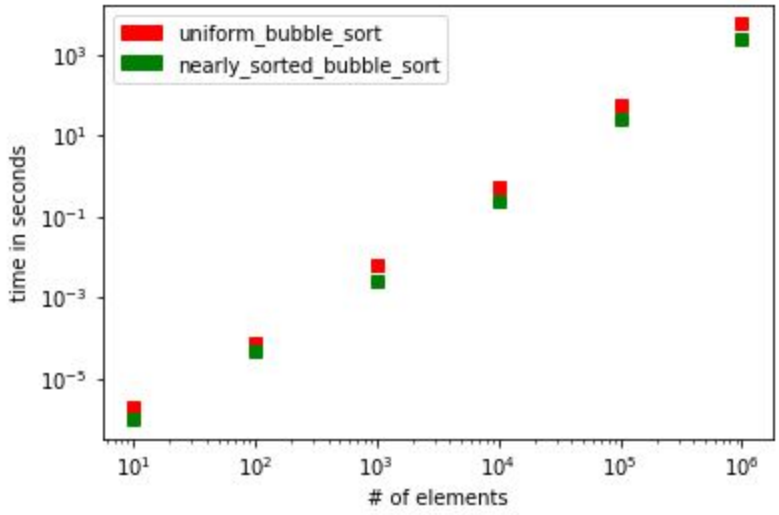
Won Joon Lee
15647075
CS165
Due 4/29/19

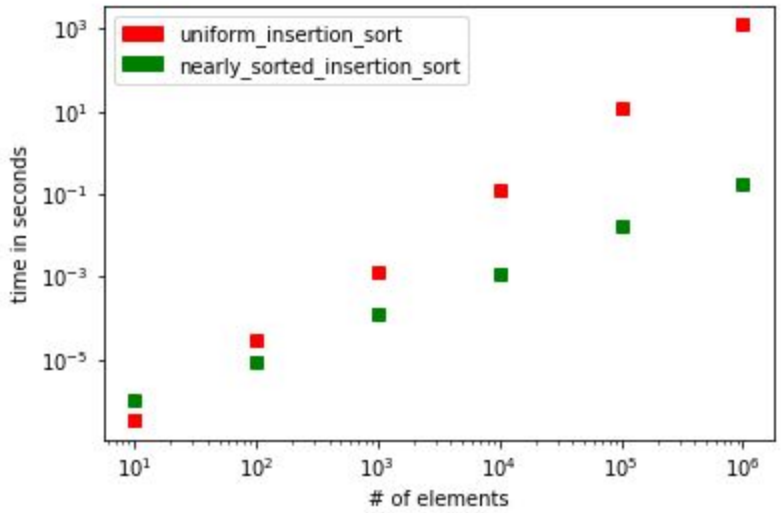
Project 1 Report

Bubble Sort:



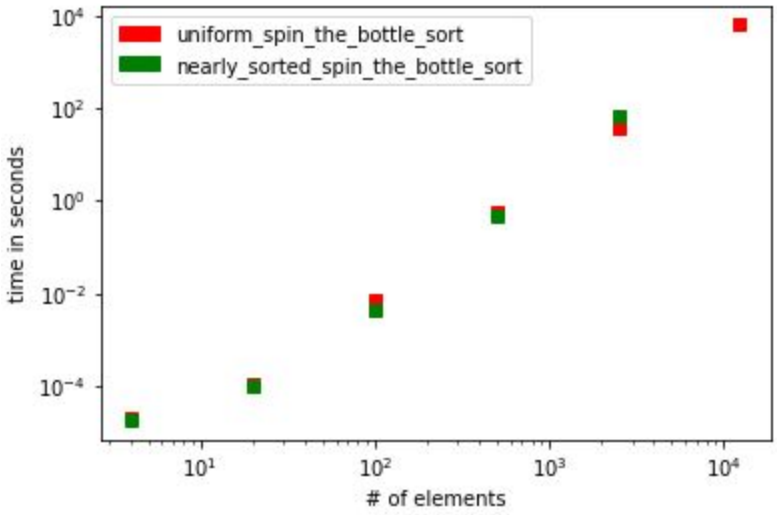
Bubble Sort showed an increase in time taken at approximately 100-fold as n increased tenfold, (at n = 100,000 taking around 58.35 seconds for uniform permutation and at n = 1,000,000 taking around 5,837 seconds). This showed that bubble sort is an **O(n^2)** algorithm. The ratio between uniform and nearly sorted permutations was ~2.32:1.

Insertion Sort:



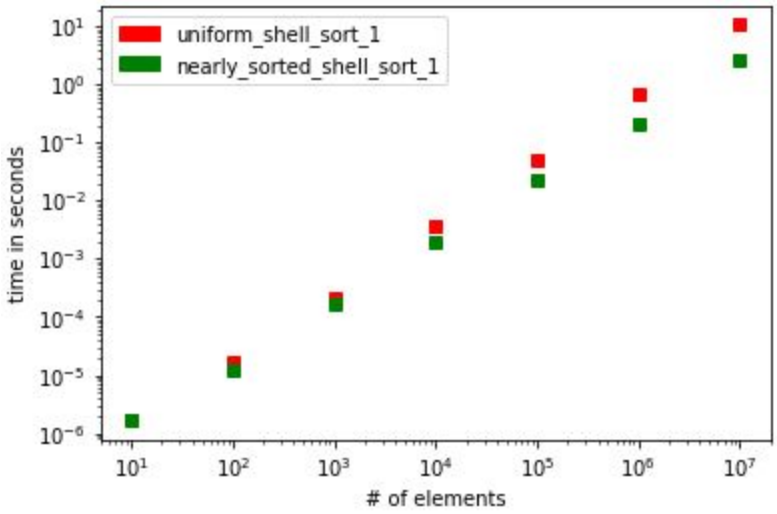
Insertion Sort, like Bubble Sort, also showed an **O(n^2)** behavior for uniformly distributed permutation (at n = 100,000 taking around 12.31 seconds and at n = 1,000,000 taking around 1,232.7 seconds - as n increased by tenfold, time increased by 100-fold). However, for nearly sorted permutation, because of the nature of Insertion Sort, it shows an O(n) behavior of time linearly increasing with n (at n = 100,000 taking around 0.0167 seconds for uniform permutation and at n = 1,000,000 taking around 0.177 seconds - time scaled linearly with n).

Spin-the-bottle Sort:



Spin-the-bottle Sort was definitely the worst sorting algorithm in terms of complexity, ranging between O(N^2) and O(n^2*lg n) (mostly **O(n^2*lg n)**). It took so long that I had to decrease the increments by 5 rather than 10 as well as stop at much lower number. (at n = 2,500 taking around 36.71 seconds and at n = 12,500 taking around 6,281.2 seconds for uniformly distributed permutation). Because of the nature of the algorithm, which involved swapping random indexes around, it performed better for nearly sorted permutations for smaller n and did slightly worse as n got bigger for the nearly sorted permutation, hence the flip in position between the green dot (nearly_sorted) and the red dot (uniform) in the graph at n = 2,500.

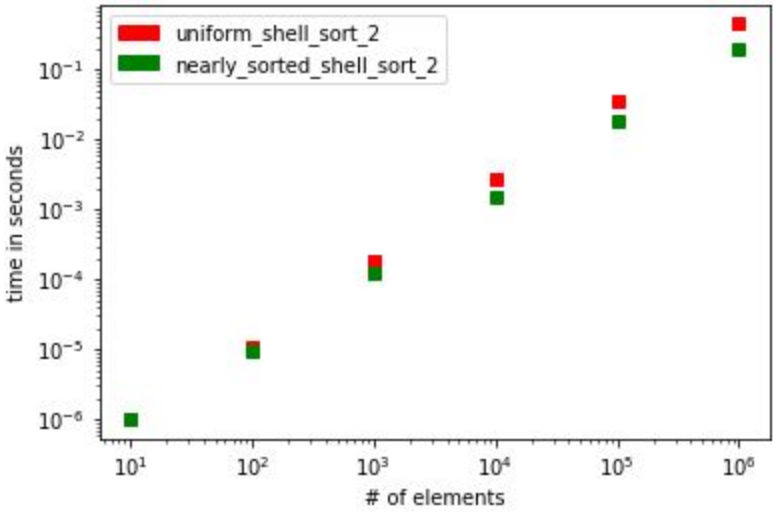
Shell Sort (Ver.1):



This version of Shell Sort used the gaps from the slides, starting the gap at n / 2 and dividing by 2 each time. I feel like this was a fairly effective gap sequence, and it mostly showed an O(n lg n) behavior for uniformly distributed permutations (at n = 1,000,000 taking around 0.71

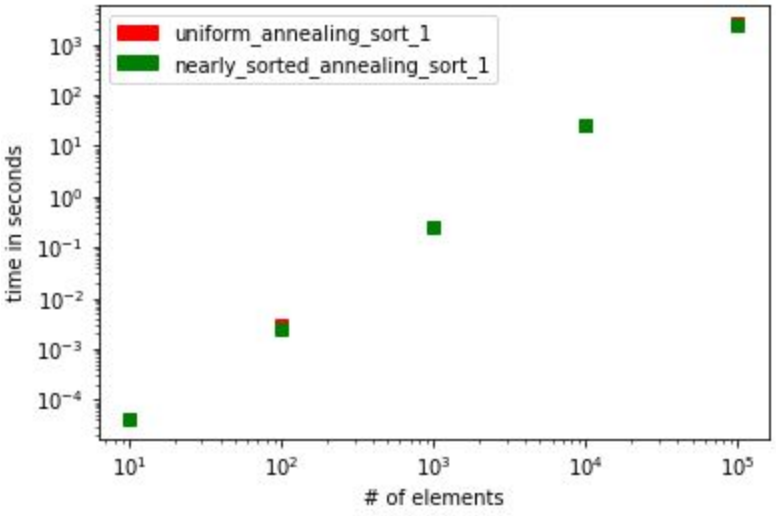
seconds and at n = 10,000,000 taking around 10.72 seconds - as n increased by tenfold, time increased by approximately a ratio of 10*lg 10). Because Shell Sort uses Insertion Sort to sort the gaps, it did significantly better for nearly sorted permutations, showing an O(n) behavior (at n = 1,000,000 taking around .212 seconds and at n = 10,000,000 taking around 2.62 seconds - time scaled close to linearly with n). In general, however, the algorithm will show an **O(n lg n)** behavior.

Shell Sort (Ver.2):



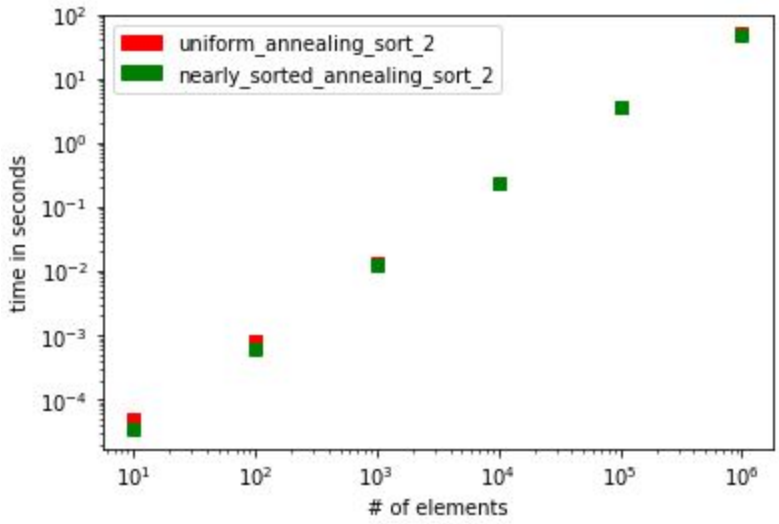
This version of Shell Sort used the popular gap sequence (A102549) created by Gunther Piez that I found on Wikipedia; the gap sequence goes 1, 4, 10, 23, 57, 132, 301, 701, … After 701, the next gap is floor(2.25 * prev gap). This gap sequence was not much of an improvement over the first Shell Sort for nearly sorted permutations, but it did perform slightly better for uniformly distributed permutations. The time complexity still lies in **O(n lg n)**.

Annealing Sort (Ver.1):



This version of Annealing Sort used a Temp and Rep sequence of (n/2, n/4, … 1, 0). However, I realized that, as a result of the first few elements of Rep being so large, this version performed very poorly for both uniformly distributed and nearly sorted permutations. This Annealing Sort showed an **O(n^2)** behavior across the board (at n = 10,000 taking around 24.88 seconds and at n = 100,000 taking around 2532.5 seconds for uniformly distributed permutation - as n increased by tenfold, time increased by 100-fold).

Annealing Sort (Ver.2):

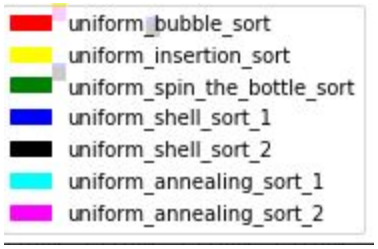


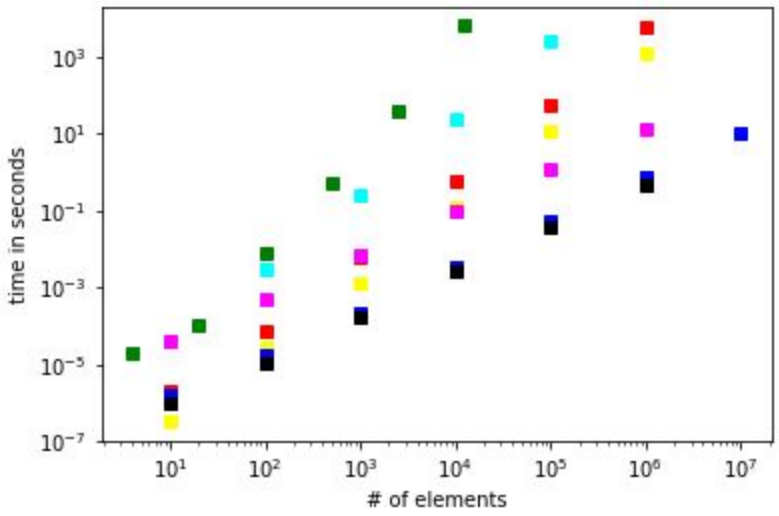
This version of Annealing Sort used the same Temp as the other Annealing Sort but a constant (lg n) as every element in Rep. This significantly reduced the time spent down to **O(n lg n)** (at n = 100,000 taking around 3.47 seconds and at n = 1,000,000 taking around 48.88

seconds for uniformly distributed permutation - as n increased by tenfold, time increased by 10*lg 10).

Compilation:

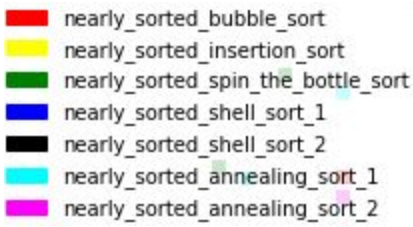
Uniform:

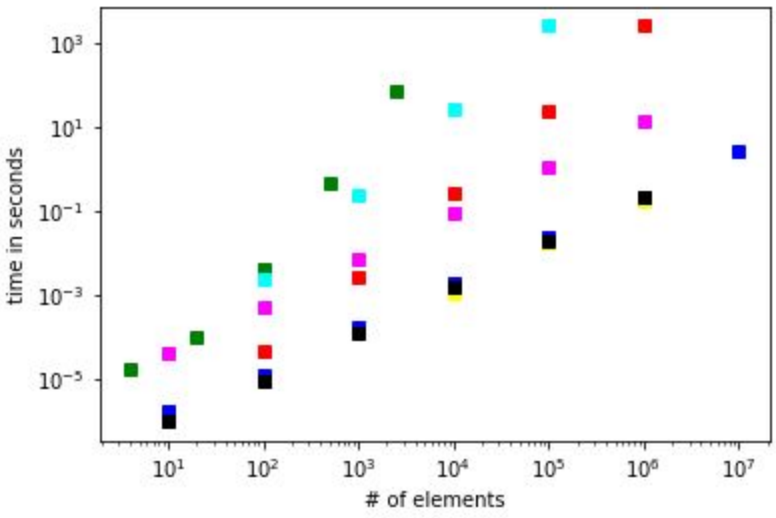




The clear loser in this graph is Spin-the-bottle Sort, which increases extremely fast compared to the better sorting algorithms, such as Shell Sort. The O(n^2) algorithms, Bubble Sort, Insertion Sort, and Annealing Sort (Ver.1) follow a similar increase in time (all with a slight curve at 10^6), with Insertion Sort performing the best. The best algorithm for uniformly distributed permutations would be both versions of Shell Sort (in blue and in **black**). Annealing Sort (Ver.2) is also O(n lg n), but it seems to have a bit of a higher coefficient, making it not the best. It's interesting to note that at very low n, Insertion Sort performs better than any other sorting algorithms, which explains why other sorting algorithms, namely Quick Sort, uses Insertion Sort at very low n.

Nearly Sorted:





For nearly sorted permutations, most of the sorting algorithms show the same pattern as the uniformly distributed permutations except for Insertion Sort, which shows a great increase in performance, at the same level as the two Shell sorts.

Conclusion:

All in all, in terms of consistent performance, regardless of the type of permutation, Shell Sort seems to be the best sorting algorithm. As mentioned, Annealing Sort, despite being quite good, seems to have too high of a coefficient value compared to Shell Sort, while Insertion Sort is too situational.