Experimentally Deriving the Running Time of Various Sorting Algorithms

Johnny Ceja
COMPSCI 165
University of California, Irvine

Prologue

Introduction

Here we compare the experimental running time of various sorting algorithms. The sorting algorithms used in this experiment are: Annealing Sort with two different temperature sequences, Bubble Sort, Insertion Sort, Shell Sort with two different gap sequences, and Spin the Bottle Sort. The running time is deduced via sorting a shuffled array of integers and sorting a partially-sorted array of integers, both of such arrays contain no duplicate values.

Test Machine and Environment

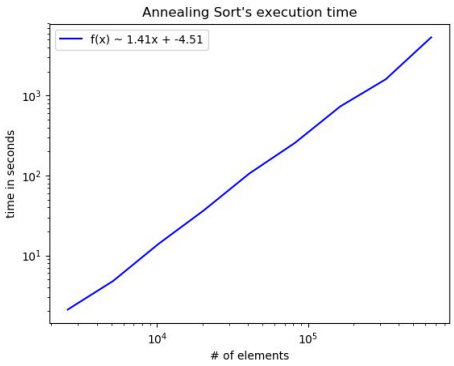
The algorithms have been executed on a shared server owned by the University of California, Irvine. The specific machine used is named "Odin" and has a four 16-core AMD Opteron 6378 CPUs @ 2.93 GHZ, for a total of 64-cores on the server. It has 512 GB of RAM and is running on a 64-bit CentOS Linux distribution. Each algorithm execution (ie: running Bubble Sort) was ran on a dedicated core from the server with no other applications running. Fully shuffling the array consisted of running the Fisher-Yates algorithm to completion, and partially shuffling the array consisted of inverting $2log_2(N)$ pairs in the sorted array. Every algorithm was sorted three times and averaged for each discrete array size.

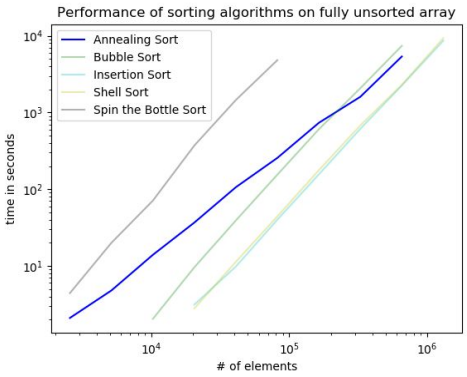
Results

Annealing Sort

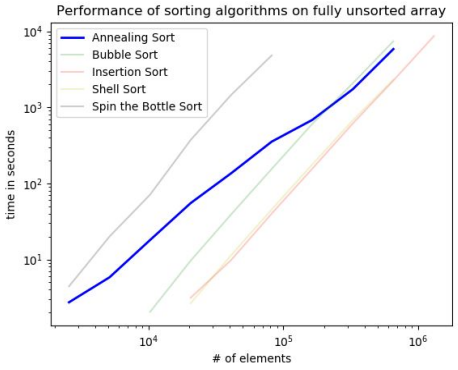
Fully Shuffled Array

Temperature Sequence #1



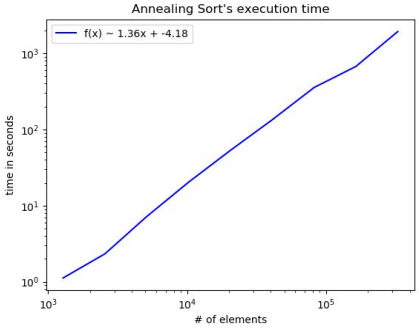


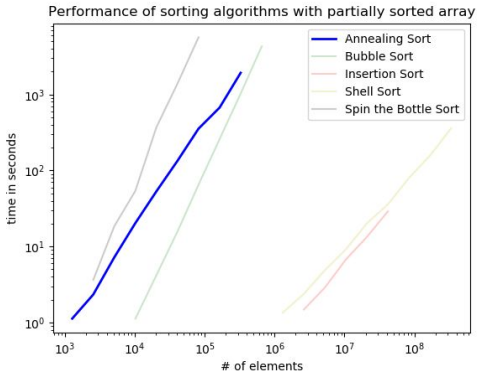
Temperature Sequence #2



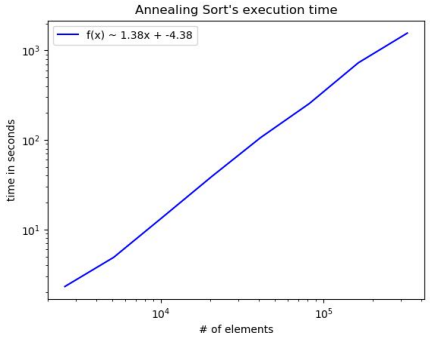
Partially Shuffled Array

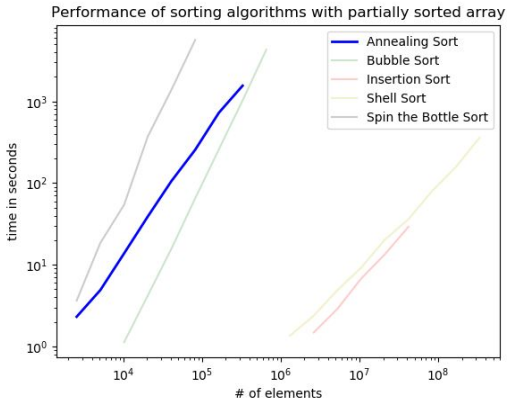
Temperature Sequence #1





Temperature Sequence #2





Annealing sort is a probabilistic sorting algorithm. This means that its performance greatly varies, the above graphs were generated by averaging the results of repeatedly sorting a fully shuffled array five times and averaging the results for each discrete array size. This algorithm works by considering r_j elements that are a valued-distance T_j from a given element, where $r_j \subset R \; and \; T_j \subset T$. The Annealing Sort algorithm requires us to define two sequences, the *temperature sequence* and the *repetition sequence*.

For our implementation of Annealing Sort, we have, using the hand-waving theorem, derived two temperature sequences and two repetition sequences. The temperature sequences use a similar paradigm to Goodrich's temperature sequence in that it is split up into 3 phases. The following shows how to acquire the first temperature sequence:

$$T_1(N) = \{S_1(N)\}$$

$$T_{11}(N) = fib(log_2(N) + 1), \; where \; fib \; returns \; the \; first \; log_2(N) \; fibonacci \; numbers$$

$$T_{11}(n) = log_2(T_{11}(n) + 1)^{\frac{\pi}{2}} \; * \frac{\pi}{2} log_2(T_{11}(n-1) + 1), \; for \; all \; valid \; n \geq 2 \; up \; until \; log_2(N)$$

$$T_{12}(n) = \{1, \; 2\} \; to \; be \; built \; gradually \; until \; of \; size \; log_2(N)$$

$$T_{12}(n) = T_{12}(n-1) + log_2(N) * (n * e^{-\pi}), \; for \; n \geq 3 \; up \; until \; N$$

$$T_{12}(n) = T_{12}(n) \; + \; offset, \; for \; n \geq 1 \; up \; until \; N, \; where \; offset \; is \; the \; last \; element \; of \; T_{11}(N)$$

$$X'(n) = \{T_{11}(n),\ T_{12}(n)\}$$
$$X'(n) = X'(n) + log_2(N/3) + 1\ ,\ for\ all\ n$$
$$X(n)\ =\ reverse(X'(n))$$
$$T_{13}(N) = \{log_2(N), log_2(N) - 1,\ log_2(N) - 2,\ ...,\ 0\}$$
$$S_1(N)\ =\ \{X(n),\ T_{13}(N)\}$$

What results is a sequence of numbers in non-increasing order, and the last element being 0, representing the first temperature sequence. It may be easier to simply look at the code/implementation if one were to want to use this temperature sequence.

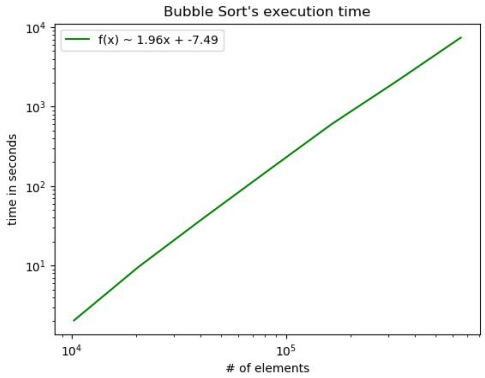
The following shows how to acquire the second temperature sequence:

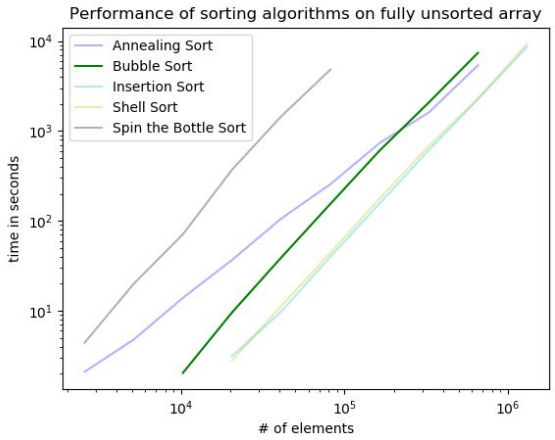
$$T_2(N) = \{S_2\},$$
$$We\ first\ define:\ dogeSequence(n,\ g,\ o)\ =\ \{1,\ 2\},\ to\ be\ gradually\ built\ up\ until\ log_2(N)$$
$$T_{21} = dogeSequence(n,\ 0.8,\ 0)\ =\ \tfrac{7n}{10} * dogeSequence(n-1,\ g,\ o)^g + n,\ for\ n\ \geq 3$$
$$T_{22} = dogeSequence(n,\ 0.75,\ last\ element\ of\ T_{21})\ =\ \tfrac{7n}{10} * dogeSequence(n-1,\ g,\ o)^g + n,\ for\ n\ \geq 3$$
$$X'(n) = \{T_{21},\ T_{22}\}$$
$$X'(n) = X'(n) + log_2(N/3) + 1\ ,\ for\ all\ n$$
$$X(n)\ =\ reverse(X'(n))$$
$$T_{23}(N) = \{log_2(N), log_2(N) - 1,\ log_2(N) - 2,\ ...,\ 0\}$$
$$S_2(N)\ =\ \{X(n),\ T_{23}(N)\}$$

The two repetition sequences are just borrowed from the first and second phase of the first temperature sequence respectively. As you can see from the graphs, the experimental running times for a fully shuffled array and partially shuffled array is about $O(n^{1.39})$ and $O(n^{1.37})$ respectively when using my temperature sequences. The temperature sequences can be improved to make the running time closer to $O(nlog(n))$.

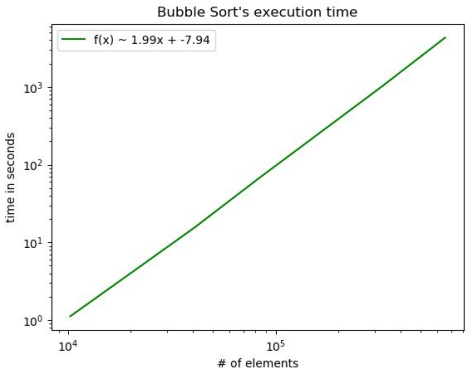
Bubble Sort

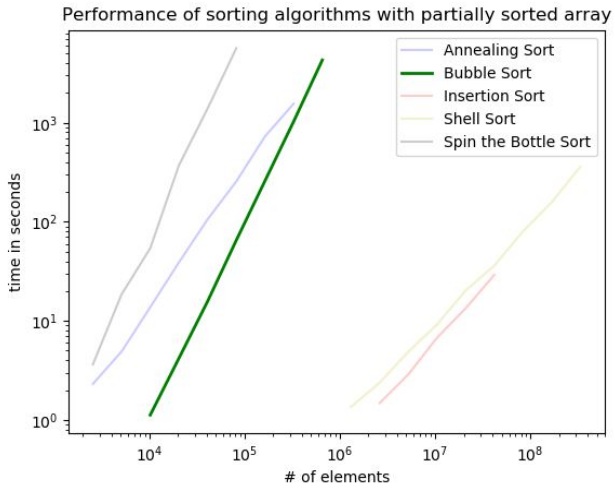
Fully Shuffled Array





Partially Sorted Array

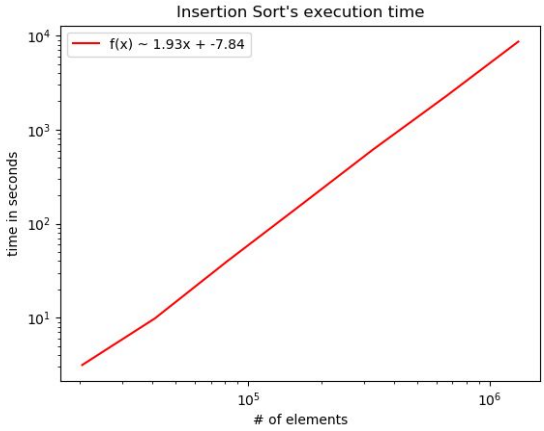


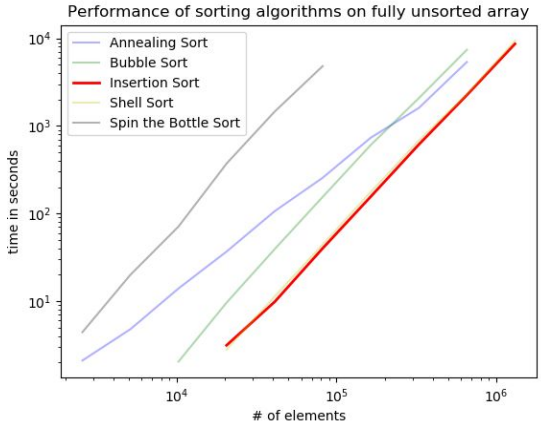


Bubble Sort is a classic sorting algorithm that is considered by most to be a bad sorting algorithm. It works by moving the the biggest element in the array to the last position in the array, and on the next iteration moves the second biggest element to the second to last spot of the array, and continues this paradigm until it is fully sorted. As you can see, the experimental running time, $O(n^{1.96})$, is close to the theoretical running time, $O(n^2)$. One thing to note about this algorithm is that the theoretical running time is the same in the best-case and worst-case, and the experiments seem to be in accordance with that.

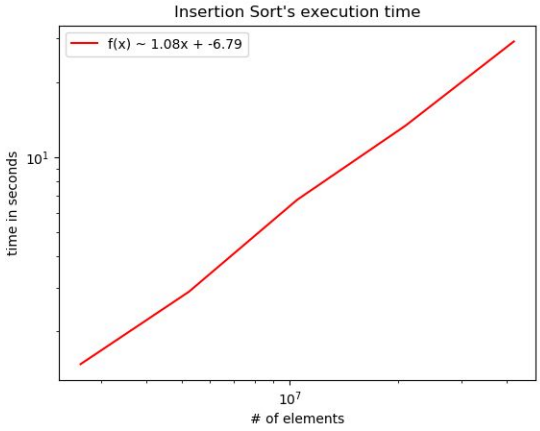
Insertion Sort

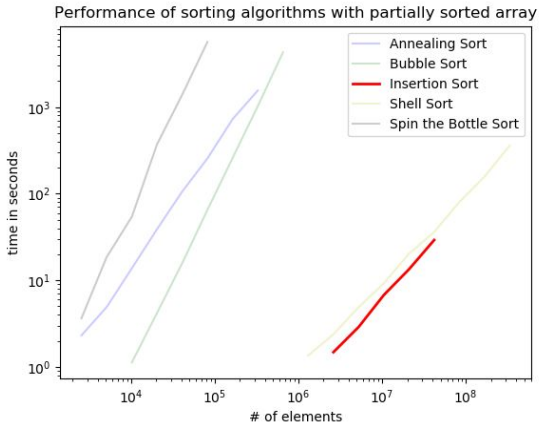
Fully Shuffled Array





Partially Shuffled Array





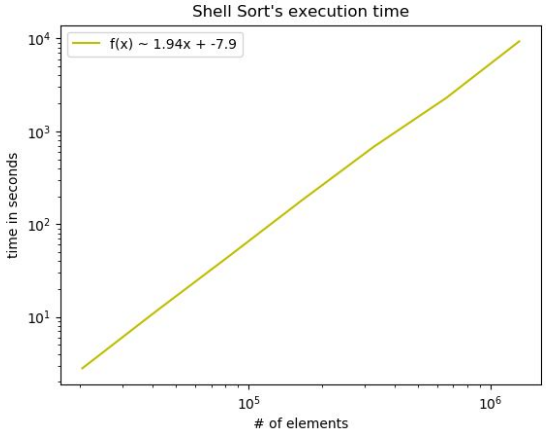
Insertion sort is another popular sorting algorithm. This algorithm works by iterating the array from left to right, and at each iteration take the current element and place it to where it belongs (somewhere to the left) and shift all elements accordingly. As you can see, the experimental runtime, $O(n^{1.93})$, is close to the theoretical runtime, $O(n^2)$. The constant of the polynomial is rather small constant since it's also such a simple algorithm. One thing to note about this algorithm is that its best-case runtime is $O(n)$, which occurs when the input array is

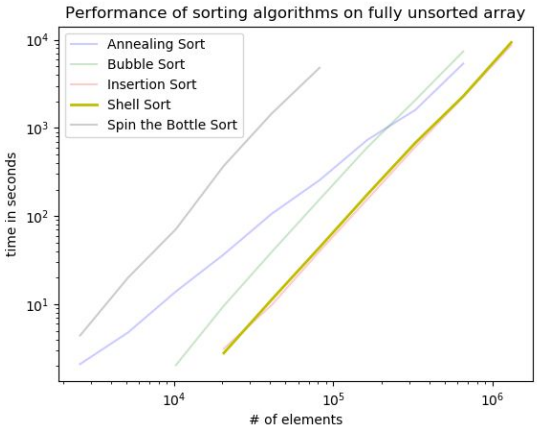
already sorted. Very good running time is also observed when the array is partially sorted, as one can see from the experimental $O(n^{1.06})$ running time we derived for a partially sorted array.

Shell Sort

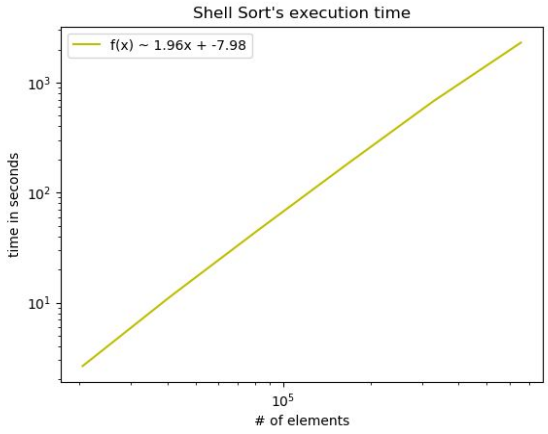
Fully Shuffled Array

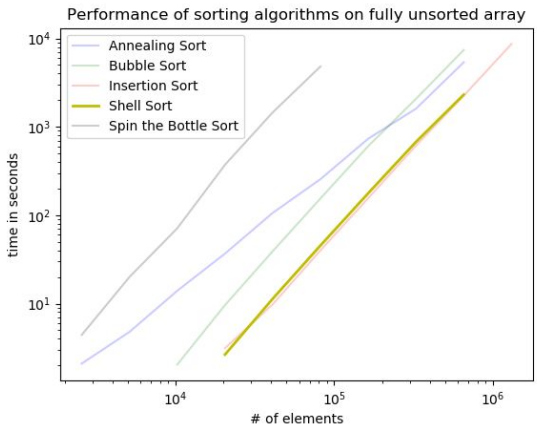
Gap Sequence #1





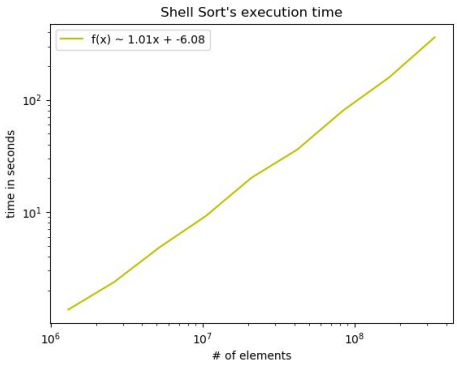
Gap Sequence #2

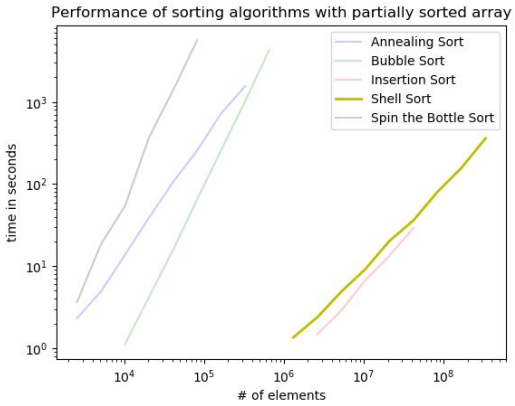




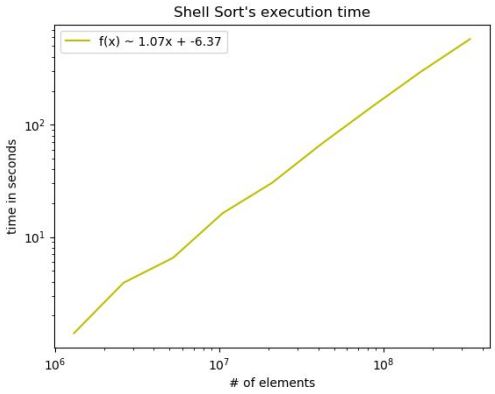
Partially shuffled array

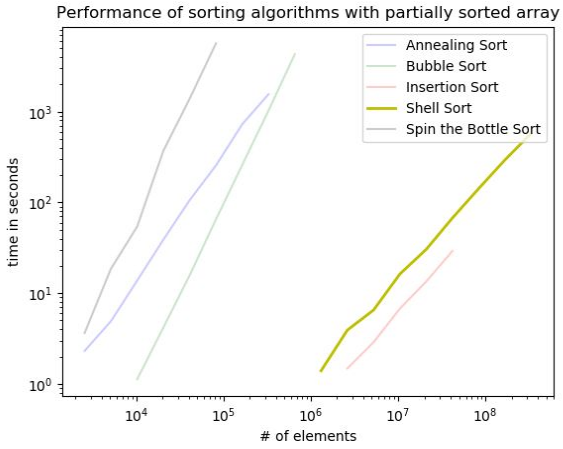
Gap Sequence #1





Gap Sequence #2



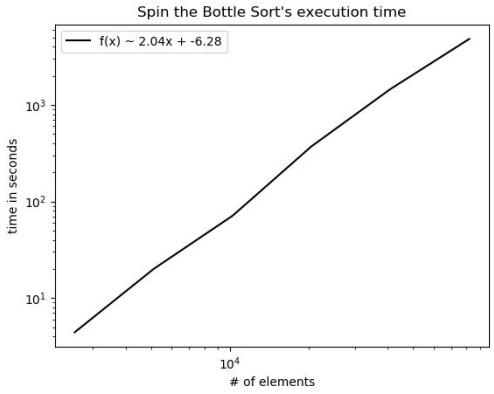


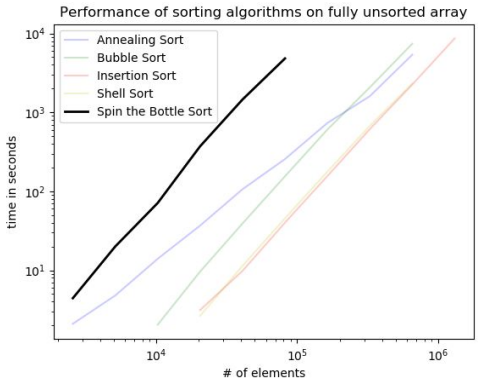
Shell Sort is similar to insertion sort. It introduces the notion of a gap sequence, which it uses to insert elements to their correct position only the place it may belong in is at least some distance away. As you can see, the experimental runtime, $O(n^{1.94})$ when using the first gap sequence and $O(n^{1.96})$ when using the second gap sequence. Very good running time is observed on a partially sorted array, $O(n^{1.01})$ and $O(n^{1.07})$ for the first and second gap sequences respectively. This shows that Shell Sort and Insertion Sort are indeed similar.

Gap sequences #1 and #2 are the same as the first phase of the first temperature sequence and the first phase of the second temperature sequence respectively.

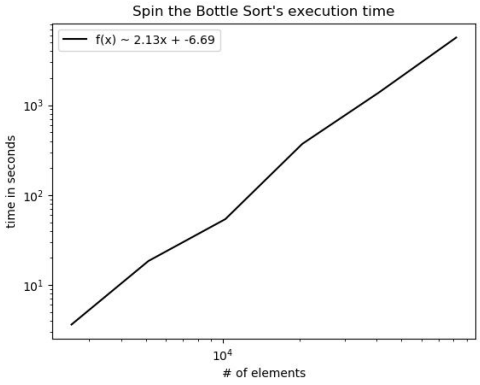
Spin-the-Bottle Sort

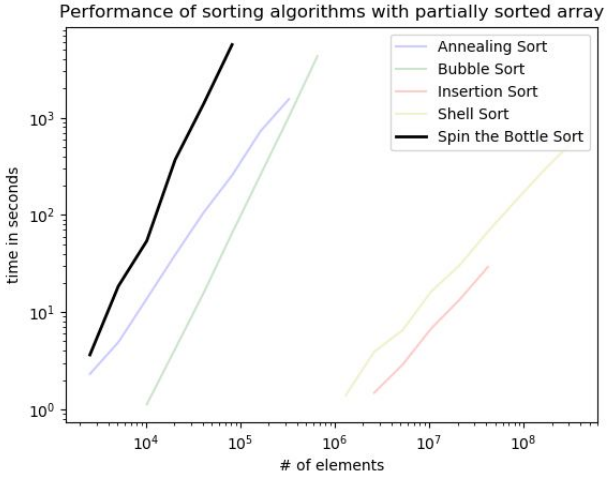
Fully Shuffled Array





Partially Shuffled Array





Spin-the-Bottle Sort has been said by its creator (Michael T. Goodrich) to be a "joke" sorting algorithm. It works by iterating through the array, randomly picking an index in the array and swapping the element at that randomly chosen index and the current index (via iteration) if elements are out of place with respect to each other; it repeats this (even iterating through the entire array again) until the array is sorted. The experimental runtime is $O(n^{2.04})$, and its theoretical running time $\Omega(n^2 \, log(n))$, which is worse than Bubble Sort.