Jonah Fidel

[jfidel@colgate.edu](mailto:jfidel@colgate.edu)

Computer Science 102

Lab #2

Sorting Detective

Mystery Algorithms

Note: all data given assumes an array of 100,000 items

1. Quicksort (with median of [first, median, last] as pivot)

Mystery Algorithm 1 is quicksort with a pivot as the median of the first, median, and last numbers. No matter what the data set, this algorithm was the most efficient all-around for sorting the information, taking averages of 11.5ms, 16ms, and 17ms for ascending, descending, and random data sets respectively. It makes sense that this would be the best algorithm all-around, as quicksort is the most efficient algorithm to begin with an average runtime of O(nlogn) and this specific algorithm chooses a ‘good’ number as a pivot (a number relatively close to the actual median of the data in most cases).

1. Selection Sort

Mystery Algorithm 2 is selection sort. The runtimes of this algorithm were the same for all types of data (ascending, descending or random values), and extremely long compared to some of the other algorithms (around 14000ms). This means the algorithms must be selection sort as selection sort has the same runtime no matter how the data set is ordered, and it always takes longer than many other algorithms with a guaranteed runtime n squared.

1. Merge Sort

Algorithm 3 was best discovered by process of elimination. Merge sort makes sense for this data set, as it has a consistent runtime of 25ms for both ascending and descending order, and a slightly higher average runtime of 35ms for random ordered data. Merge sort acts by continuously splitting and reconstructing the data set so these numbers represent a fairly efficient algorithm, but one that is slightly slower than a best-case quicksort.

1. Quicksort (with last number of data set as pivot)

Algorithm 4 had an interesting set of runtimes. When the data was placed in ascending order, the runtime was 2.3ms, however if the data were random or descending, the runtime was around 22ms. A quicksort using the last number as the pivot would yield a very good runtime with the data in ascending order as the pivot is guaranteed to be the largest number of the data set, which will only need to be compared once to each other number in the set. In other cases, however, this will not yield as efficient an algorithm, as the pivot will be completely random or the smallest number of the set.

1. Insertion Sort

Algorithm 5 is insertion sort. Insertion sort has a guaranteed runtime of O(n) when the data is in ascending order. This was demonstrated, as the algorithm sorted an ascending array of 100,000 items in only 0.3ms. Insertion sort is awful at sorting data in the worst case (descending order), however, which was demonstrated by a runtime of 14,000ms for this algorithm.

1. Shell Sort

Algorithm 6 is shell sort. When the data set is in ascending or descending order, the average runtime was around 12,900ms, as opposed to an average of only 17ms when the data was arranged randomly. This makes sense because when the data is arranged in ascending or descending order, the random shell choice will do nothing but inefficiently rearrange data until it ultimately returns to its original position. When the data is random, however, the shells will serve a better purpose of randomly grabbing data and moving it toward its proper position in the sorted array.