Lakshya Mehta

Sorting

# objects	Selection Sort	Quick Sort
1000	13.4256	1.029
10000	150.8297	27.0896
20000	570.6815	36.0866
50000	8227.7048	103.8605
100000	33809.184	267.216

The selection sort algorithm roughly follows the predicted big O notation of n^2. This can be observed since the time for 20000 objects is roughly 4x the 10000 objects for selection sort. The time for 50000 is roughly 50x the amount for 10000 objects, as opposed to the predicted 25x, but the amount for 100000 is roughly 4x the amount of time for 50000 objects which is predicted.
The quicksort algorithm has a predicted big O of $n*\log_2(n)$. However, the quicksort algorithm slightly underestimates the predicted relationship, since the time for 20000 objects is only roughly 1.3x the amount for 10000 as opposed to the predicted 2x; the amount for 50000 is 3.8x the time as opposed to the predicted 5x. The only relationship where it is relatively close is for the relationship between the time for 100000 vs 50000 and 100000 vs 10000 which had predicted factors of 2x and 10x and actual factors of 2.6x and 9.85 respectively. However, these discrepancies are probably due to some overhead associated with accessing an object's attributes, and based on the higher accuracy of the relationships between large datasets, the big O notation probably holds as the set sizes get bigger.

Lakshya Mehta

Searching

# objects	Linear Search (sorted, fixed searchClock)	Linear Search (unsorted)	Binary Search
1000	0.487	0.2116	0.0174
10000	1.6629	0.4113	0.0235
20000	6.6453	1.6306	0.0312
50000	6.8739	8.4171	0.0272
100000	10.9986	11.7673	0.0451

The linear search algorithm doesn't hold up to the expected big O notation. The linear search does not scale linearly, since increasing the set size by 10x results in either a 2x between 1000 and 10000 objects or ~30x between 100000 and 10000 objects, and the ~5x jump between the 20000 and 50000 set sizes is not correct. Again, this is probably associated with some overhead for accessing class info, or has to do with how there are only 50 discrete values for the clock speed so searching for any one is almost guaranteed to be in the dataset.

For the binary search, in some cases the time for larger datasets is less than the time for smaller datasets, but generally, the binary search should stay around a similar value for large data sets, increasing by factor of $\frac{log_2(ax)}{log_2(x)}$ where a is the factor between two set sizes. The limit as x approaches infinity for any value of a is 1, so it makes sense that the time hovers around something similar for all the trials.