K. Hohmeier

CSC 380

Homework 4 Written Report

Due September 24, 2020, 11:59 PM

Problem 1: Implement Insertion Sort

For this problem, the insertion sort algorithm, as described in chapter 4 of the class textbook, were implemented in Python. The code was also configured so that the implementation of the insertion sort function counted and returned the number of comparisons performed for a randomly generated list of a given size. The actual number of comparisons, along with the expected number of comparisons generated by the sequence $n^2/4$, were graphed against the size of the list (see Figure 1). Since the average case for insertion sort is $n^2/4$ comparisons (where n is the size of the list), we would expect that the actual number of comparisons performed would be approximately equal to $n^2/4$. The graph shown in Figure 1 confirms this prediction. The scatterplots match very closely.

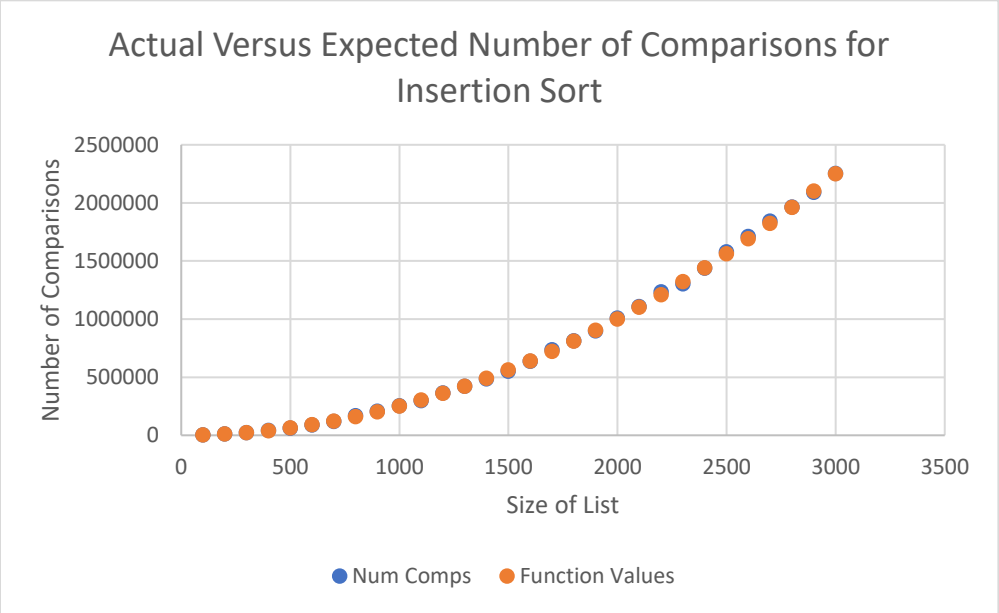

Figure 1. Graph of times for bubble sort and selection sort

Problem 3: Russian Peasant Multiplication

For this section, the Russian Peasant Multiplication algorithm was implemented. The code was also designed so that the number of multiplications, divisions and additions were counted, summed, and returned. The algorithm accepts as input two positive integers, n and m. We wish graph the size of n against the number of comparisons performed by the algorithm; we expect that this will produce a logarithmic curve. To do this, n and m were initially set to be around 100,000 and iteratively increased until they were around size 100,000,000. These results of this experiments are shown in the graph, Figure 2. As can be seen from this graph, we obtained a logarithmic curve. From this, we can conclude that the Russian Peasant Multiplication algorithm has logarithmic efficiency.

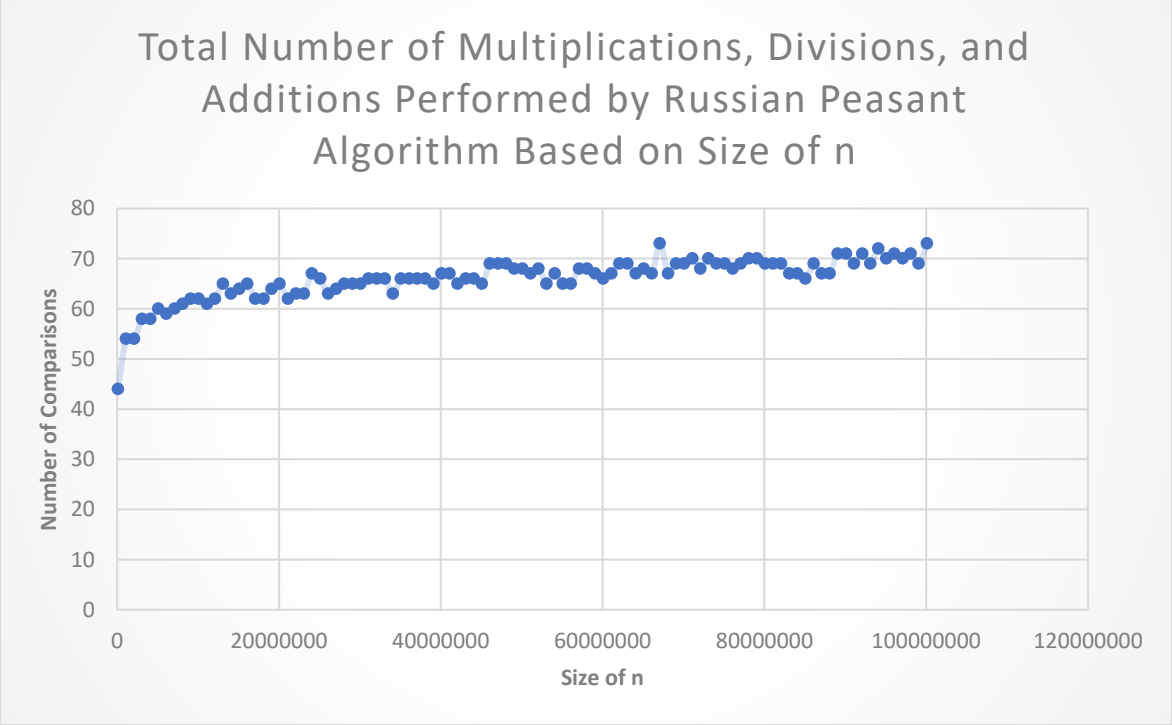

Figure 2. Graph of total number of multiplications, divisions, and additions performed in the Russian Peasant Multiplication Algorithm versus size of input n

Problem 4: Lomuto Partition and Quick Select Algorithm

For the last problem, the Lomuto Partition algorithm was implemented in Python and subsequently utilized in the Quick Select algorithm. Our goal is to test whether finding the kth element of a list can be performed in O(n) time with these two algorithms. To test this, we require the total number of comparisons that occur within the Lomuto partition algorithm. The size of the list would then graphed against this result. First, a list of randomly generated numbers was created via a helper function. Then, a series of experiments were performed to find the total number of comparisons performed by the Lomuto partition when Quick Select is called to find the kth element of the list. The value of k for this experiment was found by taking the middle element of the list. This experiment generated the graph shown below in Figure 3.

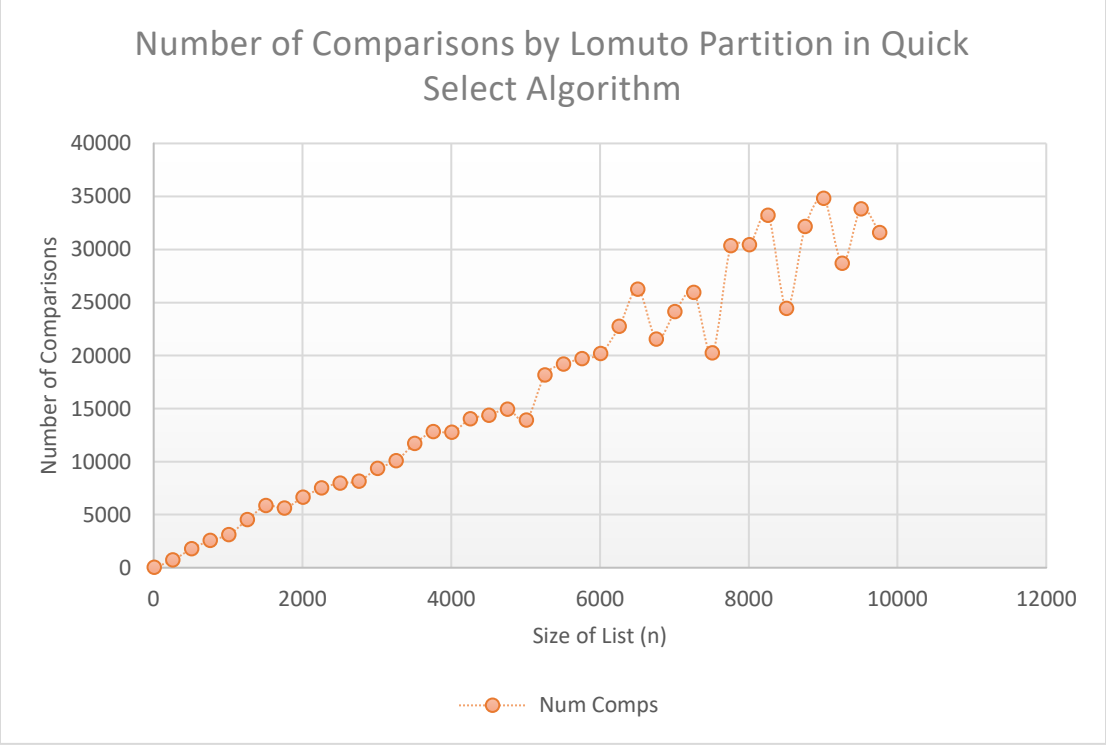

Figure 3. Graph of number of comparisons performed by the Lomuto Partition algorithm within the Quick Select algorithm

As can be seen from Figure 3, the graph obtained was roughly linear. This is the expected result, since we claimed that finding the kth element of a list can be performed in O(n) time with these two algorithms.