<u>Part A.</u> State the function I believe my plot represents, and explain why.

I believe that my plot represents a logarithmic function. In terms of big O, I believe it represents an O(log N) function due to its average time complexity explained in part B. In terms of a specific function of x, I believe that the function is similar to the function:

$$(9/10)\ log(x)^{1.5}$$

I believe that quick sort's function O(log N), in terms of the collected data, has a function similar to this because this function only has an average standard deviation of 0.6449 from the output, and an average variance of -0.1094. Please reference the attached excel sheet for more information.

<u>Part B.</u> What does the function have to do with the average case complexity of quicksort?

The function tells me that because it's logarithmic, the average case complexity is the time it takes for the quicksort algorithm to sort through an array increases linearly, while the total number of items within the array increases exponentially. This is proven by the data as well: 10 items calls for an average of 0.9 exchanges, 100 calls for approximately 3.0, and 1000 calls for approximately 5.4. Each time the number of items exponentially increases, the average number of exchanges increases by a little less than 3.

Part C. Explain how I designed the simulation part of my program.

My simulation begins with the declaration of four global variables: rnd, S, C, and num hits; each limited to the scope of the program class. The variable rnd is a random variable which will be used to fill array S. Array S is an array of type long. S will be used as an array that holds an unsorted list of random integers, which will be sorted within the quicksort algorithm. Array C is an array of type long. C will be used to hold the number of times the quicksort method “sorts through” the S array in the simulation. The integer numHits is a variable that will hold the total number of exchanges the quicksort algorithm will parse through the current unsorted array. Following the declaration of the global variables, The main method initializes an integer array named lim with the following values: 10, 50, 100, 500, 1000, 5000, 10000, 25000, 50000, 75000, 100000, 200000, 300000, 400000, 500000. Array lim will be used to simulate different array sizes. The main method then parses through each value in lim, through the use of a for loop. The current position in lim is held by an integer named i. For each value in lim, array S is set to be a new instance of an array with its size being of the item in lim at i. Array C is set as a new instance of an array of size 50, and numHits is set to 0. The next for loop increments 50 times, its position held by an integer named j. For each increment, with the use of rnd, array S is filled with a random set of numbers. Each number ranges from 0 to the item in lim at i. Once array S is filled, the quicksort method is called, and array S is sorted. For each “exchange” within the quicksort algorithm, integer numHits is incremented up by one. Once array S has been sorted, array C at position j is set to the integer numHits. Following this statement, numHits is set back to zero. Once integer j has incremented up to 50, the for loop which incremented j is exited. A variable of type double is then declared, named average, and initialized to 0. A new for loop is created, and it parses through every value in C, using integer j to hold the current position in C. For each item in C, the item in C at j is added to the variable

average. Once every item in C is added to average, the for loop exits. Average is then set to be the average divided by 50. This was done to obtain the average number of exchanges from all 50 trials. Average is then divided by the value at i in lim. This was done to obtain the average number of exchanges for the quicksort algorithm to parse through, at the value of i at lim. Once all the items in lim have been parsed through, the for loop using integer i is exited. The proper output is printed, and the simulation has ended.