Donovan Griego

Blackbox Project Summary

My program set out to run the unknown functions found in blackbox.o and report the run-times of each iteration ran. When running the functions, it splits the dataset into a given number of iterations and runs each iteration three times to weed out any possible error. These three values are then averaged and the average is recorded as the final value. This happens for each iteration of running so the data should be reliable. Using this strategy I was able to get some pretty accurate fits to for each function.

To use my program run with the following:

```
./main -s size -f function -i iterations -F filename -S -R -M
```

Size [REQUIRED]: The number to feed into the function or the size of the array to pass to the function if running for function 3 or 6.

Function [REQUIRED]: The function to run the tests for. [1-7]

Iterations [REQUIRED]: The number of iterations, or samples, to get from the function. Must be less than or equal to Size.

Filename [REQUIRED]: The directory of the file to write the data to.

-S [OPTIONAL]: Will sort the array prior to feeding it into the function.

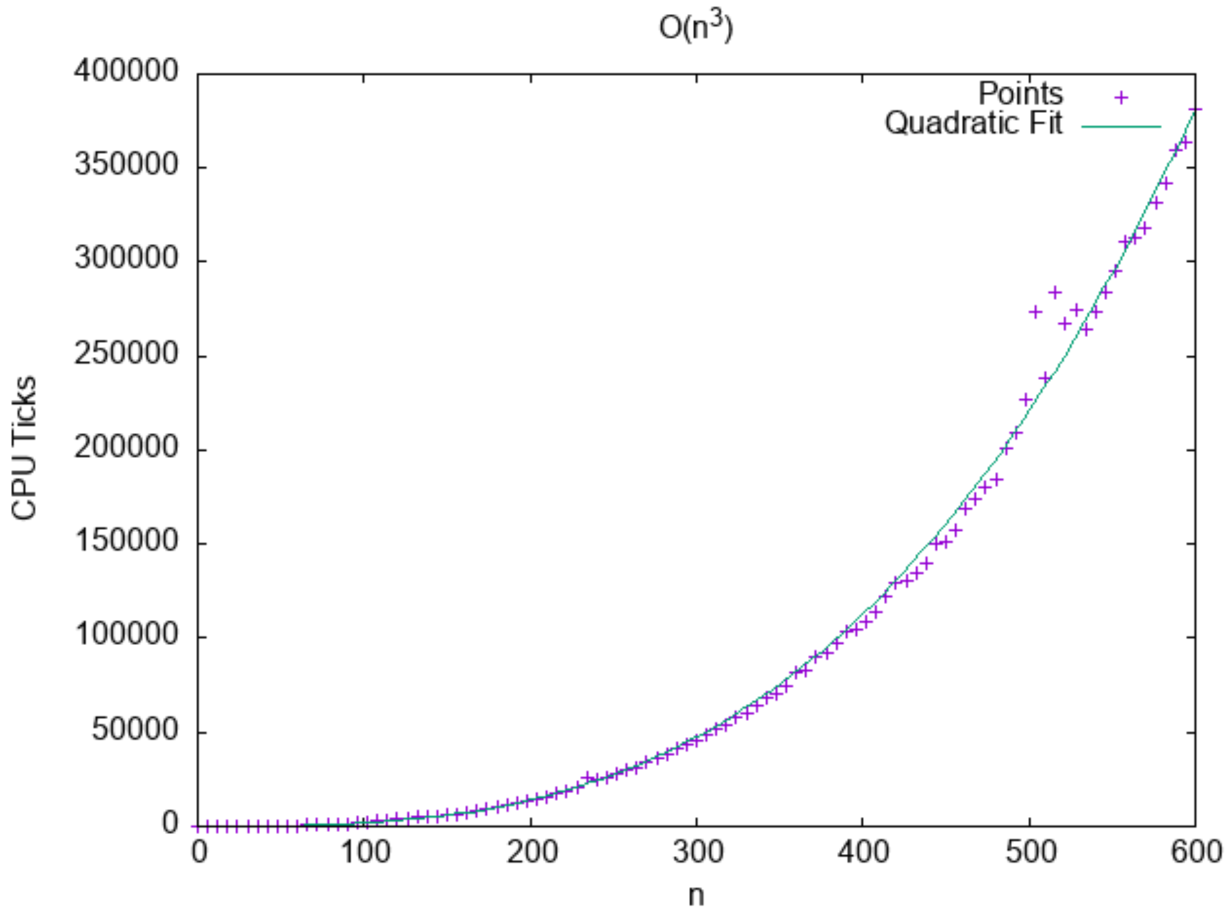
-R [OPTIONAL]: Will sort the array in reverse order prior to feeding it into the function.

-M [OPTIONAL]: Will mix up (shuffle) the array prior to feeding it into the function.

Note: While only one can be selected at a time, if none of -S, -R, or -F are specified -S is defaulted to and the array will be sorted.

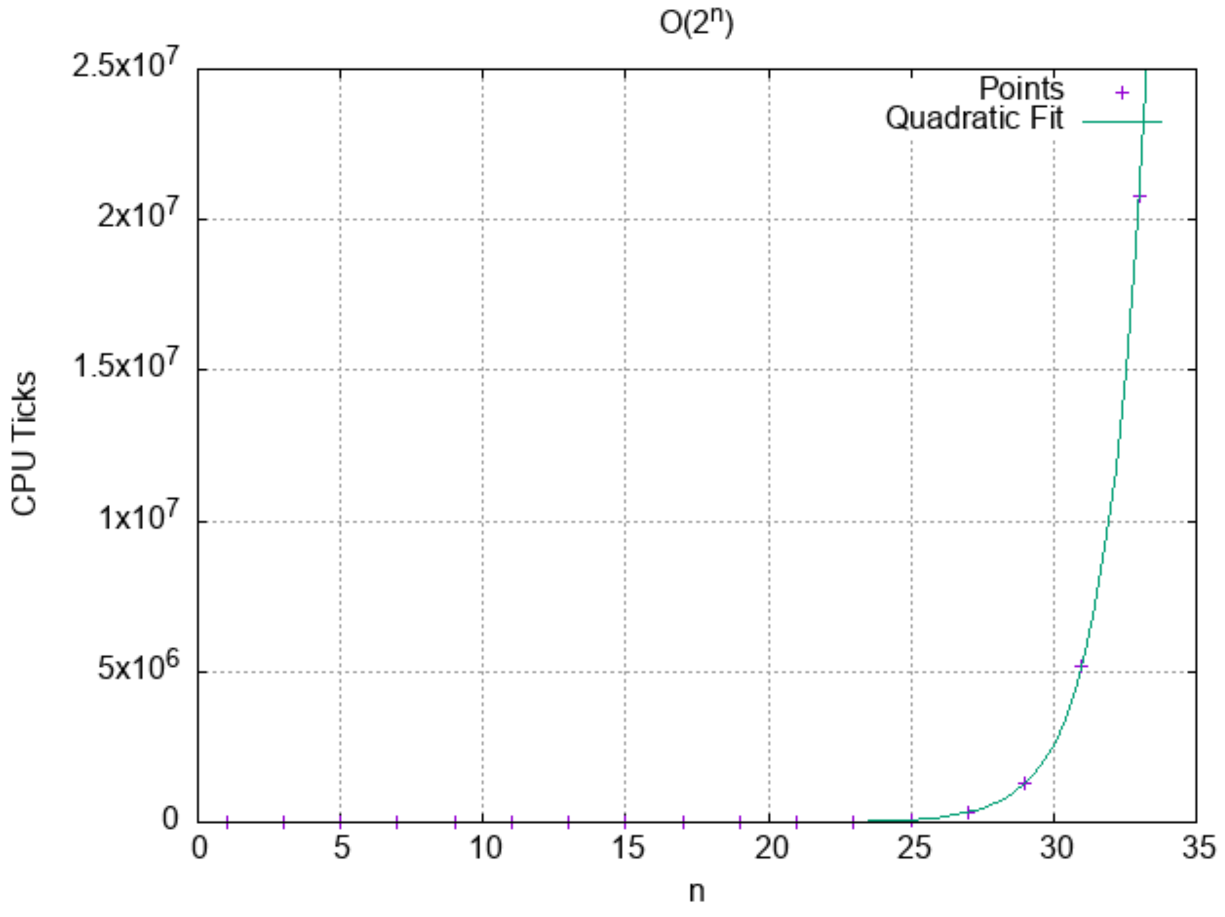
Function 1.

Testing out this function revealed that it is of $O(n^3)$ complexity. The data fits very nicely with this equation with a c value of 0.00176619 and an asymptotic standard error of +/- 9.078e-06 (0.514%).



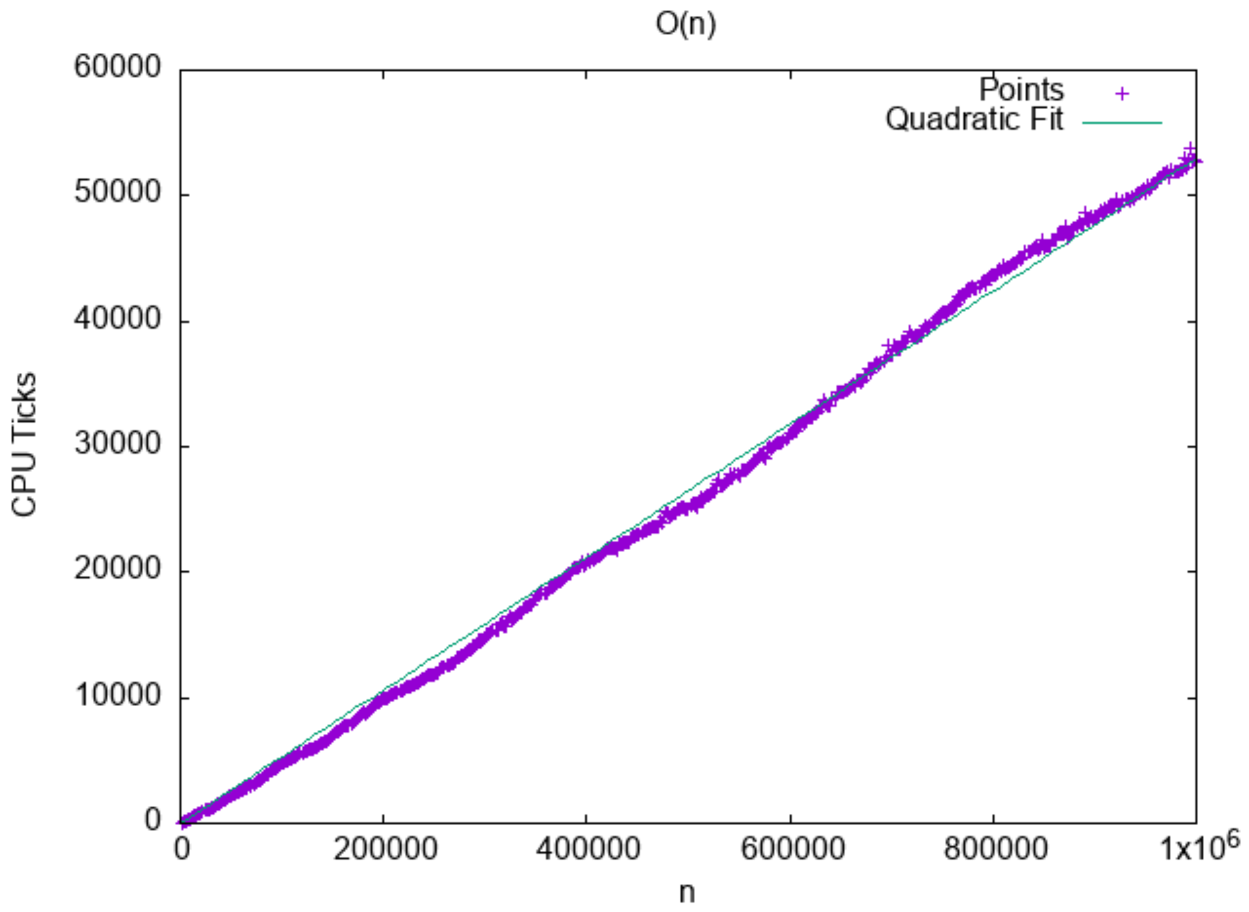
Function 2.

Testing out this function revealed that it is of $O(2^n)$ complexity. The data fits very nicely with this equation with a c value of 0.00241703 and an asymptotic standard error of +/- 2.391e-07 (0.009893%).

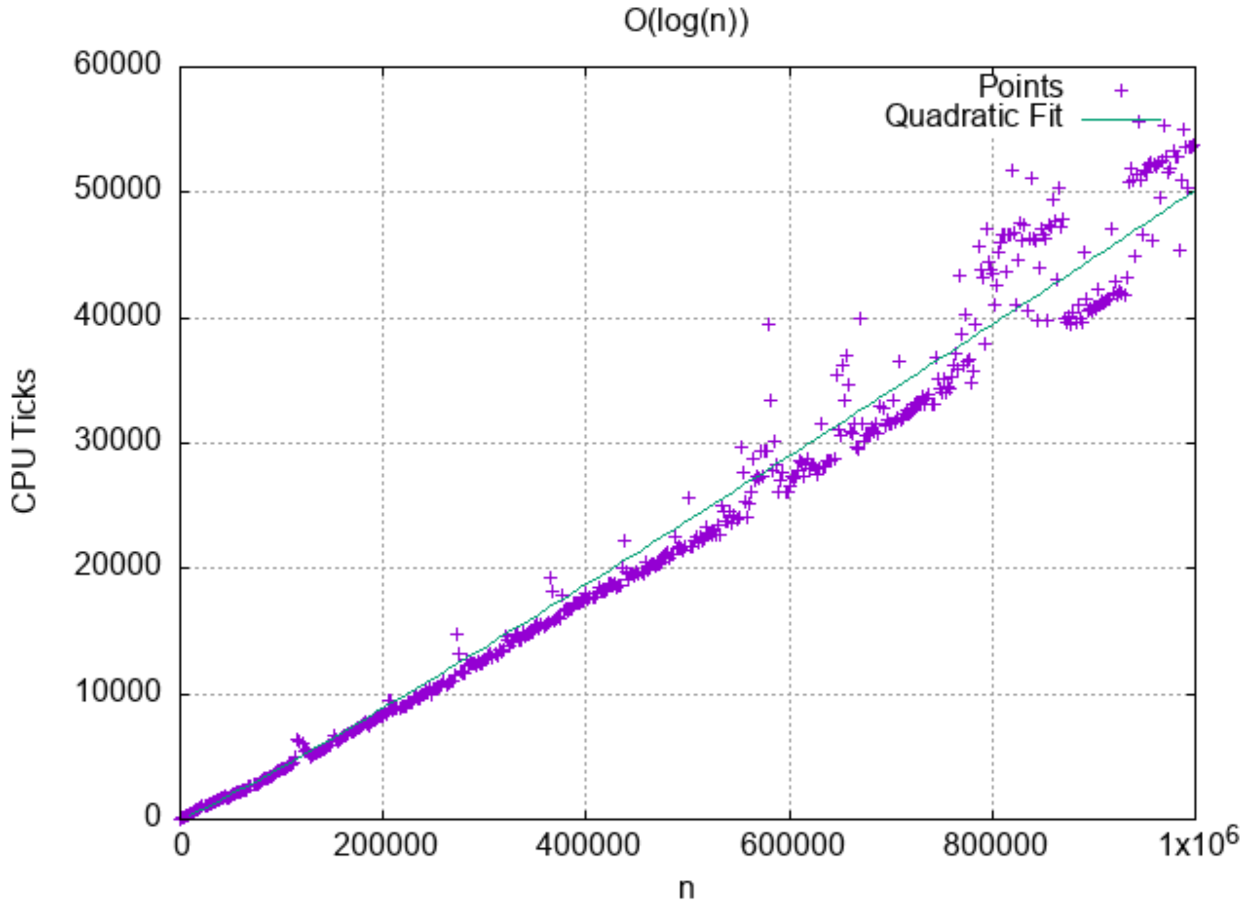


Function 3.

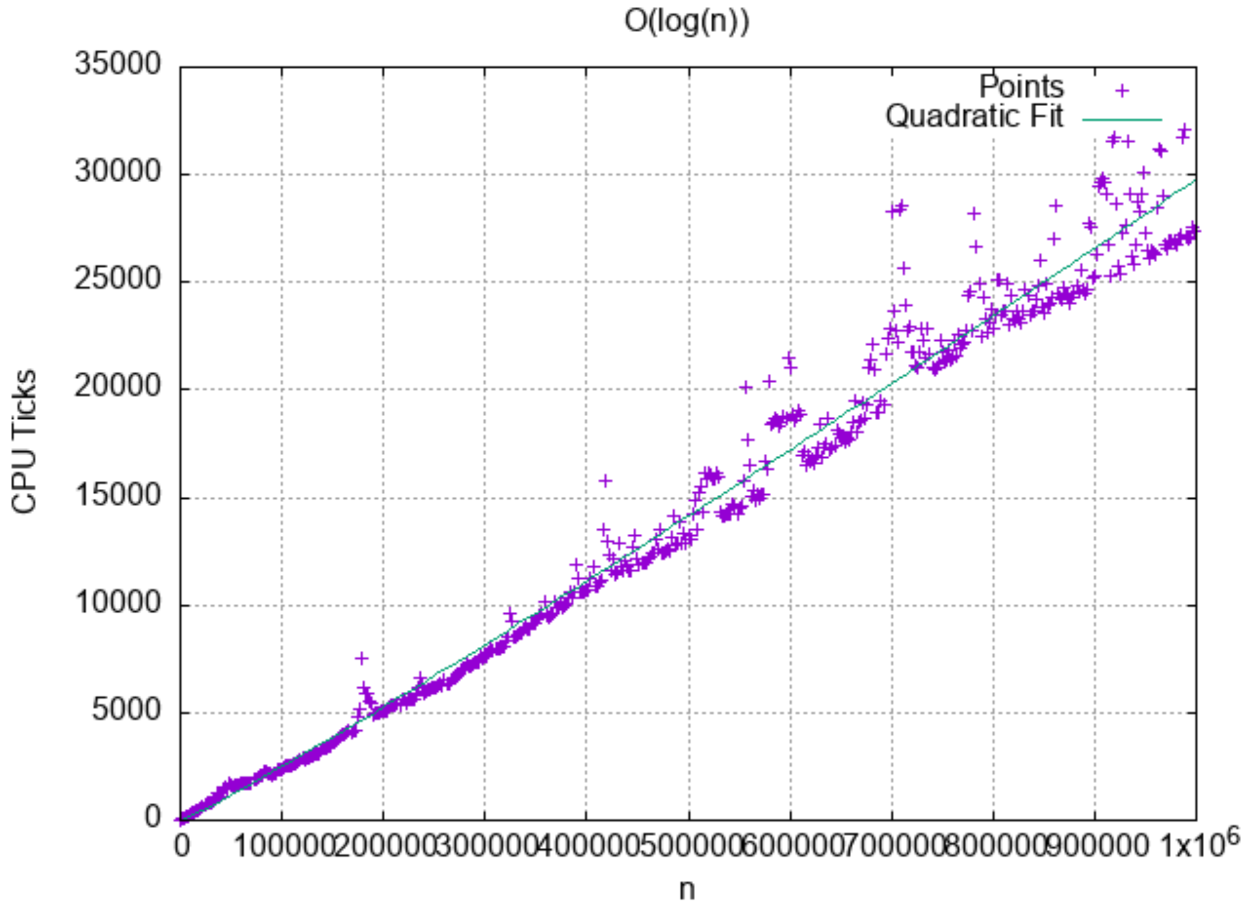
Testing out this function revealed that its longest complexity is $O(n)$ when the array was sorted. It is $O(logn)$ for shuffled and reverse order. This could mean it is a sorting algorithm sorting in normal order.



Sorted: c = 0.052985; asymptotic standard error = +/- 6.655e-05 (0.1256%)



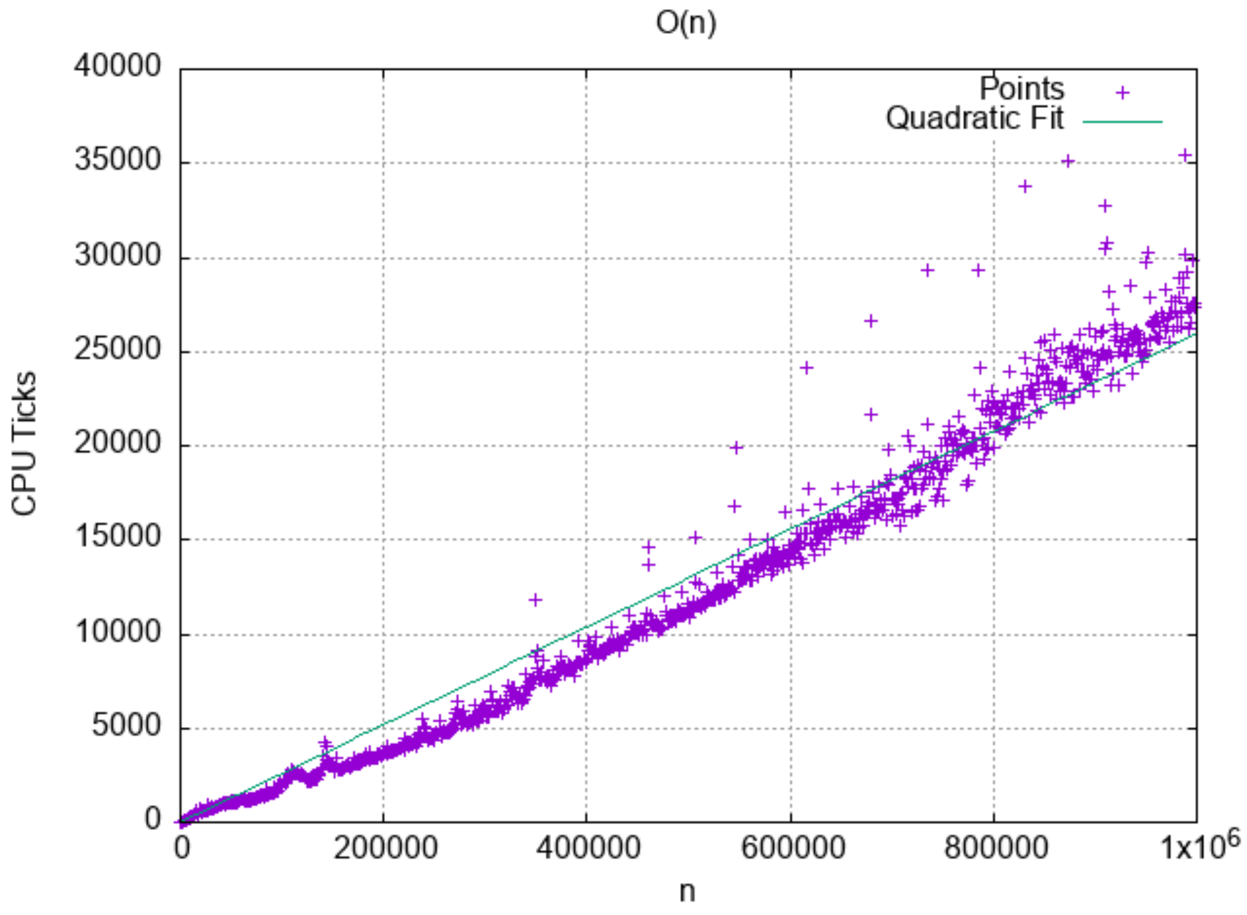
Shuffled: c = 0.00362982; asymptotic standard error = +/- 1.491e-05 (0.4107%)



Reversed: c = 0.00215518; asymptotic standard error = +/- 7.737e06—05 (0.359%)

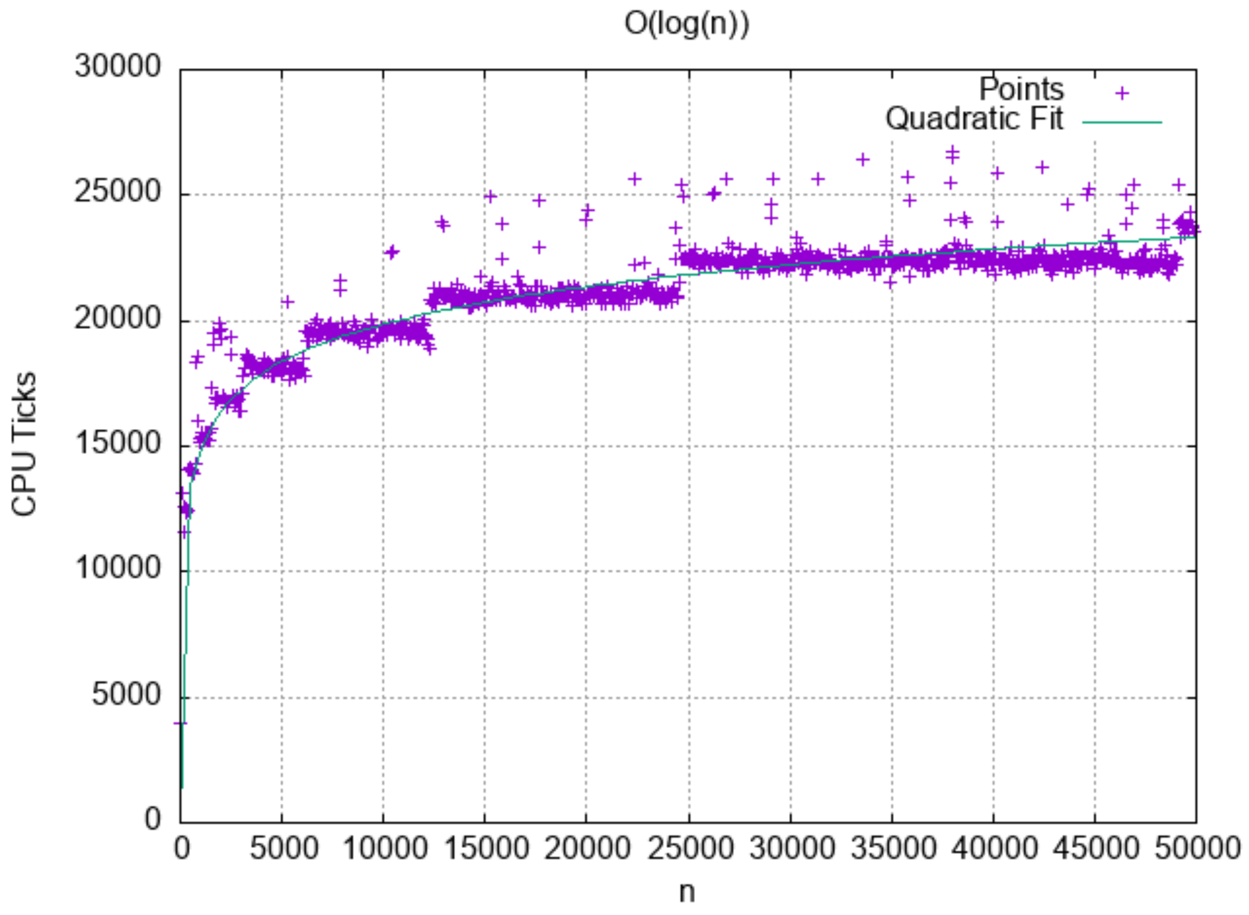
Function 4.

Testing out this function revealed that it is of $O(n)$ complexity. The data fits very nicely with this equation with a c value of 0.0259594 and an asymptotic standard error of +/- 9.605e-05 (0.37%).



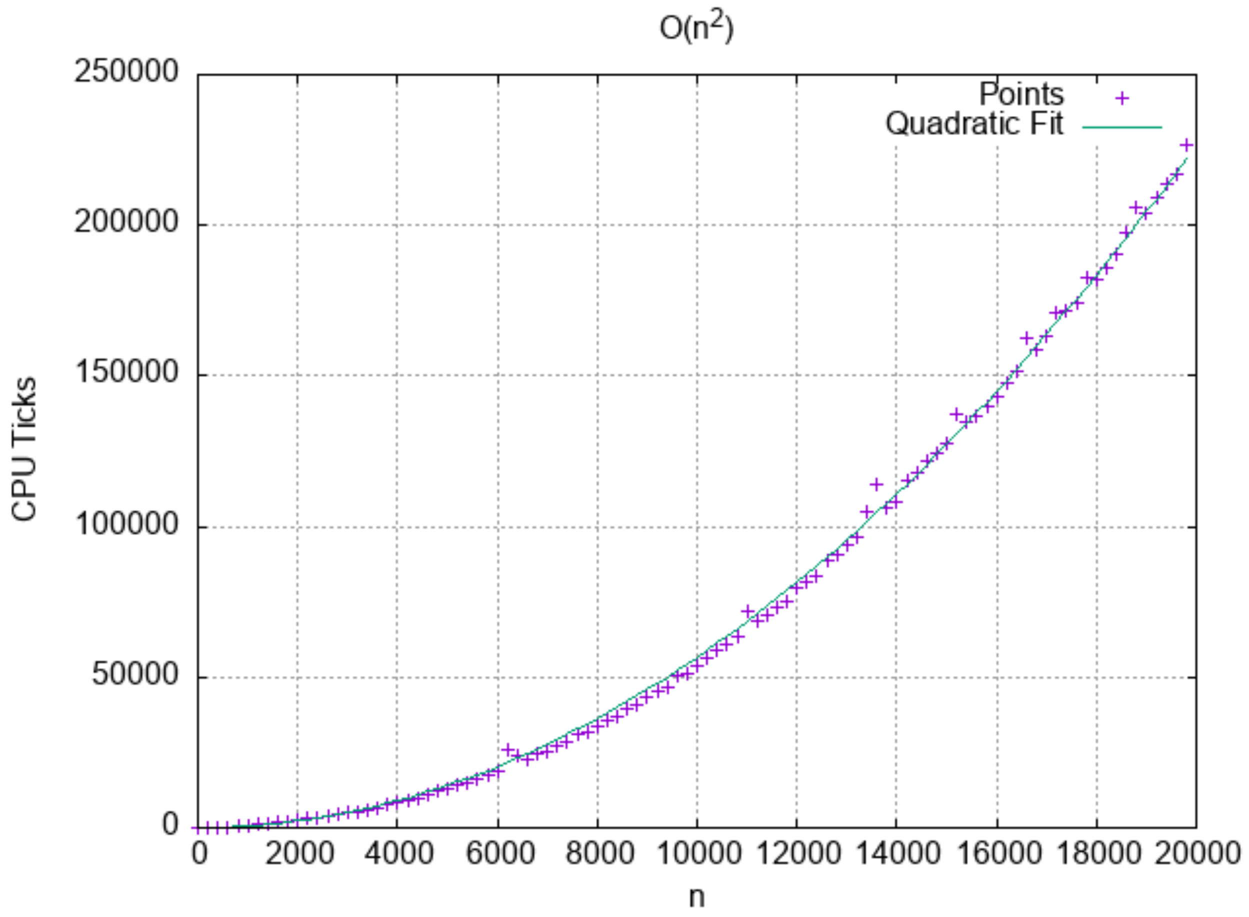
Function 5.

Testing out this function revealed that it is of $O(log(n))$ complexity. The data fits very nicely with this equation with a c value of 2156.53 and an asymptotic standard error of +/- 2.778 (0.1288%).

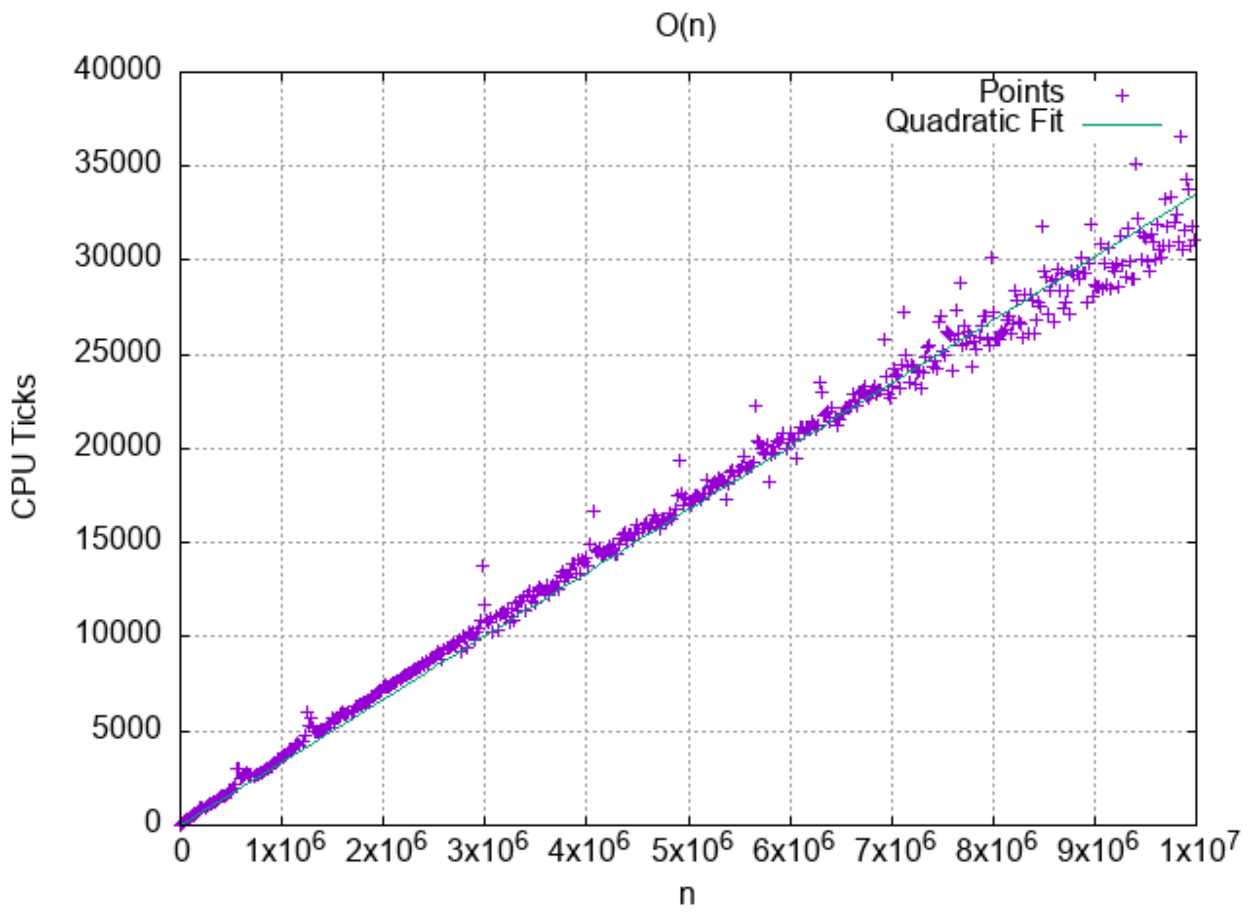


Function 6.

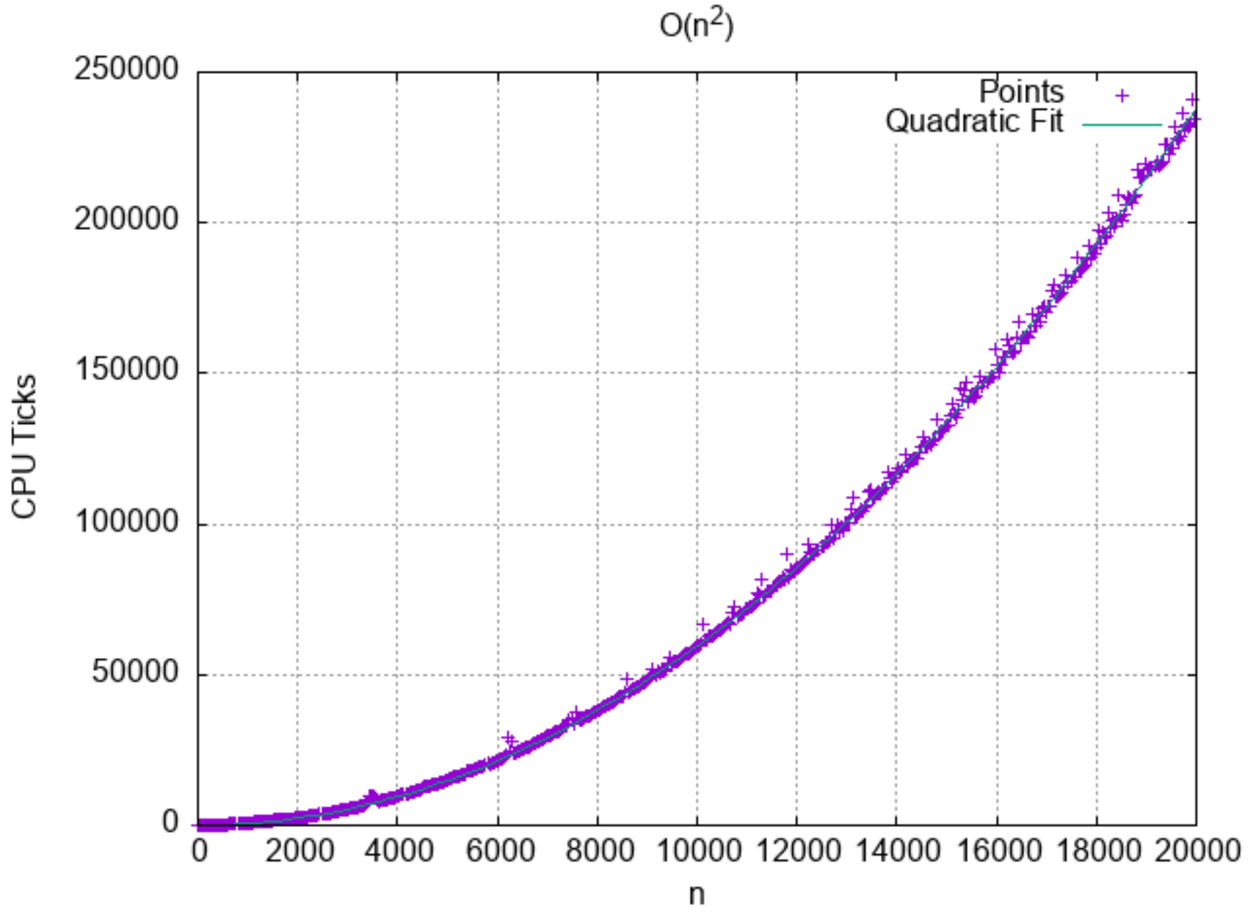
Testing out this function revealed that its longest complexity is $O(n^2)$ when the array was sorted and shuffled. It is $O(n)$ for reverse order. This could mean that it is a sorting algorithm but it sorts into a reverse order.



Shuffled: c = 0.000566166; asymptotic standard error = +/- 1.343e-06 (0.2373%)



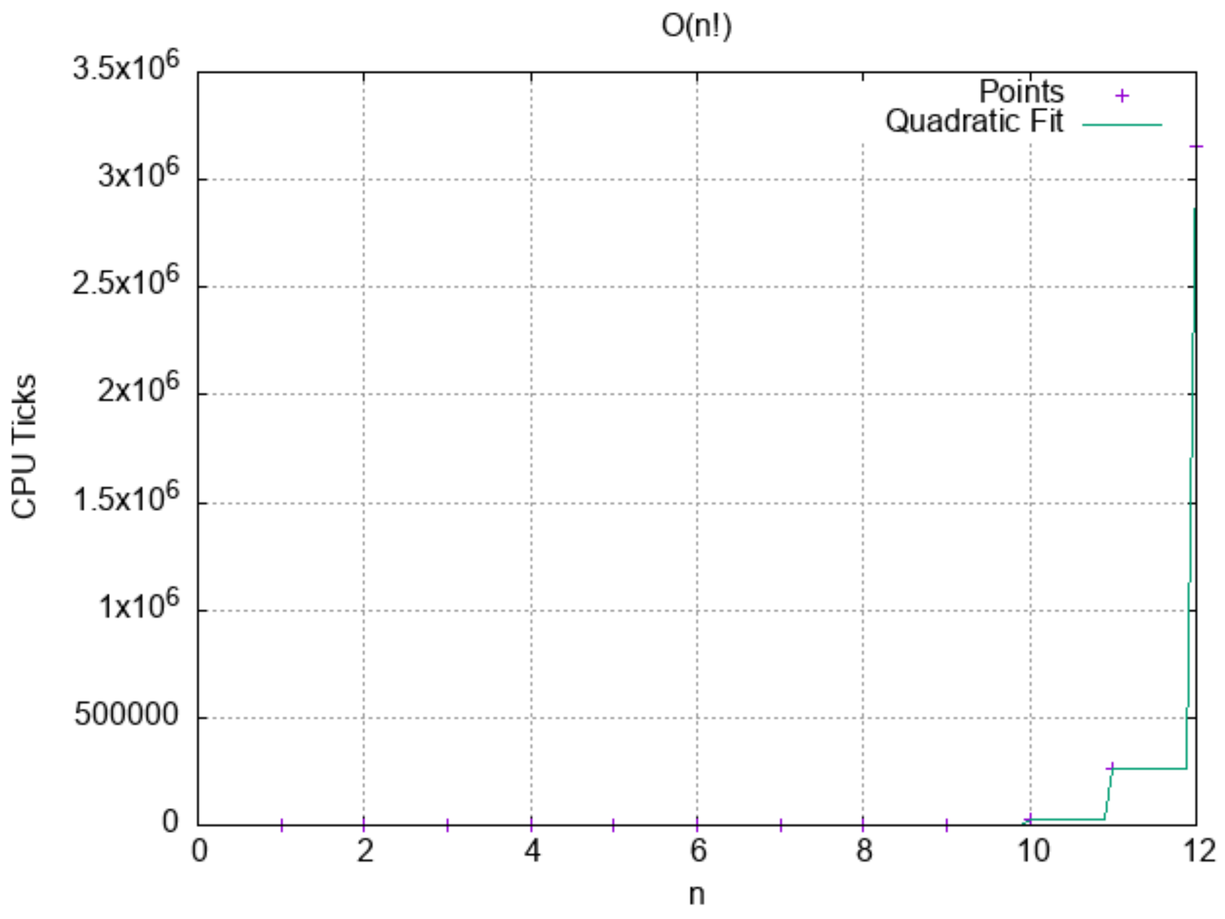
Reversed: c = 0.00335263; asymptotic standard error = +/- 7.243e-06 (0.216%)



Sorted: c = 0.00059568; asymptotic standard error = +/- 3.946e-07 (0.06624%)

Function 7.

Testing out this function revealed that it is of $O(n!)$ complexity. The data fits very nicely with this equation with a c value of 0.00 658174 and an asymptotic standard error of +/- 3.941e-06 (0.05988%).



Conclusion

In this project I created a program that ran the unknown functions inside of blackbox.o and reported the run-times for different sets of data. With these samples I was able to construct a plot to find the complexity of each of the functions. I was able to get a good estimate for each of the functions with very little error in the end. While my program did a great job and worked well, next time I think I am going to add more command line flags to make the experience easier and precise (i.e. number of times to run each iteration to get even more precise and error-free numbers). I might also add a way for the program to run gnuplot to plot and export the data. There are a lot of way I can improve. In the end, I am very pleased with my results.