

*Runtime Analysis of Programs Using Python***ABSTRACT**

**PURPOSE:** Compare the computational efficiency of binary search to linear search on small and large sorted data sets of integers in order to find the optimal time of implementing each ( $F(n)$ ).

**METHODS:** The two search algorithms were implemented and tested on the randomly generated data sets. Time taken to complete was measure using Python function `timeit`. The most efficient algorithm was determined to be the algorithm that took the smallest amount of time.

**RESULTS:** Sorting first then performing binary search outperformed linear search in every case except for ones with smallest possible list size, and fewest possible target checks (fewest elements in target list).

**CONCLUSIONS:** Both algorithms asymptotically approached the theoretical number of integer comparisons  $O(n)$  for linear search and  $O(n \log n)$  for binary as the size of the data increased. Binary search was found to be more efficient for the relatively large length of lists found in the experiment. However for smaller lists and target lists, linear search is advised.

**INTRODUCTION:** The scientific purpose of this experiment is to determine the runtime differences between differing searching algorithms. Specifically, binary search versus linear search. Despite binary search having a complexity of  $O(\log n)$  and linear search having a complexity of  $O(n)$ , the most obvious assumption of binary search being faster than linear search is not always true. This is because of the need to have a presorted array for binary search, meaning that the time taken to sort the array must also be included in the timing for binary search. The sorting algorithm used has a complexity of  $O(n \log n)$ . This means that depending on the number of elements in the list and how many values are checked in the list, the runtime of linear search may be better than binary and vice versa. The scientific question for this experiment is: In which scenarios of target element list length and list length does binary search outperform linear search. The question was tested using mainly Python.

## METHODS

Python was used as the main vehicle to explore this experiment. First methods were written in python to perform linear and binary search. For linear search, simply iterating through the list and making comparisons and for binary, first a sort function had to be implemented using Python's built in sort() function, next binary search was implemented as making a comparison of the median data, and only searching remaining values lower or smaller than the one compared. In order to test the time taken to complete each search, the Python library timeit was used. The specifications of the test cases done are, 4 binary and linear searches at lengths of lists 1000, 2000, 5000, and 10000, and for number of elements in the target list 10, 100, and 1000. This resulted in a total of 24 timings done. In order to specifically time linear search, using the search function as a parameter of the timeit function for every element in the target list and summing the time taken in total. In order to specifically time binary search, the timeit function was again used but first to time the amount of time to sort the array, then this value was added to the time taken to complete the binary search function for every element in the target array.

## Results

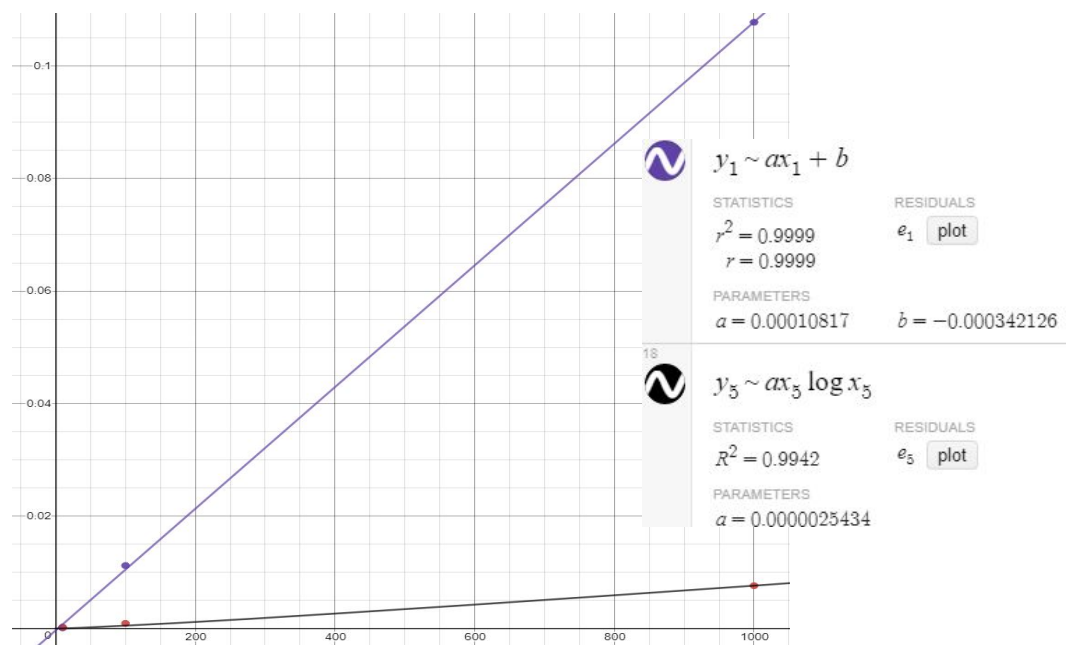
X is number of elements in the target list (k) and y is time taken in seconds.

linear 1k


$x_1$	$y_1$
10	0.000096938
100	0.01118172099999
1000	0.10776333799999

Binary 1k


$x_5$	$y_5$
10	0.00023632600000
100	0.00089344800000
1000	0.00760385800001

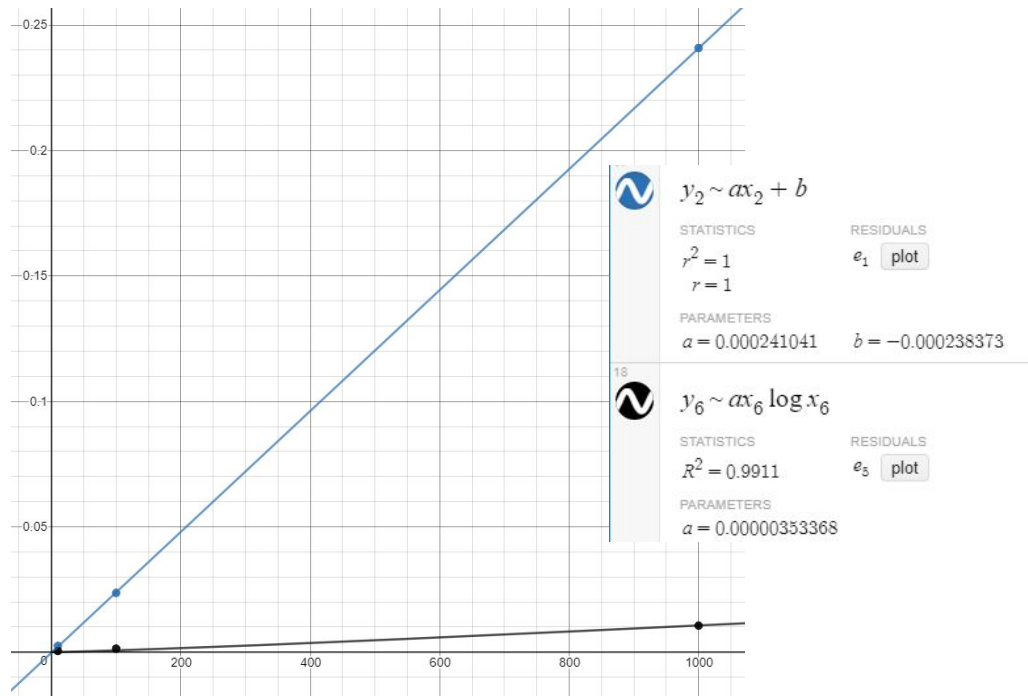


## linear 2k


$x_2$		$y_2$
10		0.00245703500000
100		0.02335222500000
1000		0.24083108500000

## Binary 2k


$x_6$		$y_6$
10		0.00043969900000
100		0.00133467499999
1000		0.01055782199999

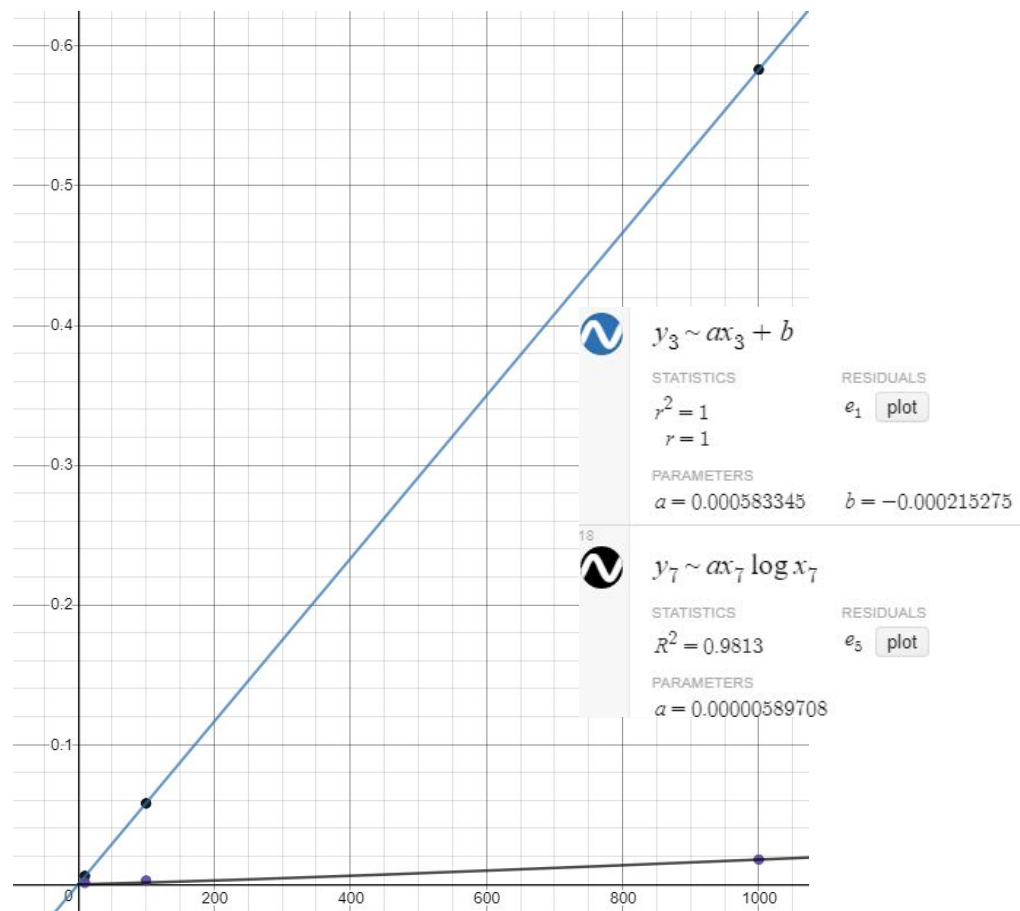


## linear 5k


$x_3$		$y_3$
10		0.00586474399999
100		0.05784801700000
1000		0.58315454899999

## Binary 5k


$x_7$		$y_7$
10		0.00109379299999
100		0.00260164099999
1000		0.01759295900000

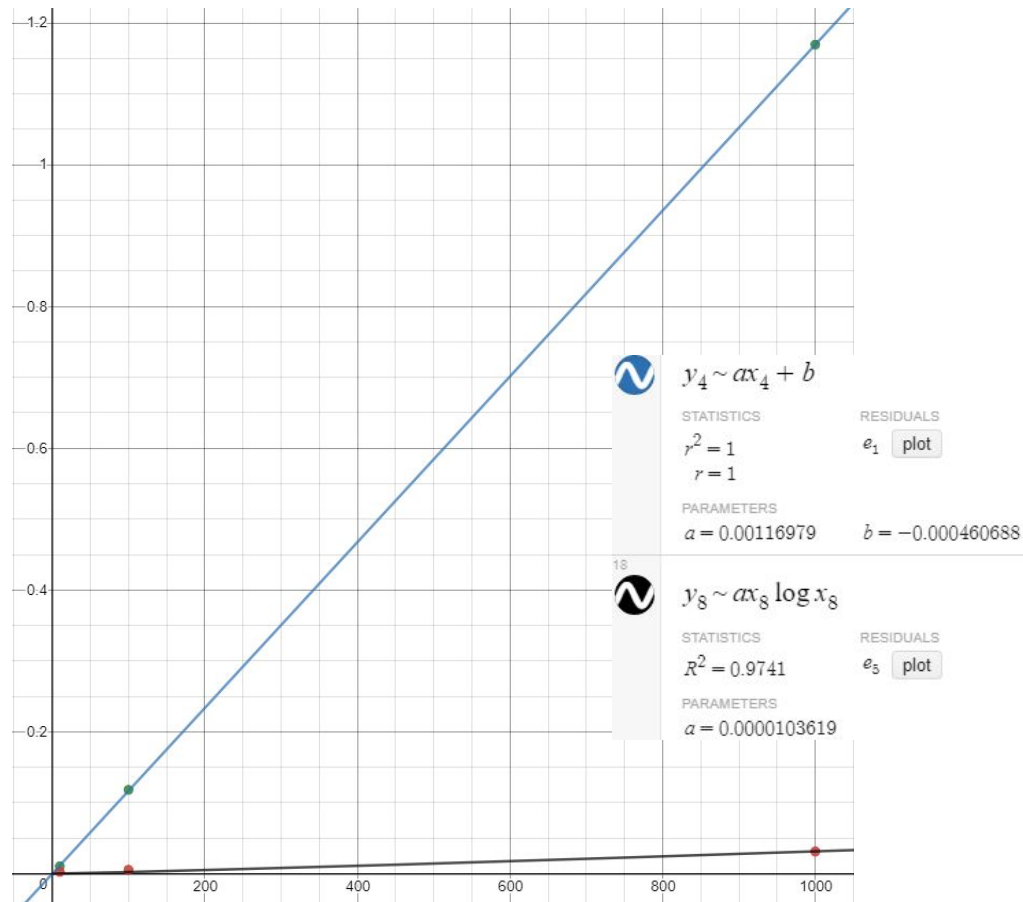


Linear 10k

$x_4$		$y_4$
10		0.01004275100000
100		0.11783267900000
1000		1.16921401000001

Binary 10k

$x_8$		$y_8$
10		0.00229224500000
100		0.00491790800000
1000		0.03088858399998



## DISCUSSION

It was found through experimentation that binary search outperformed the time taken by linear search for almost every test case. The only time linear searched outperformed binary search was when the number of elements in the list was only 1000 (the minimum tested) and when the number of elements in the target array,  $k$ , was only 10. Using this result and specifically the regressions made from the timing data, the  $F(n)$  value for  $n = 1000$  is around  $f(n) = 20$ . These results were not completely as expected, it was thought that linear search would outperform binary for more test cases allowing for a more accurate calculation of  $f(n)$ . Yet, since binary search outperformed linear in every other test case those test are useless in finding  $f(n)$ . However, in general these results conform with the expectations of the experiment as it's clear with every graph and data finding that as the number of elements in the target increase and

<sup>1</sup> <https://www.desmos.com/calculator/7lc0yeoujh> (see for an electronic version of the graphs)

as the number of elements in the original array increase, binary search is exponentially more efficient than linear search. This is demonstrated partially within the graphs above, how the vertical distance between points exponentially increases with every k increase, and partially by how the distance between points increased as the size of the array increased (1k, 2k, 5k 10k).

## REFERENCES

-

