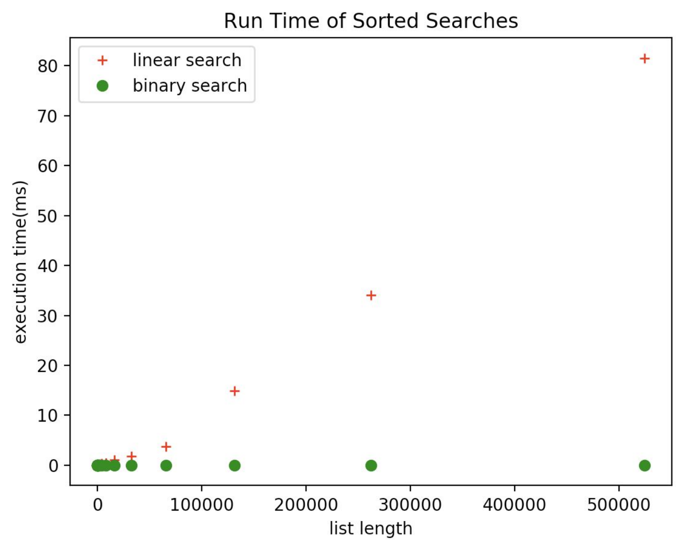
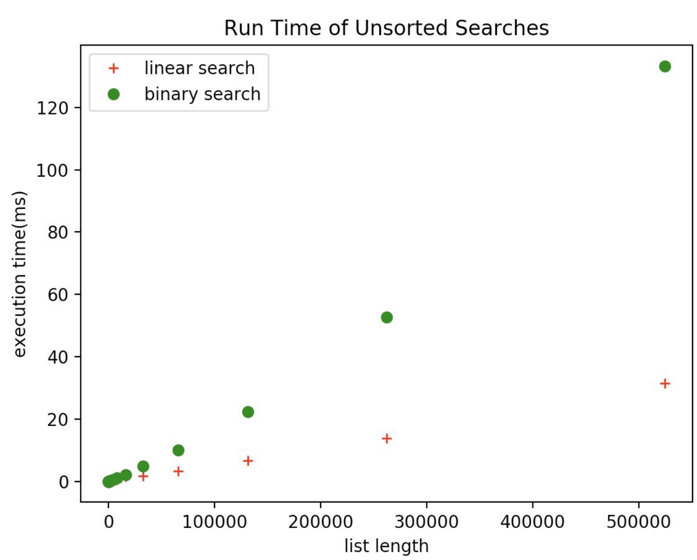
Hannah Mandell
CS51P - Tuesday Lab

A9: Search Performance

The graph generated by plot_comparison when given the results of sorted_comparison as the first argument:



The graph generated by plot_comparison when given the results of unsorted_comparison as the first argument:



A comparison between your experimental results and a big-O analysis of the algorithms you implemented.

Note that both of these comparisons were done under worst-case scenarios.

For the sorted comparison, binary_search was clearly more efficient. Its run time was practically zero for every list length while the run time of linear_search increased quite dramatically as the list length increased. These results line up with the big-O analysis of the algorithms:

- Linear: Is O(n) because it iterates through one for loop once for each element in the inputted list.
- Binary: Is O(log(n)) because after every comparison with the middle term, our searching range gets divided into half of the current range. In order to find the desired value in a list of *n* elements, the function will have to recurse *k* times.
  n (½) ^ k = 1
  Simplifying this equation show that:
  n (1/2^k) = 1
  n = 2^k
  log(base 2)n = k
  You end up with a O(log(n)) function for binary_search.

O(n) is a higher order than O(log(n)) and thus takes longer to compute.

For the unsorted comparison, linear_search was more efficient. As the list lengths increased, linear_search run times increased at a much lower rate than binary_search run times. These results also line up with the big-O analysis of the algorithms:

- Linear: Is O(n) because it iterates through one for loop once for each element in the inputted list.
- Binary: The function binary_search is still O(log(n)). However Python's built in .sort( ) function is also called in this algorithm. It is a merge-sort and is O(nlog(n)). Therefore, the highest level of BigO in the unsorted_comparison binary algorithm is O(nlog(n)).

O(nlog(n)) is more steps than the O(n) of the linear sort and thus takes longer to compute.

A few sentences, grounded in the data you collected, addressing how you would advise someone choosing between using your linear search and your binary search codes.

Linear_search is most efficient if the list(s) in question are unsorted. I would advise someone to use the binary_search only if their list(s) in question is already sorted. Since binary_search can only be used on sorted lists, the list must be sorted before it can be used as a parameter in binary_search. The extra time it takes to sort the list adds to the overall run time, therefore making binary_search slower than a linear_search for an originally unsorted list.