** I try my best to make you understand. Please be mercy
1) SelectionSort

amount running time: O(n^2)

divide list into two parts : 1) sublist that sort(Sorted list) 2) sulist of the rest of item(Unsorted list) keep sorting until the order right

In graph blue line

2) insertionSort

amount running time: O(n^2)

remove 1 element from list and then insert at right index in list that sort and then keep goinh until the order right

In graph red line

3) mergeSort

amount running time: O(nlog(n))

Make a list that starts with the tester being correct (Unsorted list) until N sublists are obtained, each sublist has one element that is tested correctly (Unsorted and N is the number of element array element arrays).
Challenging Merge into the sub-list Targets are formed as sub-items. New potential can be negotiated

In graph green line

4) builtinSort
It’s sort() method
We can use sorted() to create new sorted list

In graph pink line

5) shellsort

amount running time: O(n^2)

Shell Sort is a method that improves Insertion Sort to reduce the number of shuffles. This is because the Insertion Sort method may produce a Worst-Case in extreme cases. at small numerical values push to the back Make the scan find the right location. It takes a very long time The Shell Sort method is improved by sorting subsets. Each subset is called k-sorted, where k is an ever-increasing ordinal number.

In graph yellow line

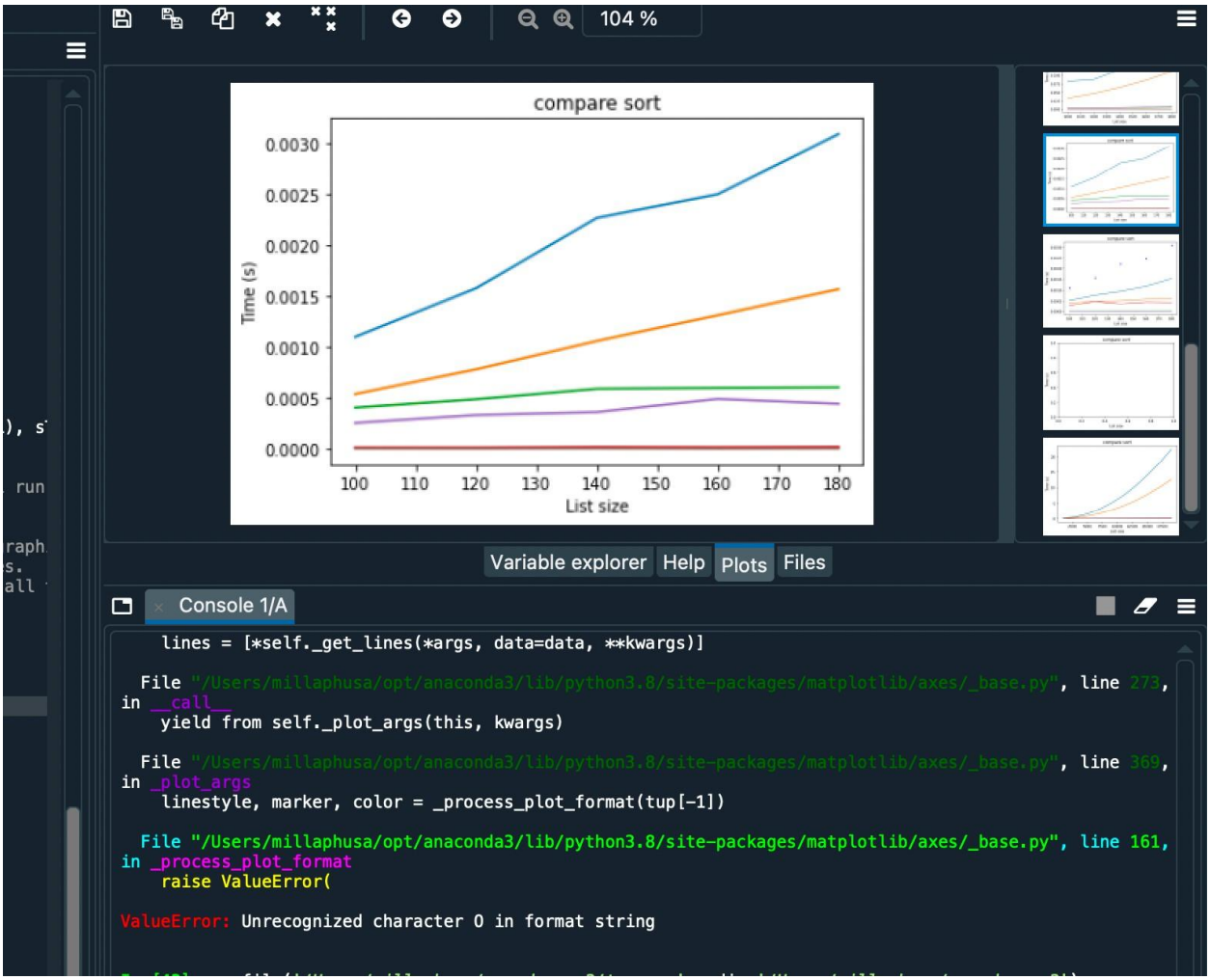
6) quicksort

amount running time: O(log n)

First, it selects one element from the array, which is called a pivot.
Move the element with less than pivot to the left of the pivot and move greater than pivot to the right. This is called partition operation.
Now that we have 2 sublists, go back and repeat steps 1 and 2 of each sublist over and over until they are sorted correctly.

In graph orange line

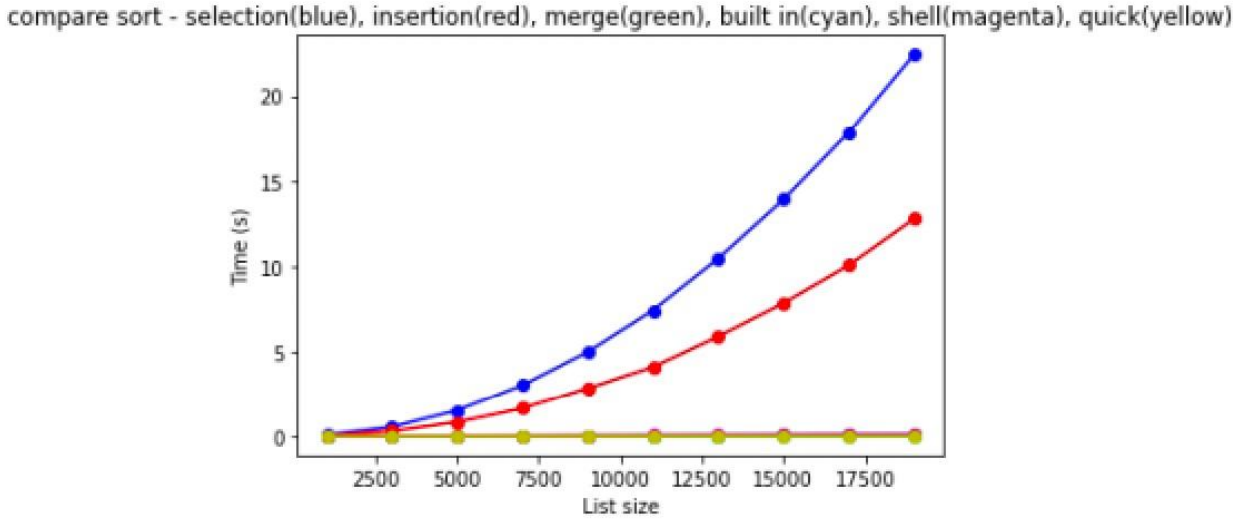
Graph for example



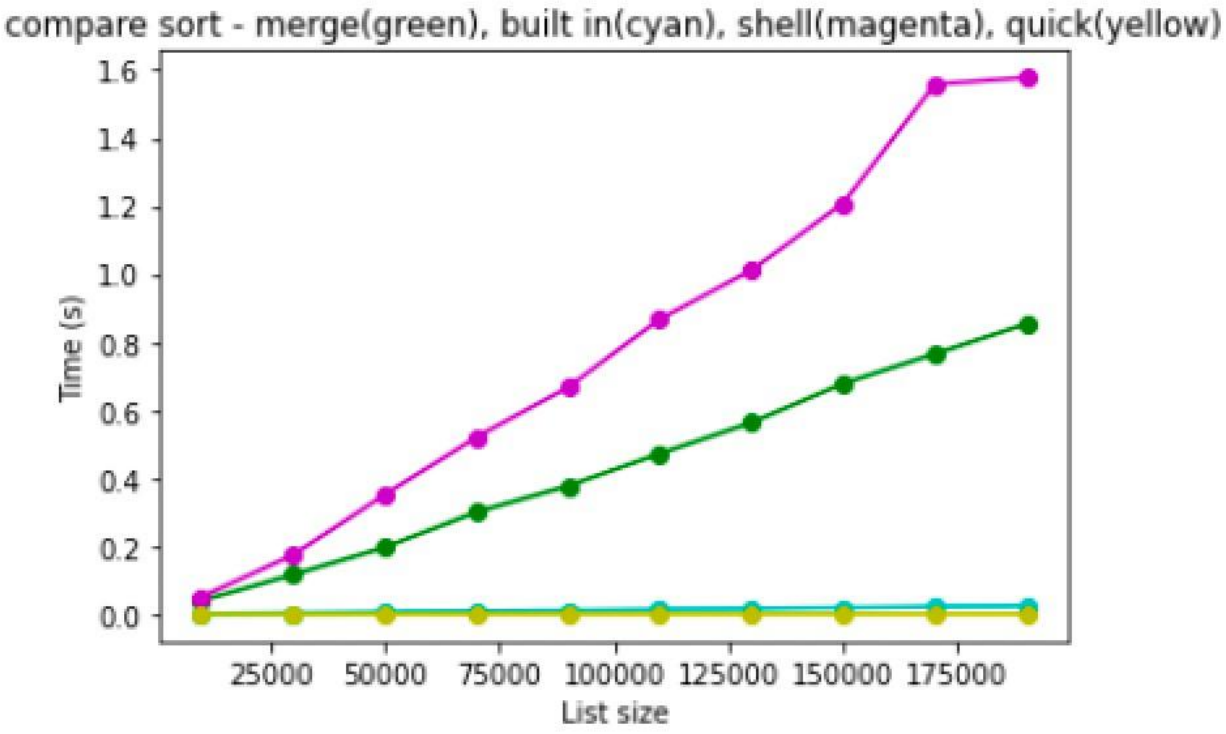
1. Comparison of all sorts on random data. This section should have two charts, one for all the sorts or just the relatively slow sorts, with data up to sizes in the tens of thousands. You won't likely be able to see much difference between the faster sorts on this first chart. The second chart should be just for the faster sorts on data up to sizes of hundreds of thousands or even millions of items.

Random data

1.1 blue is a selection sort. As you can see it is the slowest one. The second slowest is red(insertion). And the rest of them is same



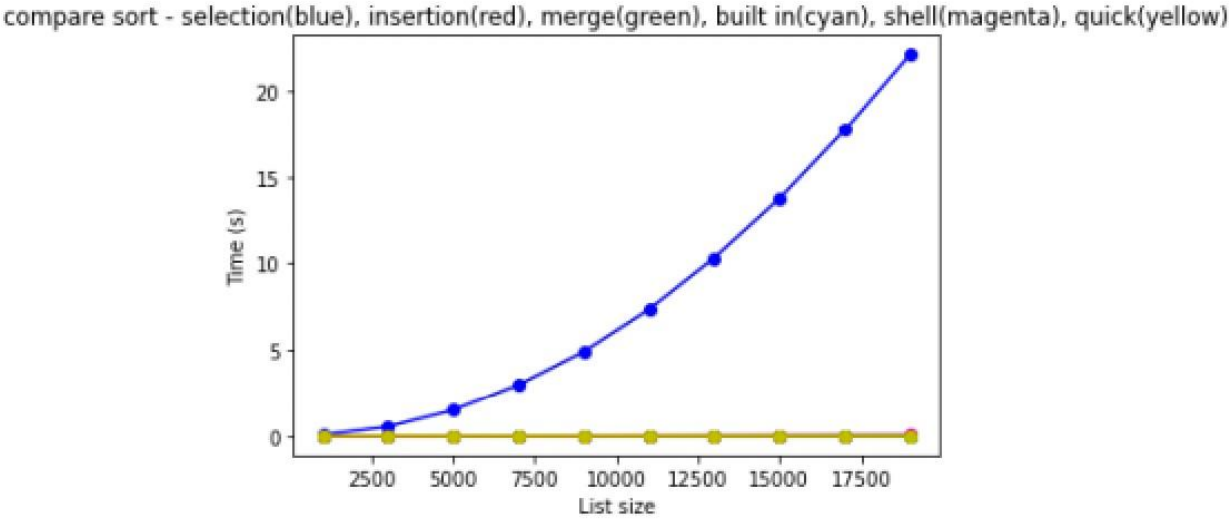
1.2 It is the rest of four sort(sizes of hundreds of thousands) compare



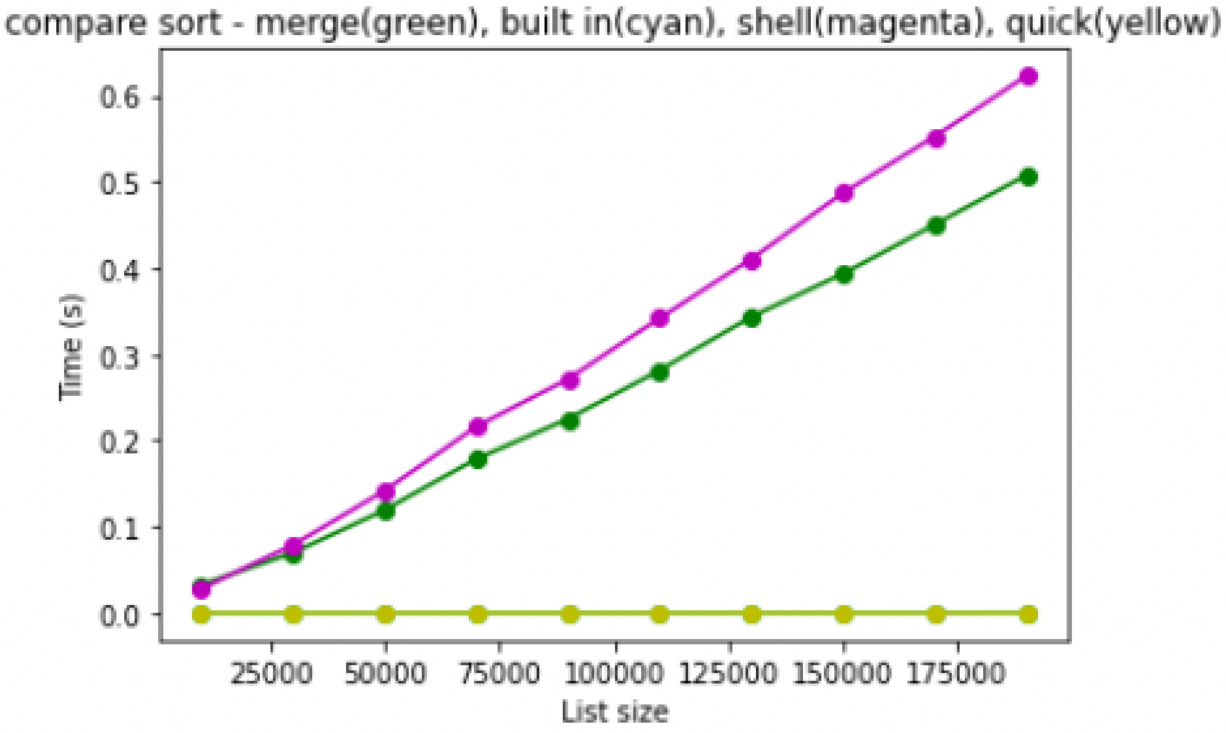
2 Comparison of all sorts on already sorted data. Again, there should be two charts.

Sorted data

2.1 blue is a selection sort. As you can see it is the slowest one. The second slowest is red(insertion). And the rest of them is same



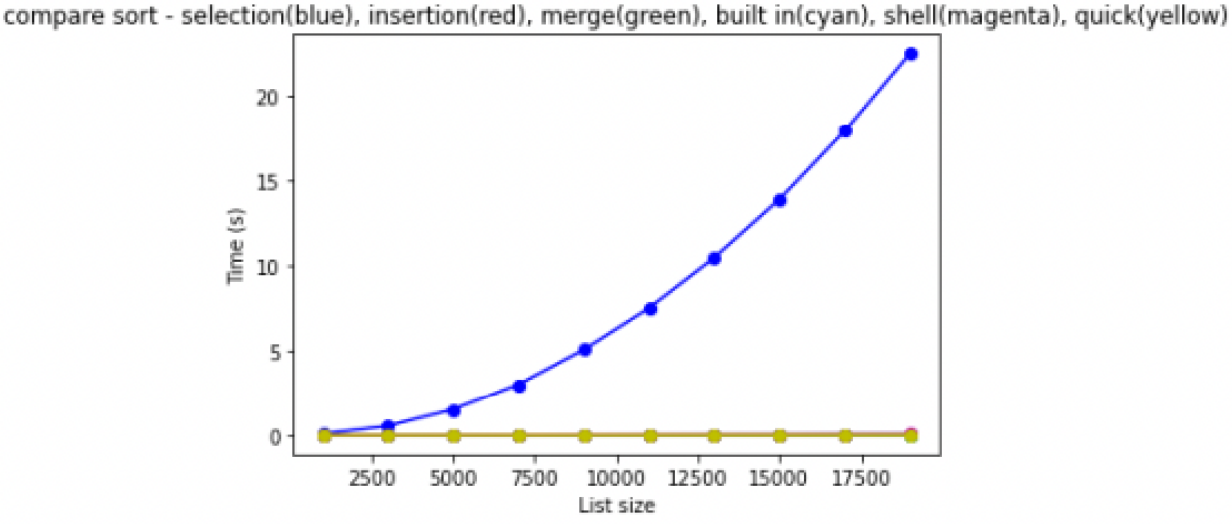
2.2 It is the rest of four sort(sizes of hundreds of thousands) compare



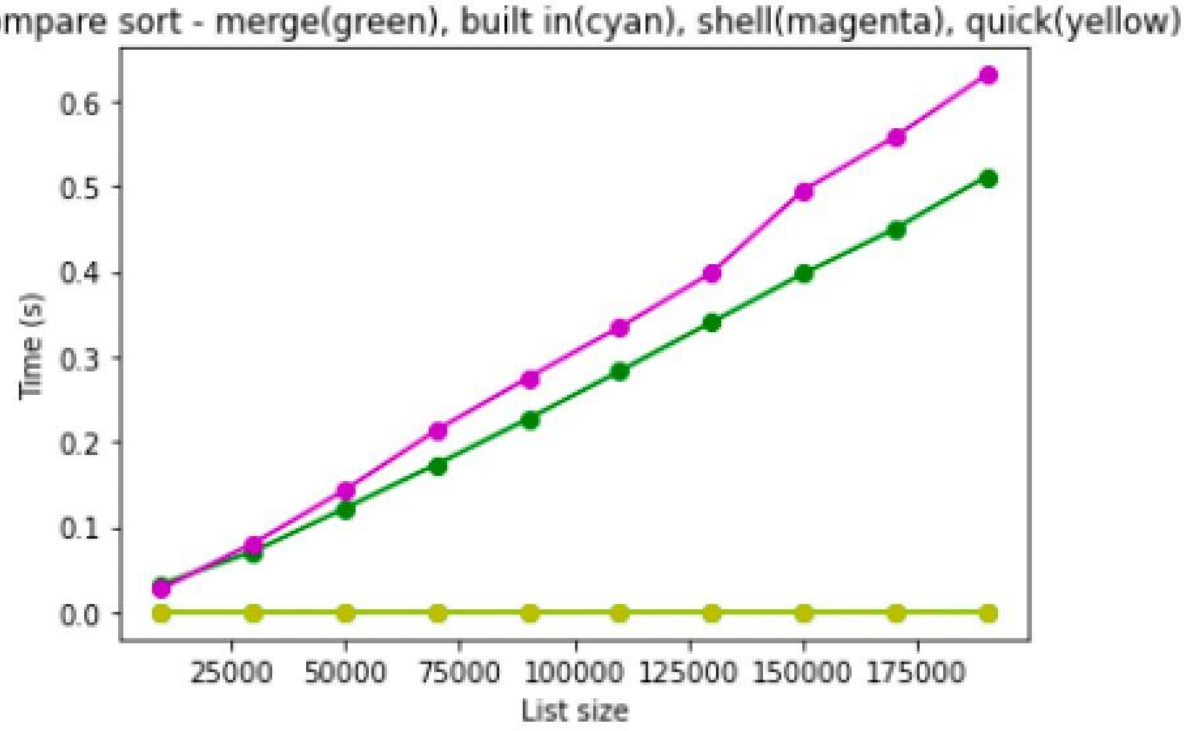
3 Comparison of all sorts on reverse sorted data. Again, there should be two charts.

Reverse sorted data

3.1 blue is a selection sort. As you can see it is the slowest one. The second slowest is red(insertion). And the rest of them is same



3.2 It is the rest of four sort(sizes of hundreds of thousands) compare



Reverse sorted data and sorted data is not that different