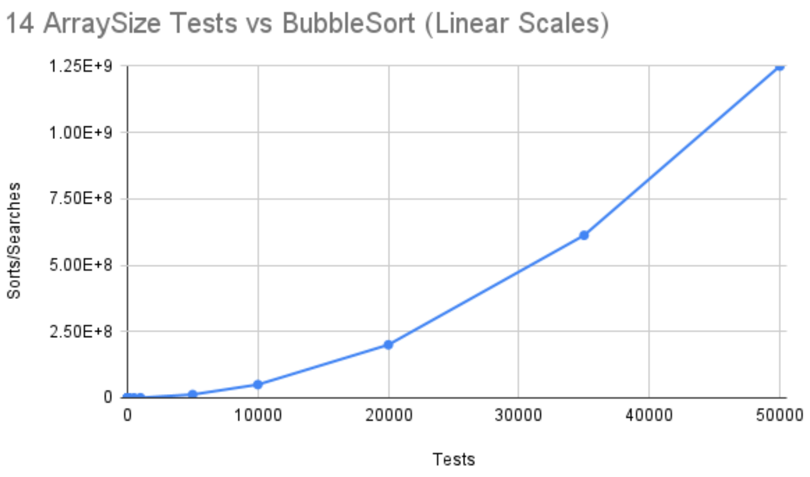
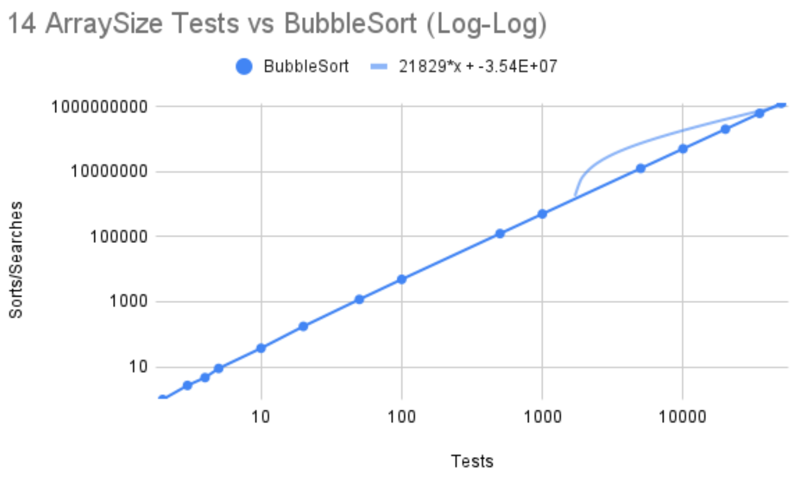
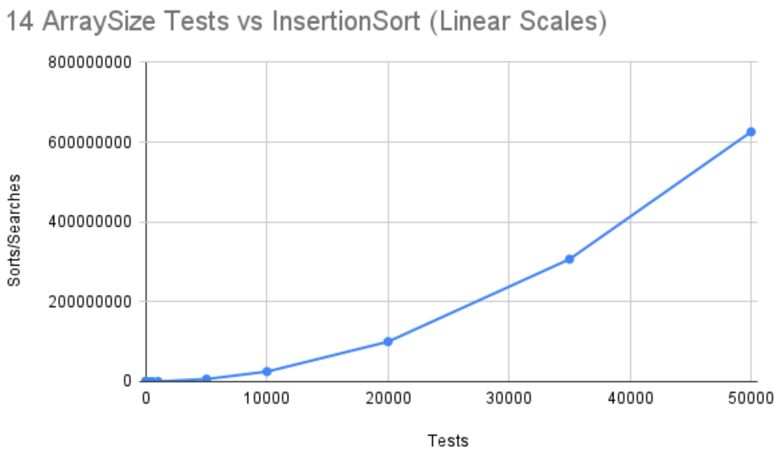
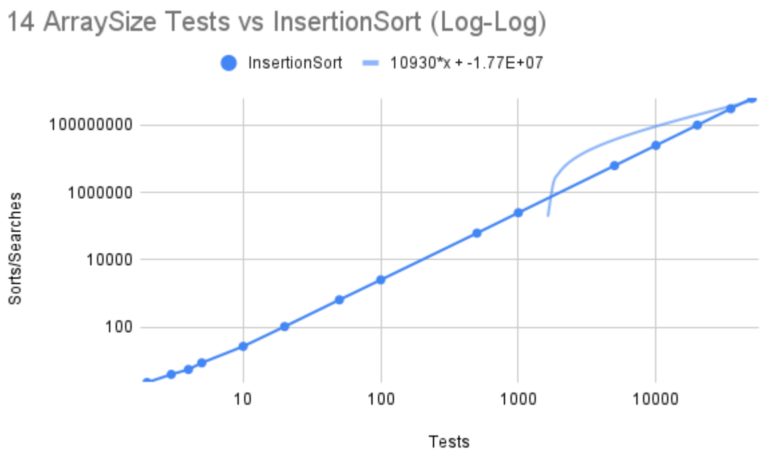
HENRY WANDOVER — DATA STRUCTURES





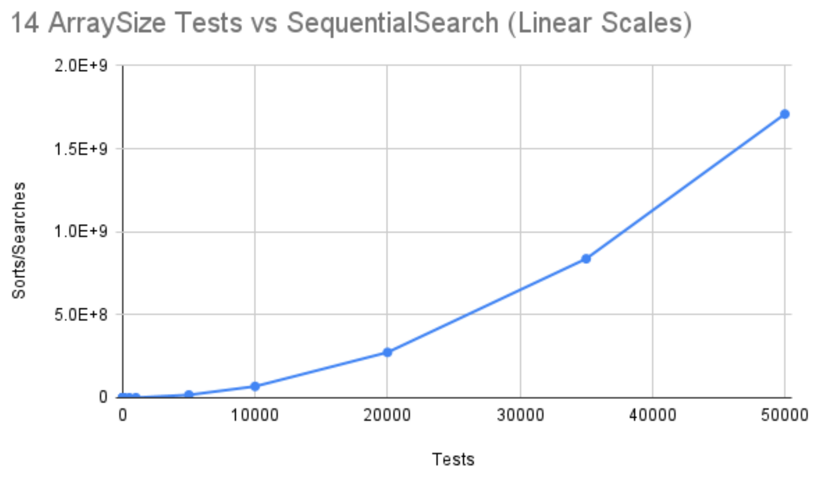
- BubbleSort is a sorting method that moves the larger values to the end of the array (bubble up) which proves useful at smaller arrays, however, at larger arrays, it requires many comparisons and longer runtime.
  - This proved to be the algorithm that produced the largest comparison count
  - Has an exponent of 21,829
- Worst case: N^2
  - Yes, looking at the plots in both there is a slow start and points are closer together because when squared at a smaller N there isn't a huge difference in total—but looking at the linear scales the larger the N, the bigger the jump to the subsequent increase.

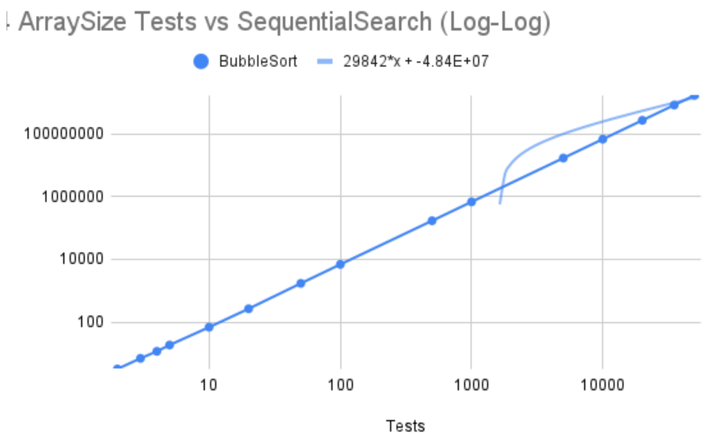




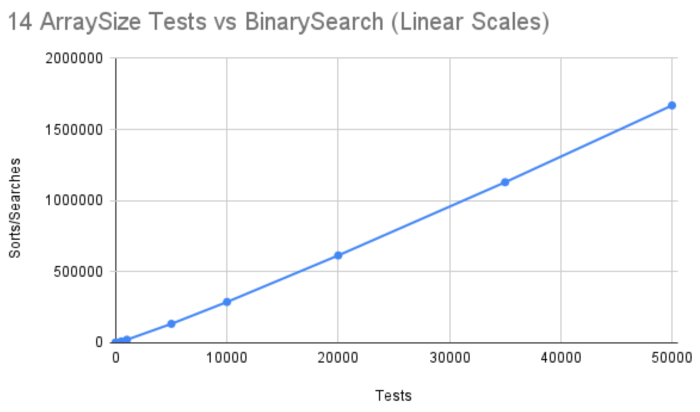
- InsertionSort while also sharing the same worse case was slightly more efficient, it works by creating a sort of subarray that starts from the front and compares one and one, and if the latter is larger then it gets inserted into that subarray in order—akin to shuffling cards.
  - It starts off with more comparisons but doesn't exponentially grow as much as BubbleSort, when reaching 50,000 it has a somewhat significant amount less, as reflected in the graph and averages, it's exponent is also 10,930—half that of the former sort method

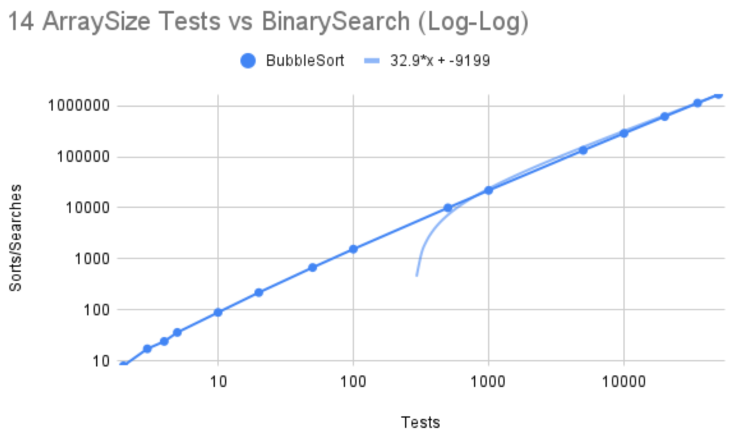
- This method is definitely the superior form of sorting as it has less comparisons and has a smaller exponential growth.
- Worse case: N^2
    - A similar sort of result as the BubbleSort, although it is certainly more efficient in some regard—the worst-case scenario comes out the be the same. The curve on the logarithmic chart is similar to that of BubbleSort.





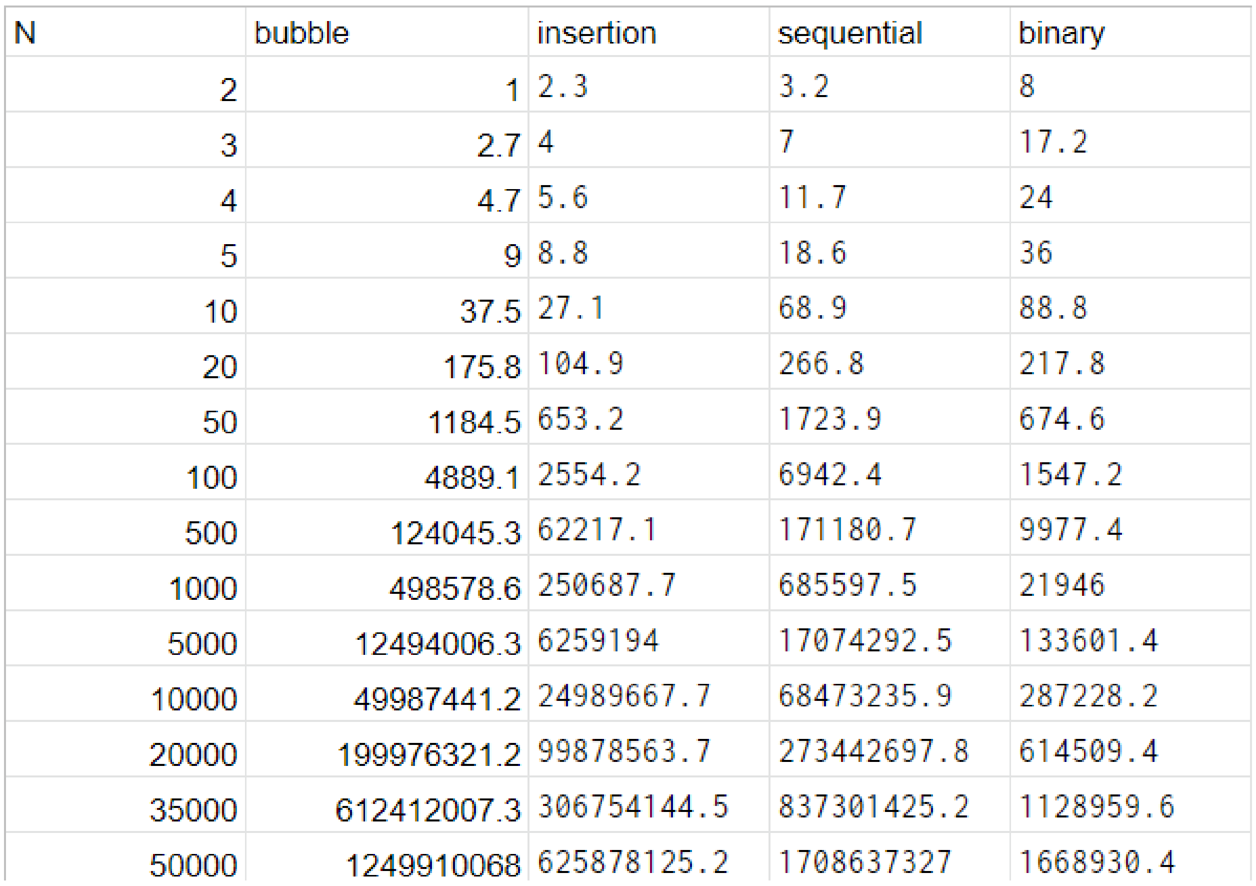
- SeqeuntialSearch works simply by checking every item in an array one by one, start to finish. As reflected in the averages and graph, its comparison amounts reach as high as bubble sort. The exponent is 29,842 which is higher than the bubble itself.
- Worse case: N
    - I had calculated N however the graphs are exhibiting a similar pattern to the previous two sorts, based on my calculations the worst-case scenario would have been something that BinarySearch looks like below—a slower increase model. However, here it is clearly shown that this search method produces an N^2 like worst-case.





- BinanrySearch is not so much more complicated, it splits the array in half and checks if the target is in the first or second half, then enters that and halves again until it isn’t able to anymore. Making it more efficient than the previous example, reflected in its curve of 32.9, showing that there isn’t a large jump from point to point as array size increases like the other search.
- N
    - For the final search and algorithm, I calculated N and the linear scales graph proves my point of a slower increase with the upscaling of N. Unlike the previous search, this one was proven to N in math and through testing.

- Finally, based on my testing I’d say that no it’s not. Looking at the averages we can identify that after the N is at 100 BinanrySearch is then better at its job. Before that with smaller increments, it makes significantly more comparisons than SequentialSearch. And as stated before at 100 there are 5000 fewer comparisons already and since sequential isn’t exactly N in practicality, there are large jumps that binary doesn’t make, proven by its worse case not being as bad. So no not all, smaller array sizes its not but large ones it is superior by a long shot.



N	bubble	insertion	sequential	binary
2	1	2.3	3.2	8
3	2.7	4	7	17.2
4	4.7	5.6	11.7	24
5	9	8.8	18.6	36
10	37.5	27.1	68.9	88.8
20	175.8	104.9	266.8	217.8
50	1184.5	653.2	1723.9	674.6
100	4889.1	2554.2	6942.4	1547.2
500	124045.3	62217.1	171180.7	9977.4
1000	498578.6	250687.7	685597.5	21946
5000	12494006.3	6259194	17074292.5	133601.4
10000	49987441.2	24989667.7	68473235.9	287228.2
20000	199976321.2	99878563.7	273442697.8	614509.4
35000	612412007.3	306754144.5	837301425.2	1128959.6
50000	1249910068	625878125.2	1708637327	1668930.4