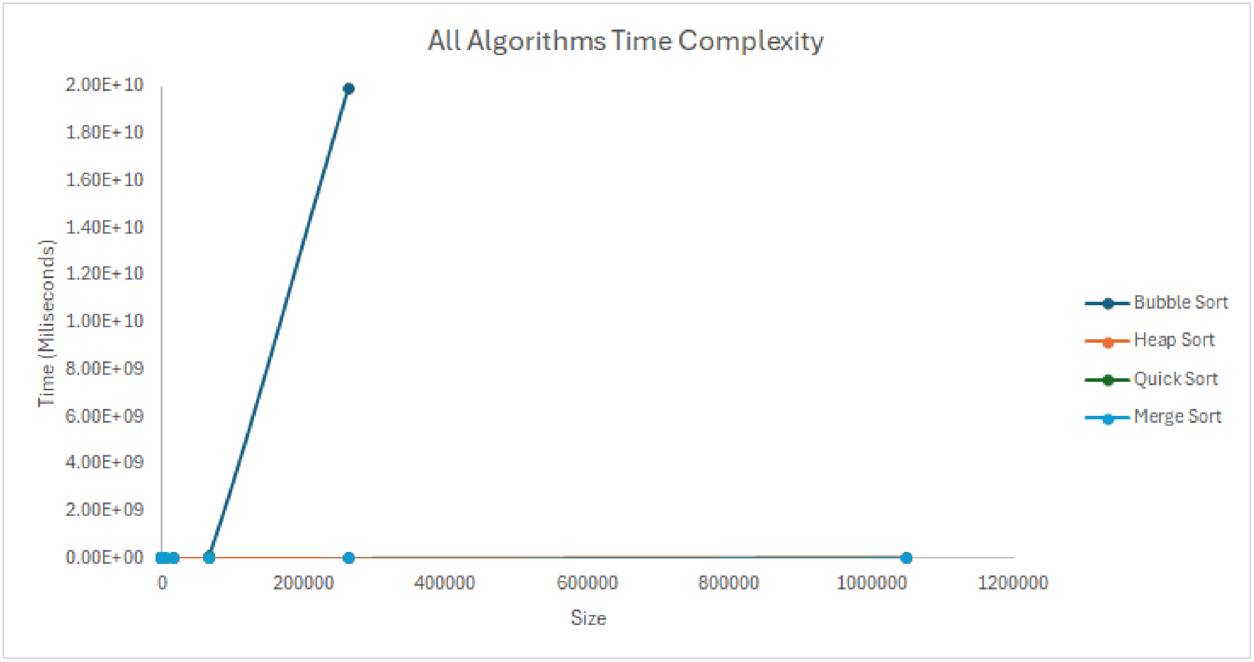
Introduction:
The purpose of this experiment is to implement various sorting algorithms and observe both the time and space complexity held by each algorithm, as its size gets increasingly larger.

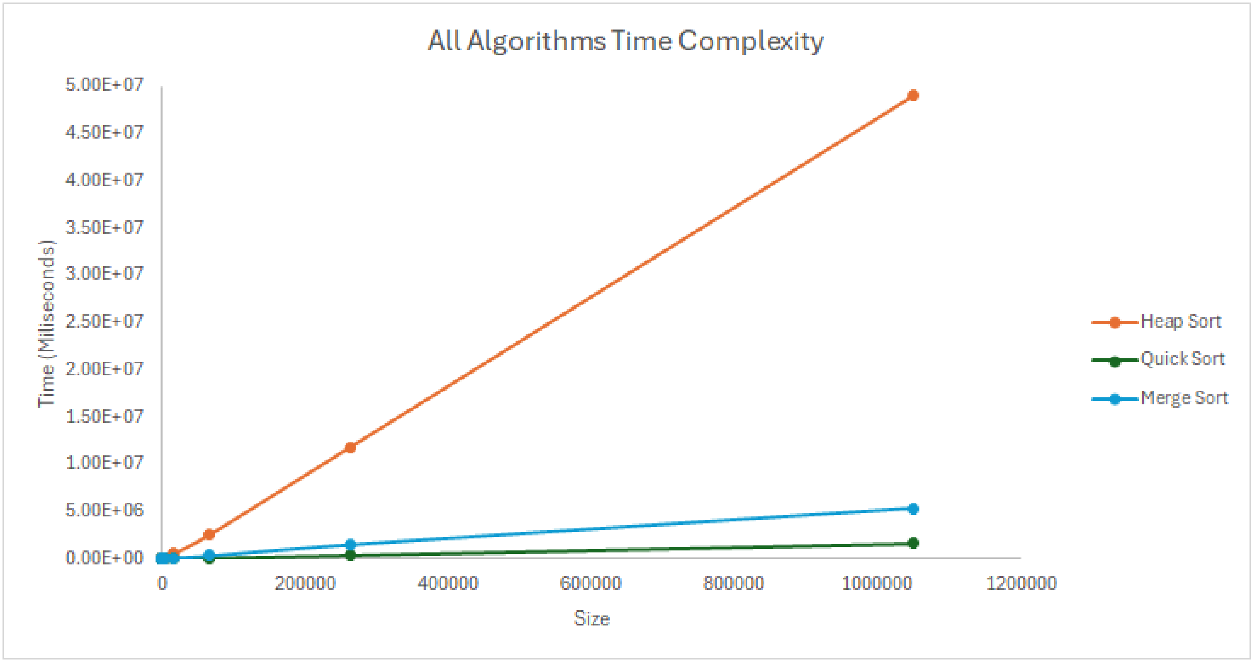
Theoretical Analysis:

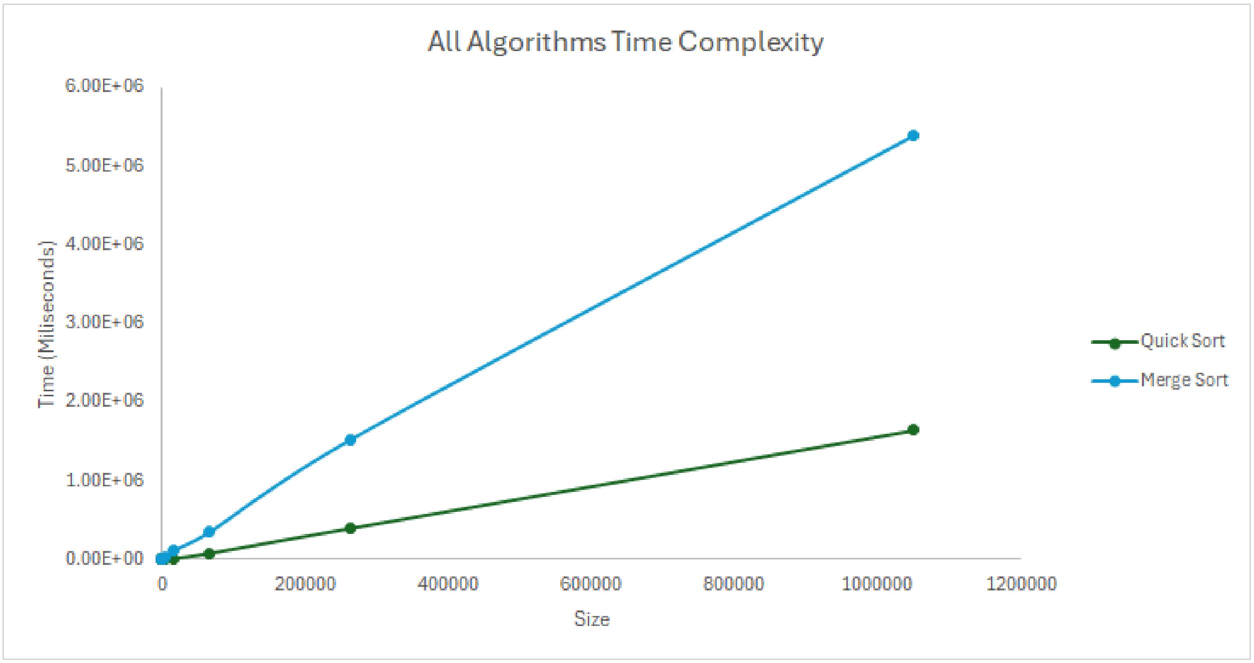
- In theory, Bubble Sort should easily take the longest, due to its O(n^2) average complexity. It can only reach best case of O(n) when the list is already sorted, which isn't possible in a random list. This complexity is mainly caused by the double for loop that bubbles each element in the list to on average the list's end.
- Quick sort should be the fastest on average, due to the lack of extra space needed to sort. The algorithm is in-place sorting, meaning there are fewer memory writes and faster memory access. One thing to note is that it's worst case can be O(n^2) if poor pivot points are chosen.
- Merge sort would likely be second fastest to quick sort because of its stability and divide-and-conquer approach. Though it requires extra memory writing, we are able to take special liberties in sorting due to how the algorithm works (ex: being able to simply add all remaining elements to the end of of a list being sorted if we already got the first hald sorted).
- Heap sort would likely be second slowest due to it having to main the properties of a heap. Though it runs in nlogn speed like the other two above algorithms, many more conditions have to be true before sorting is allowed to happen (to maintain its integrity as a heap). More code executions inevitably means slower runtime.

Experimental Setup:
I experimented by making 4^k random elements and adding them to a list to be sorted, where k is the exponent. I did a loop that increased K up to 9 for bubble sort, and 10 for all other sorts. I collected the time taken to sort the algorithms and plotted them on a graph for analysis.

Experimental Results:


All Algorithms Time Complexity
Time (Miliseconds)
2.00E+10
1.80E+10
1.60E+10
1.40E+10
1.20E+10
1.00E+10
8.00E+09
6.00E+09
4.00E+09
2.00E+09
0.00E+00
0
200000
400000
600000
800000
1000000
1200000
Size
Bubble Sort
Heap Sort
Quick Sort
Merge Sort


All Algorithms Time Complexity
Time (Miliseconds)
5.00E+07
4.50E+07
4.00E+07
3.50E+07
3.00E+07
2.50E+07
2.00E+07
1.50E+07
1.00E+07
5.00E+06
0.00E+00
0
200000
400000
600000
800000
1000000
1200000
Size
Heap Sort
Quick Sort
Merge Sort



- Quick sort easily performed the best of all the algorithms, while bubble sort performed the worst.
- Bubble sort was the worst naturally because of its O(n^2) average case, which is unlike the other algorithm's O(nlogn) average case.
- Quick sort was the fastest due to not needing as many memory writes or conditions to be satisfied before completing its loop, as well as good pivot points being chosen throughout.
- Merge sort was slower than Quick due to its need to create and manage extra memory, and Heap even slower because of it needing to manage heap properties.
- Certain conditions might make algorithms run faster than others. For example, Quick sort can be the worst algorithm if bad pivots are chosen every time, and Bubble sort to be the best algorithm if the random list is already sorted.
- If you know certain things about the array, sometime's its best to use another algorithm over one to expedite the process. For example, if you know the list is already sorted, or majority sorted, it might be best to do bubble sort instead, or merge sort.

- The experimental results agreed with my theoretical analysis, though from the looks of it some algorithm's time complexities are being multiplied by a constant scalar instead of being the same slope throughout. This can be attributed to the nuances that come with how each sorting algorithm is created.
- Another interesting thing is that Quicksort seems slightly more smooth than Merge sort in complexity. Though this is a minuscule detail, it might mean that there is something in my code that could be optimized or the nuances of merge sort (such as memory creation/deletion) is affecting the way the algorithm grows.
- Other than that, I don't see any major discrepancies.