```
jcabigas@ecc-linux:~/CS311/P1$ ./project_1

Vector 1:  |bubble done| |insert done| |selection done| |quick done|

Vector 2:  |bubble done| |insert done| |selection done| |quick done|

Vector 3:  |bubble done| |insert done| |selection done| |quick done|

Vector 4:  |bubble done| |insert done| |selection done| |quick done|

Vector 5:  |bubble done| |insert done| |selection done| |quick done|

Vector 6:  |bubble done| |insert done| |selection done| |quick done|

Vector 7:  |bubble done| |insert done| |selection done| |quick done|

Vector 8:  |bubble done| |insert done| |selection done| |quick done|

Vector 9:  |bubble done| |insert done| |selection done| |quick done|

Vector 10:  |bubble done| |insert done| |selection done| |quick done|
-----------------------------------------------
Bubble sort on 10 vectors: Successful
Minimum: 0.000109101
Maximum: 0.000155603
Mean: 0.000124132
Standard Deviation: 1.48767e-05
-----------------------------------------------
Insertion sort on 10 vectors: Successful
Minimum: 2.22e-05
Maximum: 3.0801e-05
Mean: 2.50205e-05
Standard Deviation: 2.88215e-06
-----------------------------------------------
Selection sort on 10 vectors: Successful
Minimum: 4.17e-05
Maximum: 5.4101e-05
Mean: 4.48605e-05
Standard Deviation: 4.10864e-06
-----------------------------------------------
Quicksort on 10 vectors: Successful
Minimum: 0.000130502
Maximum: 0.000163202
Mean: 0.000140772
Standard Deviation: 1.01453e-05
jcabigas@ecc-linux:~/CS311/P1$
```

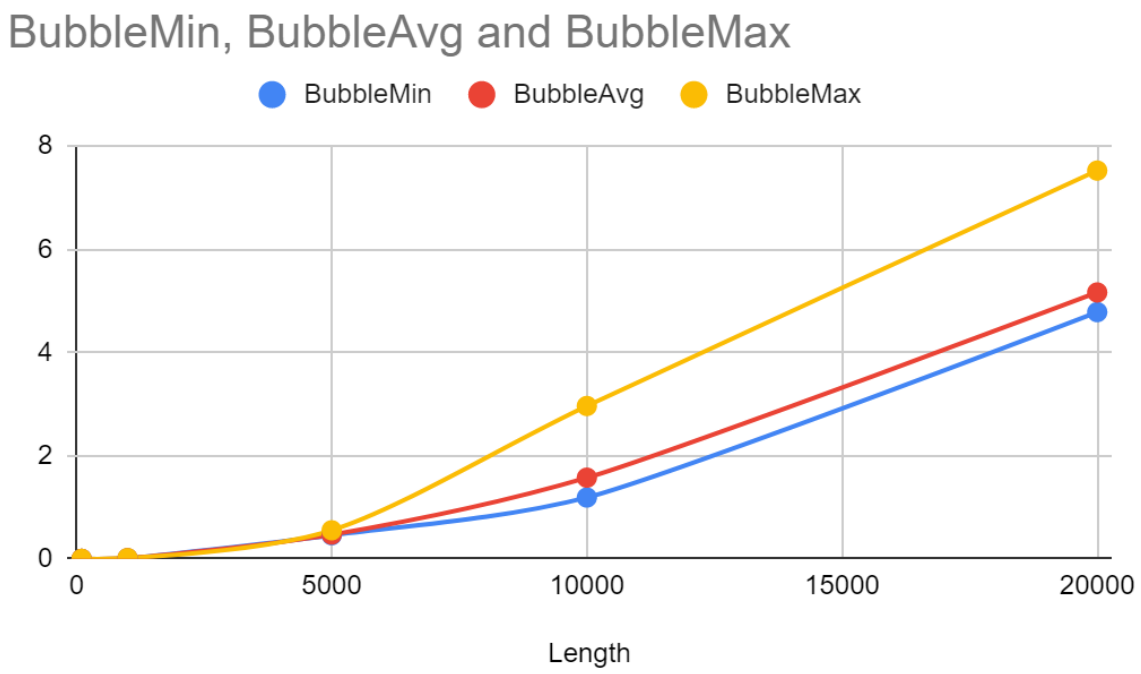
First phase output

The array lengths I chose were 100, 1000, 5000, 10,000, 50,000. I decided to modify the output format in my project so it would save the times to a file in csv format. The plan was to turn a csv into a spreadsheet.

The resulting spreadsheet is below. Using this data, I graphed each algorithm independently.

Length	BubbleMin	BubbleAvg	BubbleMax	InsertionMin	InsertionAvg	InsertionMax	SelectionMin	SelectionAvg	SelectionMax	QuickMin	QuickAvg	QuickMax
100	0.000110402	0.000139224	0.000210503	1.83E-05	2.18E-05	3.60E-05	4.13E-05	4.34E-05	6.79E-05	0.000131102	0.000146414	0.000231704
1000	0.0113334	0.0117705	0.0149449	0.0017397	0.0018447	0.0031395	0.0034792	0.003534	0.0054848	0.0014676	0.0015135	0.00159332
5000	0.458362	0.470571	0.554007	0.0822507	0.0859201	0.0893244	0.14821	0.152573	0.218713	0.0122103	0.0128415	0.0196412
10000	1.18914	1.57343	2.96262	0.17841	0.252056	0.497952	0.339187	0.480291	0.874646	0.0163691	0.0221518	0.0442799
20000	4.78197	5.16282	7.52489	0.719199	0.827528	1.39059	1.36575	1.52713	2.51205	0.0335832	0.0374243	0.0619286

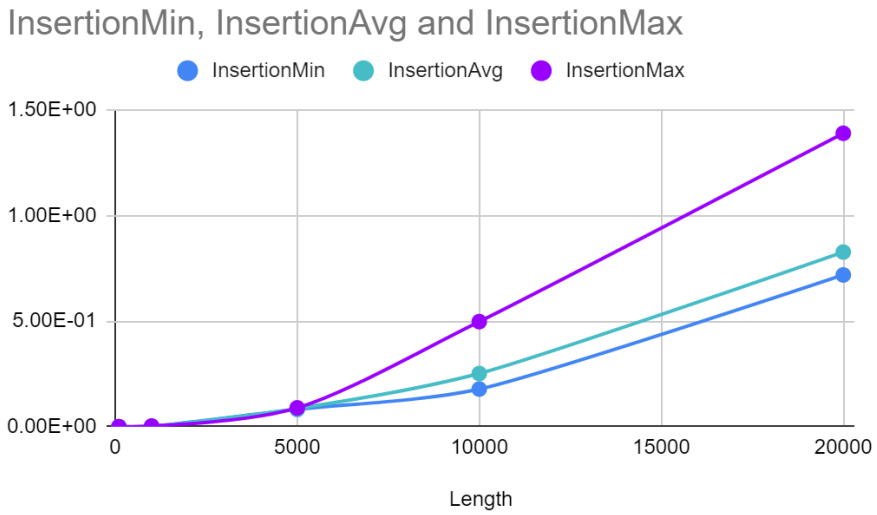
Bubble Sort Graph



Theoretically, the best case for Bubble Sort is an already sorted list. In this case its time complexity is O(n). The worst case is a completely reversed list, in which its time complexity is $O(n^2)$. The average case for Bubble Sort also happens to be $O(n^2)$. This is because of the nested loop structure of the algorithm. Unless the inner loop isn't actually doing anything (a presorted list for instance) the nested loops create a time complexity of $O(n^2)$. Much like the algorithm that birthed it, the curves of the graph are very linear, and also very large time-wise.

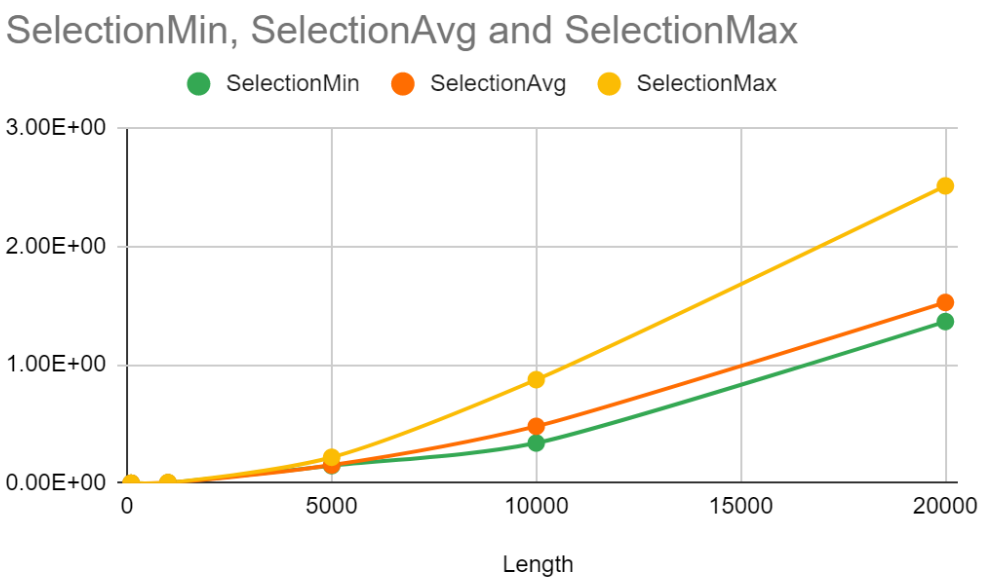
We can see the growth of the time, and if we were to imagine the x value growing towards infinity, an n^2 growth can be seen.

Insertion Sort Graph



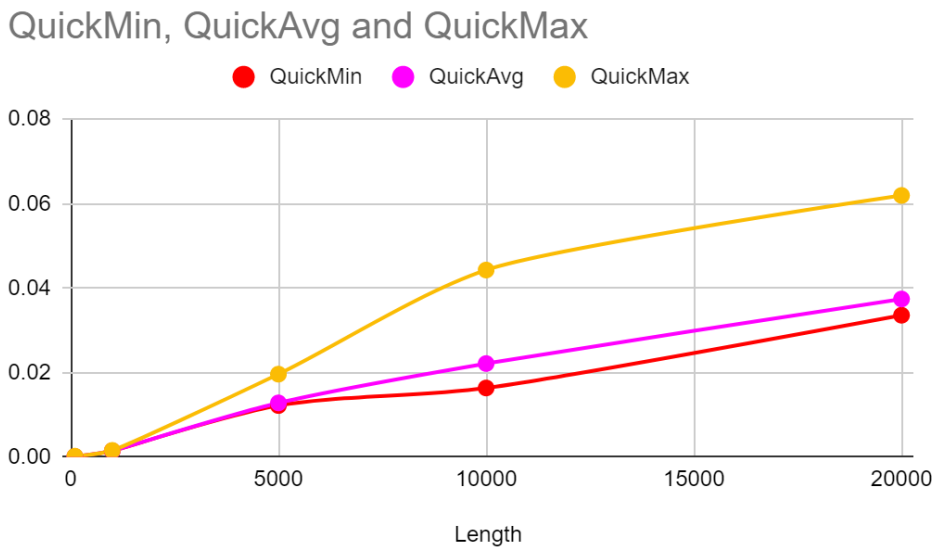
The time complexity for the best case of Insertion Sort is very similar to that of bubble sort, in the worst case it is $O(n^2)$, and in the best case it is $O(n)$. The average case is yet again $O(n^2)$, all for the same reasons as Bubble Sort. Insertion and bubble sort yet again show similarities, this time in appearance. The likelihood of generating a sorted list at random is incredibly low, so what we see is three different n^2-esque curves.

Selection Sort Graph



Selection Sort is where the time complexities start to take a turn. The time complexity for selection sort is $O(n^2)$ in the best, worst, and average cases. This is because the inner loop has no mechanism for ending early, and will always check every value. As expected, the curves demonstrate this behavior, as they follow the same general pattern.

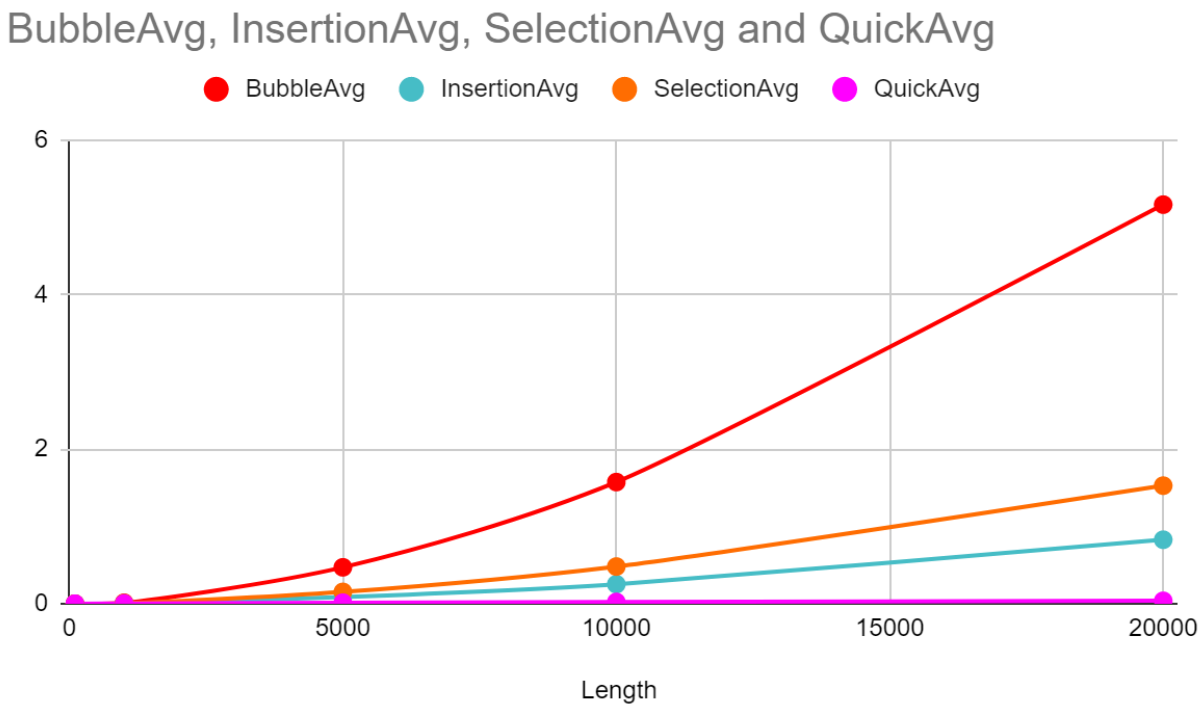
Quicksort Graph



Quicksort is the only recursive algorithm of the group. Because of this, it is also the fastest on average. Its best case occurs when the pivot happens to be the exact middle element of the list, allowing the recursion to end as soon as possible. This best case scenario, as well as the average, is O(nlog(n)). The worst case is $O(n^2)$. The graph seems to follow the expected growth, the worst case being closer to O(n) than the rest, and the average and minimum curves being

relatively similar. The reason we don't see the maximum become a curve resembling n^2 is because the likelihood of choosing the beginning or end of a list becomes increasingly low the more random elements there are.

Averages of all algorithms:



I included this graph just to visualize the differences between the algorithms. When looking at each of the graphs, they all seem to exhibit the same general curve, but when compared to each other directly, it becomes clear that it is not the case. Had I been able to zoom in more with the y-axis, we would see a more logarithmic curve for quicksort compared to the rest.