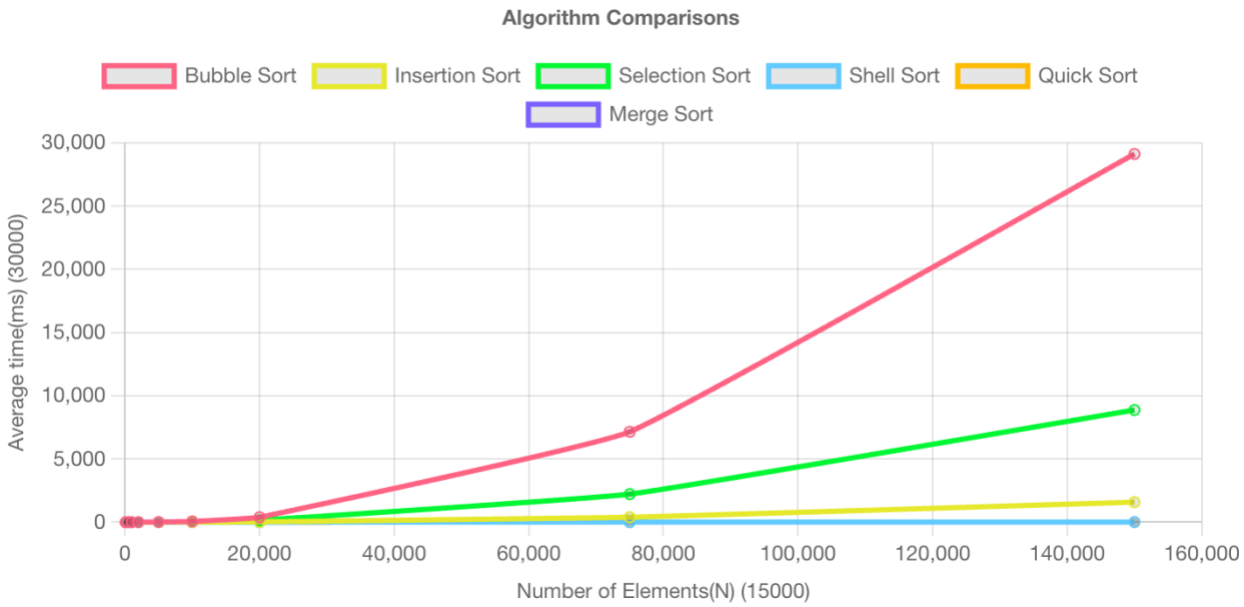
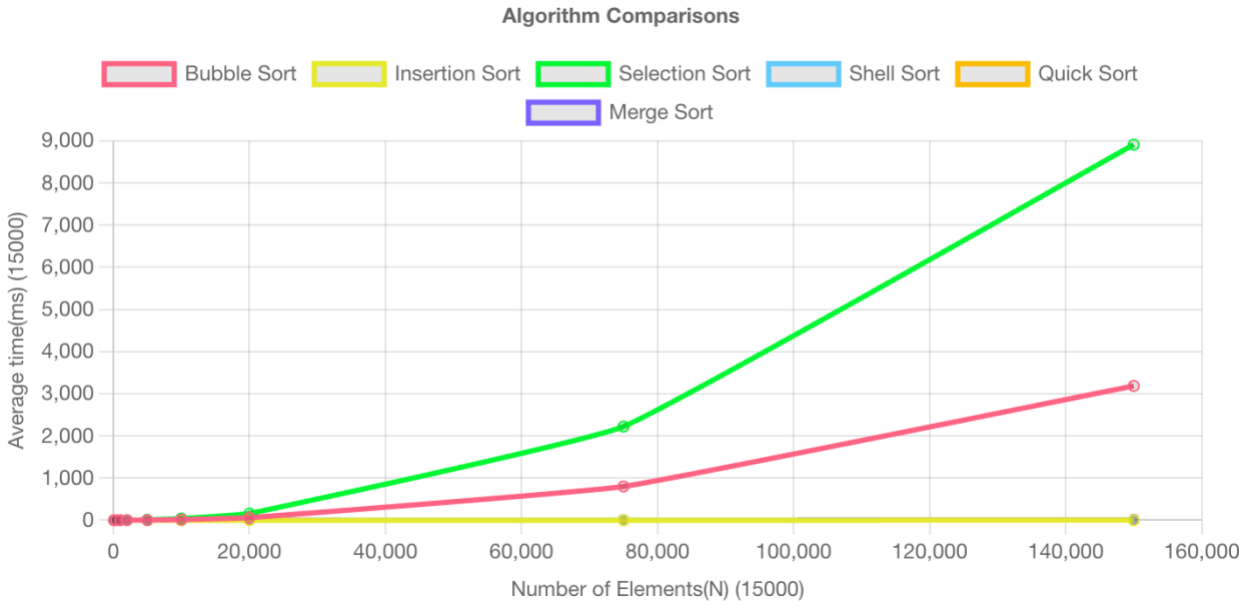


10. After running my code, it took around 25 minutes which I got scared because I thought my computer was bugging, but then I saw on the instructions to be patient. At first all of them seem to be done at the same time or close enough where they are nearly identical. We can really see the differences between each space complexities and the ones we “cut” off (like their constants and focus on fastest growing) are important in the bigger stages like Bubble Sort and Insertion sort even though they are both O(n^2) they are different. They ended up being accurate as I ranked them. I was kind of worried about Shell Sort though as it’s one that can be trickier or at least for me to decide where it goes.



12. After running the algorithms this time, I saw that most cut their average time by a lot. Bubble sort drastically improved from 29108 the first time to 3184. Insertion sort was also improved by a lot. I think because with random data it hit its worst case scenario but with the kperformamce it hit just O(n). There were improvements on everyone except Selection sort because it always does $O(n^2)$ comparisons regardless of input order. It ended up going even more. It ended up being that sorting them this way than random the quadratic algorithms benefited the most except selection.

4. [15, 14, -6, 10, 1, 15, -6, 0]
G = 8/2 = 4
C: 15 1, 14 15, 10 0
[1, 14, -6, 0, 15, 15, -6, 10]
G = 4/2 = 2
C: 1 -6, 14 0, -6 15, 0 -6
[-6, 0, -6, 1, 14, 10, 15, 15]
G = 2/2 = 1
C: 0 -6, -6 0, -6 1, 1 14, 14 10, 14 15, 15 15
[-6, -6, 0, 1, 10, 14, 15, 15]

5. ~~1. Quick Sort - O(nlogn)~~
~~2. Merge Sort - O(nlogn)~~
~~3. Heap Sort - O(nlogn)~~
~~4. Shell Sort - O(nlogn) BUT can be O(n²)~~
~~5. Insertion Sort - O(n²)~~
~~6. Bubble Sort - O(n²)~~

5. 1. Quick Sort O(nlogn) - Least Computer memory
2. Merge Sort O(nlogn) - Uses extra space
~~3. [illegible]~~
3. Shell Sort O(nlogn) - not consistent can be O(n²)
4. Insertion Sort O(n²) - fast because moves elements directly to correct position
6. Bubble Sort O(n²) - slowest b/c swaps adjacent elements repeatedly
5. Selection Sort O(n²) - O(n²) comparisons but O(n) swaps

Assignment 2 - Sorting Algorithms

1. [1, 25, 31, 16] & [-3, 0, 16, 27] but actually [1, 16, 25, 31] & [-3, 0, 16, 27]
   C = compare [ ] = result
   - C: -3 & 1 → [-3]
   - C: 1 & 16 → [-3, 0, 1]
   - C: 25 & 27 → [-3, 0, 1, 16, 25]
   - Add 31 → [-3, 0, 1, 16, 25, 27, 31]
   - C: 0 & 1 → [-3, 0]
   - C: 16 & 16 → [-3, 0, 1, 16]
   - C: 31 & 27 → [-3, 0, 1, 16, 25, 27]

2. [-1, -5, 67, -10, 21, 8, 4, 1] = original
   - [-5, -1, 67, -10, 21, 8, 4, 1]
   - [-5, -1, -10, 67, 21, 8, 4, 1]
   - [-5, -10, -1, 67, 21, 8, 4, 1]
   - [-10, -5, -1, 67, 21, 8, 4, 1]
   - [-10, -5, -1, 67, 8, 21, 4, 1]
   - [-10, -5, -1, 8, 67, 21, 4, 1] → [-10, -5, -1, 8, 21, 67, 4, 1]
   - [-10, -5, -1, 8, 21, 4, 67, 1]
   - [-10, -5, -1, 8, 4, 21, 67, 1]
   - [-10, -5, -1, 4, 8, 21, 67, 1]
   - [-10, -5, -1, 4, 8, 21, 1, 67]
   - [-10, -5, -1, 4, 8, 1, 21, 67]
   - [-10, -5, -1, 4, 1, 8, 21, 67]
   - [-10, -5, -1, 1, 4, 8, 21, 67]

3. [-5, 42, 6, 19, 11, 25, 26, -3]
   P = -3
   [-5] -3 [42, 6, 19, 11, 25, 26]
   [-5, -3] P = 26
   [6, 19, 11, 25] 26 [42]
   P = 25 [26, 42]
   P = 11 [6, 19, 11] 25
   [6] 11 [19]
   [-5, -3, 6, 11, 19, 25, 26, 42]