Bubble Sort
Worst and average Time complexity: O(n^2)
n^2 is for double nested for loops

Best Time Complexity: O(n)
N for just singular loop

My implementation: O(2n + n^2)
2 for loops and one double nested for loops

I experimented with using arrays that would have the worst case scenario which is where it'll be in reverse order and the number of comparisons that I was able to figure out was n!. I would also use the base best case scenario which wouldn't even run into the loop since everything was already sorted. There for there would be n numbers of compare and 0 moves I learned about the amount of comparisons and the best and worse cases of bubble sort.

Shell Sort
Worst Time Complexity: O(n^2)
n^2 is for double nested for loops

Best Time Complexity: O(n)
The best case is when the array is already sorted because the inner loop would never run and the array would be left alone.

My implementation: O(4n + n^3)
4 for loops and triple nested for loops

There are many ways to reduce the gap that lead to better time complexity. One way that was mentioned in the piazza that sparked my interest was the Prat Sequence. Prat Gap Sequence greatly reduces the runtime of the code and would make the gap sequence more efficient. The equation for pratt sequence is in the form of 2^p*3^q which would be no larger than the maximum gap size of the sort. I tried to implement this sequence for my gap but failed miserably and I just kept my old one.

Quick Sort
Worst Time Complexity: O(n^2)
n^2 is for double nested for loops
This happens when the partition is chosen as

Best Time complexity: O(nlogn)

My implementation: O(5n^2 + n)
One double nested for loops with 2 for loops inside and a for loop and a double nested for loop

I experimented with trying to get the best partition value. I searched online that it was best to get the median of the array and set the sequence to that to greatly reduce the runtime. It would create the best case scenario everytime if the partition was the median instead of possibly being the extreme. There are other ways to get the median of the array other than the psuedo code provided to us so I decided to implement my own method

Binary Insertion Sort
Best Time Complexity: O(n^2)
Worst Time Complexity: O(nlogn)
My Implementation: O(4n^2 + 2n)
2 double nested for loops and two for loops

In the worst case binary insertion sort performs log(base 2)(n) comparisons because it needs to determine the correct location to insert the new elements. The Binary search makes sure that the array is split into halves to compare with the halves and greatly reduces the amount of comparisons compared to using just the insertion sort. I experimented with using the insertion without the binary search after researching it. It would greatly increase the comparisons and it doesn't split the array in halves to compare with. The binary search insertion would reduce the runtime for the sort.