Writeup

Binary Insertion: Binary insertion actually has the least comparisons, because the way that binary insertion sort compares is to calculate the pointer mid rather than compare the element that next to the current element. Therefore, it only takes log(n) to search. The reason that makes binary insertion sort inefficient because this sort method requires insertion. While insertion is not recommended in array because insertion basically means to shift all the other elements which take o(n) time, therefore, we see a lot of moves there. Since it requires a lot of insertion, the average time complexity is o(n2).

Bubble sort: Bubble sort is the simplest sort method and the easiest to understand. It loops through all the element and compare the to the element next to them. It allows us the find the smallest elements in the array and place them on the left each time. However, this method requires a lot of moves and comparisons. The best case for bubble sort is when the array is already sorted, so it will only need to loop once. However, the average time complexity of bubble sort would be o(n2) because it almost needs to loop through each element twice. It takes one loop to find the smallest value in the unsorted portion of the array, and it needs to do it n times.

Shell sort has way less movements compare to bubble and binary. But you have to pay for that too. Shell sort has a better sort algorithm to reduce the amount of swapping, but it also requires more comparisons. The shell sort time complexity base on the gap sequence. By looking at the result of my program, the average time complexity is o(n2) because it has about n2 comparisons.

Quick sort: Apparently, quick sort is the fastest sorting method because it has the least moves. The partition function place one element to its final position. The partition function only takes o(n) because its using two pointer approach. The partition function itself split the array in half, so it only needs 0(log(n)) to finish to rest of the array. Therefore, the average time complicity of quick sort is o(nlogn).

I realize constant is not a big deal when comparing O(nlogn) and O(n2). Also, I learned that a good sort algorithm shouldn't involve insertion because insertion itself takes time. Also, it is important to make necessary moves and comparisons only. That means we should try to place the element to its final position at once.

I tested the sorted with different size of arrays. I realized shell sort could be faster when dealing with small arrays. But generally, quick sort is still the faster sort. When the arrays size goes bigger, the moves and compares get significant larger except quick sort. Therefore, a good algorithm is really important when n getting bigger.