1.

Bubble sort has a time complexity of O(n^2). This is said to be the worst time complexity for all the sorts. By looking at it, the bubble sort seems the easiest to code and to understand. It goes through and checks each adjacent element and switches only if necessary. I can understand why this is the slowest because it goes though the whole array in a for loop, and once it reaches the end, it goes back to the beginning to check again. I would probably not use this sort unless it's with very little elements, or only to check if elements are in order after knowing it's sorted.

Shell sort has a time complexity of O(n^2) at worst case but varies and can be O(n^3/2) or O(n logn^2). Shell sort is a little better than bubble sort because of the initial steps. Shell sort takes gaps of the elements starting from first index and last index and then increments down by half. It's a little more extra to code and understand but a little more efficient. The only bad thing is that it still has to go through 2 for loops which makes the O(n^2) time complexity. I don't like Shell sort because I have to use another function to find the gap.

Quick sort has a time complexity of O(n^2) at worst but on average has a O(n*logn). Quick sort uses a recursion and a pivot point. Quick sort divides the array into two around pivot and separates them separately. This is a lot faster because it doesn't require and for loops and checking each element in the array. It will sort as it goes without needing repetition. This is pretty fast, because it requires less steps to sort even though the worst time is O(n^2).

Binary insertion sort has a time complexity of O(n) at worst case but can even get to O(log*n). Binary insertion sort is the fastest because it finds the position of the element needed to be sorted and places it at the right location which doesn't require any for loops to indent through. The sort keeps swapping until it finds the right position. The code is simple to code and understand and would recommend using this sort because its very fast.

2.

I have learned that each sort has some pros and cons. It's either if you want faster time for sorting or less comparisons. The bubble sort although is the worst for all cases. It has a huge number of moves and comparisons in the long run. Shell sort has very little need for swapping but the comparisons are way too big which ruins the time. Quick sort is really nice with very fast swapping but the time is a little slower because it compares more than it swaps. I believe that binary sort will be almost the best because it barely compares elements which saves a lot of time. The only problem is that it swaps elements as much as bubble sort. The only sort with low swap moves is Shell sort. The lowest ratio of numbers for both would be quick sort. A lot of trade offs.

3.

I experimented with the sorts by trying sorting big small numbers and then big numbers. I tried with sorted arrays and then will unsorted arrays. If the array is more sorted than the worst case complexity would be better, otherwise it's the other way around with super unsorted arrays.