1. Who is your programming partner? Which of you submitted the source code of your program?

   My programming partner is Aaron Smith; I submitted our code under my name.

2. What did you learn from your partner? What did your partner learn from you?

   I learned how to create our comparators so that they worked as expected. He learned from me how to create the shell sort.

3. Evaluate your programming partner. Do you plan to work with this person again?

   My programming partner was very helpful and I plan on working with him again.

4. Analyze the run-time performance of the `areAnagrams`method.
   - What is the Big-O behavior *and why*? Be sure to define *N*.

     The big O behavior is N^2. This is because our areAnagrams method uses our sort method, which uses a selection sort. Selection sorts have BigO(N^2).

   - Does the growth rate of the plotted running times match the Big-O behavior you predicted?
     The graphs of the runtimes do match the predicted Big-O behavior, it appears to have exponential growth.

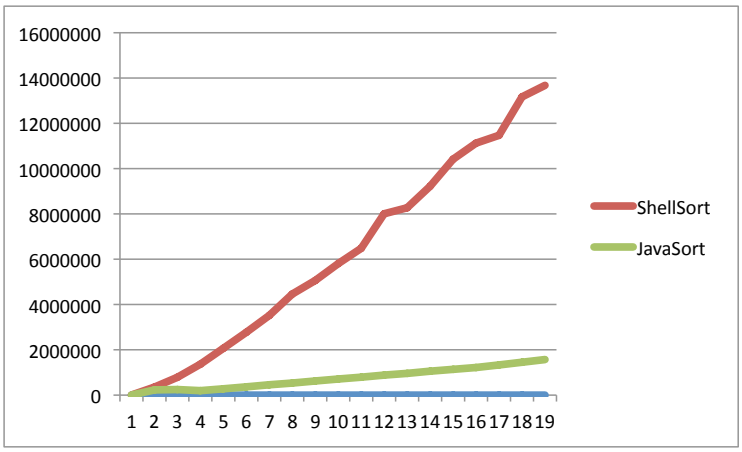
5. The shell sort and insertion sort both have a worse case of O(N^2), when the array is in reverse order, and the best case of O(N), when the array is in order. The shell sort though is able to gap over large portions of the list though making it able to preform much better then the selection sort, which only compares with items next to it. Our graphs appear to portray these complexities. For smaller values of N they seem to be around the same run time but for larger values of N you can see that the shell sort is way more efficient.

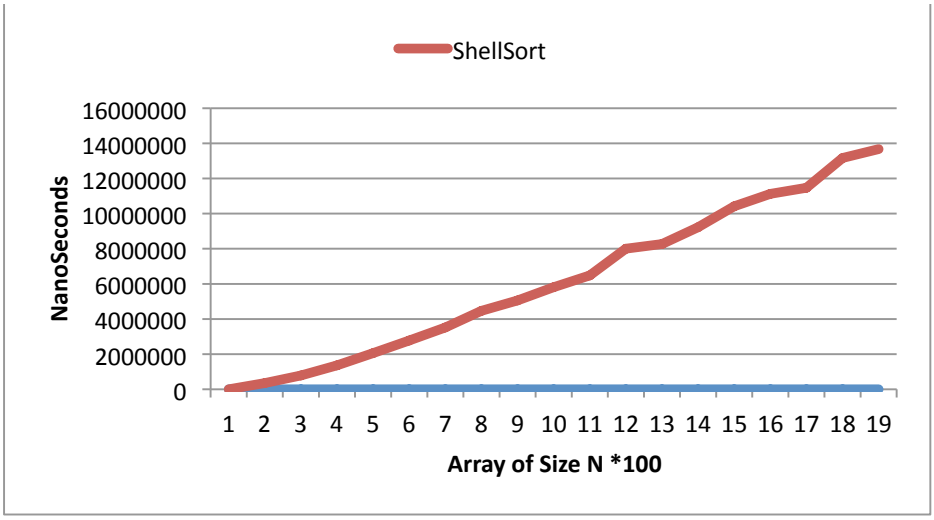
6. What is the run-time performance of the `getLargestAnagramGroup` method if we use Java's`sort` method instead? How does it compare to using insertion sort and Shell sort? (Use the same list of guiding questions as in #4.)

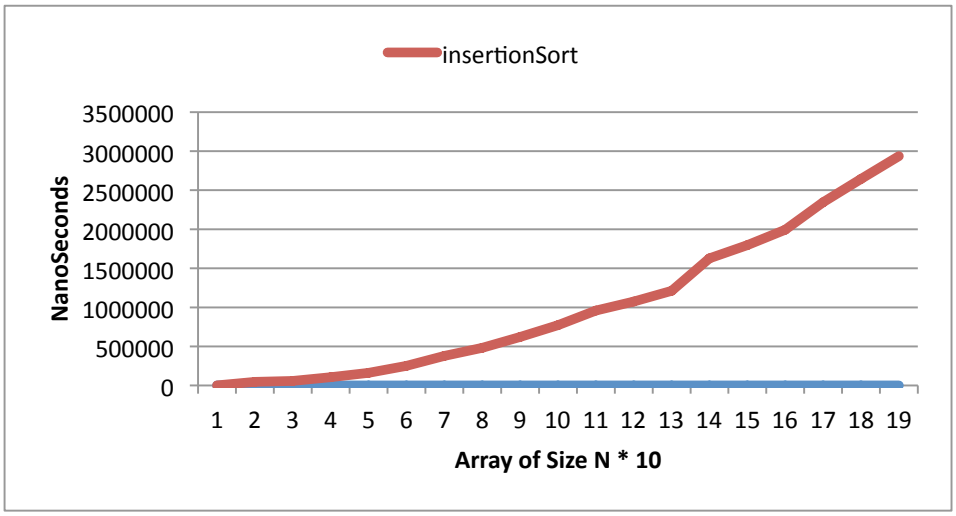
   It appears that Java's Sort algorithm uses a merge sort with the complexity of O(Nlog(N)). Our shell sort depending on how sorted the array is has a complexity from O(n) to O(n^2), while our insertion sort has a complexity of O(n^2). Due to this the java sort does dramatically better then our selection sort where the shell sort sits in between the two. For higher values of N though it appears that it starts preforming dramatically better then the selection sort too.

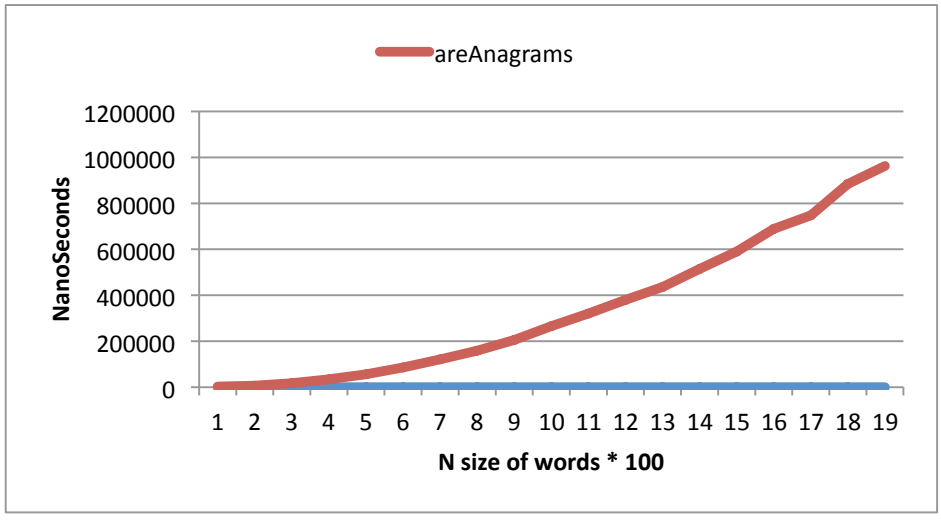
7. How many hours did you spend on this assignment? We spent about 9 hours on this assignment.

N Items	areAnagrams	insertionSort	ShellSort	JavaSort
0	2963	1180	1185	1270
1	6811	45476	352313	231676
2	17413	57376	788866	245412
3	34385	105491	1365569	198894
4	56195	161307	2065607	276591
5	85668	251024	2776885	363738
6	120808	379202	3521408	458055
7	158862	480956	4463286	530077
8	205812	620223	5062343	626479
9	265636	772924	5807761	710868
10	320207	959451	6483400	792166
11	379933	1074544	8003943	883441
12	436057	1211219	8273261	957793
13	514766	1628360	9224315	1054175
14	590130	1797436	1.04E+07	1.14E+06
15	688959	1994414	1.11E+07	1.22E+06
16	747875	2345683	1.15E+07	1.33E+06
17	884484	2644530	1.32E+07	1.45E+06
18	962352	2937263	1.37E+07	1.57E+06




ShellSort
16000000
14000000
12000000
10000000
8000000
6000000
4000000
2000000
0
NanoSeconds
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Array of Size N *100


insertionSort
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
NanoSeconds
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Array of Size N * 10


areAnagrams
1200000
1000000
800000
600000
400000
200000
0
NanoSeconds
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
N size of words * 100