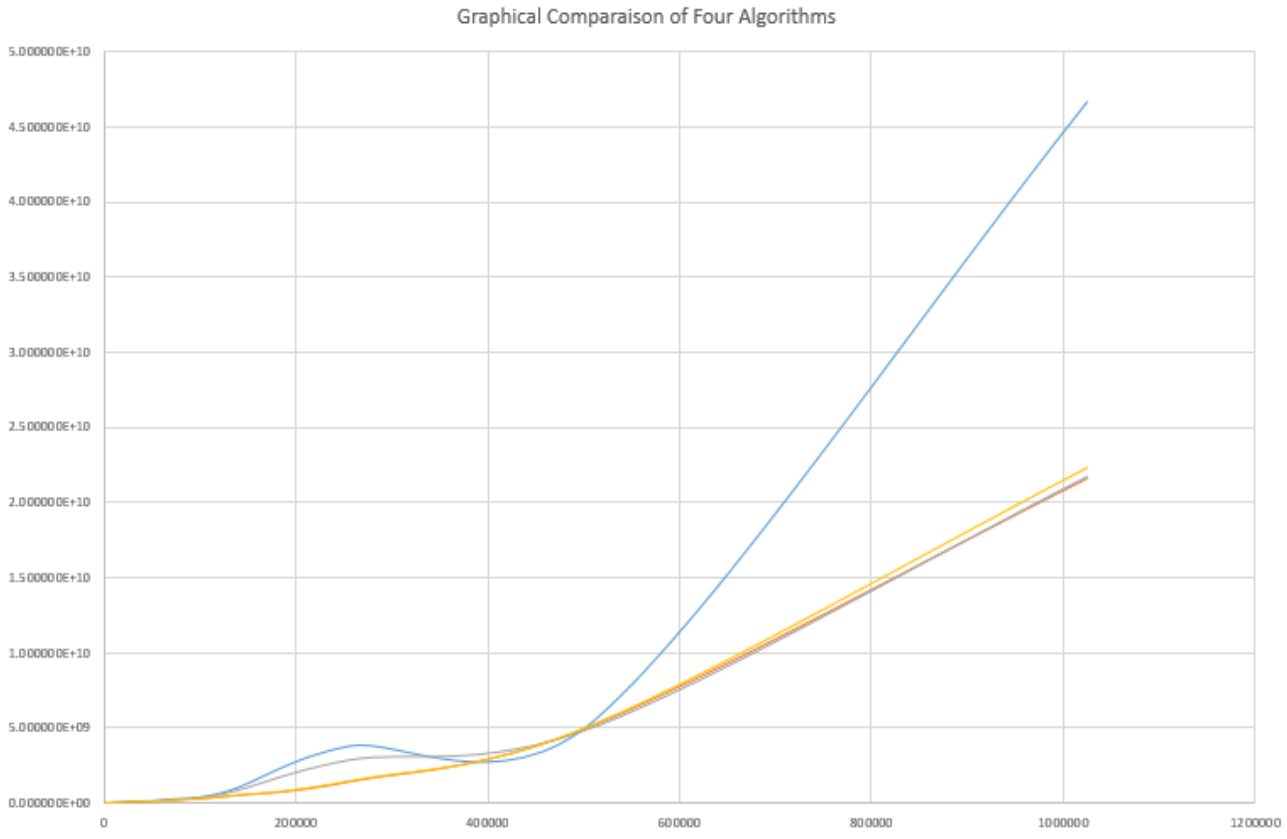
Alexander Bratyshkin
260684228
COMP 250 - ASSIGNMENT #1

Question 5

a)

Size	Nested (ns)	Binary (ns)	Parallel (ns)	Merge (ns)
1000	1.169466E+06	1.173085E+06	1.208853E+06	1.125666E+06
2000	2.639482E+06	2.648571E+06	2.598186E+06	2.645781E+06
4000	5.267546E+06	5.155253E+06	5.281252E+06	5.507562E+06
8000	1.169920E+07	1.121219E+07	1.140605E+07	1.205262E+07
16000	2.607496E+07	2.958304E+07	2.798685E+07	2.891251E+07
32000	6.628529E+07	6.696677E+07	6.775638E+07	6.622797E+07
64000	2.130831E+08	1.549993E+08	1.516659E+08	1.559839E+08
128000	7.456040E+08	4.346708E+08	6.498679E+08	4.461288E+08
256000	3.774177E+09	1.408536E+09	2.857577E+09	1.458976E+09
512000	5.542317E+09	5.215264E+09	5.088588E+09	5.281195E+09
1024000	4.663912E+10	2.155829E+10	2.174567E+10	2.232267E+10

b)



Nested Loops: $T(n) = n^2 \Rightarrow \boldsymbol{O}(n^2)$
Binary Search: $T(n) = 2n^* \log(n) + n \Rightarrow \boldsymbol{O}(2n^* \log(n) + n) \Rightarrow \boldsymbol{O}(n \log(n))$
Sort and Parallel: $T(n) = 2n^*\log(n)+ 4n \Rightarrow \boldsymbol{O}(2n^*\log(n)+ 4n) \Rightarrow \boldsymbol{O}(n \log(n))$
Merge and Sort: $T(n) = 2n^*\log(2n)+ 2n-1 \Rightarrow \boldsymbol{O}(2n^*\log(2n)+ 2n-1) \Rightarrow \boldsymbol{O}(n \log(n))$

c)

Type	Nested (ns)	Binary (ns)	Parallel (ns)	Merge (ns)
1	7.1906673E+09	9.2581744E+07	2.9091402E+08	3.2347501E+08
2	7.5903903E+09	2.5650402E+08	2.5428824E+08	3.2740333E+08

Type 1: First list with 1024000 students, second list with 32000
Type 2: First list with 32000 students, second list with 1024000

Nested Loops: In this case, we can see the values for both types are relatively similar to each other. This can be explained by the fact that it doesn't matter if the target element of a larger array is iterated through a smaller array, or if the target element of a smaller array is iterated through a larger array, because the total amount of iterations end up being the same, as illustrated by the fact that the average complexity of the algorithm is $\boldsymbol{O}(m*n)$, where m and n are the sizes of the two arrays. Since m and n are commutative, the difference in sizes has little effect on the run-time of the algorithm.

Binary Search: We can clearly see that the second type takes much longer to run than the first one. This is due to the fact that, in the second type, the list that is to be sorted happens to be the larger one of the two (1024000 students). Since we are comparing by binary search, there are going to be in average many more divisions in half of the sorted array (and thus many more comparisons) than with a 32000-sized list (as it is in the first type), because, naturally, the array is much larger in size.

Sort and Parallel: We can observe that the values for the runtimes of both types are relatively equal. This can be explained by the fact that this algorithm depends on how many times the two pointers go through their respective arrays. However, since each pointer is "specific" to its own array, it doesn't matter if the sizes of the two arrays are interchanged, because both pointers go through all the indexes of their respective arrays, and are compared to the other array in function of their *own* size. This can also be interpreted mathematically by the fact that the actual running time of the function, with two different arrays of sizes m and n, is expressed by the function $m*\log(m) + n*\log(n)+ 4n$, where addition is commutative and each array relates to its own.

Merge and Sort: Similarly to the previous algorithm, we see that this one has also a similar running time for both cases. The difference in sizes between both arrays doesn't matter at all in this case, because both of them end up being merged into one big one, and a pointer runs through this merged array of combined size.