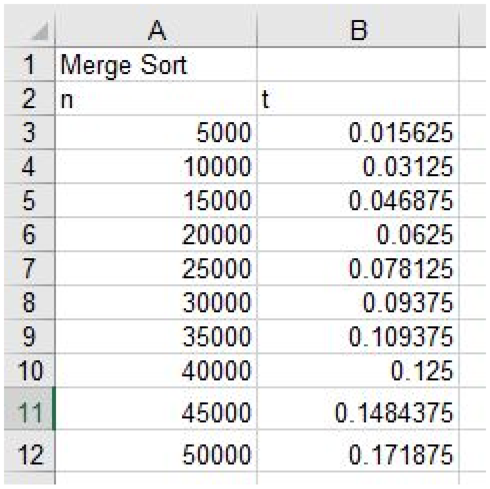
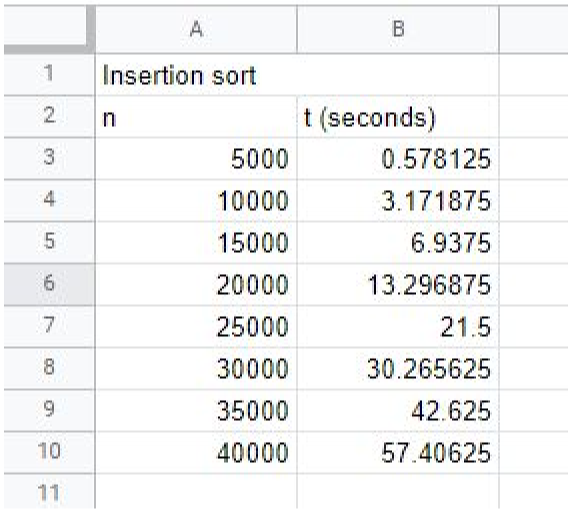
a)



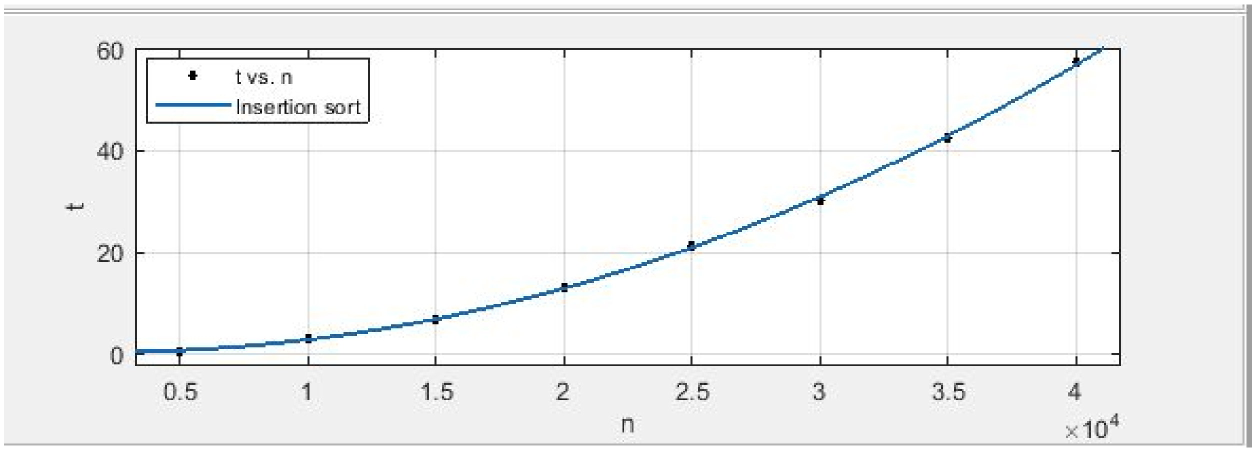
	A	B
1	Merge Sort	
2	n	t
3	5000	0.015625
4	10000	0.03125
5	15000	0.046875
6	20000	0.0625
7	25000	0.078125
8	30000	0.09375
9	35000	0.109375
10	40000	0.125
11	45000	0.1484375
12	50000	0.171875



	A	B
1	Insertion sort	
2	n	t (seconds)
3	5000	0.578125
4	10000	3.171875
5	15000	6.9375
6	20000	13.296875
7	25000	21.5
8	30000	30.265625
9	35000	42.625
10	40000	57.40625
11		

The arrays were generated using python’s “random” library. The program would generate n many random numbers within the range [1 - 10,000], and store them in a list. This allowed for numbers to show up more than once.

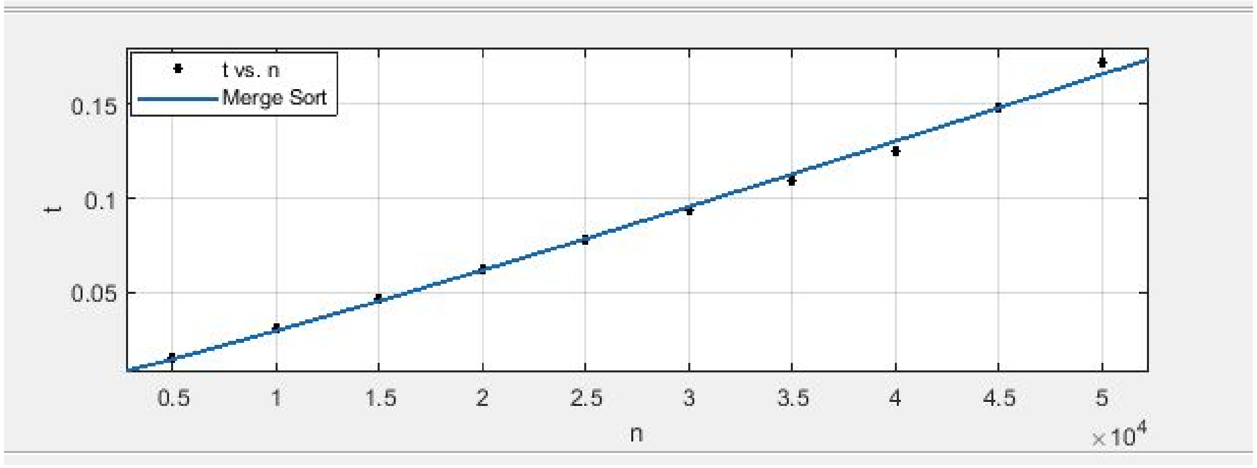
b)
Insertion sort:



f(x) = p1*x^2 + p2*x + p3

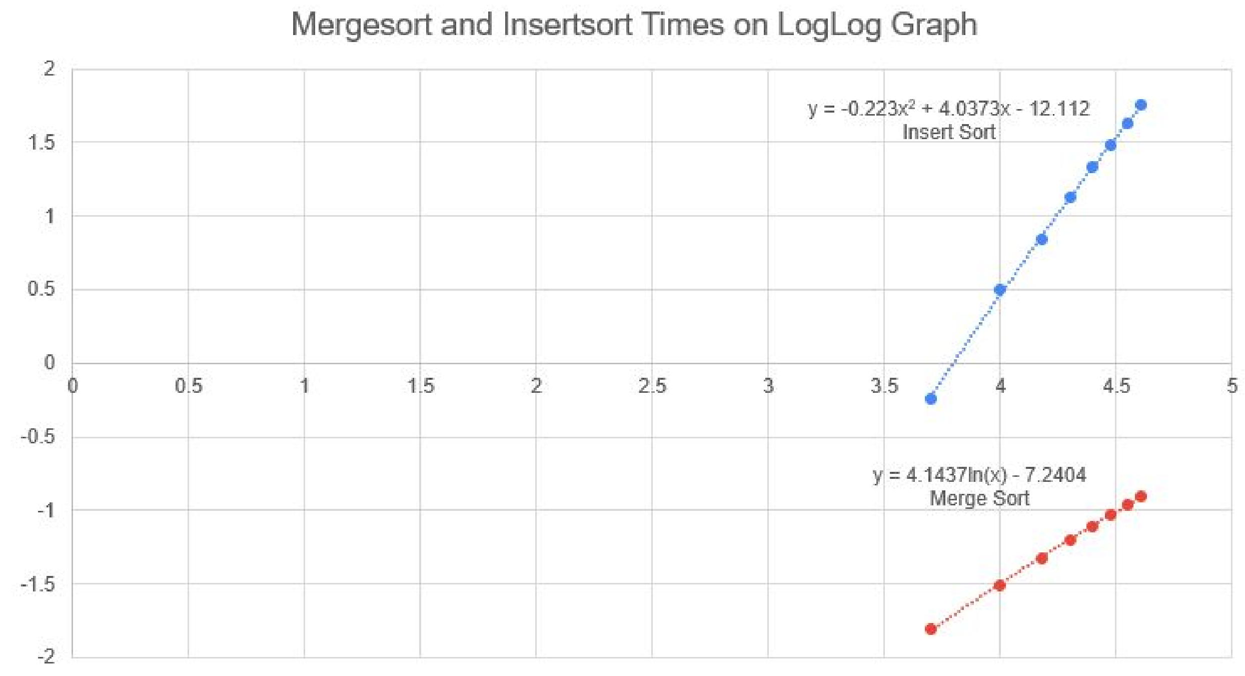
p1 = 3.955e-08
p2 = -0.0001766
p3 = 0.7355

Merge sort:



f(x) = a*x*log(x)+b
a = 3.025e-07
b = 0.001983

c)



d)
The expected time for insertion sort to sort 200,000 elements is 1547.42 seconds.
The expected time for merge sort to sort 200,000 elements is 0.32 seconds.

e)
Insertion sort: 1935.515625 seconds
This one was pretty in the range of the expected value, though with so many elements it's easy for the array to be in a starting state that isn't completely even in its randomness.

Merge sort: 0.84375 seconds
I think this one was pretty far off because the original data set was fairly linear looking, so for large numbers like 200,000, we wouldn't get close to the expected logarithmic graph. Something like a much wider spanning data set would probably fix this issue.