Question 1:

$8n^2 <= 64\ n \log n$
$n^2 <= 8\ n \log n$
$n <= 8 \log n$
$n - 8 \log n <= 0$

According to Wolfram Alpha:
$n = 1$
$n = 8$

This indicates that insertion sort is faster between values 1 and 8. When putting those values into the table below, we see that the insert sort actually gets slower at around n = 7.

n	Insert Sort	Merge Sort
2	32	38.53
3	72	91.6
4	128	154.13
5	200	223.67
6	288	298.81
7	**392**	**378.6**
8	**512**	**462.38**

Question 2:

	Relationship	Explanation		
a.	f(n) = O(g(n))	$\log n^{.25}$ .25 log n	$\log n^{.5}$ .5 log n	By taking the log of both sides, it's apparent that f(n) grows slower than g(n) given its smaller coefficient.
b.	f(n) = Ω(g(n))	log n	$\log(\log^2 n)$ log(log n * log n) log * log n + log * log n	Because log(log n) is smaller than log n, we can say that f(n) grows faster than g(n). Thus, g(n) forms the lower bound for f(n).
c.	f(n) = O(g(n))	log n	ln n	The natural log grows faster once n is greater than 1 than normal log. Thus, ln n forms the upper bound of log n.
d.	f(n) = Θ(g(n))	$1000n^2$ n^2	$.0002n^2$ - 1000n $.0002n^2$ n^2	We can disregard the coefficients and the 1000n (ignore because dominated by n^2) as we're looking at this asymptotically. Thus, it's obvious that both grow at the same rate n^2.
e.	f(n) = O(g(n))	n log n log (n log n) log n + log(log n)	$n^{sqrt(n)}$ $\log n^{sqrt(n)}$ sqrt(n) log n	Because the square root of n is greater than log n, we can say that (n) < g(n) and that g(n) is the upper bound of f(n).

f.	f(n) =O(g(n))	e^n	3^n	Because e is less than 3, we can say that it grows slower than 3^n and thus 3^n forms the upper bound for e^n.
g.	f(n) = O(g(n))	2^n log 2^n n log 2	2^{n+1} log 2^{n+1} (n+1) log 2	Because n < n + 1, we can say that f(n) grows slower than g(n). Thus, g(n) forms the upper bound of f(n).
h.	f(n) = O(g(n))	2^n log 2^n n log 2	2^{2n} log 2^{2n} (2n) log 2	Because n < 2n for all values greater than equal to 1, we can say that f(n) grows slower than g(n). Thus, g(n) forms the upper bound of f(n).
i.		2^n	n!	Unable to determine the relationship as n! has closest upper and lower bounds that move depending on the value of n. Only thing that can be determined about n! is that the lower bound is Ω(1) and the upper bound is O(n^n).
j.	f(n) = Ω(g(n))	log n	sqrt(n) $n^{1/2}$ log $n^{1/2}$ 1/2 log n	Because g(n) grows at a faster rate (1/2 log n < log n), we can say that f(n) forms the upper bound for g(n).

Question 3:

Part A:

f1(n) = Θ(g(n)) and f2(n) = Θ(g(n))

f1(n) = Θ(g(n)) and g(n) = Θ(f2(n)) => symmetric property

f1(n) = Θ(f2(n)) => transitive property

Thus, through use of the symmetric and transitive properties, we're able to prove the conjecture.

Part B:

f1(n) = O(g1(n)) and f2(n) = O(g2(n))

f1(n) <= c1 * g1(n) and f2(n) <= c2 * g2(n) => above can be rewritten given general properties

f1(n) + f2(n) <= c1 * g1(n) + c2 * g2(n) => logically, given both are less than this must also be true

f1(n) + f2(n) <= c(g1(n) + g2(n)) => this is also true when factoring out the coefficient to just c

This is the definition for O, but in order to be Θ, it would also need to be true for Ω

The following is an example that disproves the function:
f1(n) = n, f2(n) = n^2, g1(n) = n^3, g2(n) = n^4

This would not satisfy the reverse Ω function (*f1(n) >= c1 * g1(n) and f2(n) >= c2 * g2(n)*) as n is not greater than n^3 and n^2 is not greater than n^4.

Question 5:

Insert sort running times:

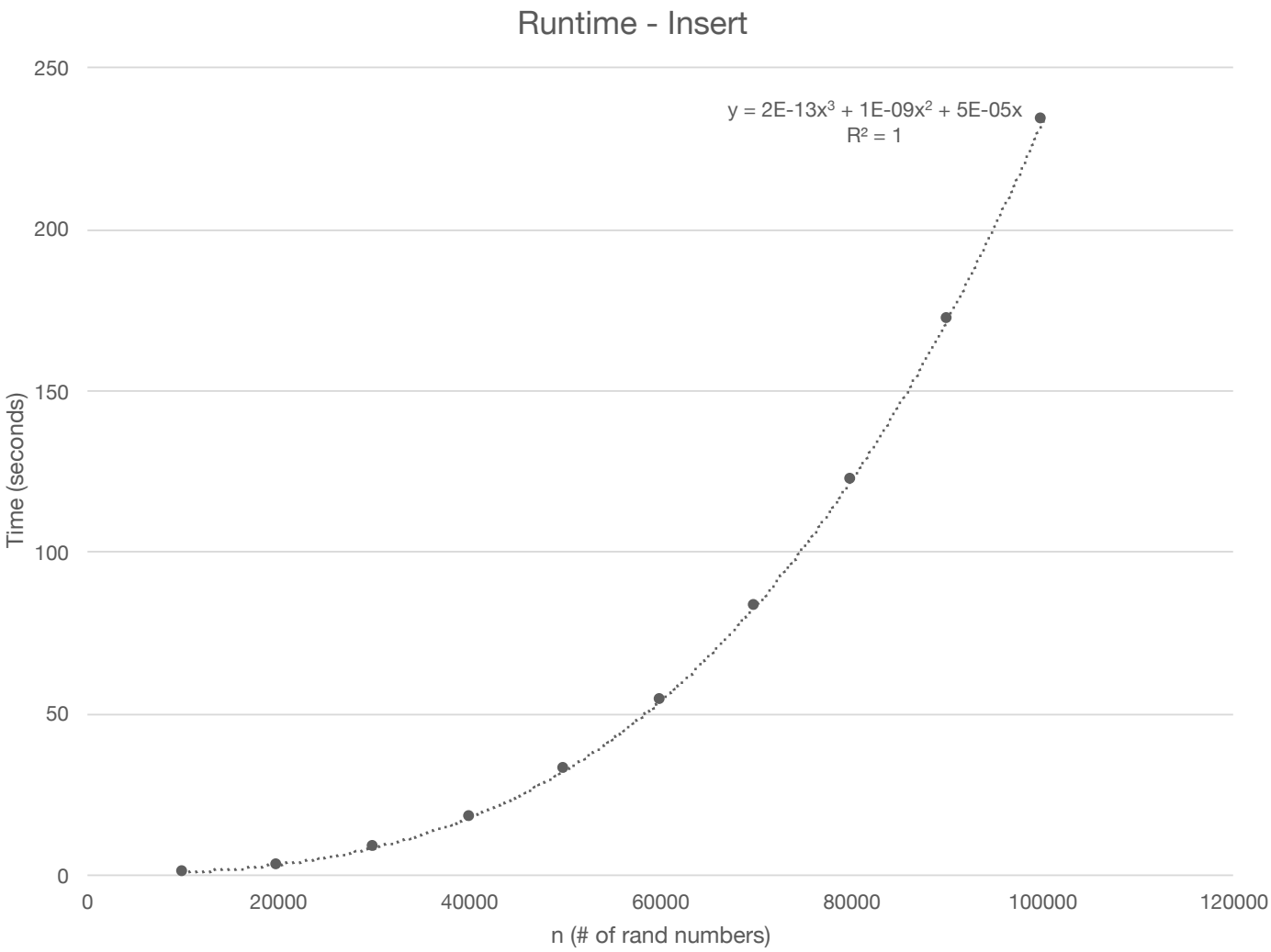
n	Time 1	Time 2	Time 3	Time 4	Time 5	Average
10000	0.24	0.5	0.74	1	1.24	0.74
20000	1.79	2.31	3.07	3.7	4.43	3.06
30000	5.61	7	8.18	9.35	10.73	8.17
40000	12.98	15.31	17.61	19.9	22.31	17.62
50000	26.07	29.36	32.86	36.37	39.84	32.9
60000	44.67	49.38	54.1	58.86	63.76	54.15
70000	70.22	76.7	83.21	89.72	96.53	83.28
80000	105.12	113.92	122.48	130.98	139.23	122.35
90000	150.03	161.12	171.6	182.49	193.28	171.7
100000	206.94	220.1	233.43	247.83	260.87	233.83

Merge sort running times:

n	Time 1	Time 2	Time 3	Time 4	Time 5	Average
10000	0	0.01	0.01	0.02	0.03	0.01
20000	0.04	0.05	0.07	0.08	0.1	0.07
30000	0.12	0.14	0.16	0.18	0.2	0.16
40000	0.23	0.26	0.28	0.32	0.35	0.29
50000	0.39	0.42	0.46	0.49	0.53	0.46
60000	0.57	0.61	0.65	0.7	0.74	0.65
70000	0.79	0.84	0.9	0.96	1.01	0.9
80000	1.06	1.12	1.18	1.24	1.3	1.18
90000	1.36	1.43	1.5	1.57	1.64	1.5
100000	1.71	1.78	1.86	1.93	2.01	1.86

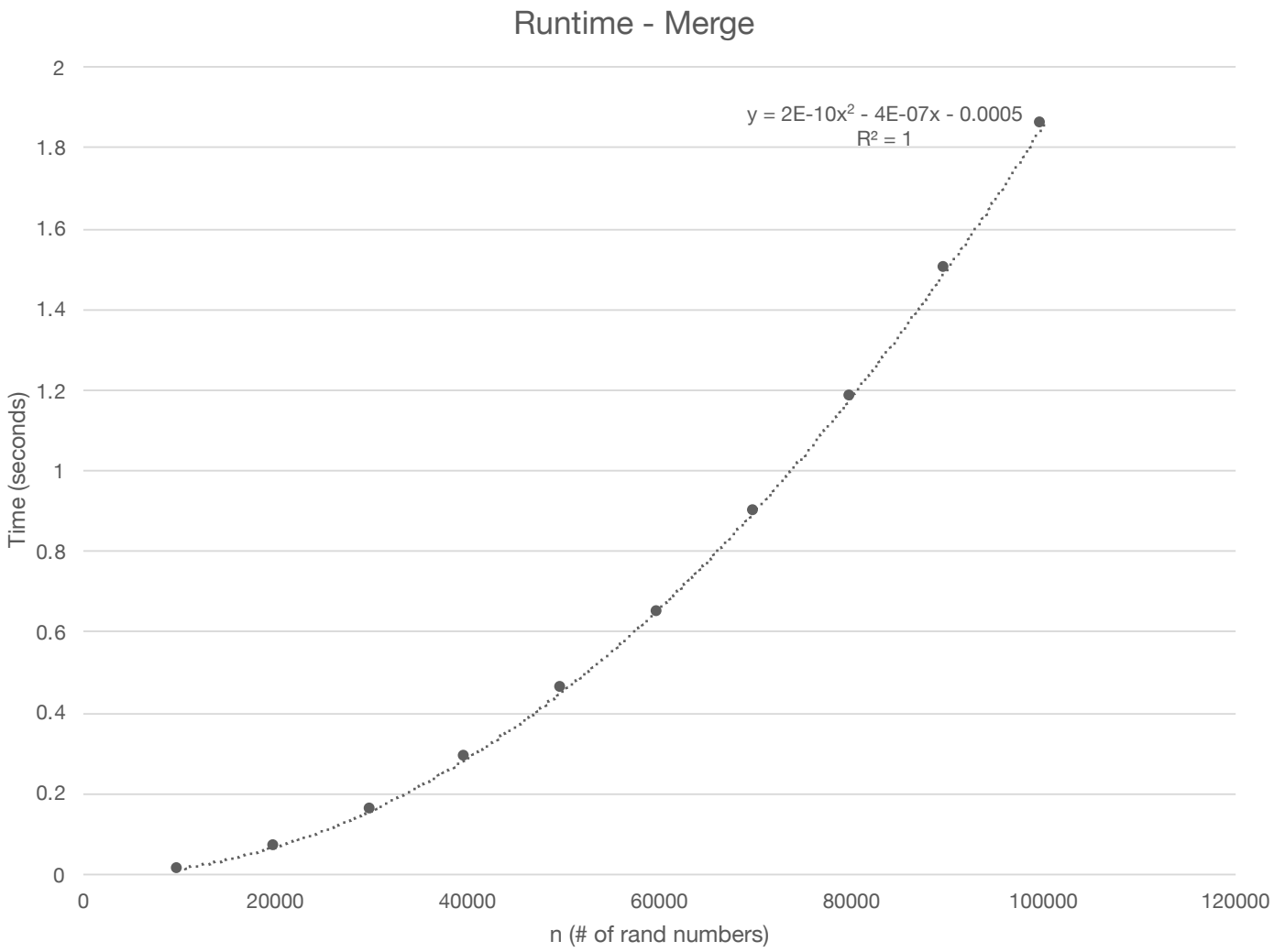
Insert sort plot:

Polynomial curve - 3rd order

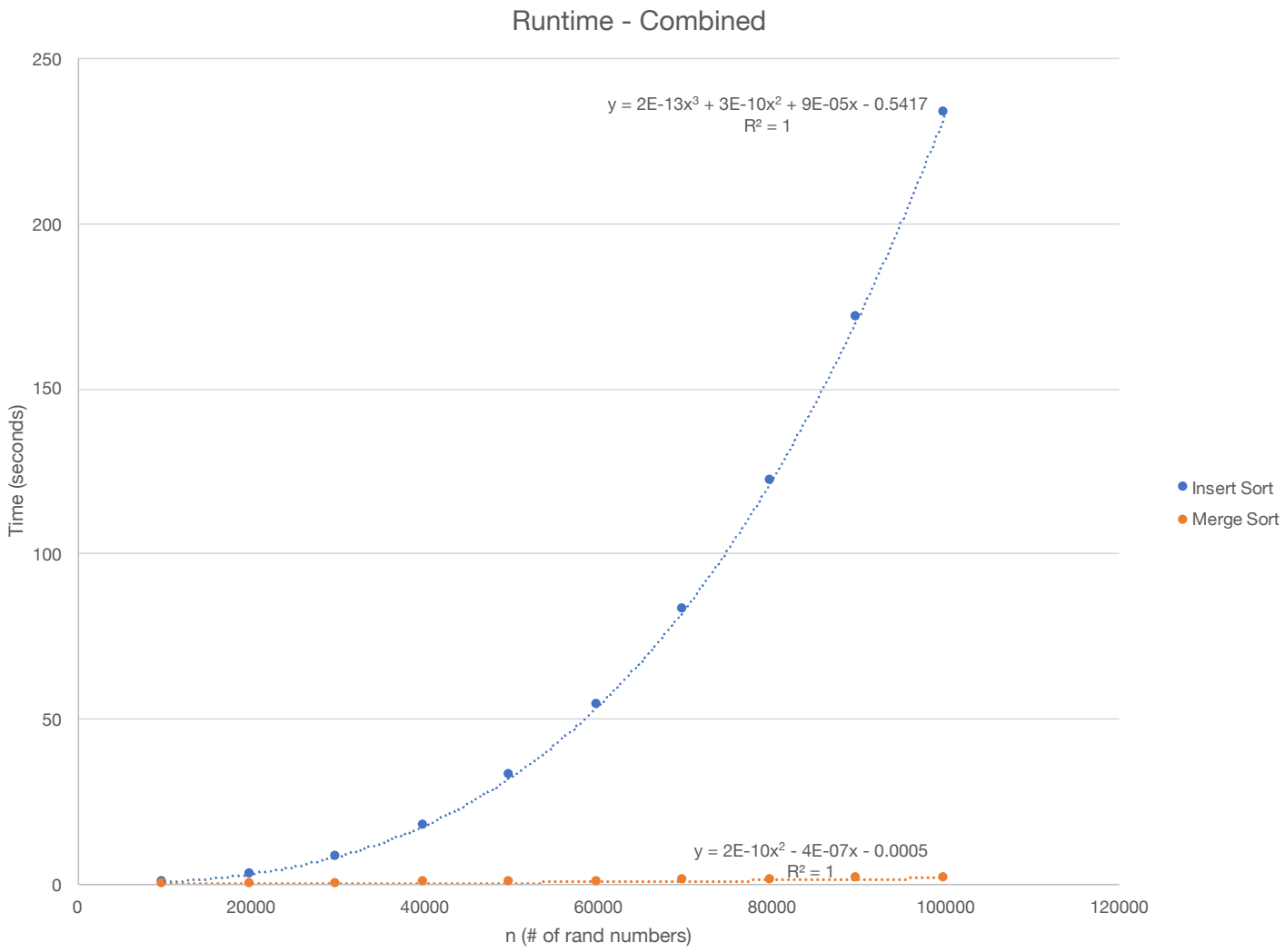


Merge sort plot:

Polynomial curve - 2nd order



Combined plot:



Comparison:

The theoretical running time of insert sort is $O(n^2)$. The result from my experimental run times was $O(n^3)$. The theoretical running time of merge sort is n log n. The result from my experimental run times was surprisingly $O(n^2)$. I'm guessing that this was largely due to a lack of inputs used when testing, and if I were to increase the number of data points the function would closer resemble the theoretical.

As expected, the insert sort function had a much steeper curve than the merge sort. This indicates that insert sort gets less efficient as the number of inputs increases, compared to the merge sort function.