CS370 HW1: Lyell Read

Problem 1

a) lim(n → inf)(n^0.25/n^0.5) = 0. f(n) is O(g(n))
b) lim(n → inf)(log n^2 / log n) = 2. f(n) is Theta(g(n))
c) lim(n → inf)(log log n / log n) = 0. f(n) is O(g(n))
d) lim(n → inf)(5000n^3 + n^2 / 0.000001n^4) = 0. f(n) is O(g(n))
e) lim(n → inf)(n log n + n / n sqrt(n)) = 0. f(n) is O(g(n))
f) lim(n → inf)(e^n / 2^n) = inf. f(n) is Omega(g(n))
g) lim(n → inf)(2^n / 2^n+1) = ½. f(n) is Theta(g(n))
h) lim(n → inf)(n^n / n!) = inf. f(n) is Omega(g(n))

Problem 2

a) The outer for loop executes n times, while the inner for loop runs from ~n times to 0 times as the outer loop progresses. Therefore, it can be estimated to run n/2 times. An approximation fort the runtime of this algorithm is therefore n * n/2 = 1/2n^2, which becomes Theta(n^2)
b) Given that the for loop runs for k runs for input z, where z = 2^2^k. This function's inverse (that will provide the 'runs' based on input) is the inverse of said function, which results in logbase[1](2, logbase(2, z)) = k. The instructions executed inside the loop would occur k times as they are constant timed. Therefore, the expression for the runtime is Theta(logbase(2, logbase(2, x)).
c) The first for loop operates n/2 times. The second operates m times. The third, outer loop operates n times, while the inner loop operates m times. Therefore, given that what happens inside the loops is the same (variable increment, printout), I will simplify these all to c. The expression for the running time for n and m is cn+cm+cnm, where cnm dominates the expression. Therefore, the running time is Theta(nm).

Problem 4

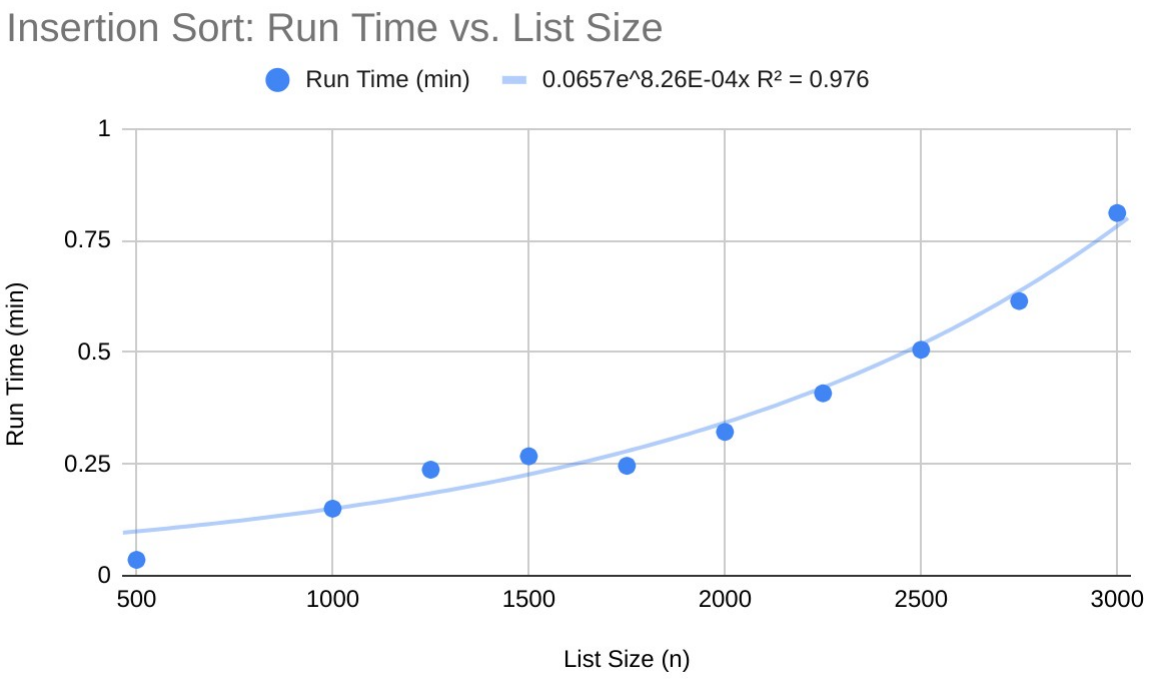
b)

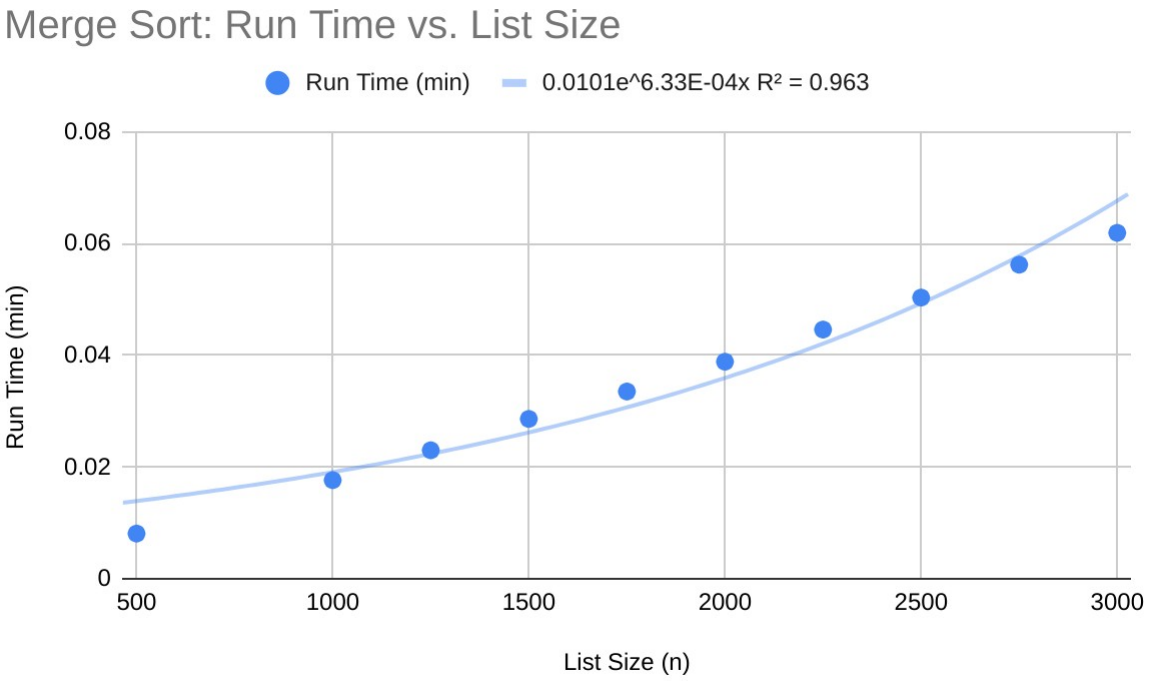
| Insert | Run 1 | Run 2 | Run 3 | Run 4 | Average |
|---|---|---|---|---|---|
| 500 | 0.0354282856 | 0.03532505035 | 0.03603959084 | 0.03522872925 | 0.03550541401 |
| 1000 | 0.1510493755 | 0.1492519379 | 0.149787426 | 0.1500480175 | 0.1500341892 |
| 1250 | 0.2387955189 | 0.2388181686 | 0.235714674 | 0.2355716228 | 0.2372249961 |
| 1500 | 0.2687392235 | 0.2369318008 | 0.2797861099 | 0.2835576534 | 0.2672536969 |
| 1750 | 0.2509710789 | 0.2470357418 | 0.2423295975 | 0.2433462143 | 0.2459206581 |
| 2000 | 0.33116889 | 0.3202552795 | 0.3170866966 | 0.3185894489 | 0.3217750788 |
| 2250 | 0.4203233719 | 0.4061934948 | 0.3998165131 | 0.406409502 | 0.4081857204 |
| 2500 | 0.5201966763 | 0.5000331402 | 0.5001089573 | 0.5017826557 | 0.5055303574 |
| 2750 | 0.626871109 | 0.6061623096 | 0.6058232784 | 0.6190369129 | 0.6144734025 |
| 3000 | 0.7425122261 | 0.7242071629 | 0.723626852 | 1.055630445 | 0.8114941716 |

| Merge | Run 1 | Run 2 | Run 3 | Run 4 | Average |
|---|---|---|---|---|---|
| 500 | 0.008046388626 | 0.008167982101 | 0.008062601089 | 0.008062124252 | 0.008084774017 |
| 1000 | 0.0175101757 | 0.01782727242 | 0.01747560501 | 0.01778435707 | 0.01764935255 |
| 1250 | 0.02272057533 | 0.02336001396 | 0.02272391319 | 0.02316570282 | 0.02299255133 |
| 1500 | 0.0281457901 | 0.02907085419 | 0.02832937241 | 0.028870821 | 0.02860420942 |
| 1750 | 0.03305387497 | 0.03388023376 | 0.03341984749 | 0.0336856842 | 0.03350991011 |
| 2000 | 0.03877997398 | 0.03912258148 | 0.03846979141 | 0.03899884224 | 0.03884279728 |
| 2250 | 0.04427671432 | 0.0450398922 | 0.04439425468 | 0.04466748238 | 0.0445945859 |
| 2500 | 0.0496430397 | 0.05084180832 | 0.04995250702 | 0.05086088181 | 0.05032455921 |
| 2750 | 0.05556297302 | 0.05680966377 | 0.05590558052 | 0.05656909943 | 0.05621182919 |
| 3000 | 0.06100416183 | 0.06277751923 | 0.06145882607 | 0.06239295006 | 0.0619083643 |

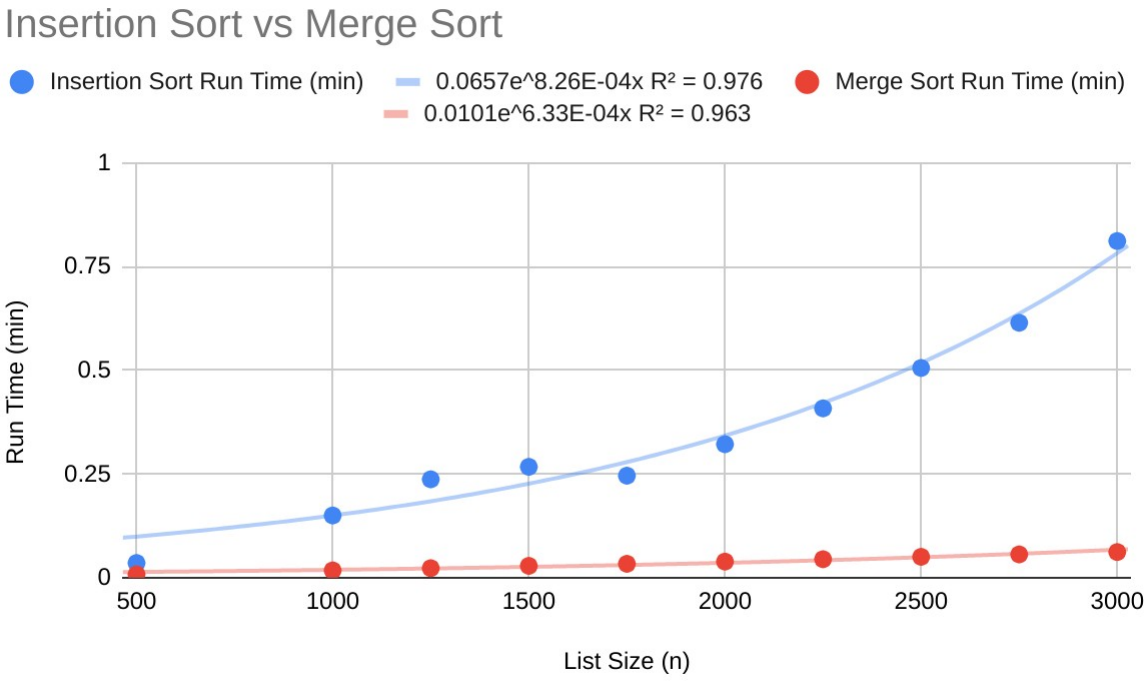
1 The theoretical function logbase(x, y) returns the log of y in base x.

c) The best fitting trendline is an exponential one.



d)




e)The running times compare quite well with the expected runtimes of Merge and Insertion sort. As we can see, as the list size gets greater, the merge sort remains fast, while insertion sort starts to take exponentially longer. That is because merge sort has time complexity of n*log(n), as opposed to the slower n^2 complexity of Insertion Sort.