Using Arrays.sort()

Data

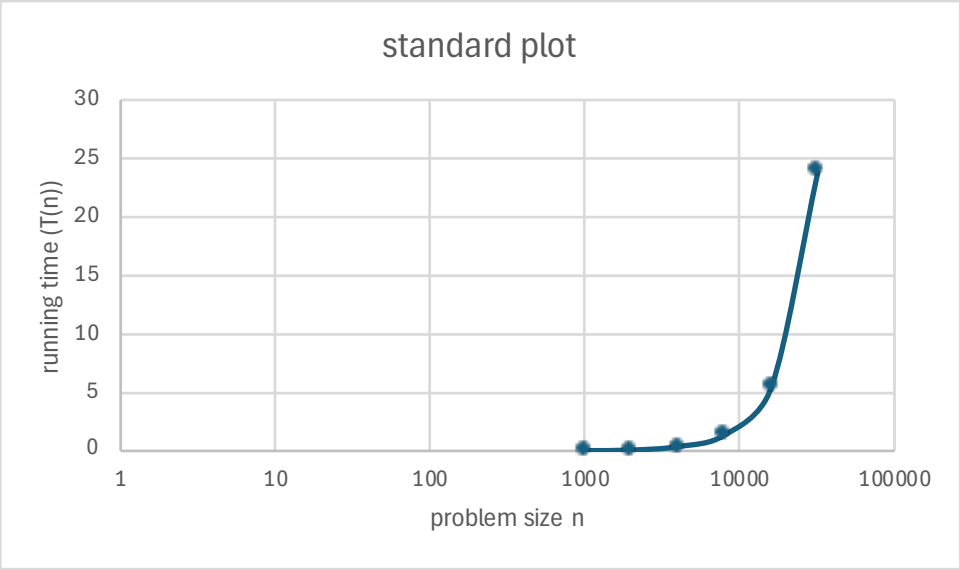
problem size n	running time (T(n))
1000	0.024
2000	0.073
4000	0.368
8000	1.362
16000	5.424
32000	23.866

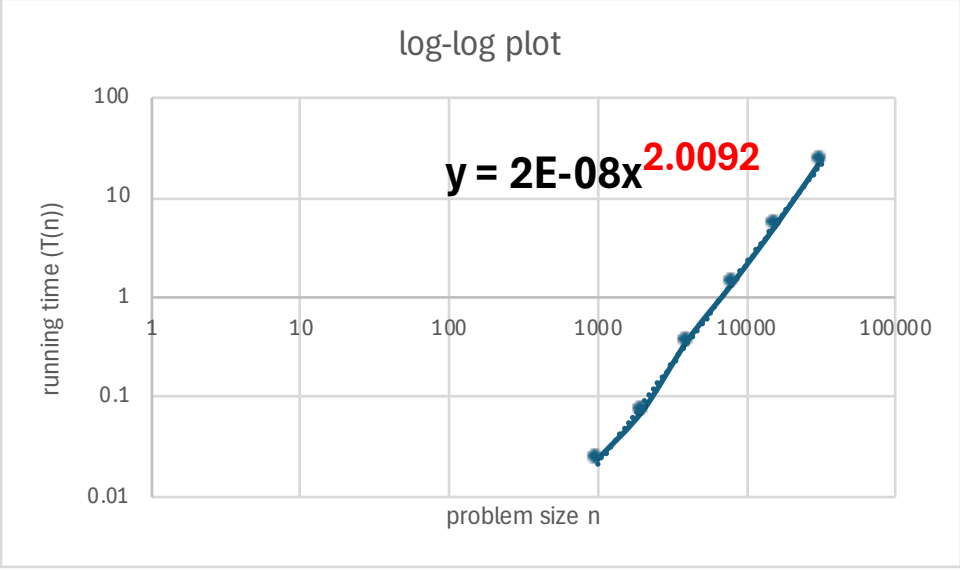
Prediction:

64 K → $T(64K) = 2 \times 10^{-8}(64K)^{2.0092}$ ≈ 91

128 K → $T(128K) = 2 \times 10^{-8}(128K)^{2.0092}$ = 365

1 M → $T(1M) = 2 \times 10^{-8}(1M)^{2.0092}$ = 22711





I used the equation given by the log log graph to predict values T(n) when n was 64K, 128K, 1M.

Using Insertion.sort()

Data

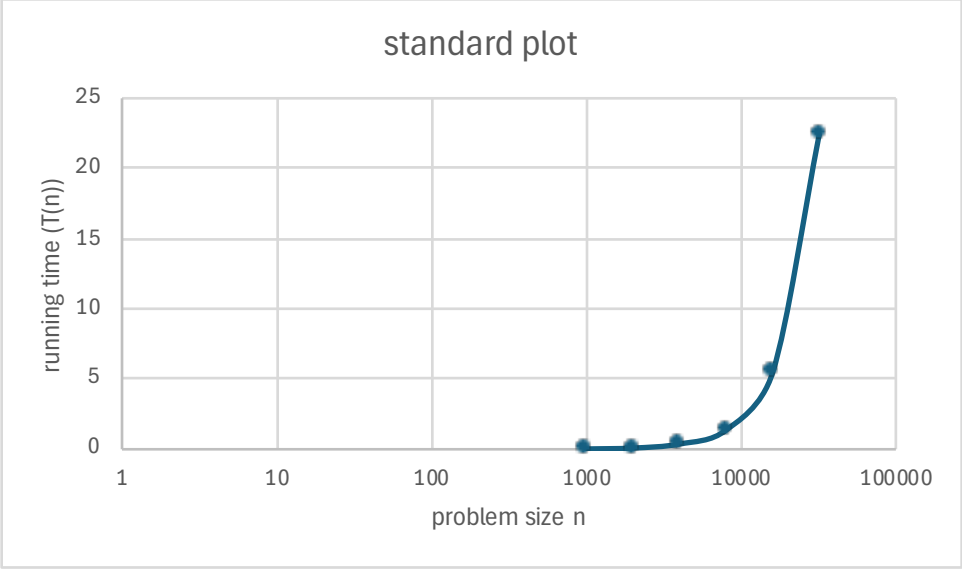
problem size n	running time (T(n))
1000	0.023
2000	0.086
4000	0.371
8000	1.364
16000	5.466
32000	22.437

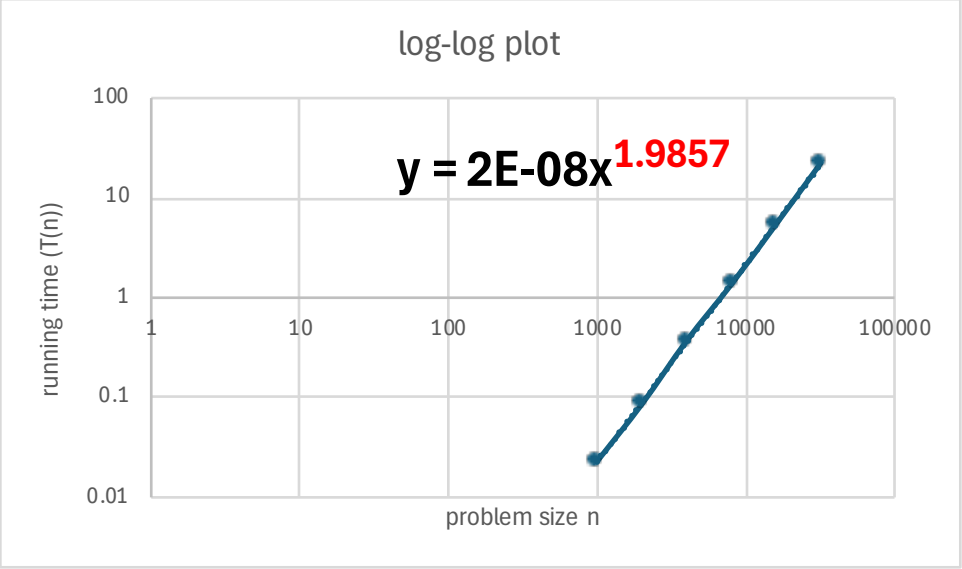
Prediction:

64K → $T(64K) = 2\times10^{-8}(64K)^{1.9857}$ = 70

128K → $T(128K) = 2\times10^{-8}(128K)^{1.9857}$ = 277

1M → $T(1M) = 2\times10^{-8}(1M)^{1.9857}$ = 16415





I used the equation given by the log log graph to predict values T(n) when n was 64K, 128K, 1M.

Were you suprised by the results using insertion vs the Arrays.sort method? what explains this?

I was suprised to get a similar value for both Arrays.sort() and Insertion.sort() considering their different time complexities. Since Arrays.sort() has a time complexity of $n\log n$, I expected a lower T(n) value for a larger problem size. The data from my runs shows very little difference in run time between the two types of sort methods. This could be due to the fact that the input data is nearly sorted, in which case they may have similar runtime.