A2: Benchmarking

Ryan Brooks

Input size 500:

Round	DnC Time (s)	Naive Time (s):
1	0.0019991397857666016	0.0004990100860595703
2	0.000997304916381836	0
3	0.0015006065368652344	0.0005013942718505859
Average execution time	0.001499	0.000333

Input size 1,000:

Round	DnC Time (s)	Naive Time (s):
1	0.0025010108947753906	0.0004973411560058594
2	0.0025017261505126953	0.0009984970092773438
3	0.002499818801879883	0.0010001659393310547
Average execution time	0.002501	0.000832

Input size 5,000:

Round	DnC Time (s)	Naive Time (s):
1	0.010998964309692383	0.004498958587646484
2	0.011500358581542969	0.004002809524536133
3	0.011999368667602539	0.004000186920166016
Average execution time	0.011410	0.004167

Input size 10,000:

Round	DnC Time (s)	Naive Time (s):
1	0.021014690399169922	0.008498668670654297
2	0.020000696182250977	0.009000778198242188
3	0.02000570297241211	0.008998155593872070
Average Execution time	0.020340	0.008833

Input size 50,000:

Round	DnC Time (s)	Naive Time (s):
1	0.0835108757019043	0.05202198028564453
2	0.08850574493408203	0.04951310157775879
3	0.11700892448425293	0.05351138114929199
Average Execution time	0.096342	0.051682

Input size 100,000:

Round	DnC Time (s)	Naive Time (s):
1	0.17002129554748535	0.1095120906829834
2	0.1790175437927246	0.11601734161376953
3	0.20952939987182617	0.11451435089111328
Average Execution time	0.186189	0.113348

Input size 300,000:

Round	DnC Time (s)	Naive Time (s):
1	0.6635937690734863	0.38704895973205566

2	0.6815929412841797	0.3895449638366699
3	0.6840996742248535	0.38518738746643066
Average Execution time	0.676429	0.387260

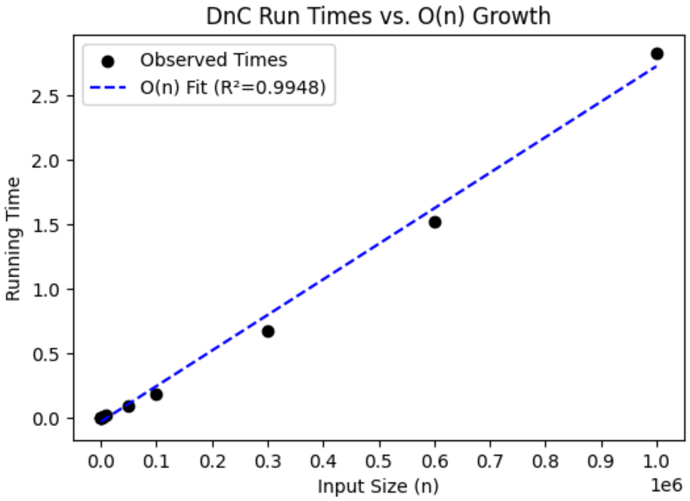
Input size 600,000:

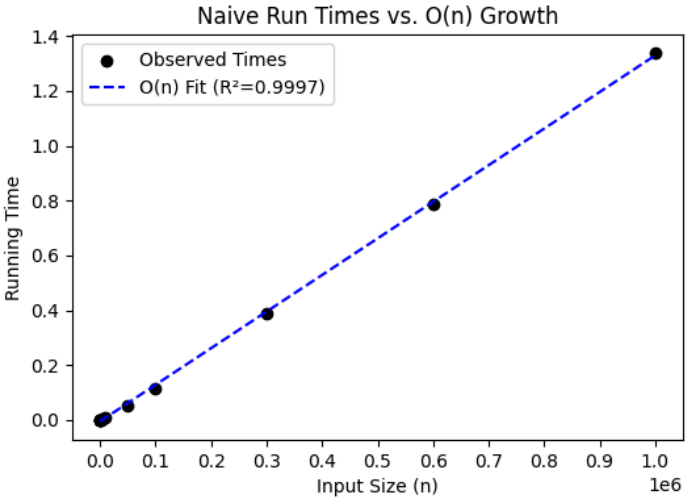
Round	DnC Time (s)	Naive Time (s):
1	1.5286791324615479	0.7835958003997803
2	1.4890220165252686	0.7901062965393066
3	1.5587081909179688	0.7870998382568359
Average Execution time	1.525470	0.7869339

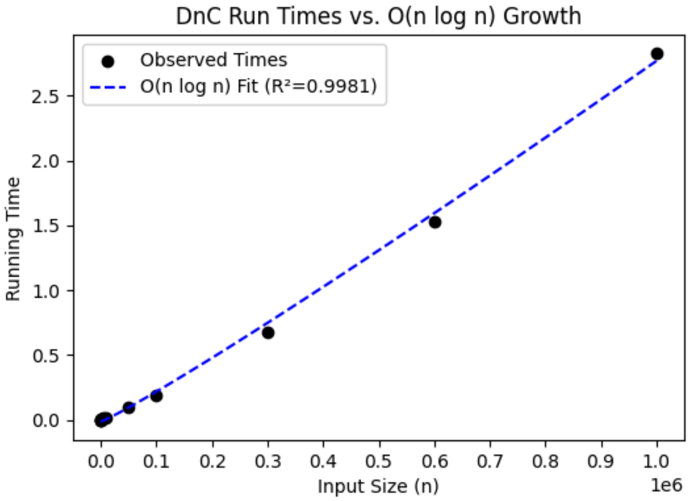
Input size 1,000,000:

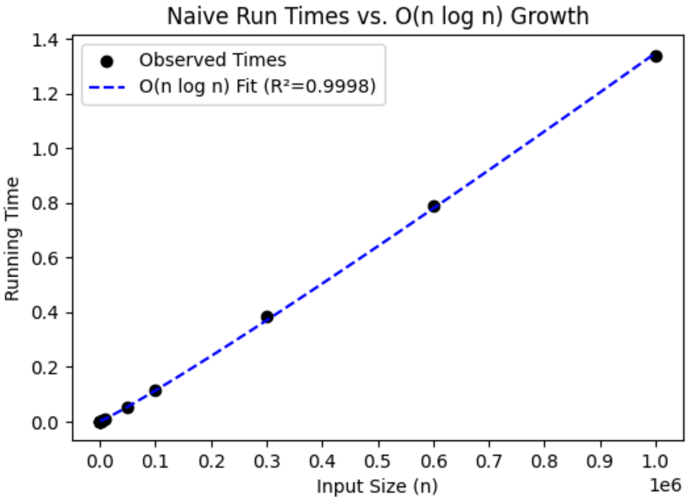
Round	DnC Time (s)	Naive Time (s):
1	2.830418825149536	1.3381831645965576
2	2.809966564178467	1.336704969406128
3	2.8408994674682617	1.3411941528320312
Average Execution time	2.827095	1.338694

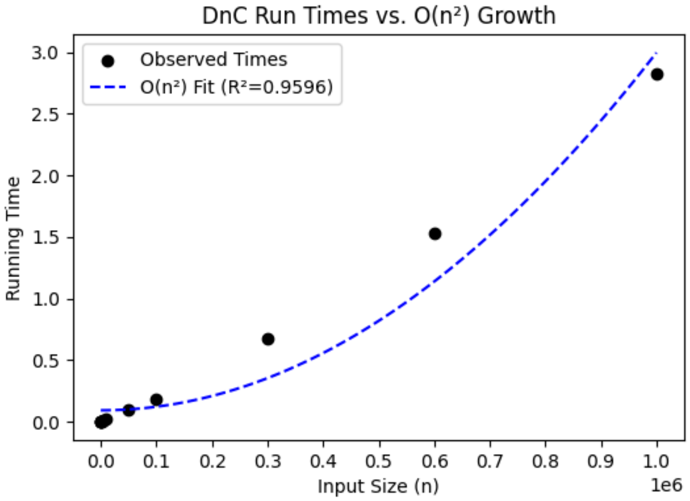
Curve Fits:

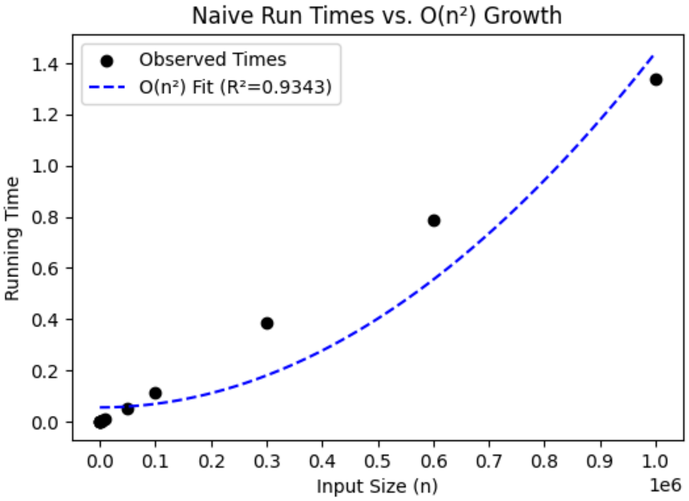

DnC Run Times vs. O(n) Growth


Naive Run Times vs. O(n) Growth


DnC Run Times vs. O(n log n) Growth


Naive Run Times vs. O(n log n) Growth


DnC Run Times vs. O(n²) Growth


Naive Run Times vs. O(n²) Growth

Based on the plot of input size vs. running time for both algorithms, we can see that the O(n*log(n)) fit the best based on R^2 values. Both algorithms did not fit well to n^2, as expected, but the naive one (and to a lesser extent, the DnC one) fit closer to O(n) than expected. I believe that if the sample size increased beyond 1 million points, there would be a sharp increase in the running time which would fit much closer to the increase in n*log(n) over n. However, it's clear the divide and conquer implementation clearly fit the best on O(n*log(n)) based on R^2 and the fact that when I tried to run with higher sizes it took much longer than the ~2.7s observed. I believe this to be due to memory constraints on my machine, leading to very slow access patterns, poor caching, and a tremendously large stack of recursive calls. Overall, I can conclude that the running times match my prediction of O(n*log(n)) for the DnC one and O(n*log(h)) for the naive one (where h is the number of points on the hull) given the nature of the merging algorithm used and the Monotone Chain algorithm I chose for my base case.