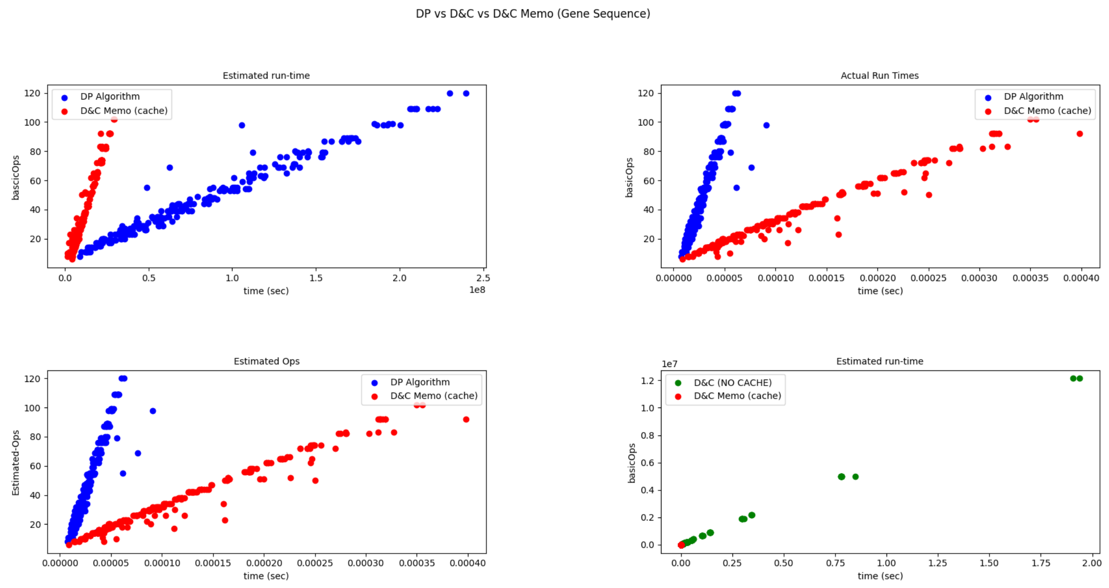
Project 2 Genetic Algorithm

Divide and Conquer vs Dynamic Programming vs Divide and Conquer (Memoized)

Skyler Ballard and Michael Shipley

In this document, one will find the research conducted by Skyler and Michael on a few of the ways of deploying a solution to the Gene Sequencing problem. For the purposes of this document, it is worth clarifying a few things. When we use ‘n’ it can refer to several different contexts. For example, our graphs are formed by testing n times against our algorithm. This can be confusing because n also refers to the size of one of the strings, and in part also the size of the memory complexities.

(Below Skyler’s test)



Now onto the Questions:

Which is generally faster: DP or Memoized D&C? Why?

Dynamic Programming solution is generally significantly faster, especially as the size of the two strings grows.

This is because Dynamic Programming has only to check against a table which is already in memory. But wait, isn't that what Memoized Divide and Conquer is doing? Almost. When Divide and Conquer searches for its memory it does so at the beginning of the algorithm, which means it makes a stack frame for a linear operation. This one difference appears to make all the difference. Really the only difference is the cost of a function call. Looking at the graphs the higher constant c belongs to Divide and Conquer in practice as the estimated ops over time is a more shallow line for D&C, which implies that with less operations it takes more time.

Were your estimated times consistent?

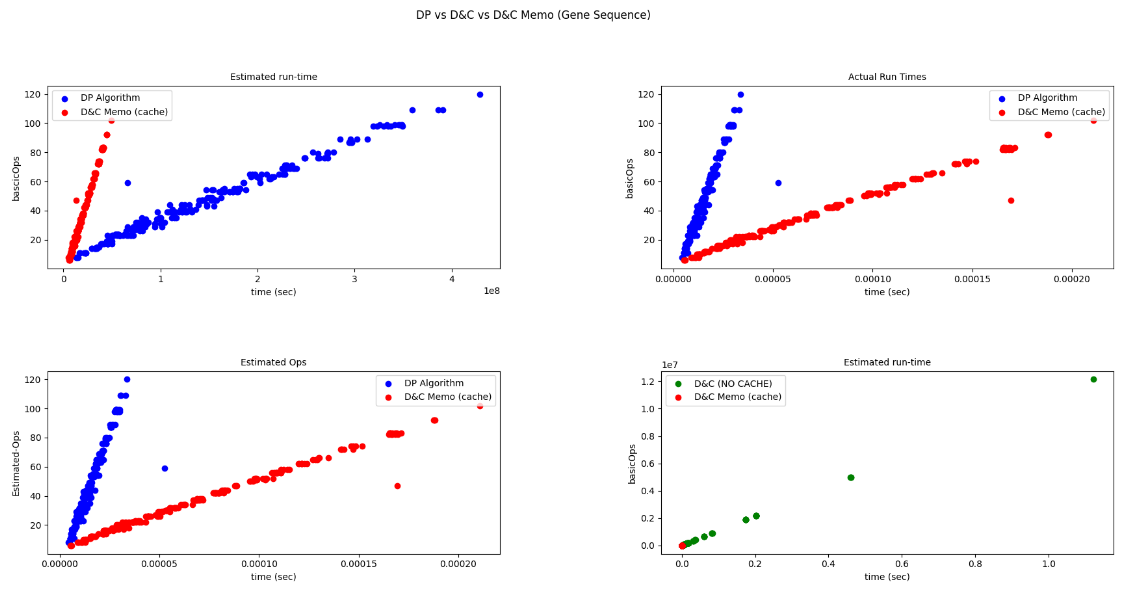
No. This is actually interesting as the deltas of the two algorithms flip with their real run times. I find this generally amusing as the blue and red on the graph look to be mislabeled as a result. They are not. The time taken to make a stack frame are not in our algorithm analysis, so it does make sense that it changes.

How does the number of basic operations for DP and D&C with Memoization compare? Why?

The number of basic ops for DC is lower for its time cost then DP. This is yet again because it takes more time to make a stack frame and that would still be considered a part of one basic op. It is worth noting that the number of ops is remarkably similar despite the time cost. Both tend around 120 with a string's largest size being 10.

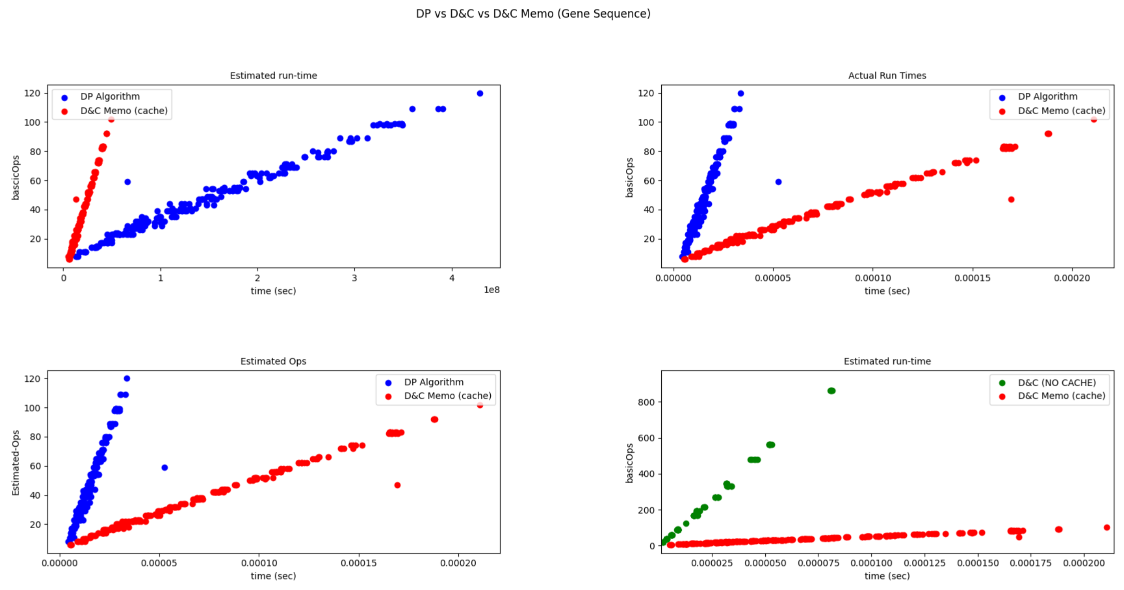
How does the number of basic operations for D&C without memoization compare to D&C with Memoization? Why?

(below Michael's Test:)



It is hard to compare them. Without caching, each operation calls three more operations until each of those recursive trees reaches a base case. This is horrible with any strings larger than size 11. The bottom right graph represents this data visually.

(below Michael's Test: Graph 4 zoomed in)



Here I have zoomed in to the corner where the one red dot appeared to show you that the data was, in fact, still on the graph. That is 100 (red) vs 1.7 million (green).

Were your estimated basic ops consistent for DP and Memoized D&C? Why or Why not?

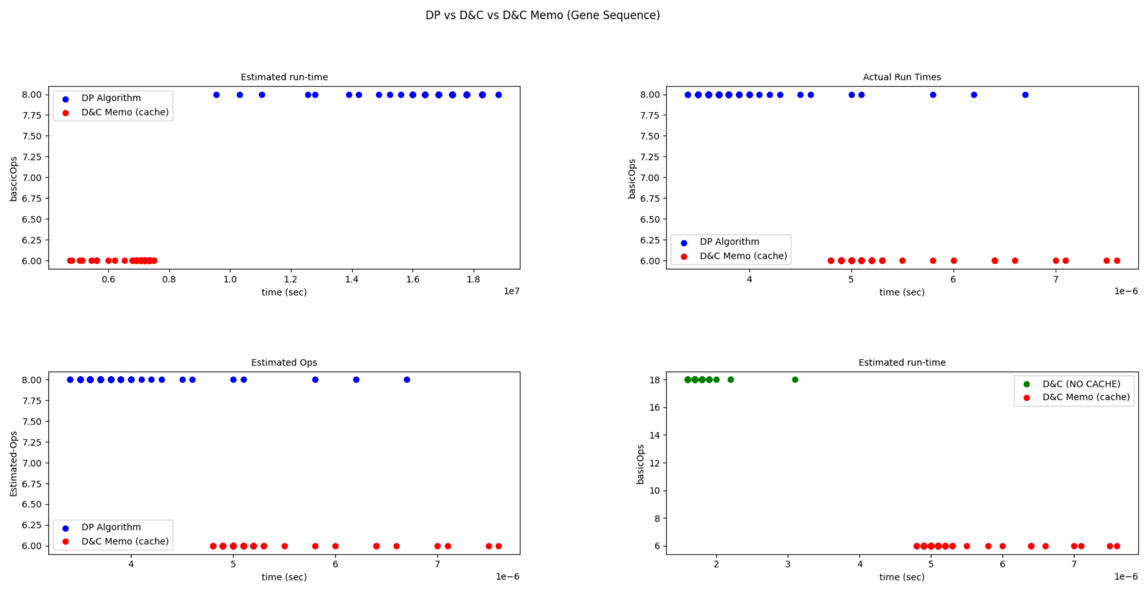
Yes. There is not a variation in the number of ops per string. I'm sure this means the functions themselves have a higher order, but for us this means our estimated ops were incredibly accurate. I actually had to check to see if we were accidentally displaying the same data, we aren't.

How close was your D&C (without memoization) estimated basic op count, based on the recurrence relation, in lecture 9b, to the actual basic operations for D&C (without memoization)?

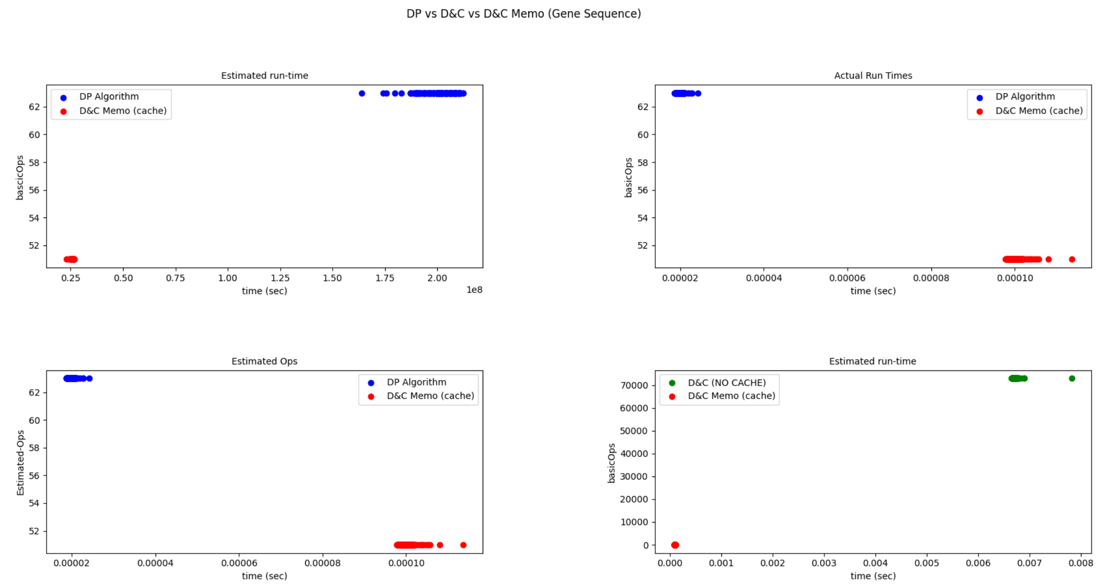
The points on the green graph match that equation root_base_100(k final), which is 100 root 1.2 million = roughly 1.1, which is remarkably what showed up as our estimated run time. So I would say it was close. Since our estimation was close, I do not know what could improve the result.

Bonus:

(below Strings of both Size 2:)



(below Strings of both Size 7:)



These were still run 250 times each. This is meant to display the variance in the data flow, and perhaps give you insight into how our graph was plotting its data. They hold no significant and unique data value otherwise to this analysis.