Project 2 – Dynamic Programming
Implementation and Experimental Analysis

CS 4120/5120 — 50 pts for 5120/60 for 4120
Fall 2021 — Due: Mon, Nov 29, 2021

In this assignment, you will develop a deeper understanding of dynamic programming by coding (1) a straightforward recursive solution (with no dynamic programming), (2) a memorized top-down version, and finally (3) a bottom-up version to solve the following two familiar problems:

a) Computation of the n^{th} Fibonacci number (as defined on page 59)
b) Identifying the optimal parenthesization for a matrix chain multiplication

You will implement three versions of the solution for each of these two problems, and also time them and report the timing results, much as the way you did in Project 1. The results for the two problems need not be compared, but for each problem, the results with the three approaches and for different problem sizes would clearly be of interest. *You will be expected to choose problem sizes that are practical (i.e., that do not cause overflows, and yet run in times that can be distinguished for different problem sizes and the different approaches.* For instance, make sure your programs do not all finish in a fraction of a microsecond, disallowing any possible distinction between the solution variants and different problem sizes. Also, even some modest problem sizes (e.g., 50^{th} Fibonacci number) might take too long in the plain recursive version!

Input (and output):

For the Fibonacci problem, the input for one computation would be a single non-negative integer. You may read in several inputs one after another, but each run should be independent of the other runs, so performance for each input size can be timed. (If you read in increasing input sizes, we do not want results from smaller input sizes to be used by larger input sizes.) Using each of the three approaches, besides computing and reporting the value of the specified Fibonacci numbers, your program should, more importantly, also report the time taken to compute each of the numbers. Sample input will be as follows:

2, 7, 30, 10, 0, 8, 50
(with or without the commas), which means compute the 2^{nd}, 7^{th}, 30^{th}, 10^{th}, 0^{th}, 8^{th}, and 50^{th} Fibonacci number and report results as specified above.
AS A SPECIAL STEP (you may skip it or get up to 10 bonus points): identify the largest n for the n^{th} Fibonacci number that you feel is practical in each approach, and the time it took for that case. (I will let you decide how much of a wait is practical for you, but would just say a minute or two is too short and ten hours is too long. ☺)

For the optimal matrix chain multiplication problem, the input for one run will be a positive integer n (indicating the number of matrices), followed by n+1 positive integers representing the dimensions $p_0, p_1, p_2, \ldots, p_n$, as defined on page 375. Using each approach, your program should output the optimal number of scalar multiplications, the optimal parenthesization, and also the time taken to identify the optimal parenthesization. Sample input will be as follows:

5, 2, 3, 10, 5, 4, 20
6, 30, 35, 15, 5, 10, 20, 25

(with or without the commas). The first input corresponds to our Homework 4 question 4, and the second one corresponds to Figure 15.5.

IMPORTANT NOTE: Although we didn't discuss all versions of the algorithms for the optimal matrix chain multiplication problem in class (we did do so for the cut-rod problem), DO note that the textbook includes all versions, as well as an algorithm to output optimal parenthesization between Sections 15.2 and 15.3; study them carefully.

Your program should handle as many input sets as there are in the input file. You may assume that the input data are correctly entered.

Submit:

Using logically related names (e.g., Fibonacci.cpp, FibonacciInput.txt, Matrix.cpp, MatrixInput.txt, Report.pdf, etc.) for your files, submit the following:

- Your code and the executable (we will also test it)
- Input data
- Results (Can be part of a clearly identified section in the paper below)
- The timing results in tabular and graphical form for easy interpretation and comparison of the different methods for different input sizes (Can be part of a clearly identified section in the paper below)
- A short paper that discusses the results, including any of the following that you can include: introduction, why is dynamic programming interesting/useful, expected vs actual results and timing, any anomalies, explanation for the anomalies, any special observations. If you did the bonus part for Fibonacci numbers, include a brief discussion of your observations and insights for that part, as well.

Approximate Grading Rubric for Project 2 (some changes possible):

Item	Max Possible	CS 5120
Program code:		
Straight recursive approaches	3+4 = 7 points	2+3 = 5 points
Memoized top-down versions	7+7 = 14 points	5+5 = 10 points
Bottom-up versions	7+7 = 14 points	6+6 = 12 points
Runs/results/timing	10 points	7 points
Track time	10 points	6 points
Presentation in paper	5 points	10 pints
Total	**60 points**	**50 points**