Objective: To gain experience with a simple, recursive, divide-and-conquer implementation and how to develop faster solutions using an iterative dynamic programming implementation.

To start the lab: Download and unzip the file at: www.cs.uni.edu/~fienup/cs1520s17/labs/lab6.zip

Part A: We’ve seen several recursive “divide-and-conquer” algorithms: fibonacci and coin-change problem. The general idea of divide-and-conquer algorithms is:
- dividing the original problem it into small problem(s) (e.g., fib(n-1) and fib(n-2))
- solving the smaller problem(s) recursively
- combine the solution(s) to smaller problem(s) to solve the original problem (e.g., return fib(n-1) + fib(n-2))

Mathematics has several simple recursively defined functions. For example, the factorial function can be recursively defined as:

\[ n! = n \times (n - 1)! \quad \text{for } n \geq 1, \text{ and } 0! = 1 \]

Implement a recursive factorial(n) function using this recursive definition and test it with several small examples (e.g., 3! = 3*2! = 3*2*1! = 3*2*1*0! = 3*2*1*1 = 6, and 5! = 5*4*3*2*1*1 = 120).

In Discrete Structures (CS 1800) you used (or will use) the binomial coefficient formula:

\[ C(n, k) = \frac{n!}{k!(n-k)!} \]

In general, (NOTE: When implementing your recursive code, be sure to use DC for the recursive calls)

\[ C(n, k) = C(n - 1, k - 1) + C(n - 1, k) \quad \text{for } 1 \leq k \leq (n-1), \text{ and } \]
\[ C(n, k) = 1 \quad \text{for } k = 0 \text{ or } k = n \]

Implement the recursive “divide-and-conquer” binomial coefficient function using equation (3). Call your function DC(n, k) for “divide-and-conquer”. Notice the difference in run-time between calculating the binomial coefficient using C(24, 12) vs. DC(24, 12), C(26, 13) vs. DC(26, 13), and C(28, 14) vs. DC(28, 14).

Completed your binomial coefficient function, DC, raise your hand and demonstrate your code.

Part C: Much of the slowness of your “divide-and-conquer” binomial coefficient function, DC(n, k), is due to redundant calculations performed due to the recursive calls. For example, the recursive calls associated with DC(5, 3) = 10 would be:
Pascal’s triangle (named for the 17th-century French mathematician Blaise Pascal, and for whom the programming language Pascal was also named) is a “dynamic programming” approach to calculating binomial coefficients.

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
1 & 2 & 1 & \\
1 & 3 & 3 & 1 \\
1 & 4 & 6 & 4 & 1 \\
1 & 5 & 10 & 10 & 5 & 1 \\
\vdots
\end{array}
\]

Recall that dynamic programming solutions eliminate the redundancy of recursive divide-and-conquer algorithms by calculating the solutions to smaller problems first, storing their answers, and looking up their answers if later needed instead of recalculating it. Abstractly, Pascal’s triangle relates to the binomial coefficient as in:

\[
\begin{array}{cccc}
C(0,0) & C(1,0) & C(1,1) & \ldots & C(n-1,k) \\
C(1,0) & C(1,1) & & & \\
C(2,0) & C(2,1) & C(2,2) & & \\
C(3,0) & C(3,1) & C(3,2) & C(3,3) & \\
C(4,0) & C(4,1) & C(4,2) & C(4,3) & C(4,4) \\
C(5,0) & C(5,1) & C(5,2) & C(5,3) & C(5,4) & C(5,5) \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
C(n-1,k-1) & C(n-1,k) & \ldots & \ldots & \\
C(n,0) & C(n,1) & C(n,2) & \ldots & C(n,k) & \ldots & C(n,n-1) & C(n,n) & \ldots & n
\end{array}
\]

For Part C, your job is to implement the “dynamic programming” binomial coefficient function using a Python list of lists to store Pascal’s triangle and loops (no recursion needed). Call your function \( \text{DP}(n, k) \) for “dynamic programming”. **Hints for Part C:**

- Review the dynamic programming fibonacci example from Lecture 9. File lab6/fibonacci.py contains the recursive divide-and-conquer, and two dynamic programming versions of fibonacci. The first dynamic programming version, \( \text{fib}_\text{DP} \), stores the answers to all of the smaller problems. The second dynamic programming version, \( \text{fib}_\text{DP2} \), reduces the amount of memory used by only storing the answers for the previous two smaller problems.

Notice the difference in run-time between calculating the binomial coefficient using \( \text{DC}(24, 12) \) vs. \( \text{DP}(24, 12) \), \( \text{DC}(26, 13) \) vs. \( \text{DP}(26, 13) \), and \( \text{DC}(28, 14) \) vs. \( \text{DP}(28, 14) \).

**Completed your binomial coefficient function, DP, raise your hand and demonstrate your code.**

**Part D: EXTRA CREDIT**

- Your function \( \text{DP}_\text{Extra_Credit}(n, k) \) should use the idea of \( \text{fib}_\text{DP2} \) to avoid storing all the answers to the smaller problems. **Notice that the calculation of the next row in the picture above only needs the previous row and none of the older rows.**