WESTERN NEW ENGLAND UNIVERSITY
COLLEGE OF ARTS AND SCIENCES

CS 366 Design and Analysis of Algorithms October 16, 2019

Programming Assignment #3
Due Date: October 28, 2019

Recall that in Programming Assignment #1, you calculated the binomial coefficient $\binom{n}{k}$ using three different methods: (1) by **definition**, (2) by **cancellation**, and (3) using **recursion**. The objective for that assignment was to realize that methods (1) and (2) are efficient, but are not always accurate due to overflow; method (3) is more accurate, but could be highly inefficient due to the volume of recursive calls. Your new task is to use dynamic programming, both a top-down approach (memoization) and a bottom-up approach, to improve upon the efficiency of the recursive method.

Directions: Add two new methods to your program from Programming Assignment #1. (You are also strongly advised to first fix any errors that you had in that program. See my grading comments or make an appointment with me, if needed.) The specifications are:

Input parameters: any integer $n \geq 0$ and any integer k, $n \geq k \geq 0$ (type `int`)

Return parameter: the value of $\binom{n}{k}$ if n and k are integers with $n \geq k \geq 0$, otherwise return -1. The return type is of type `long`.

1. **Method 4** (memoization): Write a method that uses **memoization** to calculate $\binom{n}{k}$.

The recursive method was built upon the following recurrence:

Pascal's Identity: $$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

with boundary conditions $\binom{n}{0} = 1$ and $\binom{n}{n} = 1$.

Using dynamic programming top-down approach (memoization):

`long memoizedBinomial(int n, int k)`

1. Create an array `c` of values of type long of dimension `n+1` by `k+1`.
2. Fill the array `c` with all values of –1 (flag as not yet calculated).
3. Fill the "boundary values" of the array (where `k == 0` or where `k == n`).
4. Call on / return `memoizedBinomialRecurse(n, k, c)`.

`long memoizedBinomialRecurse(int n, int k, long[][] c)`

1. Use the recursive calculation in `binomialRecurse`, but <u>before</u> making any recursive calls, check if the value has been calculated or not (e.g., is `c[n-1][k-1] == -1`?). Only make a recursive call to `memoizedBinomialRecurse` if the value has not been calculated yet.
2. Once the value(s) have been calculated, store them in the array `c` in the proper location(s).
3. Return the value `c[n][k]`.

2. **Method 5** (dynamic programming): Write a method that uses **dynamic programming** (bottom-up) to calculate $\binom{n}{k}$ using Pascal's Identity and storing values in the array.

   Using dynamic programming bottom-up approach:
   `long dynamicBinomial(int n, int k)`
   1. Create an array `c` of values of type long of dimension `n+1` by `k+1`.
   2. Fill the array `c` from upper left to lower right. (Think about necessary boundary conditions and default values in the array.)
   3. Return the value `c[n][k]`.

3. **Testing**: In your main method, call on these two new methods (in addition to the original three methods). For each method call, determine the processing time by making use of a **system time function**. Test each method **extensively** to determine the accuracy and efficiency of each method.
   In particular, test the limits of the two new dynamic programming methods.
   - How do they perform in terms of accuracy? Are there any incorrect calculations due to overflow error?
   - How do they perform in terms of efficiency?

   Create an output table (or output tables) that shows various values for *n*, *k,* the value calculated for $\binom{n}{k}$ by each of the <u>**five**</u> methods, and the processing time for each of the methods.

4. **Summary**: Analyze the results and briefly describe your observations regarding accuracy and efficiency of the various methods for calculating binomial coefficients.