Data Structures (810:052) Lab #3 Name:________________

Objective: To experiment with recursion, and see the improvement between a recursive divide-and-conquer implementation and an iterative dynamic programming implementation.

Part A: In Mathematics the factorial function is usually written as n!. For example, 5! = 5 x 4 x 3 x 2 x 1. Implement a recursive factorial function, called factorial(n) using the recursive definition:

(1) $n! = n * (n - 1)!$ for $n \geq 1$, and
$0! = 1$ for $n = 0$

After you have implemented AND fully tested your factorial function, raise you hand and demonstrate your factorial function.

Part B: In Discrete Structures (810:080) you will (or have) use the binomial coefficient formula:

(2)
$$C(n,k) = \frac{n!}{k!(n-k)!}$$

to calculate the number of combinations of "n choose k," i.e., the number of ways to choose k objects from n objects. For example, the number of unique 5-card hands from a standard 52-card deck is C(52, 5).

Using your factorial function from Part A, implement the binomial coefficient function **C(n, k),** directly using equation (2).

After you have implemented AND fully tested your binomial coefficient function C(n,k), raise you hand and demonstrate it.

Part C: One problem with using the above binomial coefficient formula (2) directly in most languages is that n! grows very fast and overflows an integer representation before you can do the division to bring the value back to a value that can be represented. (Python does not suffer from this problem because it switch to "long integers" automatically, but lets pretend that it does.)

When calculating the number of unique 5-card hands from a standard 52-card deck (e.g., C(52, 5)) for example, the value of

52! = 80,658,175,170,943,878,571,660,636,856,403,766,975,289,505,440,883,277,824,000,000,000,000

is much, much bigger than can fit into a 64-bit integer representation.

Fortunately, another way to view C(52, 5) is recursively by splitting the problem into two smaller problem by focusing on the hands containing a specific card, say the ace of clubs, and those that do not. For those hands that do contain the ace of clubs, we need to choose 4 more cards from the remaining 51 cards, i.e., $C(51, 4)$. For those hands that do not contain the ace of clubs, we need to choose 5 cards from the remaining 51 cards, i.e., $C(51, 5)$. Therefore, $C(52, 5) = C(51, 4) + C(51, 5)$.

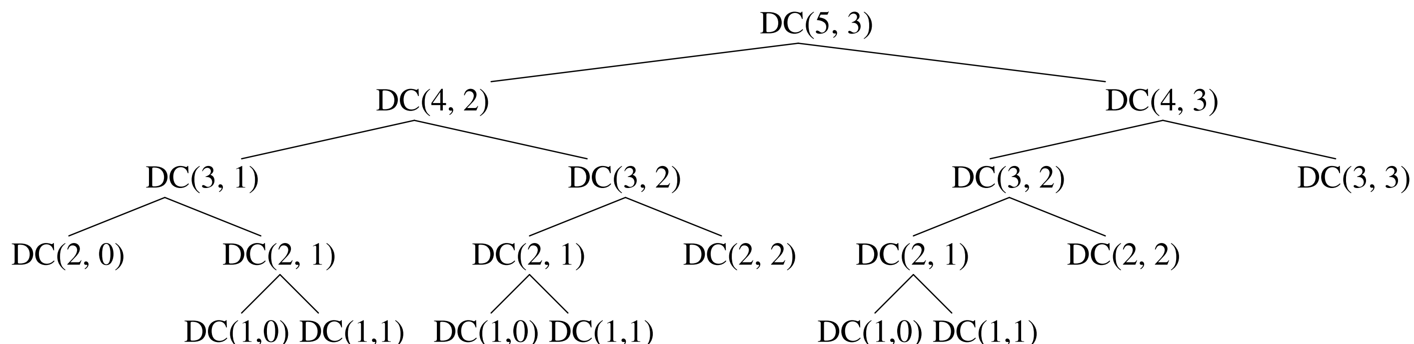
In general,

(3) $C(n, k) = C(n-1, k-1) + C(n-1, k)$ for $1 \leq k \leq (n-1)$, and
$C(n, k) = 1$ for $k = 0$ and $k = n$

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) of Part B vs. DC(24, 12) of Part C, C(26, 13) vs. DC(26, 13), and C(28, 14) vs. DC(28, 14).

After you have implemented AND fully tested your binomial coefficient function DC, raise you hand and demonstrate it.

Part D: 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 call tree for DC(5, 3) = 10 is:



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. Recall that dynamic programming solutions eliminate the redundancy of 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:

```
                                                                              Row #
C(0,0)                                                                          0
C(1,0)  C(1,1)                                                                  1
C(2,0)  C(2,1)  C(2,2)                                                          2
C(3,0)  C(3,1)  C(3,2)  C(3,3)                                                  3
C(4,0)  C(4,1)  C(4,2)  C(4,3)  C(4,4)                                          4
C(5,0)  C(5,1)  C(5,2)  C(5,3)  C(5,4)  C(5,5)                                  5
  .                                                                             .
  .                                                                             .
                                C(n-1,k-1)  C(n-1,k)                           n-1
                                           +
C(n,0)  C(n,1)  C(n,2)  . . .              C(n,k)        C(n, n-1)  C(n,n)      n
```

However, it is general written with numeric values in the form:

```
                                                            Row #
                    1                                         0
                 1     1                                      1
              1     2     1                                   2
           1     3     3     1                                3
        1     4     6     4     1                             4
     1     5    10    10     5     1                          5
                    .
                    .
                    .
```

For Part D, you job is to implement the “dynamic programming” binomial coefficient function using Python lists and loops (no recursion needed). Call your function DP(n, k) for “dynamic programming”. Notice the difference in run-time between calculating the binomial coefficient using DC(24, 12) vs. DP(24, 12), DC(26, 13) vs. DP(26, 13), and DC(28, 14) vs. DP(28, 14).

After you have implemented AND fully tested your binomial coefficient function DP, raise you hand and demonstrate it.