Data Structures and Algorithms, Spring '23 section 1 **Dr. Hristescu**

Lab 7

Assigned: Thursday, March 9
Due: Pre-lab(P0,P1) due beginning of lab, Thursday, March 9
Entire lab due: **Monday, March 20**

Pre-lab: Problem0: Factorial

Write an application that reads in an integer value **n** provided as input and uses the **recursive** factorial method designed in class to display the values of all the factorial of all numbers from 1 to **n**. Display in this order on subsequent lines: **n!** followed by **n** on the same line for increasing values on n. **Make sure you keep and submit the code for Solutions 1,2,3 as well as a write-up with your findings.**

Solution1: Use this program with the primitive type **int** to store the value of the factorial as well as return it when computed. Run it with increasing values of n. Determine the value of **n0** for which this implementation of factorial fails to provide a correct value. Explain the reason it failed. Can the result be improved using another **integer primitive** type, and if so, how?

Solution2: Use this **better integer primitive** data type in the frame you have designed for a). Determine the new value **n1** for which this better factorial fails to provide a correct value. Explain the reason it failed. Can the result be improved even further, and if so, how?

Solution3: Implement and thoroughly test your improved solution for factorial that **does not use** a **primitive** data type. Does this approach have any accuracy limitations? What is the largest number **n2** for which the factorial can be computed?

Include in the conclusions file the write-up the values for **n0**, **n1** and **n2.**

Use the program with very large input to exhaust the virtual address space through:

1. The **Execution Stack** by producing a **StackOverflowError**: run your method on large input.
2. The **Heap** by producing an **OutOfMemoryError: Java heap space**: allocate an otherwise unnecessary but huge array in your recursive method.

For both cases **explain what happened**.

Pre-lab: Problem1: Towers of Hanoi

Test the method designed in class that solves the problem of the Towers of Hanoi. The number of disks (n) should be an input parameter for your program. Your program should:

- Display the **sequence of disk moves** to solve the problem for **n disks**.
- In your program include 2 counters (use an **integer array of size 2 as parameter** for the solve method and let the solve method update it. For both **count in your program**. **Do not evaluate! Compare it with the theoretical analysis results from [Monday 3/6] lecture.**
  - a counter for the **number of calls** to the method that solves the problem recursively and display the value of that counter.
  - a counter for the **number of disk moves** in the recursive solution of the problem and display the value of that counter.

  Input parameters: n
  Output: Display sequence of disk moves, total number of calls, and total number of disk moves to solve the problem.
  Sample runs on 0,1,2,3,4,5.

Problem 1':

Write a simple program that takes a possibly large integer number as command line argument (args[0]) and allocates memory for a **1-dimensional array of integers** of that specified size. Report in your submission the smallest found **value of the integer** for which the memory is exhausted.

Problem2: Binomial Coefficient and Pascal's Triangle

The binomial coefficient may be defined by the following recurrence relation, which is the idea of ***Pascal's triangle***:

$C(n,0) = 1$ and $C(n,n) = 1$ for $n >= 0$

$C(n,k) = C(n-1, k) + C(n-1, k-1)$ for $n > k > 0$

This problem is about designing, implementing and analyzing and testing 3 different methods to compute the Binomial Coefficient as well as generating the coefficients to display Pascal's Triangle.

(A) Method I: Recursive method for computing the binomial coefficient: Write a recursive method to calculate $C(n,k)$ using the formula above. Call this method **BCI.**

Input parameters: n, k
Return value: value of $C(n,k)$.
Sample runs on (0,0), (100,0), (100,1), (100,99), (7,7), (6,4), (6,5), (6,3),(4,2)

(B) ***(picture/scan)*** **Drawing of recursion tree:** Draw the recursion **tree** for calculating $C(7,3)$. Submit as a separate (pdf or jpeg) file not included in allfiles. Make sure the picture looks like a tree, not a graph.

(C) Displaying of Pascal's triangle using iterative method: You **HAVE to use a 2-dimensional array** of size (n+1)by(n+1). Write an iterative (non-recursive) method to generate Pascal's triangle in row-wise manner in the lower

diagonal of the array, that is, in the entries for which $k <= n$. Display Pascal's triangle for n+1 lines (0…n). While there are also other [more memory-efficient] ways to organize the data collection for this approach, this required approach serves as an exercise for working with a 2-dimensional array. Call this method **displayPT.**

Input parameter: n
Output: Display Pascal's triangle with **n+1** lines.
Sample runs on 1,2,3,4,5,6,7,8

(D) Method II: Iterative method for computing the binomial coefficient: Adapt the method developed for **(C)** to compute C(n,k), rather than display the triangle (Compute only the **necessary** coefficients until C(n,k) is generated). As discussed in class, for efficiency make use of symmetry and do not generate anything past the k*-th column.(k* is the smallest of k and n-k). Call this method **BCII.**

Input parameters: n, k
Return value: value of *C(n,k).*
Sample runs on (0,0), (100,0), (100,1), (100,99), (7,7), (6,4), (6,5), (6,3),(4,2)

(E) Method III: Formula-based method for computing the binomial coefficient: Write a program that calculates *C(n,k)* for any given n, k where $n >= k >= 0$ using **either one of these equivalent approaches discussed in the lab**:

1. the efficient **REDUCED** formula (as discussed in class) or
2. the method that using symmetry for efficiency generates **ONLY** the coefficients **on line n** of Pascal's triangle until C(n,k*) is reached

Call this method **BCIII.**

Input parameters: n, k
Return value: value of *C(n,k).*
Sample runs on (0,0), (100,0), (100,1), (100,99), (7,7), (6,4), (6,5), (6,3),(4,2)

(F) Which one of the 3 methods used to compute the binomial coefficient Method I (A) or Method II (D) or Method III (E) is the most efficient and **why**? Include your answer in the conclusions file that will be incorporated in the pdf file.

(Extra credit) Analyze the approximate space and time complexity for each of the 3 methods (A), (D), and (E).

Problem3: Lab7Conclusions: Summarize in a few sentences what you have learned from working on this lab.

Submit: Follow the instructions to turn in the lab electronically using **Lab7** for the name of your directory.
All the code developed for this lab and all sample runs including the ones on the provided input data as well as your conclusions, scan/picture for B as well as the writeup for F.