CPSC 2430 Data Structures

Winter Quarter 2024

Lab 2 **Due: 9:00pm, Tuesday, Jan 16**

Lab 2 focuses on recursion design and time complexity analysis. In this lab, you will rewrite your Lab 1 program in a recursive approach, compare the time complexity between different implementations and explain what causes the difference using Big O notation.

Task 0: Modify your Lab1 function so it only contains the code necessary to generate the Pascal Triangle iteratively (still using a dynamically allocated 2d array). Take out any input/output statements (cin/cout), so they don't affect time measurement in Task 2. Include this function in your lab2.cpp. Your function prototype should look like:

```
int** iterativePascal(int degree)
//returns 2d array filled with Pascal Triangle values up to given degree
```

Task 1: Write function recursiveBico(int degree, int index) using recursion and use it to fill in your Pascal Triangle when called from a recursivePascal function. recursiveBico(int degree, int index) should have two parameters: the first one is the degree of binomial expression, and the second parameter is the ith coefficient. For example, bico(4,2) should return 6 because

$$(x+y)^4 = x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + y^4.$$

Like Lab 1, you will need to

(1) provide a user interface that asks the user for the desired degree and index of the coefficient,
(2) build a 2d dynamic array to store the Pascal Triangle up to the desired degree
    This time you will fill in the values of your Pascal Triangle by calling a ***recursive*** function **recursiveBico(int degree, int index).** *Hint: it will be similar to recursive Fibonacci.*
(3) print out the Pascal Triangle and requested coefficient for each approach (example below)

```
[mjilani@cs1 lab1]$ g++ -Wall -Werror -pedantic -o lab2 lab2.cpp
[mjilani@cs1 lab1]$ ./lab2
Please input the degree of the binomial: 4
Please input the index of the coefficient: 2      Input
Lab 1, iterative pascal triangle:
1
1 1
1 2 1
1 3 3 1                                      Output 1
1 4 6 4 1
The result is: 6
```

```
Lab 2, recursive pascal triangle:
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
The result is: 6
```

Output 2

Task 2: Compare the execution time between your newly implemented recursive bico() and the original you implemented in Lab 1. There are multiple ways to measure your program execution time. For example, using the time library *chrono*.

```
// include chrono library and using its own namespace
#include <chrono>
using namespace std::chrono;

// set a clock before the program executes
auto start = high_resolution_clock::now();

// program execution, call either Pascal Triangle function here
// make sure NOT to include cin or cout operations here,
// as they will invalidate your time execution measurement!

// set a clock after the program executes
auto stop = high_resolution_clock::now();

// get the execution time by calculating the difference
auto duration = stop - start;
auto duration_ns = duration_cast<nanoseconds>(duration);

// output the duration
cout<< duration_ns.count() << endl;
```

Use different input sizes to test the two different approaches. For example, suppose your old bico() in Lab 1 is **bico(int degree, int index)**, and your new recursive bico is **recursiveBico(int degree, int index)**, compare the following inputs:

Task 2.1: Will the first parameter (the degree of binomial expression) affect the execution time when the function is implemented as your Lab 1?

for example, compare the execution time when n is getting larger:

```
bico(3,2)
bico(10,5)
bico(30,10)
```

Task 2.2: Will the first parameter (the degree of binomial expression) affect the execution time when the function is implemented as your Lab 2?

for example, compare the execution time when n is getting larger:

```
recursiveBico(3,2)
recursiveBico(10,5)
recursiveBico(30,10)
```

For both Task 2.1 and Task 2.2 you may want to run the same input a few times and take an average for higher accuracy. Record execution time with correct TIME UNITS!

Task 2.3: Will the implementation approach affect the execution time?

for example, compare the execution time between 2 different approaches when using the same input:

```
bico(3,2)        vs.   recursiveBico(3,2)
bico(10,5)       vs.   recursiveBico(10,5)
bico(30,10)      vs.   recursiveBico(30,10)
```

Include tables or charts as necessary; don't forget to specify the time units for each measurement.

Task 2.4: Use Big O notation to explain your testing results in Task 2.3.

ATTENTION: you don't have to use the parameters in the example shown above. Pick whatever parameter values that could reflect the execution time change on your machine.

Lab 2 Submission

For Lab 2, you need to submit TWO things:

1) Your code in a file named **lab2.cpp** via CS1 server (must compile and be thoroughly tested as usual). To submit, run the script below from the directory containing your lab2.cpp file (assuming your files are on cs1.seattleu.edu):
```
      /home/fac/mjilani/submit/24wq2430/lab2_submit
```
2) A lab report containing all the tasks mentioned above via Canvas

Your report should include:

1. Cover sheet. This is the first page. This page will contain:
   a. Your lab title
   b. Your name
   c. Course name and section number
2. Introduction. This will have two parts:
   a. Background: What is the overall purpose of this lab?
   b. Equipment: what hardware, software, and other materials (if any) were used to complete this lab report?

3. Experiment. This will have three parts (Task 2.1, 2.2 and 2.3 above):
    a. How will the execution time change when giving different degrees of binomial expression for the approach you used in Lab 1?
    b. How will the execution time change when giving different degrees of binomial expression for the approach you used in Lab 2?
    c. What is the execution time comparison between different approaches?

    Include tables or charts as necessary; don't forget to specify the time units for each measurement.
4. Discussion and Conclusion (Task 2.4 above). In this part, you need to use Big O theory to explain the results from your experimental results in Task 2.3.

You can submit your report and code multiple times before the deadline. The last submission will be used for grading.

Lab 2 Grading Breakdown

Breakdown	**Points**	**Note**
Cover sheet	5	Your lab report includes a cover sheet.
Introduction	10	Background: 5pts; Equipment: 5pts.
Experiment	30	10 pts for each task (2.1, 2.2, 2.3).
Discussion and Conclusion	20	Use big O theory to explain your experimental results in Task 2.3.
Code	35	Correct printing of the Pascal Triangle generated with iterativePascal() 5pts And with recursivePascal() 30pts Points will be taken away for lack of functional decomposition, use of globals, improper allocation or deallocation of memory, lack of comments, messy code, etc.