

Graded Lab 3: Recursion vs Iteration

Objective

Evaluate the strengths and weaknesses of recursive algorithms in relation to the time taken to complete the program, and compare them to their iterative counterparts.

Introduction

Recursion can be difficult to grasp, but it emphasizes many very important aspects of programming, including execution speed and complexity. For this lab, you will program two well-known problems in their iterative and recursive forms: “finding the nth Fibonacci number” and “finding the sum of n odd numbers”.

Task 1 – Write code for the following functions.

Part a: Code the following functions for a Fibonacci number (fib) using these prototypes and description:

Prototype: `int iterativeFibonacci(int n);`

Description: This function finds and returns the nth Fibonacci number using iteration (i.e., using loops).

Prototype: `int recursiveFibonacci(int n);`

Description: This function finds and returns the nth Fibonacci number using recursion.

Fibonacci number (fib) is defined as the sum of the previous 2 numbers in its sequence, given that fib (1) = 1 and fib (2) = 1.

$$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$$

For example, fib (3) = fib (2) + fib (1) = 1 + 1 = 2. Similarly, fib (4) = fib (3) + fib (2) = 2 + 1 = 3.

Part b: Code the following functions for sum of n odd numbers using these prototypes:

Prototype: `int iterativeSumNOdd(int n);`

Description: This function finds and returns the sum of n odd numbers using iteration (i.e., using loops).

Prototype: `int recursiveSumNOdd(int n);`

Description: This function finds and returns the sum of n odd numbers using recursion.

Sum of n odd numbers = 1 + 3 + 5 + ... + n

For example, `iterativeSumNOdd (3) = 1 + 3 + 5 = 9`

Similarly, `recursiveSumNOdd (5) = 1 + 3 + 5 + 7 + 9 = 25`

Each of these 4 functions should be contained in its own function file, named to match the name of the function inside. (Ex. the iterative Fibonacci function will be in *iterativeFibonacci.c*). Each of these will be linked to a main.c with the following specification:

1. The main program will take a single command line input (an integer value) from the user and store it in 'n'.
2. The main program will then run each of the four functions one after another, keeping track of the time that each function takes to run to completion. On the last page of this document, you will find a sample code using the **time functions** you may find useful for this lab.

- The main program will then output the results and runtime for each function. A sample input/output scenario is shown on the next page.

Output

Output will be formatted as 2 lines per function. A sample input/output scenario is shown below:

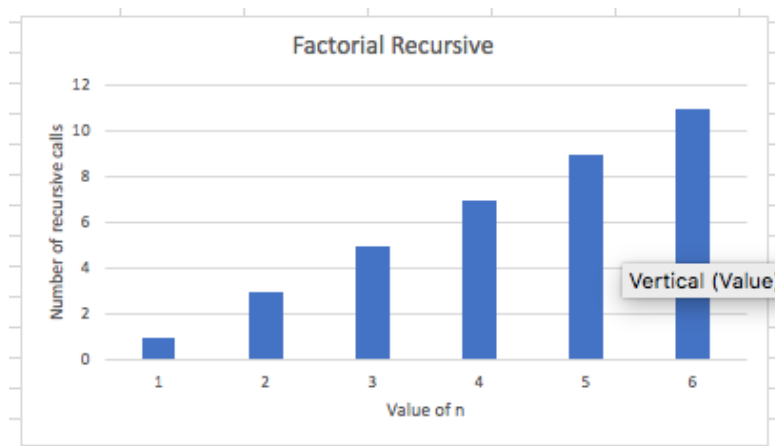
Sample Input / Output:

```
Enter an input value: 4
*****
Sum of 1 + 3 + 5 + 7
Using iteration = 16
Time elapsed for iterative sum is 0.000021 seconds
Using recursion = 16
Time elapsed for recursive sum is 0.000031 seconds
*****
Iteratively, Fibonacci (4) = 3
Time elapsed for iterative Fibonacci is 0.000040 seconds
Recursively, Fibonacci (4) = 3
Time elapsed for recursive Fibonacci is 0.000047 seconds
*****
```

Also note that the displayed time to complete may be different at different times and different on different machines. That happens because the time that a program takes to execute depends on the load that the machine has in that specific moment – so it is normal for it to vary. But it meets the purpose of this lab as it allows us to compare iterative and recursive functions – note that we are not interested in computing the exact time for these functions.

Task 2

You are required to plot 2 graphs for the **2 recursive functions**. You may use Excel or any other tool to plot the graphs. The graph must show the relationship between n and the number of recursive calls made for n . A sample graph is shown for the recursive factorial function discussed in class.



Submission

- Submit your finished code to the lab 3 repository on GitLab. Include a makefile to link the four function files to the main and output an executable named “lab3”. The makefile must also include a ‘clean’ option, which

removes all executables and (if present) intermediate .o files. All compilation should be done with the -Wall and -std=c99 flags.

b. Submit the file with graphs on Gitlab lab 3 repository as a pdf file.

Grading

Grading will be based on the number of successfully generated functions (four algorithm functions plus a working and linked main function), proper submission instructions, correct execution of the program and the question/answer session with TAs during the lab hours. Serious code style offences will incur a small penalty to the final grade for this submission. Warnings may be penalized, and submissions that do not compile will be given a zero.

Tips

- Start with the iterative functions, as they are lengthier and more laborious to complete. More importantly, they'll provide expert knowledge about the algorithm that you'll need to understand the recursive versions.
- The recursive functions, by design are short and succinct. If your recursive functions start to go beyond 50 lines to complete, reconsider your algorithm: there might be a much simpler program that you can build.
- The main function is simple, but will require you to learn about how to time things in seconds and milliseconds. Standard library `#include time.h` has the functions you're looking for. **Page 3 has a sample code that you may use.**
- If you're not seeing a major difference in timings between the functions, try scaling up your program to use larger values of n . Some good numbers to try when scaling up might be 5, 10, 25. If you get up to these values and still don't see a difference, there might be something wrong with your implementation.

Sample code using clock () and time.h

Below is given a code that finds the clock time of a code that runs a for loop 1000000 times. As can be seen in the sample run, values of elapsed time are different for each run. This happens because the time that this program took to execute depends on the load that the machine has in that specific moment – so it is normal for it to vary.

```
~/Documents/GUELPH/Teaching2019_W2021/Winter2021/CIS2500_W21/Winter2021/Labs/Lab3/sampleProgramTime.c
1  #include <stdio.h>
2  #include <time.h>      // for clock_t, clock(), CLOCKS_PER_SEC
3
4  // main function to find the execution time of a C program
5  int main()
6  {
7      double timeElapsed = 0.0;
8
9      clock_t begin, end;
10
11      begin = clock();
12
13      for (int i = 0; i < 1000000; i++) {
14          // just run the loop
15      }
16
17      end = clock();
18
19      /* calculate elapsed time by finding difference (end - begin) and
20       dividing the difference by CLOCKS_PER_SEC to convert to seconds
21      */
22
23      timeElapsed += (double)(end - begin) / CLOCKS_PER_SEC;
24
25      printf("Time elapsed is %lf seconds\n", timeElapsed);
26
27      return 0;
28  }
29
30  Lab3 — -bash — 150x24
31  Ritus-MacBook-Air:Lab3 ritu$ gcc -Wall sampleProgramTime.c
32  Ritus-MacBook-Air:Lab3 ritu$ ./a.out
33  Time elapsed is 0.003987 seconds
34  Ritus-MacBook-Air:Lab3 ritu$ ./a.out
35  Time elapsed is 0.004325 seconds
```