AUBURN UNIVERSITY

CPSC 3283 M4: Programming Assignment

Objective of this assignment:

- To **introduce/initiate** you to the next module topic: *Dynamic Programming*

What you need to do:

1. Read about Fibonacci numbers (*Wikipedia*)
2. Implement a **naïve recursive** (**top-down**) algorithm to compute the *Fibonacci* numbers.
3. Implement an improved recursive (top-down) algorithm to compute the *Fibonacci* numbers.
4. Implement an improved **iterative** (**bottom-up**) algorithm to compute the *Fibonacci* numbers.
5. Measure and compare the performance of the three algorithms.

(**Hint**: while not necessary, a glance at next week's module may help.)

Objective:

The ultimate objective is to introduce you to the next topic: *dynamic programming*. Recursive programming is powerful and often yields elegant algorithms to solve many complex problems. However, sometimes, it may lead to wasteful computations and may produce time **inefficient** solutions. This programming assignment is designed 1) to draw your attention to such inefficient recursive programming using a simple example, and 2) show you alternative efficient solutions that dramatically improve the time complexity.

Fibonacci numbers are the basis of the simple example used for this programming assignment. Fibonacci numbers Fi are defined as follows:

$F_0 = 0$, $F_1 = 1$

$F_n = F_{n-1} + F_{n-2}$ for any number $n \geq 2$

Programming

1) **(5 points) Algorithm 1**: Implement this naïve recursive (top-down) algorithm to compute the Fibonacci numbers:

```
Fibonacci(n)
  if (n == 0)
    return 0;
  if (n == 1)
    return 1;
  return (Fibonacci(n-1) + Fibonacci(n-2));
```

2) **(25 points) Algorithm 2**: The above algorithm is inefficient because it calls the function Fibonacci multiple times to **Re**compute the i^{th} Fibonacci(i) number many times. For example, trace the execution of Fibonacci(6) and compute the number n_2 of times you call Fibonacci(**2**). **Report and document this number n_2 in your final written report** (**worth 10 points**)

In order to improve the efficiency of the recursive algorithm, an idea would be to store the value Fibonacci(i) in some table **Fib**(i) whenever we compute it the **first time**. For the consecutive times when Fibonacci(i) is needed, bring it out of Table ***Fib***. This method is called ***memoization***: it is a trade-off space versus time.

Improve the above algorithm by using *memoization* (**hint**: Use and initialize the table ***Fib***(i) with -1 (-1 is not a Fibonacci number. We will use -1 as a place holder to check whether we already computed Fib(i)). Whenever you need Fibonacci(i), check if Fib(i) != -1 ….).

Provide the pseudocode of your solution (in the report) and implement it using your preferred language. (**15 points)**

3) **(10 points) Algorithm 3**: Implement an **iterative** (bottom-up) version. This means that you must implement an **iterative** algorithm to compute Fibonacci(n). The idea is that you compute the needed values by starting with computing F_0, F_1, F_2, F_3….. until you reach F_n.

Provide the pseudocode of your solution (in the report) and implement it in your preferred language.

Data collection and analysis

For **each** algorithm ***A*** (1) naïve recursive (top-down), 2) recursive with memorization, and 3) iterative (bottom-up)

1) **(3 x 2.5 points) Collect** the execution times $t_i(A)$ to compute the number Fibonacci(i) for ***i*** = 0 to ***i*** = **55** using Algorithm A
2) **(3 x 0.5 point)** What happens when you try to compute Fibonacci(n) for values larger than 60?
3) **(3 x 17 points) Plot** $\frac{t_i}{t_{i-1}}$ versus ***i*** for ***i*** = 1 to ***i*** = 55 for Algorithm A. Discuss the plot in light of the *Golden Ratio*. ***Compare, discuss and analyze*** the results of the three algorithms. (**Hint**: while not necessary, a glance at next week's module may help.)

Report

- Write a report that will contain, explain, and discuss the plot. The report should not exceed one page.
- In addition, your report must contain the following information:
  - whether the program works or not (this must be just ONE sentence)
  - the directions to compile and execute your program
- Good writing is expected.
- Recall that answers must be well written, documented, justified, and presented to get full credit.

What you need to turn in:

- Electronic copy of your source program (standalone)
- Electronic copy of the report (including your answers) (standalone). Submit the file as a Microsoft Word or PDF file.

Grading

- Program is worth 30% if it works and provides data to analyze
- Quality of the report is worth 70% distributed as follows: good plot (25%), explanations of plot (10%), discussion and conclusion (35%).