Case Study – On Fibonacci Numbers/Sequence

The following are the **objectives** of this case study:

1. To **know** more about Fibonacci numbers (or Fibonacci Sequence)
2. To **implement** two (2) different programs/functions that generate the Nth Fibonacci number. Specifically, a non-recursive and recursive implementation of Fibonacci number (in C or Java programming language).
3. To **observe** the relative performance of these two (2) programs/functions.
4. To **strengthen** the discipline of self-study, conducting research, working in a group or with a peer and documentation.
5. To **clarify**, **enhance**, and **strengthen** your knowledge in doing algorithm analysis using frequency counts, solving recurrences, determining and justifying algorithms' worst-case time complexity.

Requirement#1: Write a **C** or **Java** program that will do the following in sequence:

1. A non-recursive function that will accept a positive integer value denoted as **n**. Here **n** represents the **nth** Fibonacci number the function will generate.
2. In the process of finding the nth Fibonacci number, introduce a **count** variable to keep track of the number of operations involved. You should able to discern well as to where in your function is best to place the counting of "important operations".
3. Run the function several times for different values of **n** (in increasing order, up to the highest possible integer value where your function can generate the Fibonacci number). Obtain the processing time for each run and the "count". The processing time should be in milliseconds [if nanoseconds can be obtained, the better].

Document the observations in Step 3 by listing down the data obtained in a table format.

Draw **two** line graphs:

- A line graph with x coordinates representing the values of n and y axis coordinates representing the processing time for each run; and
- a line graph with x coordinates representing the values of n and y axis coordinates representing the number of operations. These 2 line graphs can be overlaid (note: learn how to use Excel or any plotting software for plotting graphs).

Using the non-recursive function created, compute the frequency count for each line, the total frequency count for the entire function and function's big-Oh. Use the definition of Big-oh to justify if the total frequency count is indeed the big-Oh you have concluded. [Solutions must be presented]

Provide a concise discussion about the information gathered (see above). The discussion can include but is not limited to:

- What observations can be made from it? Correlate the count and processing time on all runs made (hint: look at the table and graphs defined)
- Correlate the graphs with the big-Oh defined. Do these two information consistent in illustrating the efficiency of the non-recursive Fibonacci function? Explain elaborately.

Note: Any assumptions made must be clearly documented.

Requirement#2: Repeat the Requirement #1 but this time, implements a recursive function that generates the **nth** Fibonacci number.

1. In the process of finding the nth Fibonacci number, introduce a **count** variable to keep track of the number of times the function calls itself. You should able to discern well as to where in your function is best to place the variable count.
2. Run the function several times for different values of **n** (in increasing order, up to the highest possible integer value where your function can generate the Fibonacci number). Obtain the processing time for each run and the "count". The processing time should be in milliseconds [if nanoseconds can be obtained, the better].

Document the observations seen in Step 2 by listing down the data obtained in a table format.

Draw **two** line graphs:

- A line graph with x coordinates representing the values of n and y axis coordinates representing the processing time for each run; and
- a line graph with x coordinates representing the values of n and y axis coordinates representing the number of operations. These 2 line graphs can be overlaid (note: learn how to use Excel or any plotting software for plotting graphs).

Using the recursive function created, define the function's recurrence and big-Oh. This means that you are expected to solve the identified recurrence relation until a closed-form expression is derived. [Feel free to ask assistance from your Math teacher or someone who is fluent in Math. Make sure that proper acknowledgement is made in your document.]

Provide a concise discussion about the information gathered (see above). The discussion can include but is not limited to:

- What observations can be made from it? Correlate the count and processing time on all runs made (hint: look at the table and graphs defined)
- Correlate the graphs with the big-Oh defined. Do these two information consistent in illustrating the efficiency of a recursive Fibonacci function? Explain elaborately.
- What can you conclude in terms of efficiency given the 2 implementations of Fibonacci? Kindly elaborate/justify your answer.

General Requirements:

1. You are to submit all the works did in this case study (documentations, source codes) in both printed and electronic forms on or before the specified due date. Submission of both versions must be done on the same day. In the event that one of the requirements is submitted at the latter date, that date will be considered as the final submission date of the group or student.
2. Late submission will be penalized at the rate of 10% per day (regardless if it is a week day or a weekend). That is a submission is marked out of 90% for 1 day late, 80% for 2 days late, etc. For example: The case study that is due on a Friday but handed in on the next Monday morning is marked out of 70%.
3. Students have the option not to work on this case study, thus automatically receive a grade of 0 for the case study task. In the event that this case study is to be considered as an incentive or an add-on grade to some task as defined by your lecturer, no credit will be given to the student.
4. A signed declaration of work is needed in both printed and electronic versions.
5. For the electronic version of your submission: use the concatenated last names as filename and **`.c`** or **`.java`** as the extension (as an attachment or place the source codes as part of your email message content). For example if your last names are CRUZ and TAN, the file name should be **`cruz_tan.c`** (for C program) or **`cruz_tan.java`** (for Java program). PDF version on the tables and other needed documents. Other formats apart from your source codes (.c or .java) and PDF will not be accepted and recognized. If you used libraries that are not part of the **`.c`** and **`.java`** package, they have to be submitted as well as attachments. Do not submit or attach your program's executable file version.
6. Email the electronic version to **`projectssubmission@gmail.com`** with subject heading as: DASALGO-CS1-<Lastname of Person 1 (ID#)>-<Lastname of Person 2 (ID#)> For example: DASALGO-ASG2-Cruz(100893454)-Tan(100452323)
7. Lastly, at the last page of your documentation, you are to list down each of the members' names and their exact contribution to this assignment. In addition, any help or assistance received (including online resources, books, journals, articles) should be properly acknowledged and documented.

Submission Date: **February 3 (Tuesday), no later than 3.00pm.** This is true for both printed and electronic form. You have the option to forward your printed version to Ms. Maricar, ST secretary or directly to me. ☺