CG1112 Engineering Principle and Practice II
Semester 2 2018/2019

Week of 28th January 2019
Tutorial 1 Suggested Solutions
Pi, Git and Complexity

Tutorial in CEG EPP II is used to consolidate learning points from the studio(s). Please refer to the respective studio handouts and online materials before attempting the questions. You will get the most benefit by working out the solution before the tutorial session.

1. [Raspberry Pi] You have now used both a Raspberry Pi and Arduino. Find out the differences between these boards in the follow areas:
    a. Power Requirement (Voltage, Current for typical usage).
    b. Hardware Specification (CPU clock speed and runtime memory).
    c. Interfacing capabilities (to other devices, components, networking etc).
    d. Software environment

    Based on the above, suggest a 1-2 scenarios where Pi is more suitable for deployment.

    [For better consistency in discussion, please base your findings on **Raspberry PI 3 model B** and **Arduino Uno**.]

ANS:

	Raspberry Pi 3 Model B	**Arduino Uno**
a	**5v, 800mA (bare board)**	**3.3v, ~50mA**
b	**1.2GHz x 4 (ARM processor quad core), 1GB SDRAM**	**ATmega328 (8-bit CPU, 16MHz clock speed, 2KB SRAM, 32KB flash storage)**
c	**40 I/O pins, USB ports, HDMI port, Ethernet Port**	**20 I/O pins, USB port for power and act as an additional serial port**
d	**Full fledge computer with an OS. Multiple programs can executes in parallel (or in time sharing mode).**	**Microcontroller that run only one task by default.**

Being a computer, Pi is useful when complex software is needed. e.g. running a mini-web server, streaming device, etc.

2. [Git] Consider the following scenario, suggest how to achieve the desired outcome by utilizing Git.
    a. You are the working on a **solo C coding project**.
    b. There is one **function X** in the project that has two **possible implementations A and B (e.g. different algorithms, different data structure etc).**
    c. You want to **try both of the approaches separately**.

    Focus only on functionalities learned in the studio. Discuss the problems with this approach.

ANS:

One possible way is to:

a. Commit the original code without function X (say version 1.0).
b. Implement the first approach A and commit as version 2.0a.
c. **Checkout version 1.0**, then implement the second approach B and commit as version 2.0b.
d. **Check out version 2.0a or 2.0b as needed.**

This is workable but very cumbersome especially when the alternative approaches are much bigger than a single function (e.g. consider the case where you need multiple commits for each version). Git support the **branching** function, where you can split off and maintain two separate lines of work. This is not covered in the studio / course, but you are encouraged to explore on your own.

3. [Complexity] Give the time complexity of the workload functions from Week 2 – Studio 1 using Big-O notation:
    a. WorkA(54321 * N);
    b. WorkB(73 * N);
    c. WorkC(5 * N);
    d. WorkE(N);
    e. [Beyond Scope] WorkD(N);

ANS:

a. Amount of work = O(567) ➔ **O(1),** i.e. constant time	b. Amount of work = O(73*N) ➔ **O(N)**
c. Amount of work = $O((5*N)^2)$ ➔ $O(25*\mathbf{N^2})$ ➔ **$O(N^2)$**	

For (d) and (e), we can use a simple visual explanation for CG1112. Students will learn Master's Theorem in CS2040/C for more complicated case.

For recursive function, do the following:

1. Determine how many functions will be called and note the calling structure.
2. Determine the cost per function call **ignoring all recursive calls**.
3. Sum (2) ➔ total cost.

d. **WorkE(N)**

[WorkE(N)] → [WorkE(N/2)] → [WorkE(N/4)] → …. → [WorkE(0)]

The call is a 1D list with total floor($\log_2$(N)) + 2.

Each call has 1 unit of work.

Total complexity = O(floor($\log_2$(N)) + 2) ➔ O($\log_2$(N))

e. **WorkD(N)**

[WorkE(N)]

[WorkE(N/2)] [WorkE(N/2)]

[WorkE(N/4)] [WorkE(N/4)] [WorkE(N/4)] [WorkE(N/4)]

…..

[WorkE(0)] ……. [WorkE(0)]

The call is a binary tree with height [**floor(Log$_2$(N)) + 2**]

In this case, it is easier to note that each **level has the same amount of work in total.** e.g. Each of the WorkE(N/2) do N/2 work, so that level sum up to N/2 + N/2 = **N**, similarly for the WorkE(N/4) level, where each of the 4 calls do N/4 work ➔ total **N.**

So, total complexity = O(N * height) = O(N * $\log_2$N)

4. [Time & Space Complexity] Consider the following problem:

Given a character strings of N characters, tally the frequency of occurrences for every characters and print out the answer.
Example: "**ab!da!**" (N = 6 characters) Output: **a = 2 times** **b = 1 time** **! = 2 times** **d = 1 time** **a = 2 times** //note the result is printed for every characters in the **! = 2 times** // input string, regardless of duplication.

Suggest **two algorithms** with the following restrictions:

a. Does not store any prior tally, i.e. recalculate the frequency for every characters
b. Use additional memory space to store the prior tally somehow.

You can give pseudo code for the algorithms. Compare the time **and space** complexities of the two approaches. Space complexity apply the same idea as time complexity but focus on memory usage.

ANS:

Approach A – Pseudo Code For I = 0 to N-1 Frequency = 0 Current = String[I] For J = 0 to N-1 if (Current is the same as String[I]) Frequency ++ Print result with String[I] and Frequency
Time complexity = $O(N^2)$
Space complexity = O(1) (only I, J, Frequency and Current, independent of N)

Approach B – Pseudo Code Array Frequency[256], initialized to all zeroes For I = 0 to N-1 Frequency[String[I]]++ //Use Ascii as index For I = 0 to N-1 Print Result with String[I] and Frequency[String[I]]

Time complexity = O(N)
Space complexity = **O(1)** (only I, J, Frequency[256] and Current, independent of N)

In this case, Approach B is the obvious winner. In general, time and space are two resources that are commonly in tradeoff relationship, i.e. we can spend more memory space in order to reduce the time spent or vice versa. For example, there are many cases where we can do pre-processing on the data and store the information to help with future calculation.