CS 415, Section 001 Sonoma State University Spring, 2023

Algorithm Analysis

Instructor: Henry M. Walker
Lecturer, Sonoma State University
Professor Emeritus of Computer Science and Mathematics, Grinnell College

**Although much of this course is well developed, some details can be expected to evolve as the semester progresses.
Any changes in course details will be announced promptly during class.**

Worksheet on Analysis of Non-recursive Algorithms, Recurrence Relations, and Program Format

This worksheet is organized into three somewhat-related parts:

- analyzing non-recursive algorithms
- developing recurrence relations, and
- recurrence relations and program format.

References

A. Many Web sites provide summation formulas for sums of the form:

$$1^a + 2^a + 3^a + ... + n^a$$

for various constants a (e.g., a = 1, 2, 3, ...).

One such reference is Common Computational Formulae from the course's home page.

B. All programs submitted this semester must follow the SSU CSS Style Guide for C/C++

Algorithmic Analysis

This section complements the previous Worksheet on the Analysis of Non-recursive Algorithms.

1. Consider the following code segment:

```
int exercise1(int n) {

int sum = 0;

for (int i=1 ; i <= n*n*n ; i++)
```

```
   for (int j=1 ; j <= i ; j++)
      sum++;

return sum;
}
```

a. Perform a careful micro-analysis of the time required for each line of this code. In your analysis, use the following constants for the time it takes a machine to do specific operations:

   - A — one assignment
   - B — one evaluation of a Boolean (e.g., a comparison)
   - S — one addition or subtraction
   - M — one multiplication or division
   - P — one increment (e.g., ++) [since the line `w++` is largely equal to `x = x + 1`, P is roughly equal to A + S]
   - R — one function/procedure return

   Note: In your answer, be sure to give a separate time required for each line of the code. Likely, the amount of time for a line will involve some expression involving at least some of the variables n, A, B, S, M, and P.

b. Once time is determined for each line, add to obtain the total time needed.
c. If your answer for the total time involves a long summation of terms, use one or more Common Computational Formulae to obtain a relatively simple algebraic expression for the total time.
d. Use your result for the total time to determine the Big-O, Big-Ω, and Big-Θ running times for `exercise1`.
e. Explain your conclusions for Big-O, Big-Ω, and Big-Θ, by indicating what term[s] in the expression for total time will dominate when `n` is large. That is, for large `n`, which terms for total time will dominate and which will have minimal impact, and why?

2. Consider the following code segment:

```
 int exercise2(int n) {

 int sum = 0;

 for (int i=1 ; i <= 2*n ; i++)
    for (int j=1 ; j <= i*i*i ; j++)
       sum++;

return sum;
 }
```

   Repeat steps a-e from Exercise 1 for this code.

3. Consider the following code segment:

```
int exercise3(int n) {

int sum = 0;

for (int i=1 ; i <= n ; i*= 2)
   sum++;
```

```
        return sum;
        }
```

Repeat steps a-e from Exercise 1 for this code.

4. Consider the program simple-loop-analysis-2.c.

   a. Following steps a-e from Exercise 1, give a micro-analysis of the `iter_compute` procedure to describe its run time.

   b. On the basis of your answer to part a in this problem, determine if the resulting code has $\Theta(n)$, $\Theta(n^2)$, $\Theta(n^3)$, $\Theta(n^4)$, and $\Theta(n^5)$. In each case, give a careful argument justifying your conclusions.

   c. On a particular machine and compiler, suppose the running time for this code is 5 seconds when n is 800. Based on your answers earlier in this problem, estimate the likely time for the running of this code when n is 1600 and when n is 8000. Justify your answer briefly.

Recurrence Relations

5. Referring to program simple-loop-analysis-2.c, develop a recurrence relation that describes the run time of `rec_compute` procedure. (For this problem, you need not solve the recurrence relation — but wait until later in the semester.)

Recurrence Relations AND Program Format

6. Consider program fourthPower.c, which computes the fourth power of a positive integer. (Note that part a is not required, but you are expected to complete parts b, c, d, and e.)

   a. (optional): Run the program several times to check that it works properly in computing the fourth power of postive integers.
   b. (to be submitted): The program is terribly formatted and hard to read. Reformat the program, so that it follows the C/C++ Style Guide
   c. (to be submitted): Run Doxygen on the reformatted program, and print the resulting html pages.
   d. (to be submitted): Develop a recurrence relation that describes the run time for the `computeFourth` function.
   e. (to be submitted): Use the method of backward substitution (from Levitin, Section 2.4) to determine a solution to the recurrence relation from part d.

   Notes:
   - For this problem, turn in a listing of the reformatted program, a print out of the html pages from Doxygen, and your work that leads to the recurrence relation and its solution for the `computeFourth` function.

created December, 2021
revised December-January 2021
expanded August 1, 2022
revised January, 2023
revised Summer, 2023

For more information, please contact Henry M. Walker at walker@cs.grinnell.edu.