605.621 Foundations of Algorithms Fall 2022
Programming Assignment #2
Assigned with Module 3, Due at the end of Module 7

The goals of Programming Assignment 2 are: (1) to have you express your understanding of recursion trees through a program, and (2) to exercise the algorithm analysis techniques you studied in Modules 1 and 2.

This assignment requires you to "construct an algorithm". There are, of course, many algorithms solving this problem. Some algorithms are asymptotically more efficient than others. Your objectives are to (a) complete the assignment, and (b) submit an asymptotically efficient algorithm.

In this course, critical thinking and problem analysis results in discovering appropriate and better, the best, algorithm for a problem; defining the algorithm in pseudocode; demonstrating (we don't usually prove) that the algorithm is correct; and determining the asymptotic runtime for the algorithm using one or more of the tools for asymptotic analysis you have studied this semester.

This programming problem is not a collaborative assignment. You are required to follow the **Programming Assignment Guidelines** and **Pseudocode Restrictions** (Canvas page *Pseudocode Restrictions & Programming Guidelines*) when preparing your solutions to these problems.

Exercising the algorithm you design using one or more example data set(s) removes any doubt from the grader that the algorithm is structurally correct and all computations are correct. When a problem supplies data you are expected to use it to demonstrate that your algorithm is correct.

You are permitted to use Internet resources while solving problems, however complete references must be provided. Please follow the Sheridan Libraries' citation guidance at "*Citing Other Things* - HOW DO YOU CITE AN INTERVIEW, A TWEET, OR A PUBLIC WEB PAGE?" and continue to the APA Academic Writer *Sample References*. Additional example citations are provided at the Purdue University, Purdue Online Writing Lab (OWL), College of Liberal Arts *Reference List: Electronic Sources*. You can also use training provided by Victoria University Library, Melbourne Australia on *Harvard Referencing: Internet/websites*

1. **Algorithm Requirements**

   Design an algorithm that generates a recursion tree based on **Definition 3.5** Recursion tree rules, contained in the *ImprovedRecursionTreeMethod.pdf* in Module 2's Content and included on the PA2 page. In this assignment you will substantially extend the algorithm defined by **Definition 3.5**.

   Definition 3.5 is an algorithm (a step-by-step procedure for solving a specific problem with defined resources) but it is not written the style of our text and is not yet in compliance with the **Pseudocode Restrictions**. The figures in the *ImprovedRecursionTreeMethod.pdf* provide a guide to applying this algorithm.

   You will augment that algorithm, as necessary, to accept as input the following recurrence function forms:

   - Divide-and-Conquer: $T(n) = aT(n/b) + f(n)$, where constants
     - a is an integer such that $a \geq 1$
     - $b > 1$, and b is a rational number
     - Note you are **not** being asked to accept $T(n) = aT(n/b) + gT(n/h) + f(n)$ recurrence relations

   - Chip-and-Be-Conquered: $T(n) = aT(n - b) + f(n)$, where constants
     - a is an integer such that $a \geq 1$
     - $b > 0$, and b is an integer

   Your algorithm that must accept the following $f(n)$ forms:

   - polynomial - cn^d where
     - $0 < c < \infty$, and c is a rational number
     - $0 \leq d < \infty$, and d is a rational number
     - Example: (1/2)n^(2/3)

   (continued on next page)

- **[Extra Credit]** logarithmic- $c \log_e^d(n)$, where
  - $0 < c < \infty$, and c is a rational number
  - $0 < d < \infty$, and d is a rational number
  - $0 < e < \infty$, and e is a rational number
  - Example: (1/2)log(base 2/3)^(2/1)(n)
  - This is the form expressed in our text pages 56-57

The algorithm you design and the code you implement must meet the following objectives:

- Your must generate the recursion tree for any combination of above recurrence function and $f(n)$ forms. You must clearly output the recursive cost and nonrecursive cost of each node of the recursion tree.
- You must generate at least depth = 0, 1, 2, 3 of the recursion tree.
- Your recursion tree need not be a graphical representation of a tree.
- Your algorithm must compute the total nonrecursive cost at each depth of the recursion tree using rational numbers. Fractions must be represented by the least common denominator (LCD) of individual nonrecursive costs at a depth. The following Java programs are provided to help in this part of the assignment.
  (a) *Rational.java* - a rational number class for your use
  (b) *RationalNumbers.java* - a test program for the rational number class (just for your use in understanding rational number class)

2. **Required $T(n)$ Expressions Test Cases**
All page numbers and other identifiers are from the CLRS text.
You must submit solutions for each of these recursion forms with the exception of the extra credit items. If you elect to implement the logarithmic form you must also submit the solution to the extra credit item below.

   - Divide-and-Conquer: $T(n) = aT(n/b) + f(n)$
     - page 84, equation (4.9) $T(n) = 8T(n/2) + \Theta(1)$
     - page 86, equation (4.10) $T(n) = 7T(n/2) + \Theta(n^2)$
     - page 96, figure 4.1 $T(n) = 3T(n/4) + cn^2$
   - Chip-and-Be-Conquered: $T(n) = aT(n - b) + f(n)$
     - page 94, problem 4.3-1 (a) $T(n) = T(n - 1) + n$
     - page 95, problem 4.3-3 $T(n) = 2T(n - 1) + 1$
     - page 119, problem 4-1 (h). $T(n) = T(n - 2) + n^2$
     - **[Extra Credit]** page 121, problem 4-4 (h). $T(n) = 2T(n - 1) + lgn$

3. **Scoring**

   - [25 points] Attributes of your algorithm:
     (a) [10 points] Algorithm expressed using pseudocode consistent with the **Pseudocode Restrictions**?
     (b) [10 points] What is the worst-case big-O asymptotic running time for your recursion tree generator algorithm?
     (c) [5 points] Does the worst-case big-O asymptotic running time vary depending on the recurrence function and/or the $f(n)$ forms?

   - [45 points] Attributes of your implementation of the algorithm & code:
     (a) [10 points] Generate the recursion tree for the Divide-and-Conquer recurrence function forms?
     (b) [10 points] Generate the recursion tree for the Chip-and-Be-Conquered recurrence function forms?
     (c) [10 points] Accept polynomial $f(n)$ forms?
     (d) **[Extra Credit 10 points]** Accept logarithmic $f(n)$ forms?
     (e) [5 points] Generate the required four depths of the recursion tree?
     (f) [5 points] Generate output representing all nodes at a depth in the recursion tree?
     (g) [5 points] Compute the total nonrecursive cost at each depth of the recursion tree in LCD form?

(continued on next page)

- [20 points] Attributes of your code:
  (a) [5 points] Code follows a reasonable, consistent style with reasonable documentation, with appropriate use of structures, modularity, error checking
  (b) [5 points] Trace runs documenting that your program executes correctly.
  (c) [10 points] Instrumentation and tests to measure the asymptotic behavior of your program.

- [10 points] Analysis comparing your algorithm's worst-case big-O asymptotic running time to the asymptotic behavior of your program. In this assignment, you must probe your code's worst-case big-O asymptotic running time not by increasingly larger input length n but by varying the parameters of your recurrence functions and $f(n)$ forms.

(end of the assignment)