Missouri University of Science & Technology — Department of Computer Science

Spring 2022 — **CS 2500: Algorithms (Sec: 102)**

Homework 1: Foundations

Instructor: *Sid Nadendla* — **Due:** *February 7, 2022*

Problem 1 Asymptotic Analysis *20 points*

1. Prove the following statements:
   (i) For any $0 < x < 1$, $e^x = \sum_{i=0}^{m-1} \frac{x^i}{i!} + \Theta(x^m)$, for all $m = 1, 2, \cdots$.
   (ii) $\log(n!) = \Theta(n \log n)$. (Hint: Use Sterling's approximation for $n!$.)
2. Solve Problems 3.1, 4.1(a,f,g) and 4.3(c,h,j) in *CLRS*.
3. Use recursion-trees to solve the recurrence $f(n) = 2f(n/2)+n$, when $n = 2^k$ for any $k > 1$.

Problem 2 Validation ***20 points***

In the class, we studied different kinds of algorithms. This problem is about validating the theoretical analysis of the algorithms presented in the class.

1. Compute the average running time of *Insertion Sort*, *Merge Sort* and the *maximum element* formally.
2. Implement the algorithm to find the *maximum element* in Python, and validate its correctness and average run-time performance via programming, for different input sizes.
3. Implement *Merge Sort* algorithm in Python, and validate its correctness and average run-time performance via programming, for different input sizes.

Problem 3 Matrix Multiplication[1] ***20 points***

Consider two $n \times n$ matrices A and B. The goal is to compute the product matrix $C = A \times B$ with elements

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{ik},$$

for all $i = 1, \cdots, n$ and $j = 1, \cdots, m$.

CLRS (our textbook) presents three different algorithms to compute the above product:

A. SQUARE-MATRIX-MULTIPLY (Page 75)

B. SQUARE-MATRIX-MULTIPLY-RECURSIVE (Page 77)

C. Strassen's algorithm (Pages 79-82)

In this problem, we will study these three algorithms using the labels provided in the above list.

1. Prove that Algorithms A and B are correct.
2. Compute the running time of Algorithm A, and show that it is $\Theta(n^3)$.
3. Prove that the running time of Algorithm B is

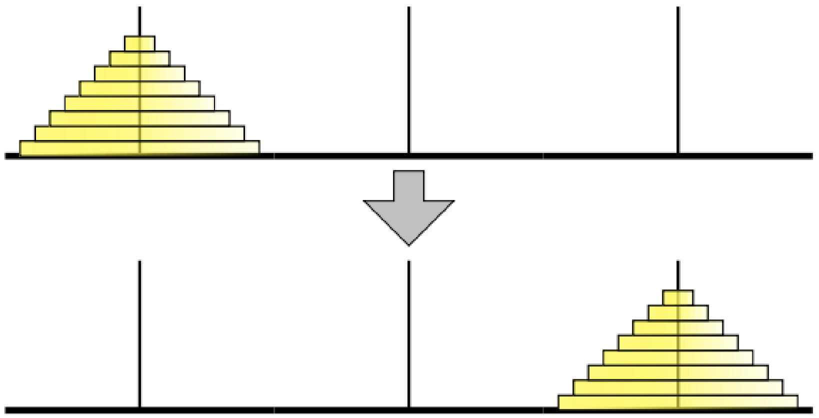
$$T(n) = \begin{cases} \Theta(1), & \text{if } n = 1, \\ 8T(n/2) + \Theta(n^2), & \text{if } n > 1. \end{cases}$$

   Also, use a recursion tree to find an asymptotically tight bound.
4. Write a pseudocode for Algorithm C.
5. Implement Algorithms A, B and C in Python, and validate the running time for different input sizes.

[1]This problem is inspired from Section 4.2 in *CLRS*.

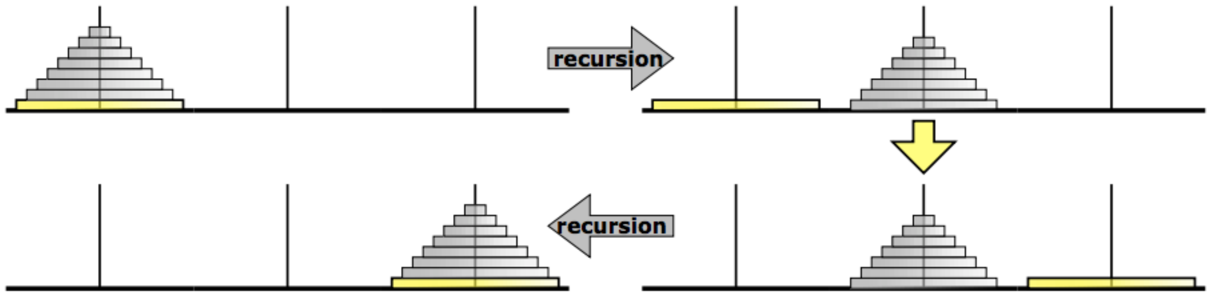
Problem 4 Towers of Hanoi/Brahma *20 points*

There are n disks of different sizes arranged on a peg in decreasing order of sizes. There are two other empty pegs. The purpose of the puzzle is to move all the disks, one at a time, from the first peg to another peg in the following way. Disks are moved from the top of one peg to the top of another. A disk can be moved to a peg only if it is smaller than all other disks on that peg. In other words, the ordering of disks by decreasing sizes must be preserved at all times. The goal is to move all the disks in as few moves as possible.



1. Design an algorithm (by induction) to find a minimal sequence of moves that solves the towers of Hanoi problem for n disks.
2. How many moves are used in your algorithm? Construct a recurrence relation for the number of moves, and solve it.

The following picture gives you a hint for designing your algorithm:



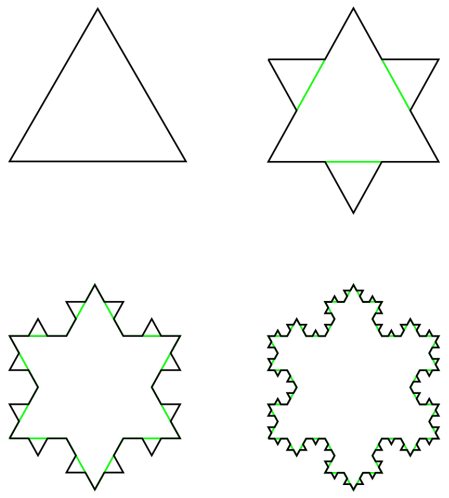
Problem 5 Koch Snowflake *20 points*

Fractals are recursive shapes that are used extensively in both 2D and 3D graphics in the movie, gaming and VR industries. In order to understand how recursion is used in constructing fractals, let us consider a simple fractal called *Koch snowflake* (shown below) which was proposed by Von Koch in 1904, decades before Mandelbrot presented a formal theory on fractals.

The Koch snowflake can be constructed by starting with an equilateral triangle as its base shape, and then recursively altering each line segment as follows:

- divide the line segment into three segments of equal length.
- draw an equilateral triangle that has the middle segment (green color) from step 1 as its base and points outward.
- remove the line segment that is the base of the triangle from step 2.

At the n^{th} iteration, the resulting curve is called the Koch snowflake of order n.



In this problem, we will study the construction of Koch snowflake and its recursive properties.

1. Write the recurrence relation for the perimeter and area of n^{th} order Koch snowflake.
2. Write the pseudocode for the above algorithm.
3. Compute the running time of your algorithm.
4. Write a program in Python to draw Koch snowflake of order n using the graphics package *Turtle*. For details about this package, please visit the following webpage:

   https://docs.python.org/3/library/turtle.html