Plan for 241 day 1: Design and Analysis of Algorithms

Read chapters 1 through 4. Lectures will start by covering asymptotic analysis (Ch. 3) and analysis of recursive algorithms (Ch. 4). Chapter 2, sections 2.1 and 2.2 give examples of how to analyze an iterative algorithm. Section 2.3 gives example of how to analyze recursive algorithms. Both algorithms in chapter 2 are solutions for the sorting problem.

___ Start by reading syllabus, a handout. Ask: who already knows LaTeX?

___ Roll call, find out who is/isn't enrolled.

___ Introduce terms: Algorithm, Algorithmic problem, Algorithm instance.

a) algorithm: computational procedure that transforms valid input(s) into valid output. The algorithms we study will also terminate and be provably correct. Most will be deterministic.
b) algorithmic problem: the set of all possible input instances and their corresponding outputs.
c) algorithm instance: one possible input to the algorithm.
Introduce sorting problem as an example (more later).

___ Given a problem, P, there may be many different solutions: A1 A2. . .Ak These algorithmic solutions may differ in efficiency and we want to find most efficient.

___ Measures of algorithm efficiency:

a) Efficiency of algorithms usually expressed only in terms of TIME taken on input size n, as n increases to infinity.

b) Another measure of efficiency is SPACE: the amount of extra memory taken beyond what is needed to store input (n).

___ Coding generally starts with a solution of the problem in pseudocode. Mention the biggest difference between pseudocode we normally write and the textbook's pseudocode: The book uses 1-based array indices. There are no braces or end statements, just indentation. Many of the steps are stated in English.

___ For this week, concentrate on chapters 3 and 4: Asymptotic analysis and Recurrence relations.

Observation 1: An algorithm that does anything with its input takes longer on larger input sizes.

Observation 2: An algorithm without iteration or recursion has running time that is not dependent on the size of the input.

___ Simplifications that make algorithms comparable:

- Notation: Asymptotic Analysis requires use of special terminology to classify the order of growth of time (or space) taken to run a particular algorithm as the data size n increases without bound.

- Computational model: A single-processor RAM machine that processes lines of the algorithm sequentially.

Observation 3: We are rarely interested in the running time of an algorithm on small inputs. Asymptotic Analysis allows us to ignore constant multiples and lower order terms in mathematical expressions.

___ Big Oh: $f(n) = \mathcal{O}(g(n))$ if there exist constants $c > 0$ and $n_0 \geq 1$ for which $f(n) \leq c(g(n))$ for some $c > 0$ and all $n \geq n_0$.

If $f(n) \leq cg(n)\ \forall\ n \geq n_0$ where $c > 0$ and $n_0 \geq 1$, then $f(n) = \mathcal{O}(g(n))$.

Ex: $2n^2 = \mathcal{O}(n^3)$ with $c = 1$ and $n_0 \geq 2$.

Observations:
$n^2 = \mathcal{O}(n^2), n^2+n = \mathcal{O}(n^2), n^2+1000n = \mathcal{O}(n^2), n^2 = \mathcal{O}(n^3), n^2 = \mathcal{O}(n^4)$

Show $n^2 + n \leq c(n^2)$. Choose $c = 2$ and $n_0 = 2$, then we have $4 + 2 \leq 2 * 4$ and $6 \leq 8$, so correct. This holds for $c = 2$ and all $n \geq 2$.

Ex: $f(n) = 5n + 4$ and $g(n) = n$. To show $f(n) = \mathcal{O}(g(n))$, let $c = 6$ and $n_0 \geq 4$. Then $c(g(n)) = c * n_0 = 6 * 4 = 24$ and $f(n) = 5 * 4 + 4 = 24$. So $f(n) \leq cg(n)$. This holds for $c = 6$ and all $n \geq 4$.

Observations:
$n = \mathcal{O}(n), n = \mathcal{O}(n^2), n = \mathcal{O}(n^3), n = \mathcal{O}(2^n), n = \mathcal{O}(n!)$

___ Big Omega: $f(n) = \Omega(g(n))$ if there exists a constant $c > 0$ and a value of $n_0 \geq 1$ for which $f(n) \geq c(g(n))$ for some c and all $n \geq n_0$.

If $f(n) \geq cg(n)\ \forall\ n \geq n_0$ where $c > 0$ and $n_0 \geq 1$, then $f(n) = \Omega(g(n))$.

Ex: $f(n) = 5n + 4$ and $g(n) = n$. To show $f(n) = \Omega(g(n))$, let $c = 5$ and $n_0 \geq 1$. $5n + 4 \geq cn$ is true for $c = 5$ and $n_0 \geq 1$. We have $5 * 1 + 4 = 9$ and $5 * 1 = 5$ and $9 \geq 5$. This holds for $c = 5$

and all $n \geq 1$.

Observations:
$n = \Omega(n), n = \Omega(lgn), n = \Omega(lglgn), n = \Omega(\sqrt{n}), n = \Omega(n^{0.9999})$

Ex: $n^{(1/2)} = \Omega(lgn)$, with $c = 1$ and $n_0 \geq 16$.

Observations:
$n^2 = \Omega(n^{1.999}), n^3 = \Omega(n^2), n^2 = \Omega(lgn), n^2 = \Omega(lglgn)$

Show $n^2 - n \geq c(n^2)$. Choose $c = \frac{1}{2}$ and $n_0 \geq 2$; then we have $4 - 2 \geq 1/2 * 4 = 2 \geq 2$.

Choose $c = \frac{1}{4}$ and $n_0 \geq 4$; then we have $16 - 4 \geq 1/4 * 16 = 12 \geq 4$.

___ Big Θ: $f(n) = \Theta(g(n))$ if there exist constants $c1, c2 > 0$ such that $\forall\ n \geq n_0$ we have $0 < c1(g(n)) \leq f(n) \leq c2(g(n))$.

$f(n) = \Theta(g(n))$ iff $f(n) = \mathcal{O}(g(n))$ and $f(n) = \Omega(g(n))$.

Ex: $n^2/2 - 2n = \Theta(n^2)$, with $c1 = 1/4, c2 = 1/2\ \ \forall\ \ n_0 \geq 8$.
(this is given on page 37 of CLRS IM)
$(8^2/2 - 2 * 8) \geq 1/4(8^2)$? $(64/2 - 16) = (32 - 16) = 16 \geq 16$.

$(8^2/2 - 2 * 8) \leq 1/2(8^2)$? $16 \leq 32$. True for all $n_0 \geq 8$. Therefore, $n^2/2 - 2n = \Theta(n^2)$.

Try with $n_0 = 10$:
$c1 = 1/4$, $c2 = 1/2$ and $n_0 \geq 10$
$(10^2/2 - 20) \geq 1/4(10^2)$? $(50 - 20) = 30 \geq 1/4(100)$ or $30 > 25$.

$(50 - 20) \leq 1/2(10^2)$? $30 < 50$.

___ little oh (strict upper bound) and little omega (strict lower bound).

Using limits to show asymptotic relationship of functions[1]:

$f(n) = o(g(n))$ iff

$$\lim_{x \to \infty} \frac{f(n)}{g(n)} = 0$$

$f(n) = \Theta(g(n))$ iff

$$\lim_{x \to \infty} \frac{f(n)}{g(n)} = c$$

$f(n) = \omega(g(n))$ iff

$$\lim_{x \to \infty} \frac{f(n)}{g(n)} = \infty$$

___ Asymptotic Notation: Big Oh, Big Omega, and Big Theta notation.

- $(\mathcal{O}(f(n)))$ is a worst case, or upper, bound;
  ("in any case, the time will not exceed f(n)").
- $\Omega(f(n))$ is a best case, or lower, bound;
  ("in any case, you cannot get a lower running time than f(n)").
- $\Theta(f(n))$ is usually given when both upper and lower bounds are the same—a tight bound

Observation 4: An algorithm without iteration or recursion has running time that is not dependent on the input size, so running time is O(1) – constant time. So we concentrate on iterative and recursive algorithms.

* NEXT TOPIC: COMPARING TIME COMPLEXITY OF ALGORITHMS

[1]c is a constant greater than 0.