ECS 60 **Lectures #2 Complexity**

I. Math Background
- A. Definitions of functions to establish the relative rates of growth of functions.
  - 1. $T(N) = O(f(N))$ if there are $c > 0$ and $n > 0$ such that $T(N) \le cf(N)$ when $N \ge n$.
    - a) $T(N) = N^2 + 25 = O(N^2)$
  - 2. $T(N) = \Omega(f(N))$ if there are $c > 0$ and $n > 0$ such that $T(N) \ge cf(N)$ when $N \ge n$.
  - 3. $T(N) = \Theta(f(N))$ if and only if $T(N) = O(f(N))$ and $T(N) = \Omega(f(N))$.
  - 4. $T(N) = o(f(N))$ if and only if $T(N) = O(f(N))$ and $T(N) \ne \Theta(f(N))$.
- B. Rules
  - 1. If $T(N) = O(f(N))$ and $V(N) = O(g(N))$ then
    - a) $T(N) + V(N) = \max(O(f(N)), O(g(N)))$
    - b) $T(N) * V(N) = O(f(N) * g(N))$
  - 2. If $T(N)$ is a polynomial of degree k, then $T(N) = \Theta(N^k)$. This means you ignore lower-order terms and multiplicative constants.
- C. Style
  - 1. Don't include constants or low order terms inside a Big-Oh.
  - 2. Don't say $T(N) \le O(f(N))$. The inequality is implied by the definition.

II. Running Time Calculations
- A. General rules
  - 1. For Loops: The running time for a for loop is at most the running time of the statements inside the for loop times the number of iterations.
  - 2. Nested Loops: Analyze these inside out. The total running time of a statement inside a group of nested loops is the running time of the statement multiplied by the product of the sizes of all the loops.
  - 3. Consecutive Statements: Just add together, i.e., the maximum is all that counts.
  - 4. If/Else: Time of test plus the maximum of the two alternatives.
  - 5. Analyze from inside out.
  - 6. Analyze function calls first. Ignore costs of the mechanism of calling functions unless there is a call by value for a data structure with a size dependent on N.

III. Intuitive interpretation of growth rate functions.
- A. O(1) implies a problem whose time requirement is constant, and therefore independent of N
- B. O(log N): Logarithmic = An algorithm that takes constant (O(1)) time to cut the problem size by a fraction (usually 1/2). Base is irrelevant. Binary search.
  - 1. Show that for all constants $a,b > 1$ $f(N)$ is $O(\log_a N)$ if and only if $f(N)$ is $O(\log_b N)$. Thus, you can omit the base.
    - Define $c = 1/(\log_b a)$ and remember that $\log_a N = \log_b N / \log_b a$
    - Then $f(N)$ is $O(\log_a N) = O(c * \log_b N) = c * O(\log_b N) = O(\log_b N)$
    - Define $c = 1/(\log_a b)$ and the proof is symmetrical
- C. O(N): Linear = Time increases directly with N. Linear Search.
- D. O(N log N): Logarithmic that has a linear component. Mergesort.
- E. $O(N^2)$ Quadratic = Two nested loops. Bubble sort and insertion sort.
- F. $O(N^3)$ Cubic = Three nested loops.
- G. $O(2^N)$ Exponential = Combinatorial. NP problems, searching permutations to find optimal, e.g. TSP.

IV. Best, average, worst cases based on data

V. Perspective.
- A. Consider the frequency of the types of operations.
- B. Some seldom-used critical operations must be efficient.
- C. If N is always small, then you can probably ignore an algorithm's efficiency.
- D. For smaller N, constants can be important. Mergesort with insertion sort.
- E. Weigh trade-offs between an algorithm's time requirements and its memory requirements. Sparse matrix vs. full matrix.