Test One CS:4050

Loop Invariant - A **loop invariant** is a condition [among program variables] that is necessarily true immediately before and immediately after each iteration of a **loop**. (Note that this says nothing about its truth or falsity part way through an iteration.)

Three necessary Conditions of a Loop Invariant:

- **Initialization**
- **Maintenance**
- **Termination**

Algorithmic Notation

Rules of Logs

1) $log_b(mn) = log_b(m) + log_b(n)$

2) $log_b(^m/_n) = log_b(m) - log_b(n)$

3) $log_b(m^n) = n \cdot log_b(m)$

https://www.cs.auckland.ac.nz/courses/compsci220s1c/lectures/2014S1C/Part1/220-03.pdf

Big O- By definition, g(n) is O(f(n)), or g(n) = O(f(n)) if a constant c > 0 exists, such that cf(n) grows faster than g(n) for all n > n0.

$$g(n) = O\big(f(n)\big)\ iff\ g(n) \leq cf(n)$$

Big Omega- iff there exists a positive real constant c and a positive integer n0 such that g(n) ≥ cf(n) for all n > n0.

$$g(n) = \Omega\big(f(n)\big)\ iff\ g(n) \geq cf(n)$$

Big Theta- iff there exists two positive real constants c1 and c2 and a positive integer n0 such that c1f(n) ≤ g(n) ≤ c2f(n) for all n > n0.

$$g(n) = \ \theta f(n)\ iff\ c1f(n) \leq g(n) \leq c2f(n)$$

Order of Function Growth

1. Exponential
2. Polynomial
3. Polylogrithmic
4. Logrithmic
5. Constant

Recurrence Relations

Iteration

Prove: $T(n) = T(n-1) + n \ = O(n^2)$

$$T(n-1) = T(n-2) + n - 1$$
$$T(n-2) = T(n-3) + n - 2$$
$$T(n) = T(n-3) + n - 2 + n - 1 + n$$
$$= T(1) + 1 + 2 \ldots n - 1 + n$$
$$= n(n+1)/2$$
$$= O(n^2)$$

Master Method

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$$a = ? \qquad b = ? \qquad f(n) = ?$$

Case 1: $f(n) = O(n^{log_b(a)-E})$

$$T(n) = \ \theta(n^{log_b(a)})$$

Case 2: $f(n) = \theta(n^{log_b(a)})$

$$T(n) = \ \theta(n^{log_b(a)} * \log(n))$$

Case 3: $f(n) = O(n^{log_b(a)+E})$

$$T(n) = \ \theta(f(n))$$

Regularity Condition: $a(f\left(\frac{n}{b}\right) \leq cf(n)$