CLRS Notes

Zhehao Wang

November 17, 2018

1 Analyzing algorithms, growth of functions

Loop invariant can help show the correctness of an algorithm. We must show three things about a loop invariant:

- initialization (true prior to the first iteration),
- maintenance (true before an iteration, remains true after an iteration), and
- termination (when loop terminates, the invariant gives us a useful property in showing the algorithm is correct.)

The first two conditions correspond with those of induction.

1.1 Asymptotic notation

Θ-notation: for a given function $g(n)$, we denote by $\Theta(g(n))$ the set of functions

$$\Theta(g(n)) = \{f(n) : \exists c_1, c_2, n_0, \text{ s.t. } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \ \forall n \geq n_0\}$$

O-notation: (asymptotic upper bound) for a given function $g(n)$, we denote by $\Theta(g(n))$ the set of functions

$$O(g(n)) = \{f(n) : \exists c, n_0, \text{ s.t. } 0 \leq f(n) \leq cg(n), \ \forall n \geq n_0\}$$

When we say "the running time is $O(n^2)$", we mean that there is a function $f(n)$ that is $O(n^2)$ s.t. for any value of n, no matter what particular input of size n is chosen, the running time on that input is bounded from above by the value $f(n)$. Equivalently, we mean that the worst-case running time is $O(n^2)$.
Ω-notation: (asymptotic lower bound) for a given function $g(n)$, we denote by $\Omega(g(n))$ the set of functions

$$\Omega(g(n)) = \{f(n) : \exists c, n_0, \text{ s.t. } 0 \leq cg(n) \leq f(n), \ \forall n \geq n_0\}$$

For any two functions $f(n)$ and $g(n)$, we have $f(n) = \Theta(g(n))$ iff $f(n) = \Omega(g(n))$ and $f(n) = O(g(n))$.

o-notation: (upper bound that is not asymptotically tight) for a given function $g(n)$, we denote by $o(g(n))$ the set of functions

$$o(g(n)) = \{f(n) : \forall c > 0,\ \exists n_0,\ \text{s.t. } 0 \leq f(n) \leq cg(n),\ \forall n \geq n_0\}$$

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

E.g. $2n = o(n^2)$ but $2n^2 \neq o(n^2)$
Similarly, we can define **ω-notation** as lower bound that is not asymptotically tight.

- $\Omega, \Theta, O, o, \omega$ are transitive
- Ω, Θ, O are reflexive
- Θ is symmetric
- O and Ω, and o and ω are transpose-symmetric.

We can then draw an analogy with real-number comparison:

- $f(n) = O(g(n))$ with $a \leq b$
- $f(n) = \Theta(g(n))$ with $a = b$
- $f(n) = \Omega(g(n))$ with $a \geq b$
- $f(n) = o(g(n))$ with $a < b$
- $f(n) = \omega(g(n))$ with $a > b$

However, **trichotomy*** of real numbers comparison does not carry over to our notation.

2 Divide and conquer

- **Divide** the problem into a number of subproblems that are smaller instances of the same problem.
- **Conquer** the subproblems by solving them recursively. If sizes are sufficiently small: solve them in a straightforward manner.
- **Combine** the solutions to the subproblems into the solution for the original problem.

Given that **mergesort** has a higher constant coefficient than selection sort, how about combine the two: we split till subsets are small enough and apply selection sort on the small enough subsets.

*For any two real numbers a and b, one of $a < b$, $a > b$ or $a = b$ must hold. Counter example with our notation: $f(n) = n^{1+\sin n}$, $g(n) = n$