CSC2100B Data Structures Analysis

Irwin King

king@cse.cuhk.edu.hk
http://www.cse.cuhk.edu.hk/~king

Department of Computer Science & Engineering
The Chinese University of Hong Kong

Algorithm

- An algorithm is a clearly specified set of simple instructions to be followed to solve a problem.
  - How to estimate the time required for a program.
  - How to reduce the running time of a program from days or years to fractions of a second.
  - What is the storage complexity of the program.
  - How to deal with trade-offs.

Running Time

- There are two contradictory goals:
  - We would like an algorithm that is easy to understand, code, and debug.
  - We would like an algorithm that makes efficient use of the computer's resources, especially, one that runs as fast as possible.

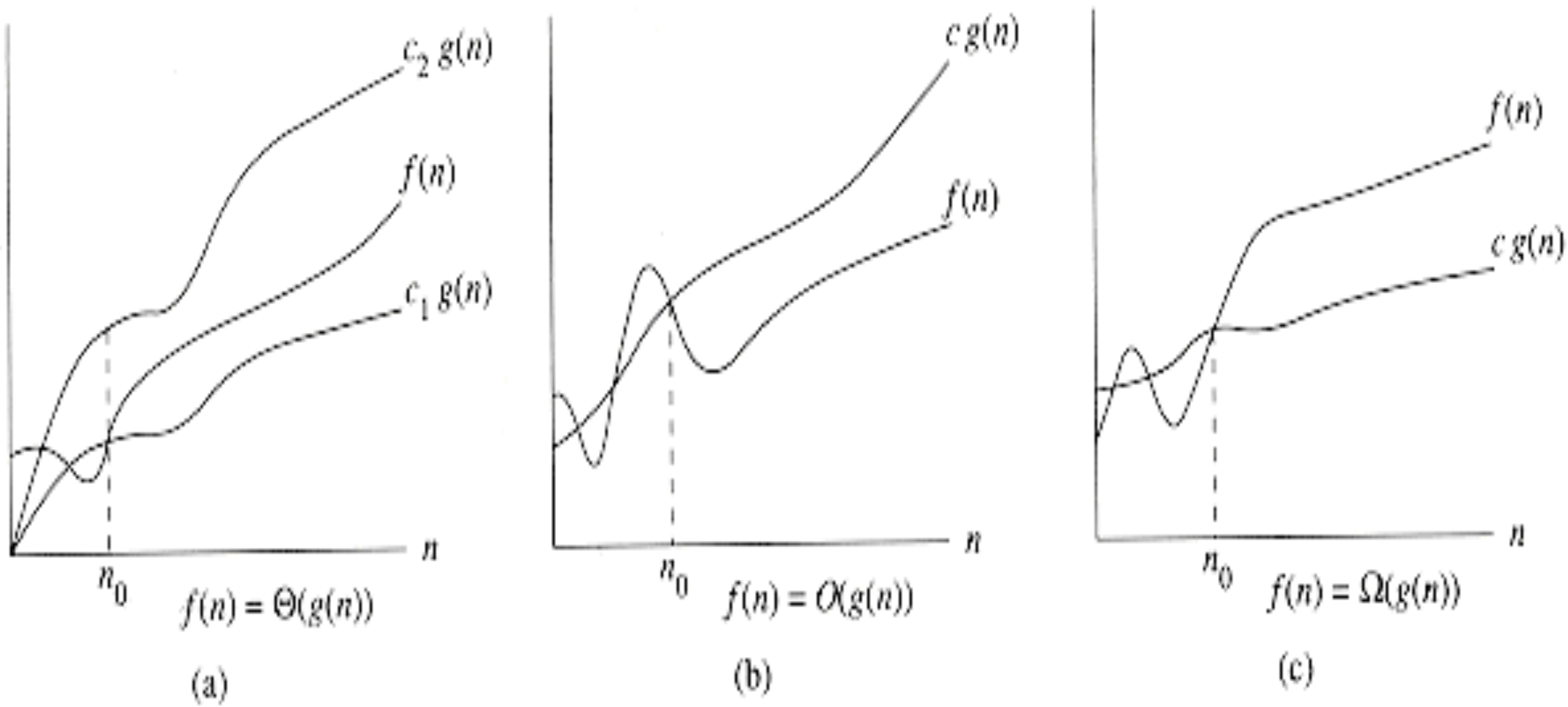
Function Comparison

- Given two functions, f(N) and g(N), what does it mean when we say that

$$f(N) < g(N)?$$

  - Should this hold for all N?
  - We need to compare their relative rates of growth.

Example



http://science.slc.edu/~jmarshall/courses/2002/spring/cs50/BigO/index.html

Why Use Bounds

- The idea is to establish a relative order among functions.
- We are more concerned about the relative rates of growth of functions.
- For example, which function is greater, 1,000N or N^2?
- The turning point is N = 1,000 where N^2 will be greater for larger N.

First Definition

- It says that there is some point n_0 past which c f(N) is always at least as large as T(N).
- In our case, T(N) = 1000N, f(N) = N^2, n_0 = 1,000, and c = 1.
- We could also use n_0 = 10, and c = 100.
- So we can say that 1000N = $O(N^2)$.
- It is an upper bound on T(N).

Other Definitions

- The second definition says that the growth rate of T(N) is greater than or equal to that of g(N).
- The third definition says that the growth rate of T(N) equals the growth rate of h(N).
- The fourth definition says that the growth rate of T(N) is less than the growth rate of p(N).

Big-O Notation

- If $f(n)$ and $g(n)$ are functions defined for positive integers, then to write $f(n)$ is $O(g(n))$.
- $f(n)$ is big-O of $g(n)$ means that there exists a constant c such that $|f(x)| \leq c|g(n)|$ for all sufficiently large positive integers n.
- Under these conditions we also say that "$f(n)$ has order at most $g(n)$" or "$f(n)$ grows no more rapidly than $g(n)$".

Examples

- $f(n) = 100n$ then $f(n) = O(n)$.
- $f(n) = 4n + 200$ then $f(n) = O(n)$.
- $f(n) = n^2$ then $f(n) = O(n^2)$.
- $f(n) = 3\ n^2 - 100$ then $f(n) = O(n^2)$.

Rules

- If $T_1(N) = O(f(N))$ and $T_2(N) = O(g(N))$, then
  - $T_1(N) + T_2(N) = \max(O(f(N)), O(g(N)))$,
  - $T_1(N) * T_2(N) = O(f(N) * g(N))$,
- If $T(N)$ is a polynomial of degree k, then $T(N) = (N^k)$.
- $\log^k N = O(N)$ for any constant k.
  - This tells us that logarithms grow very slowly.

Watch Out!

- It is bad to include constants or low-order terms inside a Big-Oh notation.
- Do not say $T(N) = O(2N^2)$ or $T(N) = O(N^2 + N)$.
- In both cases, $T(N) = O(N^2)$.

Observations

- If $f(n)$ is a polynomial in n with degree r, then $f(n)$ is $O(n^r)$, but $f(n)$ is not $O(n^s)$ for any power s less than r.

- Any logarithm of n grows more slowly (as n increases) than any positive power of n.
  - Hence $\log n$ is $O(n^k)$ for any $k > 0$, but n^k is never $O(\log n)$ for any power $k > 0$.

Common Orders

- $O(1)$ means computing time that is bounded by a constant (not dependent on n)
- $O(n)$ means that the time is directly proportional to n, and is called linear time.
- $O(n^2)$ means quadratic time.
- $O(n^3)$ means cubic time.
- $O(2^n)$ means exponential time.
- $O(\log n)$ means logarithmic time.
- $O(\log^2 n)$ means log-squared time.

Algorithm Analyses

- On a list of length n, sequential search has running time O(n).
- On a ordered list of length n, binary search has running time O(log n).
- The sum of the sum of integer index of a loop from 1 to n is $O(n^2)$, i.e., 1 + 2 + 3 + … + n.
    - For i = 1 to n
        - For j = i to n

Recurrence Relations

- Recurrence relations are useful in certain counting problems.
- A recurrence relation relates the n-th element of a sequence to its predecessors.
- Recurrence relations arise naturally in the analysis of recursive algorithms.

Sequences and Recurrence Relations

- A (numerical) sequence is an ordered list of number.
  - 2, 4, 6, 8, … (positive even numbers)
  - 0, 1, 1, 2, 3, 5, 8, … (the Fibonacci numbers)
  - 0, 1, 3, 6, 10, 15, … (numbers of key comparisons in selection sort)

Definitions

- A recurrence relation for the sequence $a_0, a_1, \ldots$ is an equation that relates an to certain of its predecessors $a_0, a_1, \ldots, a_{n-1}$.

- Initial conditions for the sequence $a_0, a_1, \ldots$ are explicitly given values for a finite number of the terms of the sequence.

Example

- A person invests \$1,000 at 12% compounded annually. If A_n represents the amount at the end of n years, find a recurrence relation and initial conditions that define the sequence A_n.
- At the end of n-1 years, the amount is A_{n-1}. After one more year, we will have the amount A_{n-1} plus the interest. Thus $A_n = A_{n-1} + (0.12) A_{n-1} = (1.12) A_{n-1}, n \geq 1$.
- To apply this recurrence relation for n = 1, we need to know the value of A_0 which is 1,000.

Solving Recurrence Relations

- Iteration - we use the recurrence relation to write the n-th term a_n in terms of certain of its predecessors a_{n-1}, …, a_0.
- We then successively use the recurrence relation to replace each of a_{n-1}, …by certain of their predecessors.
- We continue until an explicit formula is obtained.

Some Definitions of Linear Second-order recurrences with constant coefficients

- kth-order
  - Elements x(n) and x(n-k) are k positions apart in the unknown sequence.
- Linear
  - It is a linear combination of the unknown terms of the sequence.
- Constant coefficients
  - The assumption that a, b, and c are some fixed numbers.
- Homogeneous
  - If f(x) = 0 for every n.

Solving Recurrence Relations

- Linear homogeneous recurrence relations with constant coefficients - a linear homogeneous recurrence relation of order k with constant coefficients is a recurrence relation of the form

$$a_0 = c_0, a_1 = c_1, \ldots, a_{k-1} = c_{k-1},$$

- Notice that a linear homogeneous recurrence relation of order K with constant coefficients, together with the k initial conditions

$$a_n = c_1 \, a_{n-1} + c_2 \, a_{n-2} + \ldots + c_k \, a_{n-k}, c, \geq 0$$

- uniquely defines a sequence $a_0, a_1, \ldots$

Example

- Nonlinear
  - $a_n = 3\ a_{n-1}\ a_{n-2}$.
- Inhomogeneous
  - $a_n - a_{n-1} = 2_n$.
- Homogeneous recurrence relation with nonconstant coefficients
  - $a_n = 3\ n\ a_{n-1}$.

Iteration Example

- We can solve the recurrence relation $a_n = a_{n-1} + 3$ subject to the initial condition $a_1 = 2$, by iteration.
  - $a_{n-1} = a_{n-2} + 3$.
  - $a_n = a_{n-1} + 3 = a_{n-2} + 3 + 3 = a_{n-2} + 2 \times 3$.
  - $a_{n-2} = a_{n-3} + 3$.
  - $a_n = a_{n-2} + 2 \times 3 = a_{n-3} + 3 + 2 \times 3 = a_{n-3} + 3 \times 3$.
  - $a_n = a_{n-k} + k \times 3 = 2 + 3(n - 1)$.

Iteration Example

- In general, to solve $a_n = a_{n-1} + k$, $a_1 = c$, one obtains $a_n = c + k(n-1)$.
- We can solve the recurrence relation
  - $a_n = k\, a_{n-1}$, $a_0 = c$.
  - $a_n = k\, a_{n-1} = k(k\, a_{n-2}) = \ldots = k^n a_0 = c\, k^n$.

Linear Homogeneous Recurrence Example

$$a_n = 5\ a_{n-1} - 6\ a_{n-2},\ a_0 = 7,\ a_1 = 16$$

- Since the solution was of the form $a_n = t^n$, thus for our first attempt at finding a solution of the second-order recurrence relation, we will search for a solution of the form $a_n = t^n$.
  - $t_n = 5\ t^{n-1} - 6\ t^{n-2}$
  - $t^2 - 5t + 6 = 0$

Example

- Solving the above we obtain, $t = 2$, $t = 3$.
- At this point, we have two solutions S and T given by
  - $S_n = 2^n, T_n = 3^n$.
- We can verify that is S and T are solutions of the above, then $bS + dT$, where b and d are any numbers whatever, is also a solution of the above.

Example

- In our case, if we define the sequence U by the equation
  - $U_n = b S_n + d T_n$
  - $= b 2^n + d 3^n$
- To satisfy the initial conditions, we must have
  - $7 = U_0 = b 2^0 + d 3^0 = b + d.$
  - $16 = U_1 = b 2^1 + d 3^1 = 2b + 3d.$

Example

- Solving these equations for b and d, we obtain
  - $b = 5, d = 2$.
- Therefore, the sequence U defined by
  - $U_n = 5 \times 2^n + 2 \times 3^n$
    satisfies the recurrence relation and the initial conditions.

Fibonacci Sequence

- The Fibonacci sequence is defined by the recurrence relation
  - $f_n = f_{n-1} + f_{n-2}$, $n \geq 3$ and initial conditions
  - $f_1 = 1$, $f_2 = 2$.
- We begin by using the quadratic formula to solve
  - $t^2 - t - 1 = 0$.
- The solutions are

$$t = \frac{1 \pm \sqrt{5}}{2}.$$

Example

- Thus the solution is of the form

$$f_n = b\left(\frac{1+\sqrt{5}}{2}\right)^n + d\left(\frac{1-\sqrt{5}}{2}\right)^n.$$

- To satisfy the initial conditions, we must have

$$b\left(\frac{1+\sqrt{5}}{2}\right) + d\left(\frac{1-\sqrt{5}}{2}\right) = 1,$$

$$b\left(\frac{1+\sqrt{5}}{2}\right)^2 + d\left(\frac{1-\sqrt{5}}{2}\right)^2 = 2.$$

Tower of Hanoi

- Find an explicit formula for an, the minimum number of moves in which the n-disk Tower of Hanoi puzzle can be solved.
  - $a_n = 2\ a_{n-1} + 1, a_1 = 1.$
- Applying the iterative method, we obtain

$$
\begin{aligned}
a_n &= 2a_{n-1} + 1 \\
&= 2(2a_{n-2} + 1) + 1 \\
&= 2^2 a_{n-2} + 2 + 1 \\
&= 2^2(2a_{n-3} + 1) + 2 + 1 \\
&= 2^3 a_{n-3} + 2^2 + 2 + 1 \\
&\vdots \\
&= 2^{n-1} a_1 + 2^{n-2} + 2^{n-3} + \ldots + 2 + 1 \\
&= 2^{n-1} + 2^{n-2} + 2^{n-3} + \ldots + 2 + 1 \\
&= 2^n - 1
\end{aligned}
$$

Common Recurrence Types

- Decrease-by-one
  - $T(n) = T(n-1) + f(n)$
- Decrease-by-a-constant-factor
  - $T(n) = T(n/b) + f(n)$
- Divide-and-conquer
  - $T(n) = aT(n/b) + f(n)$

