Algorithm Design and Analysis (ECS 122A) Study Guide

Davis Computer Science Club
Tutoring Committee

Spring Quarter 2015

Contents

Chapter 1

Asymptotic Notation

1.1 O-Notation (Big O)

Notation

$$f(n) \in O(g(n))$$

Formal Definition

For a given function $g(n)$, $O(g(n))$ is the set of functions for which there exists positive constants c and n_0 such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

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

Informal Definition

The function $g(n)$ is an asymptotic upper bound for the function $f(n)$ if there exists constants c and n_0 such that $0 \leq f(n) \leq c \cdot g(n)$ for $n \geq n_0$.

Another way to perceive Big O notation is that for $f(n) \in O(g(n))$, the function f's asymptotic[1] growth is no faster than that of function g's.

Limit Definition

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

1.1.1 Example

Prove that asymptotic upper bound of $f(n) = 2n + 10$ is $g(n) = n^2$.

$$\begin{aligned} 0 \leq f(n) &\leq c \cdot g(n) \text{ for } n \geq n_0 \\ 0 \leq 2n + 10 &\leq c \cdot n^2 \text{ for } n \geq n_0 \end{aligned}$$

Arbitrarily choose c and n_0 values. Simplest is to turn one of the variables into the value 1 and solve. For this example, we will assign the value 1 to n_0.

$$\begin{aligned} 0 \leq 2n + 10 &\leq c \cdot n^2 \text{ for } n \geq 1 \\ 2(1) + 10 &\leq c \cdot (1)^2 \\ 12 &\leq c \end{aligned}$$

By picking $n_0 = 1$ and $c = 12$, the inequality of $2n + 10 \leq 12n^2$ will hold true for all $n \geq 1$. Since there exists a constant c and n_0 that fulfill this inequality, we have proven that $f(n) = 2n + 10 = O(n^2)$.

[1] Asymptotic: As given variable approaches infinity.

1.2 o-Notation (Little O)

Notation

$$f(n) \in o(g(n))$$

Formal Definition

For a given function $g(n)$, $o(g(n))$ is the set of functions for which every positive constant $c > 0$, there exists a constant $n_0 > 0$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$.

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

Informal Definition

The function g(n) is an upper bound that is not asymptotically tight. For all positive constant values of c, there must exists a constant n_0 such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$. The value of n_0 may not depend on n, but may depend on c.

Another way to perceive Little O notation is that for $f(n) \in o(g(n))$, the function f's asymptotic growth is strictly less than that of the function g's. In this sense, Little O can be seen as a "stronger" bound in comparison to Big O. By proving that a function is an element of Little O, it also proves that the function is an element of Big O.

Limit Definition

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

1.2.1 Example

Prove that $f(n) = 2n$ has an upper bound $o(n^2)$.

$$\begin{aligned}
0 \leq c \cdot g(n) &\leq f(n) \text{ for } n \geq n_0 \\
0 \leq c \cdot 2n &\leq n^2 \text{ for } n \geq n_0 \\
2c &\leq n \text{ for } n \geq n_0 \\
2c &\leq n_0
\end{aligned}$$

For Little O to hold true, the inequality needs to hold true for all $c > 0$ and for all $n > n_0$. From simplifying the inequality, we assert that the inequality will hold true as long as the value of n_0 is twice the value of c. Given that they are both constants, then there exists a constant value of n_0 for all positive constant c that fulfill this inequality.

Another method to solve this problem is to use the limit definition.

$$\begin{aligned}
&\lim_{n \to \infty} \frac{2n}{n^2} \\
&\lim_{n \to \infty} \frac{2}{n} = 0
\end{aligned}$$

1.2.2 Example

Prove that $f(n) = 2n^2$ does not have the upper bound $o(n^2)$.

$$\begin{aligned}
0 \leq c \cdot g(n) &\leq f(n) \text{ for } n \geq n_0 \\
0 \leq c \cdot 2n^2 &\leq n^2 \text{ for } n \geq n_0 \\
2c &\leq 1 \text{ for } n \geq n_0
\end{aligned}$$

For a function to have the Little O bound, the inequality must hold true for all positive c. However, simplification of the inequality asserts that the inequality will only hold true for all $c < \frac{1}{2}$. Therefore, $f(n) = 2n^2$ does not have the upper bound $o(n^2)$.

1.3 Ω-Notation (Big Omega)

Notation

$$f(n) \in \Omega(g(n))$$

Formal Definition

For a given function $g(n)$, $\Omega(g(n))$ is the set of functions for which there exists positive constants c and n_0 such that $0 \leq c \cdot g(n) \leq f(n)$ for all $n \geq n_0$.

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

Informal Definition

The function $g(n)$ is an asymptotic lower bound for the function $f(n)$ if there exists constants c and n_0 such that $0 \leq c \cdot g(n) \leq f(n)$ for $n \geq n_0$.

Limit Definition

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

1.4 ω-Notation (Little Omega)

Notation

$$f(n) \in \omega(g(n))$$

Formal Definition

For a given function $g(n)$, $\omega(g(n))$ is the set of functions for which every positive constant $c > 0$, there exists a constant $n_0 > 0$ such that $0 \leq c \cdot g(n) \leq f(n)$ for all $n \geq n_0$.

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

Informal Definition

The function g(n) is a lower bound that is not asymptotically tight. For all positive constant values of c, there must exists a constant n_0 such that $0 \leq c \cdot g(n) \leq f(n)$ for all $n \geq n_0$. The value of n_0 may not depend on n, but may depend on c.

Another way to perceive Little ω notation is that for $f(n) \in \omega(g(n))$, the function f's asymptotic growth is strictly greater than that of the function g's. In this sense, Little ω can be seen as a "stronger" bound in comparison to Big Ω. By proving that a function is an element of Little ω, it also proves that the function is an element of Big Ω.

Limit Definition

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

1.5 Θ-notation (Big Theta)

Notation

$$f(n) \in \Theta(g(n))$$

Formal Definition

For a given function $g(n)$, $\Theta(g(n))$ is the set of functions for which there exists positive constants c_1, c_2, and n_0 such that $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq n_0$.

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

Informal Definition

The function g(n) is an asymptotic tight bound for the function f(n) if there exists constants c_1, c_2, and n_0 such that $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for $n \geq n_0$.

Big theta implies that $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Limit Definition

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} \in \mathbb{R}_{>0}$$

Chapter 2

Recurrence Relations

2.1 Recurrence Relations

A recurrence relation is an equation that recursively defines a sequence of values. After the initial terms are given, each subsequent term is defined as a function of the previous terms.

Fibonacci

Fibonacci is an example of a recurrence relation.

$$F_n = \begin{cases} F_{n-1} + F_{n-2}, & n \geq 2 \\ 1, & n = 1 \\ 0, & n = 0 \end{cases}$$

The first two terms are defined while the subsequent terms are a function of the two previous.

2.2 Solving Recurrence Relations

- Substitution Method
- Recursion-Tree Method
- Master Theorem

2.3 Substitution Method

1. Guess the bounds.
2. Apply mathematical induction to prove the bounds.

2.3.1 Example

Find the asymptotic upper bound for the following function:

$$T(n) \begin{cases} 2T(n-1)+1, & n \geq 1 \\ 1, & n = 0 \end{cases}$$

Guess

$$T(n) \in O(2^n)$$

Inductive Basis

$$\begin{aligned} T(0) &= 2^0 \\ &= 1 \end{aligned}$$

Inductive Hypothesis
Assume that $T(n) = 2^n$ holds true for all $n = k$.

Inductive Step

$$\begin{aligned} T(n) &= 2T(n-1)+1 && \text{Base equation} \\ &= 2T((k+1)-1)+1 && \text{Substitute n with } k+1 \\ &= 2T(k)+1 && \text{Simplify parameters to T(n)} \\ &= 2(2^k)+1 && \text{Substitute T(n) with inductive hypothesis} \\ &= 2^{k+1}+1 && \text{Property of exponents} \\ & && \text{Q.E.D} \end{aligned}$$

2.4 Master Theorem

Used for divide and conquer recurrences that follow the generic form:

$$T(n) = a \cdot T(\frac{n}{b}) + f(n) \text{ where } a \geq 1, b > 1$$

2.4.1 Case 1

Condition

$$f(n) \in O(n^c)$$

$$c < log_b(a)$$

Solution

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

2.4.2 Case 2

Condition

$$f(n) \in \Theta(n^c)$$

$$c = log_b(a)$$

Solution

$$T(n) \in \Theta(n^{log_b(a)} \cdot log_2(n))$$

2.4.3 Case 3

Condition

$$f(n) \in \Omega(n^c)$$

$$c > log_b(a)$$

Regularity Condition

This case must also fulfill the regularity condition.

$$a \cdot f(\frac{n}{b}) \leq k \cdot f(n) \text{ where } k < 1$$

Solution

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

Remark

The idea behind this case is that given the generic form, the function $f(n)$ will grow far quicker than $a \cdot T(\frac{n}{b})$ and will be the primary influence of $T(n)$'s asymptotic behavior.

2.4.4 Example

$$T(n) = 64T(\frac{n}{4}) + 1000n^2$$

Given

$$f(n) = 1000n^2 \in \Theta(n^2)$$

$$a = 64$$

$$b = 4$$

$$c = 2$$

Condition

$$\begin{array}{ccc} c & ? & log_b(a) \\ 2 & ? & log_4(64) \\ 2 & < & 3 \end{array}$$

Condition satisfied for case 1

Solution

$$\therefore T(n) = \Theta(n^{log_4(64)}) = \Theta(n^3)$$

2.4.5 Example

$$T(n) = 32T(\frac{n}{2}) + 20n^5$$

Given

$$f(n) = 20n^5 \in \Theta(n^5)$$

$$a = 32$$

$$b = 2$$

$$c = 5$$

Condition

$$\begin{array}{ccc} c & ? & log_b(a) \\ 5 & ? & log_2(32) \\ 5 & = & 5 \end{array}$$

Condition satisfied for case 2

Solution

$$\therefore T(n) = \Theta(n^{log_2(32)} \cdot log_2(n)) = \Theta(n^5 \cdot lg(n))$$

2.4.6 Example

$$T(n) = 7T(\frac{n}{7}) + 19n^{11}$$

Given

$$f(n) = 19n^{11} \in \Theta(n^{11})$$
$$a = 7$$
$$b = 7$$
$$c = 11$$

Condition

$$\begin{array}{rcl} c & ? & log_b(a) \\ 11 & ? & log_7(7) \\ 5 & > & 1 \end{array}$$

Condition partially fulfilled for case 3. Must also check regularity condition.

$$\begin{array}{rcl} a \cdot f(\frac{n}{b}) & \leq & k \cdot f(n) \\ 7 \cdot \left[19(\frac{n}{7})^{11} \right] & \leq & k \cdot 19n^{11} \\ 7 \cdot \frac{n^{11}}{7^{11}} & \leq & k \cdot n^{11} \\ \frac{1}{7^{10}} \cdot n^{11} & \leq & k \cdot n^{11} \end{array}$$

Choosing $k = \frac{1}{7^{10}} < 1$ fulfills the regularity condition.

Solution

$$\therefore T(n) = \Theta(19n^{11})$$

Chapter 3

Divide and Conquer Algorithms

Chapter 4

Side Topics

4.1 Induction

Steps

1. Basis (Base Case)
2. Inductive Hypothesis
3. Inductive Step

4.1.1 Example

Prove that the following systems of equations has the solution $T(n) = n \cdot lg(n)$.

$$T(n) \begin{cases} 2T(\frac{n}{2}) + n, & n = 2^k \text{ for } k > 1 \\ 2, & n = 2 \end{cases}$$

Basis

$$\begin{aligned} T(2) &= (2) \cdot lg(2) \\ &= 2 \cdot 1 \\ &= 2 \end{aligned}$$

Inductive Hypothesis

Assume that $T(n) = n \cdot lg(n)$ holds true for all $n = 2^k$.

Inductive Step

$$\begin{aligned} T(n) &= 2T(\frac{n}{2}) + n && \text{Base equation} \\ &= 2T(\frac{2^{k+1}}{2}) + 2^{k+1} && \text{Substitute n with } 2^{k+1} \\ &= 2T(2^k) + 2^{k+1} && \text{Simplify parameters to function T(...)} \\ &= 2(2^k \cdot lg(2^k)) + 2^{k+1} && \text{Inductive hypothesis} \\ &= 2^{k+1}\left[lg(2^k) + 1\right] && \text{Distributive property} \\ &= 2^{k+1}\left[lg(2^k) + lg(2)\right] && \text{Logarithmic identity} \\ &= 2^{k+1} \cdot lg(2^k \cdot 2) && \text{Logarithmic identity} \\ &= 2^{k+1} \cdot lg(2^{k+1}) && \text{Exponent property} \\ & && \text{Q.E.D} \end{aligned}$$

4.1.2 Example

Prove that the following systems of equations has the solution $T(n) = 2F(n) - 1$ where $F(n) = F(n-1) + F(n-2)$.

$$T(n) \begin{cases} T(n-1) + T(n-2) + 1, & \text{if } n \geq 2 \\ 0, & \text{if } n = \{0, 1\} \end{cases}$$

Basis

$T(0) = 1$

Inductive Hypothesis

Assume that $T(n) = F(n) - 1$ is true for all $n = k$.

Inductive Step

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + 1 && \text{Base equation} \\ T(k+1) &= T((k+1)-1) + T((k+1)-2) + 1 && \text{Substitute n with k+1} \\ &= T(k) + T(k-1) + 1 && \text{Simplify parameters to function T(...)} \\ &= (2F(k) - 1) + (2F(k-1) - 1) + 1 && \text{Inductive hypothesis} \\ &= 2F(k) + 2F(k-1) - 1 && \text{Simplify equation} \\ &= 2(F(k) + F(k-1)) - 1 && \text{Distributive property} \\ &= 2(F(k+1)) - 1 && \text{Definition of function: } F(k+1) = F(k) + F(k-1) \\ &= 2F(k+1) - 1 && \text{Simplify} \\ & && \text{Q.E.D} \end{aligned}$$