CS4311
Design and Analysis of Algorithms

Lecture 2: Growth of Function

About this lecture

- Introduce Asymptotic Notation
  - $\Theta(\ )$, $O(\ )$, $\Omega(\ )$, $o(\ )$, $\omega(\ )$

Dominating Term

Recall that for input size n,

- Insertion Sort 's running time is:

  $An^2 + Bn + C$, (A,B,C are constants)

- Merge Sort 's running time is:

  $Dn \log n + En + F$, (D,E,F are constants)

- To compare their running times for large n, we can in fact just focus on the dominating term (the term that grows fastest when n increases)
  - That is, An^2 vs $Dn \log n$

Dominating Term

- If we look more closely, the leading constants in the dominating term does not affect much in this comparison
  - That is, we may as well compare n^2 vs $n \log n$ (instead of An^2 vs $Dn \log n$)
- As a result, we conclude that Merge Sort is better than Insertion Sort when n is sufficiently large

Asymptotic Efficiency

- In the previous comparison, we are studying the asymptotic efficiency of the two sorting algorithms
  - That is, what happens if the input size can increase without bound?
- If algorithm P is asymptotically faster than algorithm Q, P is often a better choice
- To aid (and simplify) our study in the asymptotic efficiency, we now introduce some useful asymptotic notation

Big-O notation

Definition: Given a function $g(n)$, we denote $O(g(n))$ to be the set of functions

$$\{ f(n) \mid \text{there exists positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq c\,g(n) \text{ for all } n \geq n_0 \}$$

Meaning: Those functions which can be upper bounded by a constant times of $g(n)$ for large n

Big-O notation (example)

- $4n \in O(5n)$ [proof: $c = 1$, $n \geq 1$]
- $4n \in O(n)$ [proof: $c = 4$, $n \geq 1$]
- $4n + 3 \in O(n)$ [proof: $c = 5$, $n \geq 3$]
- $n \in O(0.001n^2)$ [proof: $c = 1$, $n \geq 100$]
- $\log_e n \in O(\log n)$ [proof: $c = 1$, $n \geq 1$]
- $\log n \in O(\log_e n)$ [proof: $c = \log e$, $n \geq 1$]

Remark: Usually, we will slightly abuse the notation, and write $f(n) = O(g(n))$ to mean $f(n) \in O(g(n))$

Big-Omega notation

Definition: Given a function $g(n)$, we denote $\Omega(g(n))$ to be the set of functions

$$\{ f(n) \mid \text{there exists positive constants } c \text{ and } n_0 \text{ such that } 0 \leq c\,g(n) \leq f(n) \text{ for all } n \geq n_0 \}$$

Meaning: Those functions which can be lower bounded by a constant times of $g(n)$ for large n

Big-O and Big-Omega

- Similar to Big-O, we will slightly abuse the notation, and write $f(n) = \Omega(g(n))$ to mean $f(n) \in \Omega(g(n))$

Relationship between Big-O and Big-Ω :

$$f(n) = \Omega(g(n)) \Leftrightarrow g(n) = O(f(n))$$

Big-Ω notation (example)

- $5n = \Omega(4n)$ [proof: $c = 1$, $n \geq 1$]
- $n = \Omega(4n)$ [proof: $c = 1/4$, $n \geq 1$]
- $4n + 3 = \Omega(n)$ [proof: $c = 1$, $n \geq 1$]
- $0.001n^2 = \Omega(n)$ [proof: $c = 1$, $n \geq 100$]
- $\log_e n = \Omega(\log n)$ [proof: $c = 1/\log e$, $n \geq 1$]
- $\log n = \Omega(\log_e n)$ [proof: $c = 1$, $n \geq 1$]

Θ notation (Big-O $\cap$ Big-Ω)

Definition: Given a function $g(n)$, we denote $\Theta(g(n))$ to be the set of functions

$$\{ f(n) \mid \text{there exists positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \}$$

Meaning: Those functions which can be both upper bounded and lower bounded by of $g(n)$ for large n

Big-O, Big-Ω, and Θ

- Similarly, we write $f(n) = \Theta(g(n))$ to mean $f(n) \in \Theta(g(n))$

Relationship between Big-O, Big-Ω, and Θ:

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

$$\Leftrightarrow$$

$$f(n) = \Omega(g(n)) \text{ and } f(n) = O(g(n))$$

Θ notation (example)

- $4n = \Theta(n)$ [$c_1 = 1, c_2 = 4, n \geq 1$]
- $4n + 3 = \Theta(n)$ [$c_1 = 1, c_2 = 5, n \geq 3$]
- $\log_e n = \Theta(\log n)$ [$c_1 = 1/\log e, c_2 = 1, n \geq 1$]
- Running Time of Insertion Sort = $\Theta(n^2)$
  - If not specified, running time refers to the worst-case running time
- Running Time of Merge Sort = $\Theta(n \log n)$

Little-o notation

Definition: Given a function g(n), we denote $o(g(n))$ to be the set of functions

$$\{ f(n) \mid \text{for any positive } c \text{, there exists positive constant } n_0 \text{ such that } 0 \le f(n) < c\, g(n) \text{ for all } n \ge n_0 \}$$

Note the similarities and differences with the Big-O definition

Little-o (equivalent definition)

Definition: Given a function $g(n)$, $o(g(n))$ is the set of functions

$$\{\, f(n) \mid \lim_{n\to\infty} \left(f(n)/g(n)\right) = 0 \,\}$$

Examples:

- $4n = o(n^2)$
- $n \log n = o(n^{1.000001})$
- $n \log n = o(n \log^2 n)$

Little-omega notation

Definition: Given a function g(n), we denote $\omega(g(n))$ to be the set of functions

$\{ f(n) \mid$ for any positive c, there exists positive constant n_0 such that

$$0 \leq c\, g(n) < f(n)$$

for all $n \geq n_0 \}$

Note the similarities and differences with the Big-Omega definition

Little-omega (equivalent definition)

Definition: Given a function $g(n)$, $\omega(g(n))$ is the set of functions

$$\{ f(n) \mid \lim_{n\to\infty} \big(g(n)/f(n)\big) = 0 \}$$

Relationship between Little-o and Little-ω :

$$f(n) = \omega(g(n)) \Leftrightarrow g(n) = o(f(n))$$

To remember the notation:

- O is like $\leq$: f(n) = O(g(n)) means f(n) $\leq$ cg(n)
  - And it is possible to have g(n) = O(f(n))
- o is like $<$: f(n) = o(g(n)) means f(n) $<$ cg(n)
  - And it is not possible to have g(n) = o(f(n))

Similarly,

- Ω is like $\geq$: f(n) = Ω(g(n)) means f(n) $\geq$ cg(n)
- ω is like $>$: f(n) = O(g(n)) means f(n) $>$ cg(n)

Finally,

- Θ is like = : f(n) = Θ(g(n)) $\Leftrightarrow$ g(n) = Θ(f(n))

Note: Not any two functions can be compared asymptotically

What's wrong with it?

Your friend, after this lecture, has tried to prove $1+2+\ldots+n = O(n)$

- His proof is by induction:
- First, $1 = O(n)$
- Assume $1+2+\ldots+k = O(n)$
- Then, $1+2+\ldots+k+(k+1) = O(n) + (k+1)$

$$= O(n) + O(n) = O(2n) = O(n)$$

So, $1+2+\ldots+n = + O(n)$ [where is the bug??]