Asymptotic Notations

CS 5633 Analysis of Algorithms

Computer Science
University of Texas at San Antonio

August 28, 2024

About Homework Submission

- Please submit electronically to Canvas.
- Please submit a PDF file.
  - You can write the answers in word processors and convert them into a pdf file.
  - You can also scan or take a photo of your files and combine them as a pdf file.

Run-time Analysis of Algorithms

Run-time Analysis of Algorithms

- As mentioned last time in class, when analyzing the running time of the algorithm, we are interested in counting the number of steps the algorithm takes as a function of n (the size of the input).
- If n is small, then any algorithm will perform reasonably fast (i.e. brute force is fast). We are interested in how the running time of the algorithm grows as n tends to infinity.
- As n gets very large, the multiplicative constants and lower-order terms tend to get dominated by the effects of the input size itself, and thus the highest-order term tends to heavily influence the total number of steps the algorithm takes.

Asymptotic Efficiency

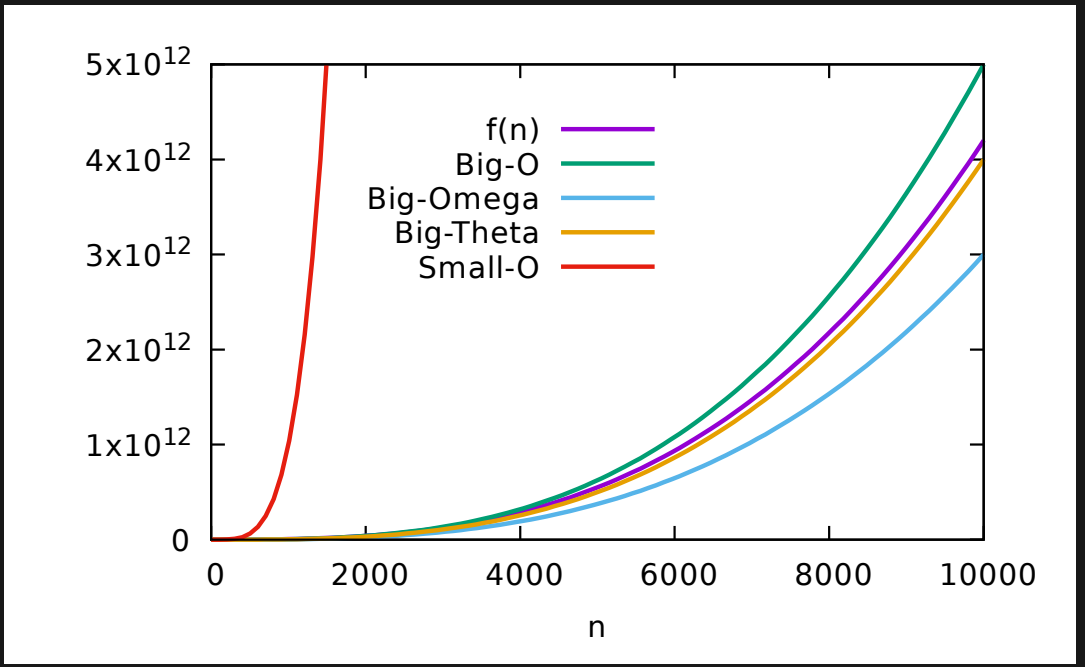
- When analyzing algorithms in this fashion, we are studying the **asymptotic efficiency** of the algorithm.
- Example: Suppose an algorithm solves a problem using $4n^3 + 2000 \cdot n^2$ steps. For large enough n, the term $4n^3$ will be much larger than $2000 \cdot n^2$ and therefore the lower-order term $2000 \cdot n^2$ will have a negligible effect on the running time on the algorithm.
- Because of this, for large enough n, a different algorithm that solves the same problem in $4n^3$ steps will have a "comparable" running time as the algorithm which takes $4n^3 + 2000 \cdot n^2$ steps, so we can ignore the term $2000 \cdot n^2$ when analyzing the running time of the original algorithm.

Asymptotic Efficiency cont.

- Similarly, for large enough n, $4n^3$ is "comparable" to n^3, and we can ignore the 4. We can then conclude the algorithm essentially takes, in an asymptotic sense, n^3 steps to solve the problem.
- When comparing the running times of two different algorithms, we are interested in comparing their asymptotic growth. That is, we are interested in comparing their running times after dropping lower order terms and multiplicative constants.
- Typical asymptotic analysis
  - We say a function $f(n)$ is asymptotically bounded by another function $g(n)$.
  - E.g., $f(n) = 4n^3 + 2000 \cdot n^2$ is asymptotically upper-bounded by $g(n) = n^3$.

Common Asymptotic Notations

- Asymptotic upper-bound, Big-O, $f(n) = O(g(n))$
- Asymptotic lower-bound, Big-Omega, $f(n) = \Omega(g(n))$
- Asymptotically similar, Big-Theta, $f(n) = \Theta(g(n))$
- Asymptotically dominated, Small-O, $f(n) = o(g(n))$
- A simple illustration of these notations.



Big-O Notation

Big-O Notation

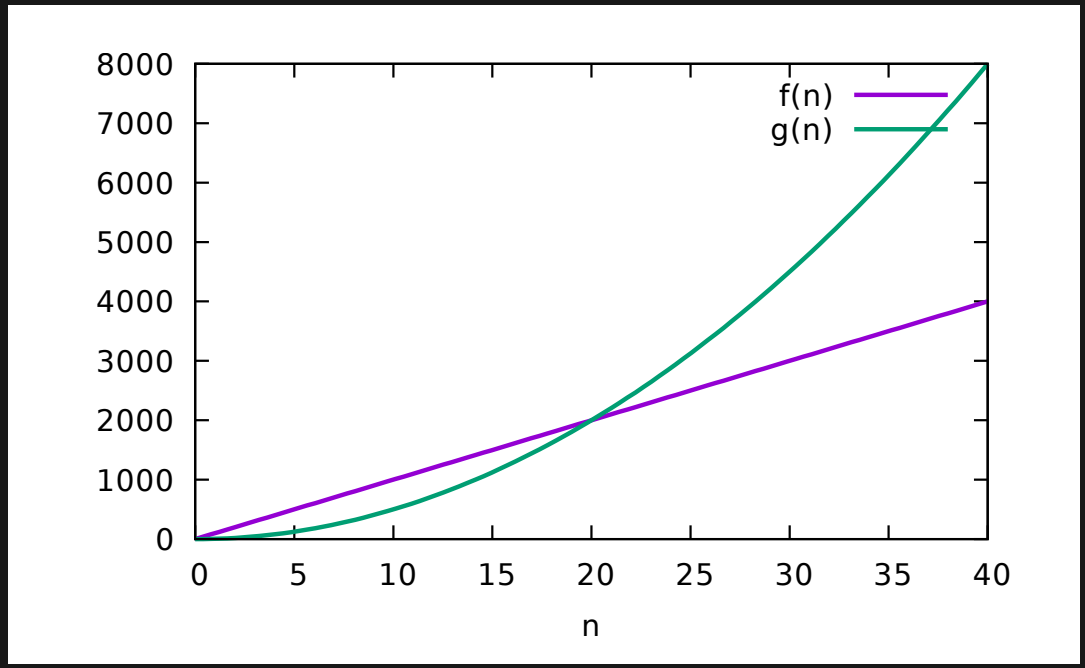
- Definition: $f(n) = O(g(n)) \iff \exists c > 0, \exists n_0 > 0, \forall n \geqslant n_0, 0 \leqslant f(n) \leqslant c \cdot g(n)$
  - c is any constant.
  - Sometimes, I will use $f(n) \in O(g(n))$ instead of $f(n) = O(g(n)$. This applies to all asymptotic notations.
- Intuitively, $f(n) = O(g(n))$ if the asymptotic growth of $f(n)$ is at most the asymptotic growth of $g(n)$. Or $g(n)$ is the upper-bound of the growth of $f(n)$.
- For example, $O(n^2) = \{n^2, 100n^2, n^2 + n, \ldots\}$

Big-O Example 1

- Example 1: $f(n) = 100n$ and $g(n) = 5 * n^2$,
  - $f(n) = O(g(n)$ because,

$$\begin{aligned} 100n \cdot 1 &\leqslant 100n \cdot n (\text{when } n \geqslant 1) \\ &= 100n^2 \leqslant 20(5n^2) = 20g(n) \end{aligned} \tag{1}$$

- An illustration of Example 1. When $n \geqslant 20$, $f(n) \leqslant g(n)$.



Big-O Example 2

]

- Let $f(n)$ be $5n^4 + 3n^2 - 7n$
- Processing

$$\begin{aligned} 5n^4 + 3n^2 - 7n &\leqslant 5n^4 + 3n^2, \text{ (b/c } (-7n) \leqslant 0) \\ &\leqslant 5n^4 + 3n^4 = 8n^4 = O(n^4), \qquad (2) \\ &\text{for } c = 8 \text{ and } n_0 = 1. \end{aligned}$$

- That is, $5n^4 + 3n^2 - 7 = O(n^4)$.

Big-O Example 3

- Let $f(n)$ be $n!$
- Processing

$$\begin{aligned} n! &= n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1 \\ &\leqslant n \cdot n \cdot n \cdot \ldots \cdot n = n^n = O(n^n) \qquad (3) \\ &\text{for } c = 1 \text{ and } n_0 = 1. \end{aligned}$$

- That is, $n! = O(n^n)$.

Big-O Example 4: Recursive Functions

- Let $f(n)$ be defined as $f(n) = 2 \cdot f(n/2) + n, \forall n > 1$, and $f(1) = 1$
- Processing

$$
\begin{aligned}
f(n) &= 2 \cdot f(n/2) + n \\
&= 2 \cdot (2 \cdot f(n/4) + n/2) + n \\
&= 4 \cdot f(n/4) + 2 \cdot n \\
&= 8 \cdot f(n/8) + 3 \cdot n \\
&\vdots \\
&= 2^{\log_2(n)} \cdot f(1) + \log_2(n) \cdot n \\
&\leqslant n + n \lg n = O(n \lg n), (\lg n \text{ is } \log_2(n)).
\end{aligned} \tag{4}
$$

More on Big-O Analysis

- We will see more recursive analysis in divide-and-conquer and Master Theorem.
- Big-O analysis is the most-commonly used run-time analysis for algorithms. More common than the other notations. However, it is very easy to reach a $g(n)$ that grows much faster than $f(n)$. For analysis accuracy, you should find a $g(n)$ that is as close to $f(n)$ as possible, especially in assignments and exams.

Big-Omega and Big-Theta Notations

Big-Omega and Big-Theta Notations

- Definition: $f(n) = \Omega(g(n)) \iff \exists c > 0, \exists n_0 > 0, \forall n \geqslant n_0, 0 \leqslant c \cdot g(n) \leqslant f(n)$.
- Intuitively, $f(n) = \Omega(g(n))$ if the asymptotic growth of $f(n)$ is at least the asymptotic growth of $g(n)$. Or $g(n)$ is the lower-bound of the growth of $f(n)$.
- Definition: $f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$.
  - Another definition: $f(n) = \Omega(g(n)) \iff \exists c_1, c_2 > 0, \exists n_0 > 0, \forall n \geqslant n_0, 0 \leqslant c_1 \cdot g(n) \leqslant f(n) \leqslant c_2 \cdot g(n)$.
- Intuitively, $f(n) = \Theta(g(n))$ if the asymptotic growth of $f(n)$ is the same as the asymptotic growth of $g(n)$.
- For example, $\Theta(n^2) = \{n^2, 100n^2, n^2 + n, \ldots\}$

Big-Omega and Big-Theta Example

- Let $f(n)$ be $5n^4 + 3n^2 - 7n$
- Processing

$$\begin{aligned} n^4 &\leqslant n^4 + 3n^2 - 7n, \text{ (when } n > \frac{7}{3}\text{, or } n > 3\text{)} \\ &\leqslant 5n^4 + 3n^4 - 7n, \\ &\text{for } c = 1 \text{ and } n_0 = 3. \end{aligned} \tag{5}$$

- That is, $5n^4 + 3n^2 - 7 = \Omega(n^4)$.
- Since we know $5n^4 + 3n^2 - 7 = O(n^4)$ from slide 11, we have $5n^4 + 3n^2 - 7 = \Theta(n^4)$.

Small-O Notation and Limit-based Definitions

Small-O Notation

- Definition: $f(n) = o(g(n)) \iff \forall c > 0, \exists n_0 > 0, \forall n \geqslant n_0, 0 \leqslant f(n) < c \cdot g(n)$.
- Intuitively, $f(n) = o(g(n))$ if the asymptotic growth of $f(n)$ is strictly less than the asymptotic growth of $g(n)$.
- Based on our definitions, $f(n) = o(g(n)) \implies f(n) = O(g(n))$.
- If is somehow hard to distinguish big-O and small-O based on these definitions.

Small-O Limit-based Definition

- All these notations can also be defined based on Limit theorem.

Big-O	f(n)=O(g(n))	$\limsup_{n\to\infty} \frac{\lvert f(n)\rvert}{g(n)} < \infty$
Small-O	f(n)=o(g(n))	$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$
Big-Omega	f(n)=Ω(g(n))	$\lim_{n\to\infty} \lvert\frac{f(n)}{g(n)}\rvert > 0$
Big-Theta	f(n)=Θ(g(n))	$f(n) = O(g(n)) \wedge$ $f(n) = \Omega(g(n))$

Table: Algorithm analysis notation definitions based on Limit Theorem.

Small-O Examples

- $n = o(n^2)$?
- $5n^2 = o(n^2)$?
- $ln(n) = o(n^2)$?
- $2^n = o(3^n)$?

Small-O Examples

- $n = o(n^2)$?
  - Yes, because $\lim_{n \to \infty} \frac{n}{n^2} = 0$.
- $5n^2 = o(n^2)$?
- $ln(n) = o(n^2)$?
- $2^n = o(3^n)$?

Small-O Examples

- $n = o(n^2)$?
  - Yes, because $\lim_{n\to\infty} \frac{n}{n^2} = 0$.
- $5n^2 = o(n^2)$?
  - No, because $\lim_{n\to\infty} \frac{5n^2}{n^2} = 5 \neq 0$.
- $ln(n) = o(n^2)$?

- $2^n = o(3^n)$?

Small-O Examples

- $n = o(n^2)$?
  - Yes, because $\lim_{n\to\infty} \frac{n}{n^2} = 0.$
- $5n^2 = o(n^2)$?
  - No, because $\lim_{n\to\infty} \frac{5n^2}{n^2} = 5 \neq 0.$
- $ln(n) = o(n^2)$?
  - Yes, because $\lim_{n\to\infty} \frac{ln(n)}{n^2} = \lim_{n\to\infty} \frac{ln(n)'}{n^{2'}} = \lim_{n\to\infty} \frac{\frac{1}{n}}{2n} = 0.$
- $2^n = o(3^n)$?

Small-O Examples

- $n = o(n^2)$?
  - Yes, because $\lim_{n\to\infty} \frac{n}{n^2} = 0.$
- $5n^2 = o(n^2)$?
  - No, because $\lim_{n\to\infty} \frac{5n^2}{n^2} = 5 \neq 0.$
- $ln(n) = o(n^2)$?
  - Yes, because $\lim_{n\to\infty} \frac{ln(n)}{n^2} = \lim_{n\to\infty} \frac{ln(n)'}{n^{2'}} = \lim_{n\to\infty} \frac{\frac{1}{n}}{2n} = 0.$
- $2^n = o(3^n)$?
  - Yes, because $\lim_{n\to\infty} \frac{2^n}{3^n} = \lim_{n\to\infty} (\frac{2}{3})^n = 0.$

Loops and Their Time Complexity

Time Complexity

The **time complexity** of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the string representing the input.[1]:226 The time complexity of an algorithm is commonly expressed using big-O notation.

Examples of Loops and Their Time Complexity

- Loop 1: $for(i = 0; i < n; i++)\{\ldots\}$
- Loop 2: $for(i = 0; i < n; i+ = 2)\{\ldots\}$
- Loop 3: $for(i = 0; i < n; i* = 2)\{\ldots\}$
- Loop 4:
  $for(i = 1; i <= n; i++)\{for(j = 0; j < n; j+ = i)\{\ldots\}\}$

- Note that, for every example here, we can replace big-O with big-Theta.

Examples of Loops and Their Time Complexity

- Loop 1: $for(i = 0; i < n; i++)\{\ldots\}$
  - n iterations. Time complexity is $O(n)$.
- Loop 2: $for(i = 0; i < n; i+ = 2)\{\ldots\}$
- Loop 3: $for(i = 0; i < n; i* = 2)\{\ldots\}$
- Loop 4:
  $for(i = 1; i <= n; i++)\{for(j = 0; j < n; j+ = i)\{\ldots\}\}$
- Note that, for every example here, we can replace big-O with big-Theta.

Examples of Loops and Their Time Complexity

- Loop 1: $for(i = 0; i < n; i++)\{\ldots\}$
  - n iterations. Time complexity is $O(n)$.
- Loop 2: $for(i = 0; i < n; i+ = 2)\{\ldots\}$
  - $n/2$ iterations. Time complexity is $O(n)$.
- Loop 3: $for(i = 0; i < n; i* = 2)\{\ldots\}$
- Loop 4:
  $for(i = 1; i <= n; i++)\{for(j = 0; j < n; j+ = i)\{\ldots\}\}$
- Note that, for every example here, we can replace big-O with big-Theta.

Examples of Loops and Their Time Complexity

- Loop 1: $for(i = 0; i < n; i++)\{\ldots\}$
  - n iterations. Time complexity is $O(n)$.
- Loop 2: $for(i = 0; i < n; i += 2)\{\ldots\}$
  - $n/2$ iterations. Time complexity is $O(n)$.
- Loop 3: $for(i = 0; i < n; i *= 2)\{\ldots\}$
  - $\lg(n)$ iterations. Time complexity is $O(\lg(n))$.
- Loop 4:
  $for(i = 1; i <= n; i++)\{for(j = 0; j < n; j += i)\{\ldots\}\}$

- Note that, for every example here, we can replace big-O with big-Theta.

Examples of Loops and Their Time Complexity

- Loop 1: $for(i = 0; i < n; i++)\{\ldots\}$
  - n iterations. Time complexity is $O(n)$.
- Loop 2: $for(i = 0; i < n; i += 2)\{\ldots\}$
  - $n/2$ iterations. Time complexity is $O(n)$.
- Loop 3: $for(i = 0; i < n; i *= 2)\{\ldots\}$
  - $\lg(n)$ iterations. Time complexity is $O(\lg(n))$.
- Loop 4:
  $for(i = 1; i <= n; i++)\{for(j = 0; j < n; j += i)\{\ldots\}\}$
  - Outer loop has n iterations.
  - Inner loop has $n, n/2, n/3, \ldots, 1$ iterations.
  - The total number of iterations is $\sum_{i=1} n\frac{n}{i} = n \cdot \sum_{i=1} n\frac{1}{i} = n \cdot O(logn) = O(nlogn)$ (check Harmonic number).
- Note that, for every example here, we can replace big-O with big-Theta.