Lecture 2: Asymptotic Notation
CSCI 700 - Algorithms I

Andrew Rosenberg

Last Time

- Review
- Insertion Sort
  - Analysis of Runtime
  - Proof of Correctness

Today

- Asymptotic Notation
  - Its use in analyzing runtimes.
- Review of Proofs by Induction
  - An important tool for correctness proofs.

Two Approaches

1. Emperical
   - Generate input.
   - Time the executable.
2. Theoretical
   - Prove the runtime as a function of the size of the input.
   - This is what we'll be focusing on.
   - The RAM model.
     - All information can be held in RAM
     - All operations take the same amount of resources each time they're called.

Asymptotic Notation

- Runtime on large inputs is more important than small input.
    - Sometimes, inefficient, but frequently called subroutines that operate on small input can have a dramatic impact on overall performance. But this isnt the kind of efficiency well be analyzing here.
- It is usually safe to assume that live inputs are larger and less well-formed than test input used during development.

Classes of Functions

- We want to develop "classes of functions" to compare the runtime of an algorithm across machines.
- Criteria
  1. We only care about the behavior of the algorithm on input with size greater than some value n_0.
  2. To transfer our analyses across machines, we consider functions that differ by at most a constant multiplier to be equivalent.

Definition

- Asymptotic Notation is a formal notation for discussing and analyzing "classes of functions".
- Criteria
  1. Capture behavior when $n_0 \leq n \to \infty$.
  2. Functions that differ by at most a constant multiplier are considered equivalent.

Example

- One one machine an algorithm may take $T(n) = 15n^3 + n^2 + 4$
- On another, $T(n) = 5n^3 + 4n + 5$
- Both will belong to the same class of functions. Namely, "cubic functions of n".
- Function classes can be thought of at "how many times we do something to each input".

O(n) definition

Definition

$f(n) \in O(g(n))$ if there exists constants $c > 0$ and $n_0 > 0$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

- "Big-O" notation
- $O(g(n))$ is an upper-bound on the growth of a function, $f(n)$.

O(n) proofs

- Prove that if $T(n) = 15n^3 + n^2 + 4$, $T(n) = O(n^3)$.

Proof.

Let $c = 20$ and $n_0 = 1$.
Must show that $0 \leq f(n)$ and $f(n) \leq cg(n)$.
$0 \leq 15n^3 + n^2 + 4$ for all $n \geq n_0 = 1$.
$f(n) = 15n^3 + n^2 + 4 \leq 15n^3 + n^3 + 4n^3$
$15n^3 + n^3 + 4n^3 = 20n^3 = 20g(n) = cg(n)$

$\square$

O(n) proofs

- Prove that if $T(n) = 15n^3 + n^2 + 4$, $T(n) = O(n^4)$.

Proof.

Let $c = 20$ and $n_0 = 1$.
Must show that $0 \leq f(n)$ and $f(n) \leq cg(n)$.
$0 \leq 15n^3 + n^2 + 4$ for all $n \geq n_0 = 1$.
$f(n) = 15n^3 + n^2 + 4 \leq 15n^4 + n^4 + 4n^4$
$15n^4 + n^4 + 4n^4 = 20n^4 = 20g(n) = cg(n)$

$\square$

O(n) recap

- $T(n) = 15n^3 + n^2 + 4$
- $T(n) = O(n^3)$.
- $T(n) = O(n^4)$.
- $O(n)$ is an upper bound
- "$=$" is not really equality. It's used as "set inclusion" $\in$ here.
- Don't use $O(n) = T(n)$

$\Omega(n)$ definition

Definition

$f(n) \in \Omega(g(n))$ if there exists constants $c > 0$ and $n_0 > 0$ such that $0 \leq c \cdot g(n) \leq f(n)$ for all $n \geq n_0$

- "Big-Omega of n"
- $\Omega(g(n))$ is lower-bound on the growth of a function, $f(n)$.

$\Omega(n)$ proofs

- Prove that if $T(n) = 15n^3 + n^2 + 4,\ T(n) = \Omega(n^3)$.

Proof.

Let $c = 15$ and $n_0 = 1$.
Must show that $0 \leq cg(n)$ and $cg(n) \leq f(n)$.
$0 \leq 15n^3$ for all $n \geq n_0 = 1$.
$cg(n) = 15n^3 \leq 15n^3 + n^2 + 4 = f(n)$

$\square$

$\Omega(n)$ proofs

- Prove that if $T(n) = 15n^3 + n^2 + 4,\ T(n) = \Omega(n^2)$.

Proof.

Let $c = 15$ and $n_0 = 1$.
Must show that $0 \leq cg(n)$ and $cg(n) \leq f(n)$.
$0 \leq 15n^3$ for all $n \geq n_0 = 1$.
$cg(n) = 15n^2 \leq 15n^3 \leq 15n^3 + n^2 + 4 = f(n)$

$\square$

$\Omega(n)$ recap

- $T(n) = 15n^3 + n^2 + 4$
- $T(n) = \Omega(n^3)$.
- $T(n) = \Omega(n^2)$.
- $\Omega(n)$ is a lower bound

$\Theta(n)$ definition

Definition

$f(n) \in \Theta(g(n))$ if there exists constants $c_1 > 0$, $c_2 > 0$ and $n_0 > 0$ such that $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ for all $n \geq n_0$

- "Theta of n"
- $\Theta(g(n))$ is tight bound on the growth of a function, $f(n)$.
- Can also say, $f(n) = \Theta(g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$
- $T(n) = 15n^3 + n^2 + 4$
- $T(n) = \Theta(n^3)$.

$\Theta(n)$ proof

- Prove that if $T(n) = 15n^3 + n^2 + 4,\ T(n) = \Theta(n^3)$.

Proof.

Let $c_1 = 15$, $c_2 = 20$ and $n_0 = 1$.
Must show that $c_1 g(n) \leq f(n)$ and $f(n) \leq c_2 g(n)$.
$c_1 g(n) = 15n^3 \leq 15n^3 + n^2 + 4 = f(n)$.
$f(n) = 15n^3 + n^2 + 4 \leq 15n^3 + n^3 + 4n^3 = 20n^3 = c_2 g(n)$.

□

Asymptotic Notation Examples

- $T(n) = 15n^3 + n^2 + 4 = \Theta(n^3)$
- $T(n) = 2n^3 + 4n + 4 = \Theta(n^3)$
- $T(n) = 2n^2 + 4n + 4 = \Theta(n^2)$
- $T(n) = 15n^3 + n^2 + 4 = \Theta(2n^3 + n^2)$
  - While true, this is never done.
- Note: $f = \Theta(g)$ is a common shorthand for $f(n) = \Theta(g(n))$

Asymptotic Notation Examples

- $T(n) = n^3 + n \log n = \Theta(n^3)$?
- If $n \log n < n^3$ for all $n > n_0$, then we can use the proof structure from before.
- Let $c_1 = 1$.
- $c_1 g(n) = n^3 \leq n^3 + n \log n$.
- Let $c_2 = 2$.
- $n^3 + n \log n \leq n^3 + n^3 = 2n^3 = c_2 g(n)$

Runtime at small n

- $T_1(n) = n^3 = \Theta(n^3)$
- $T_2(n) = 2n + 100 = \Theta(n)$

n	$T_1(n)$	$T_2(n)$
1	1	102
2	8	104
3	27	106
4	64	108
5	125	110
6	216	112

Asymptotic Notation Properties

- $f = \Theta(h)$ and $g = \Theta(h)$ then $f + g = \Theta(h)$
  - $f = \Theta(h) = c_1 h + slack$.
  - $g = \Theta(h) = c_2 h + slack$.
  - $f + g = (c_1 + c_2)h + slack$.

- $\Theta(f + g) = \Theta(\max(f, g))$
  - $f = n^3$, $g = n \log n$, $h = f + g = n^3 + n \log n$
  - $\Theta(h) = \Theta(f + g) = \Theta(\max(f, g)) = \Theta(n^3)$

Orders of growth

- Behavior of a function as $n \to \infty$.

- if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ then $f(n) = \Omega(g(n))$
  - "f is bigger than g", or "f dominates g"
- if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ then $f(n) = O(g(n))$
  - "f is smaller than g", or "g dominates f"
- if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$ then $f(n) = \Theta(g(n))$
  - "f is the same as g"

Orders of growth examples

- Show that $n^2 = \Omega(n)$
- Prove it to yourself first.

n	n^2	$n^2 - n$
1	1	0
2	4	2
3	9	6
4	16	12
5	25	19
6	36	30

Orders of growth examples

- Show that $n^2 = \Omega(n)$.
- Evaluate $\lim_{n\to\infty} \frac{f(n)}{g(n)}$
- $\lim_{n\to\infty} \frac{n^2}{n}$
- $\lim_{n\to\infty} n = \infty$ so $n^2 = \Omega(n)$

Orders of growth examples

- Show that $n = \Omega(\log n)$.
- Prove it to yourself first.

n	$\log n$	$n - \log n$
1	0	1
4	2	2
8	3	5
16	4	12
32	5	27
64	6	58

Orders of growth example.

- Show that $n^2 = \Omega(n)$.
- Evaluate $\lim_{n\to\infty} \frac{f(n)}{g(n)}$
- $\lim_{n\to\infty} \frac{n}{\log n}$
- l'Hopital's rule. (from Calculus)
- if $lim_{n\to c} f(n) = \infty$ and $lim_{n\to c} g(n) = \infty$, then $\lim_{n\to c} \frac{f(n)}{g(n)} = \lim_{n\to c} \frac{f'(n)}{g'(n)}$
- $\lim_{n\to\infty} \frac{n}{\log n} = \lim_{n\to\infty} \frac{1}{1/n}$
- $\lim_{n\to\infty} \frac{n}{\log n} = \lim_{n\to\infty} n = \infty$
- So, $n = \Omega(\log n)$

Order of growth of a summation

$$S(n) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

- Even if we don't know this, we can show that $S(n) = \Theta(n^2)$

Proof.

Show $S(n) = O(n^2)$.

$$S(n) = \sum_{i=1}^{n} i \leq \sum_{i=1}^{n} n = n \cdot n = n^2$$

$$S(n) \leq n^2$$

$\square$

Order of growth of a summation

Proof.

Show $S(n) = \Omega(n^2)$.

$$S(n) = \sum_{i=1}^{n} i \geq \sum_{i=n/2+1}^{n} i \geq \sum_{i=n/2+1}^{n} \frac{n}{2}$$

$$\sum_{i=n/2+1}^{n} \frac{n}{2} = \frac{n}{2} \cdot \frac{n}{2} = \frac{n^2}{4}$$

Thus, $S(n) = \Omega(n^2)$, and $S(n) = \Theta(n^2)$

$\square$

Arithmetic series

$$S(n) = \sum_{i=1}^{n} i$$

$$\begin{aligned} S(6) &= \sum_{i=1}^{6} i \\ &= 1 + 2 + 3 + 4 + 5 + 6 \\ &= (1 + 6) + (2 + 5) + (3 + 4) \\ &= 3 * 7 \end{aligned}$$

Arithmetic series

$$S(n) = \sum_{i=1}^{n} i$$

$$\begin{aligned} S(n) &= \sum_{i=1}^{n} i \\ &= 1 + 2 + \ldots + n \\ &= (1 + n) + (2 + (n - 1)) + ...(\frac{n}{2} + \frac{n}{2} + 1) \\ &= \frac{n}{2}(n + 1) \end{aligned}$$

Inductive Proof

$$S(n) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

- **Base Case:** S(1)

$$\begin{aligned} S(1) &= \sum_{i=1}^{1} i \\ &= 1 \\ \frac{n(n+1)}{2} &= \frac{1(1+1)}{2} \\ &= \frac{2}{2} \\ &= 1 \end{aligned}$$

- **Inductive Step:** Assume true for $n \le n_0 = 1$. Prove for $n+1$.

$$
\begin{aligned}
S(n+1) &= \sum_{i=1}^{n+1} i \\
&= (n+1) + \sum_{i=1}^{n} i \\
&= (n+1) + \frac{n(n+1)}{2} \\
&= \frac{2(n+1)}{2} + \frac{n(n+1)}{2} \\
&= \frac{2(n+1) + n(n+1)}{2} \\
&= \frac{(n+2)(n+1)}{2} \\
&= \frac{(n+1)((n+1)+1)}{2}
\end{aligned}
$$

Induction over structures

- The structure of an inductive proof can be applied to structures.
  - Sometimes using the construct of **Loop invariants**.
  - Initialization = Base Case
  - Maintenance = Inductive Step
  - Termination

Insertion Sort Pseudocode

InsertionSort(A)

for $j \leftarrow 2$ **to** $size(A)$ **do**
  $key \leftarrow A[j]$
  $i \leftarrow j - 1$
  while $i > 0$ **and** $A[i] > key$ **do**
    {**Slide the array to the right**}
    $A[i+1] \leftarrow A[i]$
    $i \leftarrow i + 1$
  end while
  $A[i+1] \leftarrow key$
end for

Insertion Sort

Start of the Loop

key:2

			j	i		
A[i]:	5	2	4	6	1	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:2

			j	i		
A[i]:	5	2	4	6	1	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:2 j i

A[i]:	5	5	4	6	1	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:2			j	i		
A[i]:	2	5	4	6	1	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

Start of the Loop

key:4			j	i		
A[i]:	2	5	4	6	1	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:4			j	i		
A[i]:	2	5	4	6	1	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:4

			j	i		
A[i]:	2	5	5	6	1	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:4 j i

			j	i		
A[i]:	2	5	5	6	1	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:4 j i

A[i]:	2	4	5	6	1	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

Start of the Loop

key:6			j	i		
A[i]:	2	4	5	6	1	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:6   j   i

A[i]:	2	4	5	6	1	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:6			j	i		
A[i]:	2	4	5	6	1	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

Start of the Loop

key:1 j i

A[i]:	2	4	5	6	1	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:1 j i

A[i]:	2	4	5	6	1	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:1 j i

A[i]:	2	4	5	6	6	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:1 | | | j | i | |
--- | --- | --- | --- | --- | --- | ---

key:1			j	i		
A[i]:	2	4	5	6	6	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:1 j i

			j	i		
A[i]:	2	4	5	5	6	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:1			j	i		
A[i]:	2	4	5	5	6	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:1 j i

A[i]:	2	4	4	5	6	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:1			j	i		
A[i]:	2	4	4	5	6	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:1 j i

			j	i		
A[i]:	2	2	4	5	6	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:1 j i

A[i]:	1	2	4	5	6	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

Start of the Loop

key:3			j	i		
A[i]:	1	2	4	5	6	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:3			j	i		
A[i]:	1	2	4	5	6	3
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:3 j i

A[i]:	1	2	4	5	6	6
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:3			j	i		
A[i]:	1	2	4	5	6	6
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:3			j	i		
A[i]:	1	2	4	5	5	6
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:3 j i

A[i]:	1	2	4	5	5	6
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:3 j i

A[i]:	1	2	4	4	5	6
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:3 j i

A[i]:	1	2	4	4	5	6
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

key:3 j i

A[i]:	1	2	3	4	5	6
idx:	*1*	*2*	*3*	*4*	*5*	*6*

Insertion Sort

Termination

key:3			j	i		
A[i]:	1	2	3	4	5	6
idx:	*1*	*2*	*3*	*4*	*5*	*6*

InsertionSort Loop Invariant

- At the start of the for loop, the subarray A[1..j-1] is sorted.
  - "Sorted" here means contains the initial items of A[1..j-1] in ascending order.

InsertionSort Induction Proof

- Induction Proof: At the start of the for loop, the subarray A[1..j-1] is sorted.
- **Base Case:** At initialization $j = 2$.
  - The subarray A[1..j-1] contains a single element, $A[1]$.
  - A single element is in sorted order.

InsertionSort Induction Proof

- Induction Proof: At the start of the for loop, the subarray A[1..j-1] is sorted.
- **Induction Step:** Assume true for $j > n_0$. Prove for $j + 1$.
  1. Show that A[j+1] is put in the correct position.
  2. Show that A[1..j+1] remains sorted.

InsertionSort Induction Proof

- Induction Proof: At the start of the for loop, the subarray A[1..j-1] is sorted.
- **Induction Step:** Assume true for $j > n_0$. Prove for $j + 1$.
  1. Show that A[j+1] is put in the "correct;; position.
     - The index i is used to find the correct position to insert $A[j+1]$, specifically, the position i such that, $A[i] \leq A[j+1] < A[i+2]$.
     - At the end of the internal while loop, i points to element with the greatest index smaller than $A[j+1]$, or 0 if no such element exists.
     - $A[j+1]$ is moved to the position $i+1$.
     - Because $A[1..j]$ is sorted, every element $A[1..i]$ is smaller than $A[j]$ and every element $A[(i+1)..j]$ is larger than $A[j+1]$.
     - Therefore $A[j+1]$ is put in the correct position.

InsertionSort Induction Proof

- Induction Proof: At the start of the for loop, the subarray A[1..j-1] is sorted.
- **Induction Step:** Assume true for $j > n_0$. Prove for $j + 1$.
  2. Show that A[1..j+1] remains sorted.
     - At the end of the internal while loop, the elements $A[i + 1..j]$ have been each been shifted up one position to $A[i + 1..j + 1]$. Therefore A[i+2..j+1] is sorted.
     - $A[1..i]$ has not been modified. Since A[1..j] was sorted at the start of the loop, the subarray $A[1..i]$ remains sorted.
     - Since A[j+1] was put in the position, i+1 such that $A[i] \leq A[j + 1] < A[i + 2]$ the resulting array $A[1..i], A[i + 1], A[i + 2..j + 1]$ is sorted.

InsertionSort Induction Proof

- Induction Proof: At the start of the for loop, the subarray A[1..j-1] is sorted.
- **Termination:** At the termination of the for loop, $j = n + 1$, therefore A[1..n] is sorted.

Bye

- Homework 2 is posted to the course website.
- Next time (9/7)
  - Recursion
- For Next Class
  - Read Sections 2.1, 2.2, 7.1, 7.2, 7.4