Design and Analysis of Algorithms
Lecture-2: Asymptotic Analysis

Prof. Eugene Chang

Overview

- Asymptotic notations
- Order of growth
- Homework-1
- Lecture materials are shared with Prof K. K. Low
- Slides are based on material from Prof. Jianhua Ruan, The University of Texas at San Antonio

Analysis of insertion Sort

```
InsertionSort(A, n) {
 for j = 2 to n {
      key = A[j]
      i = j - 1;
      while (i > 0) and (A[i] > key) {
           A[i+1] = A[i]
           i = i - 1
      }
      A[i+1] = key
 }
}
```

How many times will this line execute?

Analysis of insertion Sort

```
InsertionSort(A, n) {
 for j = 2 to n {
      key = A[j]
      i = j - 1;
      while (i > 0) and (A[i] > key) {
           A[i+1] = A[i]
           i = i - 1
      }
      A[i+1] = key
 }
}
```

How many times will this line execute?

Analysis of insertion Sort

Statement	cost	time
`InsertionSort(A, n) {`		
`  for j = 2 to n {`	c_1	n
`        key = A[j]`	c_2	(n-1)
`        i = j - 1;`	c_3	(n-1)
`        while (i > 0) and (A[i] > key) {`	c_4	S
`                A[i+1] = A[i]`	c_5	(S-(n-1))
`                i = i - 1`	c_6	(S-(n-1))
`        }`	0	
`        A[i+1] = key`	c_7	(n-1)
`  }`	0	
`}`		

$S = t_2 + t_3 + \ldots + t_n$ where t_j is number of while expression evaluations for the j^{th} for loop iteration

Analyzing Insertion Sort

- $T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 S + c_5(S - (n-1)) + c_6(S - (n-1)) + c_7(n-1)$
  $= c_8 S + c_9 n + c_{10}$
- What can S be?
  - Best case -- inner loop body never executed
    - $t_j = 1 \Rightarrow S = n - 1$
    - $T(n) = an + b$ is a linear function
  - Worst case -- inner loop body executed for all previous elements
    - $t_j = j \Rightarrow S = 2 + 3 + \ldots + n = n(n+1)/2 - 1$
    - $T(n) = an^2 + bn + c$ is a quadratic function
  - Average case
    - Can assume that on average, we have to insert A[j] into the middle of A[1..j-1], so $t_j = j/2$
    - $S \approx n(n+1)/4$
    - T(n) is still a quadratic function

Analysis of insertion Sort

Statement	cost	time
`InsertionSort(A, n) {`		
`for j = 2 to n {`	c_1	n
`key = A[j]`	c_2	(n-1)
`i = j - 1;`	c_3	(n-1)
`while (i > 0) and (A[i] > key) {`	c_4	S
`A[i+1] = A[i]`	c_5	(S-(n-1))
`i = i - 1`	c_6	(S-(n-1))
`}`	0	
`A[i+1] = key`	c_7	(n-1)
`}`	0	
`}`		

What are the basic operations (most executed lines)?

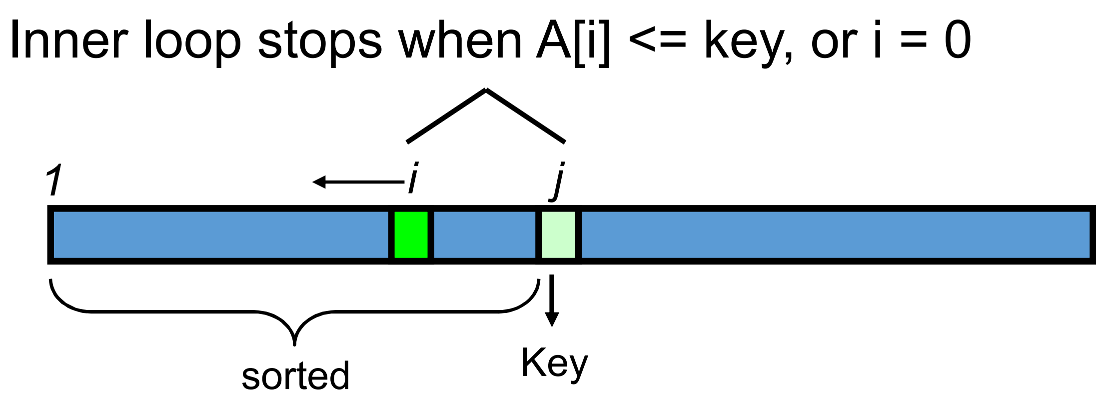
Analysis of insertion Sort

Statement	cost	time
`InsertionSort(A, n) {`		
`for j = 2 to n {`	c_1	n
`key = A[j]`	c_2	(n-1)
`i = j - 1;`	c_3	(n-1)
`while (i > 0) and (A[i] > key) {`	c_4	S
`A[i+1] = A[i]`	c_5	(S-(n-1))
`i = i - 1`	c_6	(S-(n-1))
`}`	0	
`A[i+1] = key`	c_7	(n-1)
`}`	0	
`}`		

Analysis of insertion Sort

Statement	cost	time
`InsertionSort(A, n) {`		
`for j = 2 to n {`	c_1	n
`key = A[j]`	c_2	(n-1)
`i = j - 1;`	c_3	(n-1)
`while (i > 0) and (A[i] > key) {`	c_4	S
`A[i+1] = A[i]`	c_5	(S-(n-1))
`i = i - 1`	c_6	(S-(n-1))
`}`	0	
`A[i+1] = key`	c_7	(n-1)
`}`	0	
`}`		

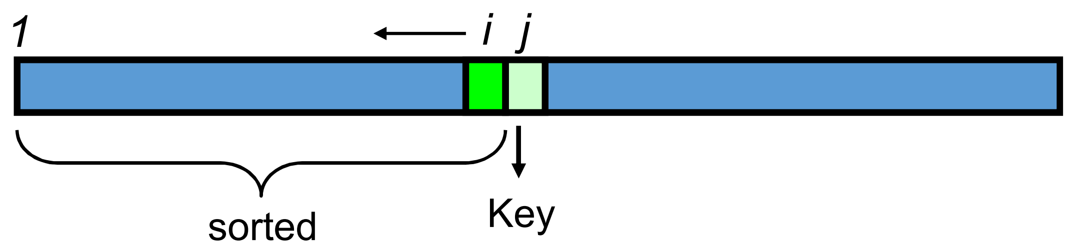
What can S be?



- $S = \sum_{j=2..n} t_j$
- Best case:
- Worst case:
- Average case:

Best case

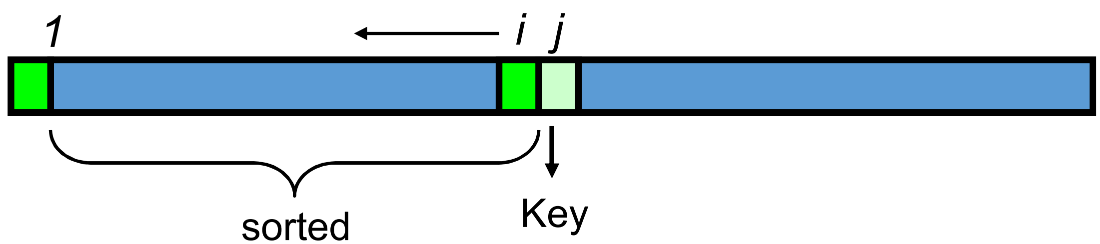
Inner loop stops when A[i] <= key, or i = 0



- Array already sorted
- $S = \sum_{j=2..n} t_j$
- $t_j = 1$ for all j
- $S = n-1 \qquad T(n) = \Theta(n)$

Worst case

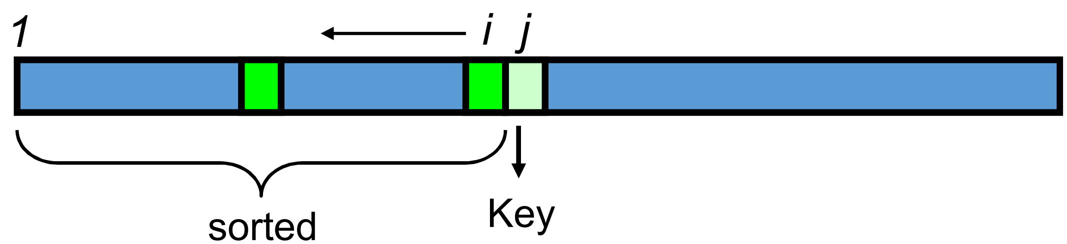
Inner loop stops when i = 0



- Array originally in reverse order sorted
- $S = \sum_{j=2..n} t_j$
- $t_j = j$
- $S = \sum_{j=2..n} j = 2 + 3 + \ldots + n = (n-1)(n+2) / 2 = \Theta(n^2)$

Average case

Inner loop stops when A[i] <= key



- Array in random order
- $S = \sum_{j=2..n} t_j$
- $t_j = j / 2$ on average
- $S = \sum_{j=2..n} j/2 = ½ \sum_{j=2..n} j = (n-1)(n+2) / 4 = \Theta(n^2)$

What if we use binary search?

Answer: still $\Theta(n^2)$

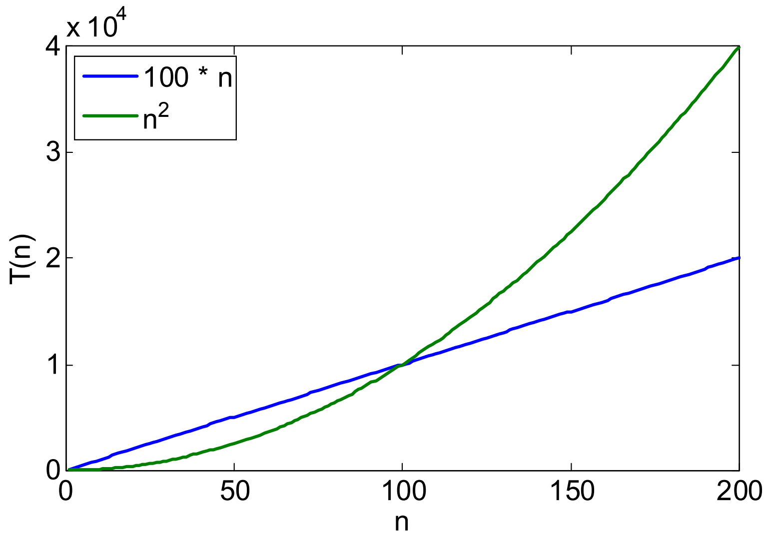
Asymptotic Analysis

- Running time depends on the size of the input
  - Larger array takes more time to sort
  - T(n): the time taken on input with size n
  - Look at ***growth*** of $T(n)$ as $n \to \infty$.

    "Asymptotic Analysis"
- Size of input is generally defined as the number of input elements
  - In some cases may be tricky

Asymptotic Analysis

- Ignore actual and abstract statement costs
- *Order of growth* is the interesting measure:
  - Highest-order term is what counts
    - As the input size grows larger it is the high order term that dominates



Comparison of functions

	$\log_2 n$	n	$n\log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10	33	10^2	10^3	10^3	10^6
10^2	6.6	10^2	660	10^4	10^6	10^{30}	10^{158}
10^3	10	10^3	10^4	10^6	10^9		
10^4	13	10^4	10^5	10^8	10^{12}		
10^5	17	10^5	10^6	10^{10}	10^{15}		
10^6	20	10^6	10^7	10^{12}	10^{18}		

For a super computer that does 1 trillion operations per second, it will be longer than 1 billion years

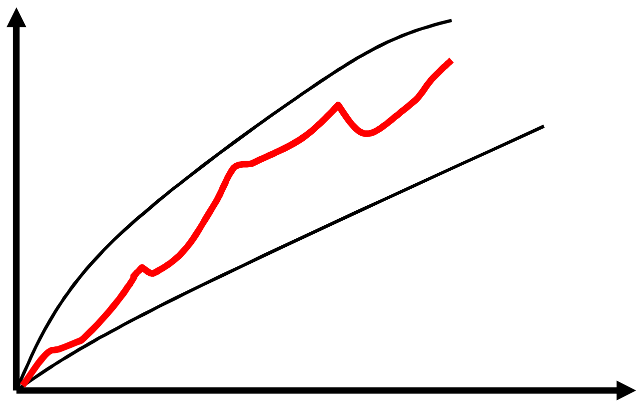
Order of growth

$1 << \log_2 n << n << n\log_2 n << n^2 << n^3 << 2^n << n!$

(We are slightly abusing of the “<<“ sign. It means a smaller order of growth).

Exact analysis is hard!

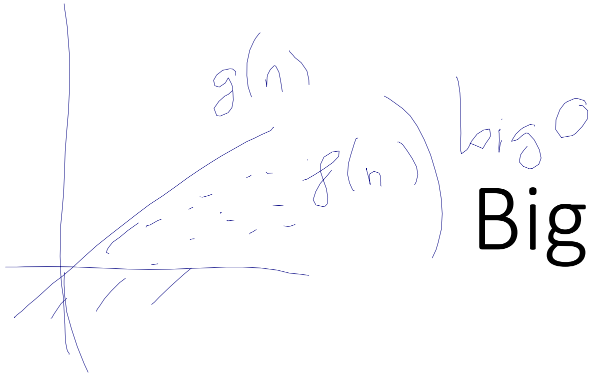
- Worst-case and average-case are difficult to deal with precisely, because the details are very complicated



It may be easier to talk about upper and lower bounds of the function.

Asymptotic notations

- O: Big-Oh
- Ω: Big-Omega
- Θ: Theta
- o: Small-oh
- ω: Small-omega



Big O

- Informally, O (g(n)) is the set of all functions with a smaller or same order of growth as g(n), within a constant multiple
- If we say f(n) is in O(g(n)), it means that g(n) is an asymptotic upper bound of f(n)
  - Intuitively, it is like f(n) ≤ g(n)
- What is $O(n^2)$?
  - The set of all functions that grow slower than or in the same order as n^2

Big O

So:

$n \in O(n^2)$

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

$1000n \in O(n^2)$

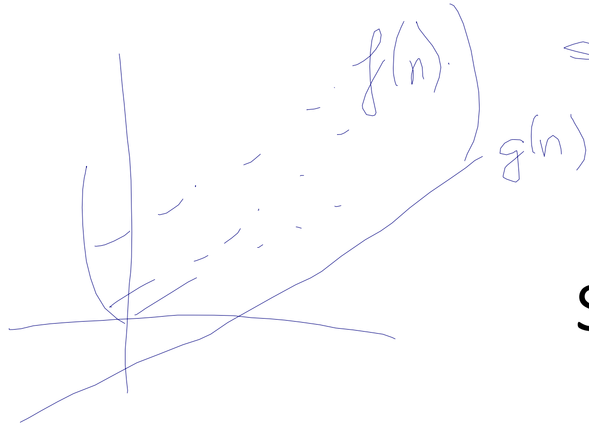
$n^2 + n \in O(n^2)$

$100n^2 + n \in O(n^2)$

But:

$1/1000\ n^3 \notin O(n^2)$

Intuitively, O is like ≤



small o

small o

- Informally, o (g(n)) is the set of all functions with a strictly smaller growth as g(n), within a constant multiple
- What is $o(n^2)$?
  - The set of all functions that grow slower than n^2

So:

$1000n \in o(n^2)$

But:

$n^2 \notin o(n^2)$

Intuitively, o is like <

Big Ω

- Informally, Ω (g(n)) is the set of all functions with a larger or same order of growth as g(n), within a constant multiple
- $f(n) \in \Omega(g(n))$ means g(n) is an asymptotic lower bound of f(n)
  - Intuitively, it is like $g(n) \le f(n)$

So:

$n^2 \in \Omega(n)$

$1/1000\ n^2 \in \Omega(n)$

But:

$1000\ n \notin \Omega(n^2)$

Intuitively, Ω is like ≥

small ω

- Informally, ω (g(n)) is the set of all functions with a larger order of growth as g(n), within a constant multiple

So:

$n^2 \in \omega(n)$

$1/1000\ n^2 \in \omega(n)$

$n^2 \notin \omega(n^2)$

Intuitively, ω is like >

Theta (Θ)

- Informally, Θ (g(n)) is the set of all functions with the same order of growth as g(n), within a constant multiple
- f(n) $\in$ Θ(g(n)) means g(n) is an asymptotically tight bound of f(n)
  - Intuitively, it is like f(n) = g(n)
- What is Θ(n^2)?
  - The set of all functions that grow in the same order as n^2

Examples

So:

$n^2 \in \Theta(n^2)$

$n^2 + n \in \Theta(n^2)$

$100n^2 + n \in \Theta(n^2)$

$100n^2 + \log_2 n \in \Theta(n^2)$

But:

$n\log_2 n \notin \Theta(n^2)$

$1000n \notin \Theta(n^2)$

$1/1000\ n^3 \notin \Theta(n^2)$

Intuitively, Θ is like =

Tricky cases

- How about sqrt(n) and $\log_2 n$?
- How about $\log_2 n$ and $\log_{10} n$
- How about 2^n and 3^n
- How about 3^n and n!?

Big-Oh

There exist

For all

- Definition:

  $O(g(n)) = \{f(n): \exists$ positive constants c and n_0 such that $0 \le f(n) \le cg(n)\ \forall\ n \ge n_0\}$

- $\lim_{n \to \infty} g(n)/f(n) > 0$ (if the limit exists.)
- Abuse of notation (for convenience):

  $f(n) = O(g(n))$ actually means $f(n) \in O(g(n))$

Big-Oh

- Claim: $f(n) = 3n^2 + 10n + 5 \in O(n^2)$
- Proof by definition

(Hint: to prove this claim by definition, we need to find some positive constants c and n_0 such that $f(n) <= cn^2$ for all $n \geq n_0$.)

(Note: you just need to find one concrete example of c and n_0 satisfying the condition, but it needs to be correct for all $n \geq n_0$. So do not try to plug in a concrete value of n and show the inequality holds.)

Proof:

$3n^2 + 10n + 5 \leq 3n^2 + 10n^2 + 5, \forall\ n \geq 1$

$\leq 3n^2 + 10n^2 + 5n^2, \forall\ n \geq 1$

$\leq 18\ n^2, \forall\ n \geq 1$

If we let c = 18 and $n_0 = 1$, we have $f(n) \leq c\ n^2, \forall\ n \geq n_0$.

Therefore by definition, $f(n) = O(n^2)$.

Big-Omega

- Definition:

  $\Omega(g(n)) = \{f(n): \exists$ positive constants c and n_0 such that $0 \leq cg(n) \leq f(n)\ \forall\ n \geq n_0\}$

- $\lim_{n\to\infty} f(n)/g(n) > 0$ (if the limit exists.)
- Abuse of notation (for convenience):

  $f(n) = \Omega(g(n))$ actually means $f(n) \in \Omega(g(n))$

Big-Omega

- Claim: $f(n) = n^2 / 10 = \Omega(n)$

- Proof by definition:

  $f(n) = n^2 / 10$, $g(n) = n$

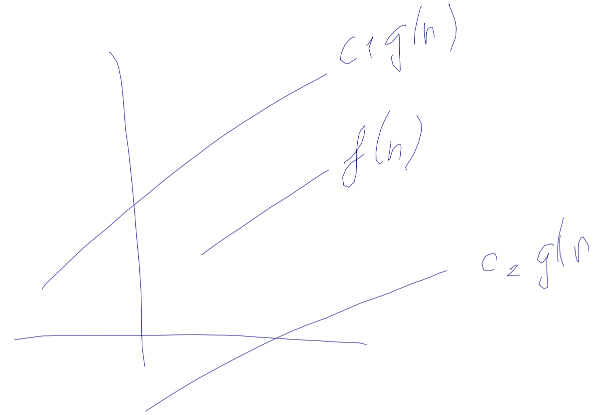
  Need to find a c and a n_o to satisfy the definition of $f(n) \in \Omega(g(n))$, i.e., $f(n) \geq cg(n)$ for $n \geq n_0$

  Proof:

  $n \leq n^2 / 10$ when $n \geq 10$

  If we let $c = 1$ and $n_0 = 10$, we have $f(n) \geq cn$, $\forall\ n \geq n_0$. Therefore, by definition, $n^2 / 10 = \Omega(n)$.

Theta



- Definition:
  - $\Theta(g(n)) = \{f(n)$: $\exists$ positive constants c_1, c_2, and n_0 such that $0 \le c_1\, g(n) \le f(n) \le c_2\, g(n), \quad \forall\ n \ge n_0\}$
- $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$
- Abuse of notation (for convenience):

  $f(n) = \Theta(g(n))$ actually means $f(n) \in \Theta(g(n))$

  $\Theta(1)$ means constant time.

Theta

- Claim: $f(n) = 2n^2 + n = \Theta(n^2)$
- Proof by definition:
  - Need to find the three constants c_1, c_2, and n_0 such that $c_1n^2 \le 2n^2+n \le c_2n^2$ for all $n \ge n_0$
  - A simple solution is $c_1 = 2$, $c_2 = 3$, and $n_0 = 1$

More Examples

- Prove $n^2 + 3n + \lg n$ is in $O(n^2)$
- Need to find c and n_0 such that

  $$n^2 + 3n + \lg n <= cn^2 \text{ for } n \geq n_0$$

- Proof:

  $n^2 + 3n + \lg n <= n^2 + 3n^2 + n$ for $n \geq 1$

  $<= n^2 + 3n^2 + n^2$ for $n \geq 1$

  $<= 5n^2$ for $n \geq 1$

  Therefore by definition $n^2 + 3n + \lg n \in O(n^2)$.

  (Alternatively: $n^2 + 3n + \lg n <= n^2 + n^2 + n^2$ for $n \geq 10$

  $<= 3n^2$ for $n \geq 10$)

More Examples

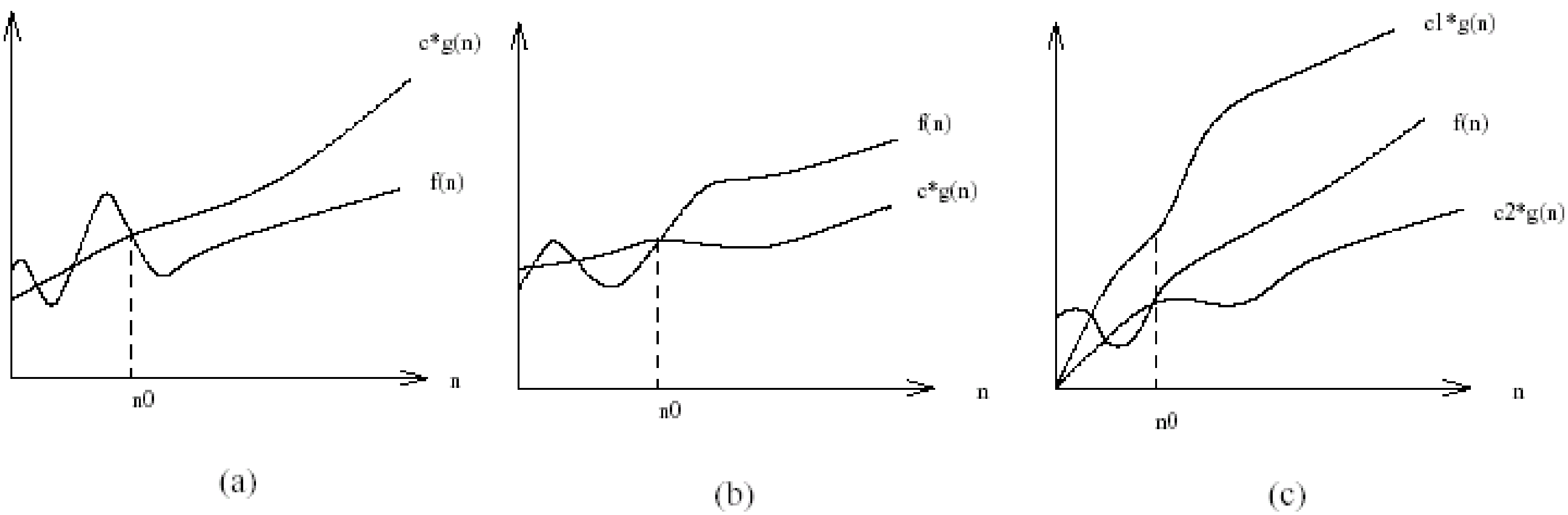
- Prove $n^2 + 3n + \lg n$ is in $\Omega(n^2)$
- Want to find c and n_0 such that

$$n^2 + 3n + \lg n >= cn^2 \text{ for } n \geq n_0$$

$$n^2 + 3n + \lg n >= n^2 \quad \text{for } n \geq 1$$

$n^2 + 3n + \lg n = O(n^2)$ and $n^2 + 3n + \lg n = \Omega\,(n^2)$
=> $n^2 + 3n + \lg n = \Theta(n^2)$

O, Ω, and Θ



The definitions imply a constant n_0 *beyond which* they are satisfied. We do not care about small values of n.

Using limits to compare orders of growth

$f(n) < g(n)$

$f(n) = c\,g(n)$

- $$\lim_{n\to\infty} f(n) / g(n) = \begin{cases} 0 & \rightarrow \quad f(n) \in o(g(n)) \\ c > 0 & \rightarrow \quad f(n) \in \Theta\,(g(n)) \\ \infty & \rightarrow \quad f(n) \in \omega\,(g(n)) \end{cases}$$

$f(n) > g(n)$

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

$f(n) \in \Theta\,(g(n)) \rightarrow f(n) \in O(g(n))$

$f(n) \in \Theta\,(g(n)) \rightarrow f(n) \in \Omega(g(n))$

$f(n) \in \omega\,(g(n)) \rightarrow f(n) \in \Omega(g(n))$

logarithms

- compare $\log_2 n$ and $\log_{10} n$

- $\log_a b = \log_c b / \log_c a$
- $\log_2 n = \log_{10} n / \log_{10} 2 \sim 3.3 \log_{10} n$
- Therefore $\lim(\log_2 n / \log_{10} n) = 3.3$
- $\log_2 n = \Theta(\log_{10} n)$

More examples

- Compare 2^n and 3^n
- $\lim_{n\to\infty} 2^n / 3^n = \lim_{n\to\infty}(2/3)^n = 0$
- Therefore, $2^n \in o(3^n)$, and $3^n \in \omega(2^n)$
- How about 2^n and 2^{n+1}?

  $2^n / 2^{n+1} = ½$, therefore $2^n = \Theta(2^{n+1})$

L’ Hopital’s rule

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

Condition:

If both lim f(n) and lim g(n) are ∞ or 0

- You can apply this transformation as many times as you want, as long as the condition holds

More examples

- Compare $n^{0.5}$ and log n
- $\lim_{n \to \infty} n^{0.5} / \log n = ?$
- $(n^{0.5})' = 0.5\ n^{-0.5}$
- $(\log n)' = 1 / n$
- $\lim (n^{-0.5} / 1/n) = \lim(n^{0.5}) = \infty$
- Therefore, $\log n \in o(n^{0.5})$
- In fact, $\log n \in o(n^{\varepsilon})$, for any $\varepsilon > 0$

Stirling's formula

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n = \sqrt{2\pi}n^{n+1/2}e^{-n}$$

$$n! \approx \text{(constant)}\ n^{n+1/2}e^{-n}$$

Stirling's formula

- Compare 2^n and $n!$

$$\lim_{n\to\infty}\frac{n!}{2^n}=\lim_{n\to\infty}\frac{c\sqrt{n}n^n}{2^n e^n}=\lim_{n\to\infty}c\sqrt{n}\left(\frac{n}{2e}\right)^n=\infty$$

- Therefore, $2^n = o(n!)$

- Compare n^n and $n!$

$$\lim_{n\to\infty}\frac{n!}{n^n}=\lim_{n\to\infty}\frac{c\sqrt{n}n^n}{n^n e^n}=\lim_{n\to\infty}\frac{c\sqrt{n}}{e^n}=0$$

- Therefore, $n^n = \omega(n!)$

- How about log (n!)?

Answer

$$\log(n!) = \log\frac{c\sqrt{n}n^n}{e^n} = C + \log n^{n+1/2} - \log(e^n)$$

$$= C + n\log n + \frac{1}{2}\log n - n$$

$$= C + \frac{n}{2}\log n + (\frac{n}{2}\log n - n) + \frac{1}{2}\log n$$

$$= \Theta(n\log n)$$

More advanced dominance ranking

$$n! \gg c^n \gg n^3 \gg n^2 \gg n^{1+\epsilon} \gg n \log n \gg n \gg \sqrt{n} \gg$$
$$\log^2 n \gg \log n \gg \log n / \log \log n \gg \log \log n \gg \alpha(n) \gg 1$$

Asymptotic notations

- O: Big-Oh
- Ω: Big-Omega
- Θ: Theta
- o: Small-oh
- ω: Small-omega
- Intuitively:

O is like ≤   Ω is like ≥

o is like <   ω is like >   Θ is like =

f(n) vs g(n)

Summary

- $f(n) = O(g(n)) \Rightarrow f(n) <= g(n)$
- $f(n) = o(g(n)) \Rightarrow f(n) < g(n)$
- $f(n) = \Omega(g(n)) \Rightarrow f(n) >= g(n)$
- $f(n) = \omega(g(n)) \Rightarrow f(n) > g(n)$
- $f(n) = \Theta(g(n)) \Rightarrow f(n) = g(n)$

Properties of asymptotic notations

- Textbook page 51
- Transitivity

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

  $\Rightarrow f(n) = \Theta(h(n))$

  (holds true for o, O, ω, and Ω as well).
- Symmetry

  $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$
- Transpose symmetry

  $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$

  $f(n) = o(g(n))$ if and only if $g(n) = \omega(f(n))$

About exponential and logarithm functions

- Textbook page 55-56
- It is important to understand what logarithms are and where they come from.
- A logarithm is simply an inverse exponential function.
- Saying $b^x = y$ is equivalent to saying that
  $x = \log_b y$.
- Logarithms reflect how many times we can double something until we get to n, or halve something until we get to 1.
- $\log_2 1 = ?$
- $\log_2 2 = ?$

Binary Search

- In binary search we throw away half the possible number of keys after each comparison.
- How many times can we halve n before getting to 1?
- Answer: ceiling (lg n)

Logarithms and Trees

- How tall a binary tree do we need until we have n leaves?
- The number of potential leaves doubles with each level.
- How many times can we double 1 until we get to n?
- Answer: ceiling (lg n)

Logarithms and Bits

- How many numbers can you represent with k bits?
- Each bit you add doubles the possible number of bit patterns
- You can represent from 0 to $2^k - 1$ with k bits. A total of 2^k numbers.
- How many bits do you need to represent the numbers from 0 to n?
- ceiling (lg (n+1))

logarithms

- $\lg n = \log_2 n$
- $\ln n = \log_e n$, $e \approx 2.718$
- $\lg^k n = (\lg n)^k$
- $\lg \lg n = \lg (\lg n) = \lg^{(2)} n$
- $\lg^{(k)} n = \lg \lg \lg \ldots \lg n$
- $\lg^2 4 = ?$
- $\lg^{(2)} 4 = ?$
- Compare $\lg^k n$ vs $\lg^{(k)} n$?

Functional iteration

- $f^{(i)}(n)$ denotes the function f (n) iteratively applied *i* times to an initial value of n
- Formally

$$f^{(i)}(n) = \begin{cases} n, & \text{if } i = 0 \\ f(f^{(i-1)}(n)) & \text{if } i > 0 \end{cases}$$

- f (n) = 2n, then $f^{(i)}(n) = 2^i n$

The iterated logarithm function

- lg* n (read "log star of n") denotes the iterated logarithm
- $\lg^* n = \min \{i \geq 0 : \lg^{(i)}(n) \leq 1\}$
- Examples
  - $\lg^* 2 = 1$
  - $\lg^* 4 = 2$
  - $\lg^* 16 = 3$
  - $\lg^* 2^{16} = 4$
  - $\lg^* 2^{65536} = 5$

Useful rules for logarithms

For all a > 0, b > 0, c > 0, the following rules hold

- $\log_b a = \log_c a / \log_c b = \lg a / \lg b$
- $\log_b a^n = n \log_b a$
- $b^{\log_b a} = a$
- $\log (ab) = \log a + \log b$
    - $\lg (2n) = ?$
- $\log (a/b) = \log (a) – \log(b)$
    - $\lg (n/2) = ?$
    - $\lg (1/n) = ?$
- $\log_b a = 1 / \log_a b$

Useful rules for exponentials

- For all $a > 0$, $b > 0$, $c > 0$, the following rules hold
- $a^0 = 1$ ($0^0 = ?$)
- $a^1 = a$
- $a^{-1} = 1/a$
- $(a^m)^n = a^{mn}$
- $(a^m)^n = (a^n)^m$
- $a^m a^n = a^{m+n}$

More advanced dominance ranking

$$n^n >> n! >> 3^n >> 2^n >> n^3 >> n^2 >> n^{1+\varepsilon} >> n\log n \sim \log n!$$
$$>> n >> n/\log n >> \sqrt{n} >> n^{\varepsilon} >> \log^3 n >> \log^2 n >> \log n$$
$$>> \log n/\log\log n >> \log\log n >> \log^{(3)} n >> \alpha(n) >> 1$$

Analyzing the complexity of an algorithm

Kinds of analyses

- Worst case
  - Provides an upper bound on running time
- Best case – not very useful, can always cheat
- Average case
  - Provides the expected running time
  - Very useful, but treat with care: what is “average”?

Plan for analyzing time efficiency of non-recursive algorithms

- Decide parameter (input size)
- Identify most executed line (basic operation)
- worst-case = average-case?
- $T(n) = \sum_i t_i$
- $T(n) = \Theta\ (f(n))$

Example

```
repeatedElement (A, n)
  // determines whether all elements in a given
  // array are distinct
  for i = 1 to n-1     {
        for j = i+1 to n {
                if (A[i] == A[j])
                        return true;
        }
  }
  return false;
```

Example

```
repeatedElement (A, n)
  // determines whether all elements in a given
  // array are distinct
  for i = 1 to n-1     {
        for j = i+1 to n {
                if (A[i] == A[j])
                        return true;
        }
  }
  return false;
```

Answers

- Best case
  - A[1] = A[2]
  - $T(n) = \Theta(1)$
- Worst-case
  - No repeated elements
  - $T(n) = (n-1) + (n-2) + \ldots + 1 = n(n-1) / 2 = \Theta(n^2)$
- Average case?
  - What do you mean by "average"?
  - Need more assumptions about data distribution.
    - How many possible repeats are in the data?
  - Average-case analysis often involves probability.

Find the order of growth for sums

- $T(n) = \sum_{i=1..n} i = \Theta(n^2)$
- $T(n) = \sum_{i=1..n} \log(i) = ?$
- $T(n) = \sum_{i=1..n} n / 2^i = ?$
- $T(n) = \sum_{i=1..n} 2^i = ?$
- …

- How to find out the actual order of growth?
  - Math…
  - Textbook Appendix A.1 (page 1058-60)

Arithmetic series

- An arithmetic series is a sequence of numbers such that the difference of any two successive members of the sequence is a constant.

  e.g.: 1, 2, 3, 4, 5

  or 10, 12, 14, 16, 18, 20

- In general:

$$a_i = a_{i-1} + d$$ ← Recursive definition

Or: $$a_i = a_1 + (i-1)d$$ ← Closed form, or explicit formula

Sum of arithmetic series

If a_1, a_2, …, a_n is an arithmetic series, then

$$\sum_{i=1}^{n} a_i = \frac{n(a_1 + a_n)}{2}$$

e.g. 1 + 3 + 5 + 7 + … + 99 = ?

(series definition: a_i = 2i-1)
This is $\sum_{i = 1 \text{ to } 50} (a_i)$ = 50 * (1 + 99) / 2 = 2500

Geometric series

- A geometric series is a sequence of numbers such that the ratio between any two successive members of the sequence is a constant.

  e.g.: 1, 2, 4, 8, 16, 32

  or 10, 20, 40, 80, 160

  or 1, ½, ¼, 1/8, 1/16

- In general:

$$a_i = ra_{i-1} \longleftarrow \text{Recursive definition}$$

Or: $$a_i = r^i a_0 \longleftarrow \text{Closed form, or explicit formula}$$

Sum of geometric series

$$\sum_{i=0}^{n} r^i = \begin{cases} (1-r^{n+1})/(1-r) & \text{if r < 1} \\ (r^{n+1}-1)/(r-1) & \text{if r > 1} \\ n+1 & \text{if r = 1} \end{cases}$$

$$\sum_{i=0}^{n} 2^i = ?$$

$$\lim_{n\to\infty} \sum_{i=0}^{n} \frac{1}{2^i} = ?$$

$$\lim_{n\to\infty} \sum_{i=1}^{n} \frac{1}{2^i} = ?$$

Sum of geometric series

$$\sum_{i=0}^{n} r^i = \begin{cases} (1-r^{n+1})/(1-r) & \text{if r < 1} \\ (r^{n+1}-1)/(r-1) & \text{if r > 1} \\ n+1 & \text{if r = 1} \end{cases}$$

$$\sum_{i=0}^{n} 2^i = \frac{2^{n+1}-1}{2-1} = 2^{n+1}-1 \approx 2^{n+1} \in \Theta(2^n)$$

$$\lim_{n\to\infty} \sum_{i=0}^{n} \frac{1}{2^i} = \lim_{n\to\infty} \sum_{i=0}^{n} (\tfrac{1}{2})^i = \frac{1}{1-\tfrac{1}{2}} = 2$$

$$\lim_{n\to\infty} \sum_{i=1}^{n} \frac{1}{2^i} = \lim_{n\to\infty} \sum_{i=0}^{n} \left(\tfrac{1}{2}\right)^i - \left(\tfrac{1}{2}\right)^0 = 2-1 = 1$$

Important formulas

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

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

$$\sum_{i=0}^{n} r^i = \frac{r^{n+1}-1}{r-1} \in \begin{cases} \Theta(1) & (r<1) \\ \Theta(r^n) & (r>1) \end{cases}$$

$$\sum_{i=1}^{n} i^2 \approx \frac{n^3}{3} \in \Theta(n^3)$$

$$\sum_{i=1}^{n} i^k \approx \frac{n^{k+1}}{k+1} \in \Theta(n^{k+1})$$

$$\sum_{i=1}^{n} i2^i = (n-1)2^{n+1} + 2 \in \Theta(n2^n)$$

$$\sum_{i=1}^{n} \frac{1}{i} \in \Theta(\lg n)$$

$$\sum_{i=1}^{n} \lg i \in \Theta(n \lg n)$$

Sum manipulation rules

$$\sum_i (a_i + b_i) = \sum_i a_i + \sum_i b_i$$

$$\sum_i ca_i = c\sum_i a_i$$

$$\sum_{i=m}^{n} a_i = \sum_{i=m}^{x} a_i + \sum_{i=x+1}^{n} a_i$$

Example:

$$\sum_{i=1}^{n} (4i + 2^i) = ?$$

$$\sum_{i=1}^{n} \frac{n}{2^i} = ?$$

Sum manipulation rules

$$\sum_i (a_i + b_i) = \sum_i a_i + \sum_i b_i$$

$$\sum_i c a_i = c \sum_i a_i$$

$$\sum_{i=m}^{n} a_i = \sum_{i=m}^{x} a_i + \sum_{i=x+1}^{n} a_i$$

Example:

$$\sum_{i=1}^{n} (4i + 2^i) = 4\sum_{i=1}^{n} i + \sum_{i=1}^{n} 2^i = 2n(n+1) + 2^{n+1} - 2$$

$$\sum_{i=1}^{n} \frac{n}{2^i} = n\sum_{i=1}^{n} \frac{1}{2^i} \approx n$$

More series

- $\sum_{i=1..n} n / 2^i = n * \sum_{i=1..n} (½)^i = ?$

- using the formula for geometric series:
  $\sum_{i=0..n} (½)^i = 1 + ½ + ¼ + ... (½)^n = 2$

- Application: algorithm for allocating dynamic memories

More series

- $\sum_{i=1..n} \log(i) = \log 1 + \log 2 + \ldots + \log n$
  $= \log 1 \times 2 \times 3 \times \ldots \times n$
  $= \log n!$
  $= \Theta(n \log n)$

- Application: algorithm for selection sort using priority queue

Recursive definition of sum of series

- $T(n) = \sum_{i=0..n} i$ is equivalent to:

$$\begin{cases} T(n) = T(n-1) + n & \longleftarrow \text{Recurrence} \\ T(0) = 0 & \longleftarrow \text{Boundary condition} \end{cases}$$

- $T(n) = \sum_{i=0..n} a^i$ is equivalent to:

$$\begin{cases} T(n) = T(n-1) + a^n \\ T(0) = 1 \end{cases}$$

Recursive definition is often intuitive and easy to obtain. It is very useful in analyzing recursive algorithms, and some non-recursive algorithms too.

Recursive definition of sum of series

- How to solve such recurrence or more generally, recurrence in the form of:
- $T(n) = aT(n-b) + f(n)$ or
- $T(n) = aT(n/b) + f(n)$