CS 560: Design and Analysis of Algorithms

Chapter 3: Asymptotic Notations

Some slides courtesy from Jeff Edmonds and Jianhua Ruan

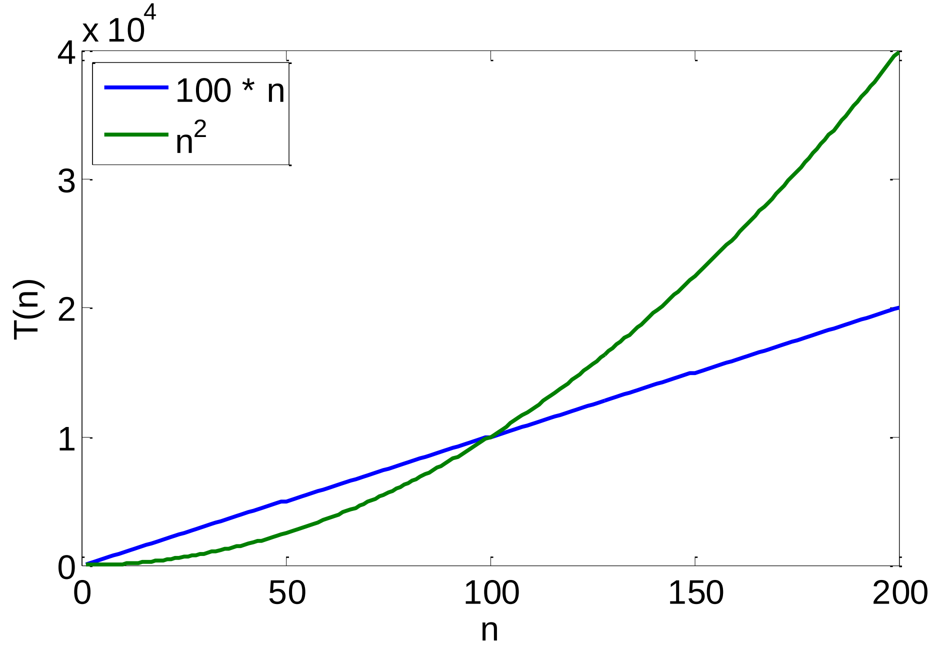
Asymptotic Analysis

- 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 $n \rightarrow \infty$.

    "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

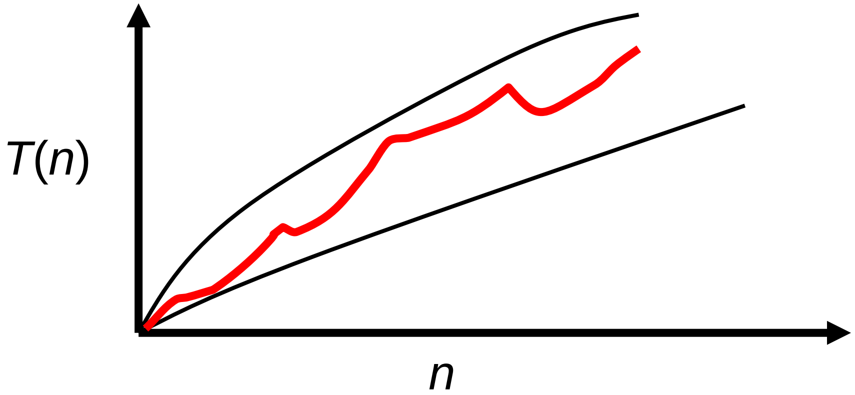
Order of growth

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

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

Exact analysis is hard!

- We have agreed that the best, worst and average case complexity of an algorithm is a numerical function of the size of instances



- However, worst-case and average-case are difficult to deal with precisely, because the details are very complicated
- Thus, it is usually cleaner and easier to talk about upper and lower bounds of the function
- There is where the big O notation comes in!
- Since running our algorithm on a machine which is twice as fast will effect the running time by a multiplicative constant of 2, we are going to have to ignore the constants anyway

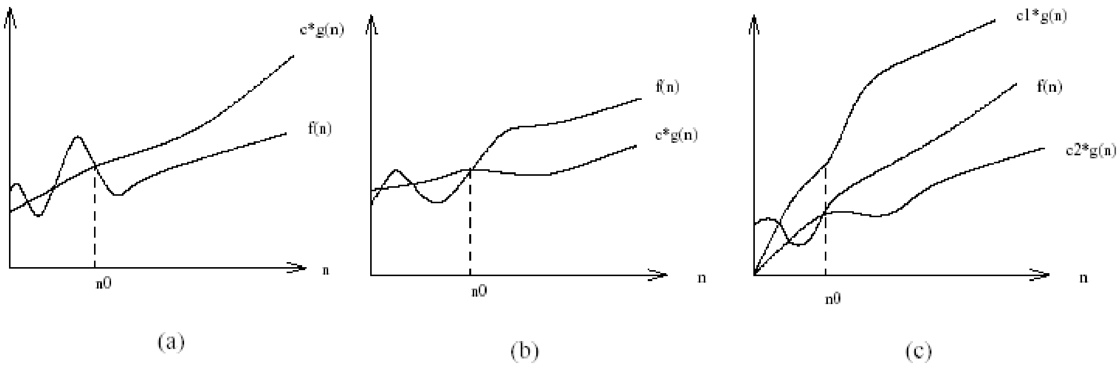
Asymptotic notations

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

Names of bounding functions

- Now that we have clearly defined the complexity functions we are talking about, we can talk about upper and lower bounds on it
  - $f(n)=O(g(n))$ means c x g(n) is an upper bound on f(n)
  - $f(n)= \Omega(g(n))$ means c x g(n) is a lower bound on f(n)
  - $f(n)= \Theta(g(n))$ means c_1 x g(n) is an upper bound on f(n) and c_2 x g(n) is a lower bound on f(n)
- Got it? c, c_1, c_2 are constants independent of n
- All of these definitions imply a constant n_0 beyond which they are satisfied
- We don’t care about the small values of n

O, Ω, and Θ



- The value of n_0 shown is the minimum possible value; any greater value will also work
  - (a) f(n)=O(g(n)) if there exists a positive constant n_0, and c* such that for all n greater than n_0 (to the right of n_0), the value of f(n) always lies on or below c* x g(n). That is, f(n) <= c* x g(n)
  - (b) f(n)= Ω(g(n)) if there exists a positive constant n_0, and c such that for all n greater than n_0 (to the right of n_0), the value of f(n) always lies on or above c* x g(n). That is, c* x g(n)<= f(n)
  - (c) f(n)= Θ(g(n)) if there exists a positive constant n_0, c^*_1, and c^*_2 such that for all n greater than n_0 (to the right of n_0), the value of f(n) always lies between c^*_1 x g(n) and c^*_2 x g(n) inclusive. That is, c^*_2 x g(n) <= f(n) <= c^*_1 x g(n)
  - Finally, f(n)= Θ(g(n)) if and only if f(n)=O(g(n)) and f(n)= Ω(g(n))

What does all these mean?

- $3n^2-100n+6 = O(n^2)$ because $3.1n^2 > 3n^2-100n+6$
- $3n^2-100n+6 = O(n^3)$ because $3n^3 > 3n^2-100n+6$
- $3n^2-100n+6 \neq O(n)$ because for any c, $cn<3n^2$ when $n>c$

- $3n^2-100n+6 = \Omega(n^2)$ because $2n^2 < 3n^2-100n+6$
- $3n^2-100n+6 \neq \Omega(n^3)$ because $3n^2-100n+6<n^3$
- $3n^2-100n+6 = \Omega(n)$ because $1000n < 3n^2-100n+6$

- $3n^2-100n+6 = \Theta(n^2)$ because O and Ω
- $3n^2-100n+6 \neq \Theta(n^3)$ because O only
- $3n^2-100n+6 \neq \Theta(n)$ because Ω only

Testing dominance

- f(n) dominates g(n) if lim $_{n\to\infty}$ g(n)/f(n) = 0, which is same as saying g(n)=o(f(n))
  - Note the **little-oh-**-- it means "grows strictly slower than"
- Knowing the dominance relation between common functions is important because we want algorithms whose time complexity is as low as possible in the hierarchy
  - If f(n) dominates g(n), f is much larger than g
  - That is f is much slower (takes more time to run) than g
- Examples
  - n^a dominates n^b if a>b
    - since $\lim_{n\to\infty} n^b/n^a=1/n^{b-a}=0$
  - $n^a+o(n^a)$ does not dominate n^a
    - Since $\lim_{n\to\infty} n^a/(n^a+o(n^a))=1$

Complexity	10	20	30	40
n	0.00001 sec	0.00002 sec	0.00003 sec	0.00004 sec
n^2	0.0001 sec	0.0004 sec	0.0009 sec	0.016 sec
n^3	0.001 sec	0.008 sec	0.027 sec	0.064 sec
n^5	0.1 sec	3.2 sec	24.3 sec	1.7 min
2^n	0.001 sec	1.0 sec	17.9 min	12.7 days
3^n	0.59 sec	58 min	6.5 years	3855 cent

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 $f(n) \leq cg(n)$ for some constant c

- What is $O(n^2)$?
  - The set of all functions that grow slower than or in the same order as n^2

Examples

So:

$n \in O(n^2)$

This says $O(n^2)$ is a set of functions and $f(n)=n$ is one of them

We will often abuse this notation and simply say $n=O(n^2)$ as we have done before

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

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

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

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

Intuitively, O is like ≤

But:

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

Why?

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 $g(n) \leq f(n)$

Intuitively, Ω is like $\geq$

So:

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

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

But:

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

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

- $f(n) \in \Theta(g(n))$ means g(n) is an asymptotically tight bound of f(n)
  - Intuitively, it is like f(n) = g(n)

- What is $\Theta(n^2)$?
  - 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: O

- Definition:

  There exist

  For all

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

- $\lim_{n\to\infty} g(n)/f(n) > 0$ (if the limit exists.)

- Abuse of notation (for convenience):

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

Big-Oh: O

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

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 $n > n_0$.

(Note: you just need to find one concrete example of c and n0 satisfying the condition.)

$$
\begin{aligned}
3n^2 + 10n + 5 &\le 10n^2 + 10n + 10 \\
&\le 10n^2 + 10n^2 + 10n^2, \forall\, n \ge 1 \\
&\le 30\, n^2, \forall\, n \ge 1
\end{aligned}
$$

Therefore, if we let $c = 30$ and $n_0 = 1$, we have $f(n) \le c\, n^2, \forall\, n \ge n_0$.

Hence according to the definition of big-Oh, $f(n) = O(n^2)$.

- Alternatively, we can show that

$$\lim_{n\to\infty} n^2 / (3n^2 + 10n + 5) = 1/3 > 0$$

Big-Omega: Ω

- Definition:

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

- $\lim_{n\to\infty} f(n)/g(n) > 0$ (if the limit exists.)

- Abuse of notation (for convenience):

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

Big-Omega: Ω

- 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 $n > n_0$

  $n \leq n^2 / 10$ 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, according to definition, $f(n) = \Omega(n)$.

- Alternatively:

  $\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty} (n/10) = \infty$

Theta: Θ

- Definition:
  - $\Theta(g(n)) = \{f(n)$: $\exists$ positive constants c_1, c_2, and n_0 such that $0 \leq c_1\, g(n) \leq f(n) \leq c_2\, g(n), \quad \forall\, n \geq n_0\}$

- $\lim_{n\to\infty} f(n)/g(n) = c > 0$ and $c < \infty$

- $f(n) = O(g(n))$ 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: Θ

- 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

    $c_1 n^2 \leq 2n^2 + n \leq c_2 n^2$ for all $n > n_0$
  - A simple solution is $c_1 = 2$, $c_2 = 3$, and $n_0 = 1$

- Alternatively, $\lim_{n \to \infty} (2n^2+n)/n^2 = 2$

More Examples

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

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

- Proof:

$$n^2 + 3n + \lg n <= 3n^2 + 3n + 3\lg n \qquad \text{for } n > 1$$

$$<= 3n^2 + 3n^2 + 3n^2$$

$$<= 9n^2$$

Or $n^2 + 3n + \lg n <= n^2 + n^2 + n^2 \qquad$ for $n > 10$

$$<= 3n^2$$

More Examples

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

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

$$n^2 + 3n + \lg n >= n^2 \text{ for } n > 0$$

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

$\Rightarrow n^2 + 3n + \lg n = \Theta(n^2)$

More Examples

- Prove that for any constant a and b,
  - $(n+a)^b = \Theta(n^b)$

Class exercise

Small-o

There exist

- Definition:

  $o(g(n)) = \{f(n)$: for any positive constant c, $\exists$ $n_0 > 0$ such that $0 \le f(n) \le cg(n)\ \forall\ n > n_0\}$

  For all

- $\lim_{n \to \infty} f(n)/g(n) = 0$ (if the limit exists.)

- Abuse of notation (for convenience):

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

Small-o

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

  $\lim_{n\to\infty} (10n+5) / (n^2) = 0$

- Claim: $f(n)=n^2+10n+5 \notin o(n^2)$
- Proof by definition

  $\lim_{n\to\infty} (n^2+10n+5) / (n^2) = 1$

Small-ω

There exist

- Definition:

  $\omega(g(n)) = \{f(n)$: for any positive constant c, $\exists$ $n_0>0$ such that $0 \le cg(n)< f(n)\ \forall\ n>n_0\}$

For all

- $\lim_{n\to\infty} f(n)/g(n) = \infty$
- Abuse of notation (for convenience):

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

Small- ω

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

  $\lim_{n\to\infty} (10n^2+5) / (n) = \infty$

- Claim: $f(n)=10n+5 \notin \omega(n)$
- Proof by definition

  $\lim_{n\to\infty} (10n+5) / (n) = 10$

Using limits to compare orders of growth

- $\lim_{n\to\infty} f(n) / g(n) = \begin{cases} 0 \\ c \quad > 0 \\ \infty \end{cases}$

$$\begin{tikzcd}
 & & f(n) \in o(g(n)) \arrow[dr] & \\
0 \arrow[urr] & & & f(n) \in O(g(n)) \\
c > 0 \arrow[rr] & & f(n) \in \Theta(g(n)) \arrow[ur] \arrow[dr] & \\
 & & & f(n) \in \Omega(g(n)) \\
\infty \arrow[rr] & & f(n) \in \omega(g(n)) \arrow[ur] &
\end{tikzcd}$$

logarithms

- It is important to understand deep in your bones what logarithms are and where they come from
- A logarithm is simply an inverse exponential function. Saying $b^x=y$ is equivalent to saying $x=\log_b y$
- Exponential functions, like the amount owed on n year mortgage at an interest rate c% per year, are functions which grow distressingly fast, as anyone who has tried to pay off a mortgage knows
- Thus inverse exponential functions, i.e., logarithms grow refreshingly slow
- Binary search is an example of an O(log n) algorithm
  - After each comparison, we can throw away half of the possible number of keys
  - Thus 20 comparisons suffice to find any name in the million-name Manhattan phone book!
- If you have an algorithm which runs in O(log n) time, take it, because this is blindingly fast even on very large instances

Properties of logarithms

- Recall the definition, $c^{\log_c x} = x$ for any c
- Asymptotically, the base of the logarithm does not matter
  - $\log_b a = \log_c a / \log_c b$
  - Thus $\log_2 n = (1/\log_{100} 2) \log_{100} n$, and note that $1/\log_{100} 2 = 6.643$ is just a constant
- Asymptotically polynomial function of n does not matter
  - Note that $\log(n^{473}+n^2+n+96)=O(\log n)$
  - Since $n^{473}+n^2+n+96=O(n^{473})$ and $\log(n^{473})=473\log(n)$
- Any exponential dominates every polynomial
  - This is why we will seek to avoid exponential time algorithms

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 $\lim(\log_2 n / \log_{10} n) = 3.3$
- $\log_2 n = \Theta(\log_{10} n)$

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

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

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

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

Examples

- 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, $2^n = o(n!)$ and $n! = \omega(2^n)$

- 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, $n^n = \omega(n!)$ and $n! = o(n^n)$

- How about log (n!)?

Examples

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

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

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

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

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 $\leq$	Ω is like $\geq$	Θ is like =
o is like <	ω is like >	

True or false?

1. $2n^2 + 1 = O(n^2)$
2. $Sqrt(n) = O(\log n)$
3. $\log n = O(sqrt(n))$
4. $n^2(1 + sqrt(n)) = O(n^2 \log n)$
5. $3n^2 + sqrt(n) = O(n^2)$
6. $sqrt(n) \log n = O(n)$

Properties of asymptotic notations

- Please read from 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

- Please read from 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 $\lg^{(k)} n$?

Useful rules for logarithms

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

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

More advanced dominance ranking

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

Next topic

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"?

General plan for analyzing time efficiency of a non-recursive algorithm

- 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;
```

Time complexity

- Best case?
- Worst-case?
- Average case?

- 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.

Runtime analysis example

- Input size is n
- What are the running times of the following pseudocodes in Big-Oh notation?

```
 I. void sunny(int n, int x) {
        for (int k = 0; k < n; ++k)
           if (x < 50) {
              for (int i = 0; i < n; ++i)
                 for (int j = 0; j < i; ++j)
                    System.out.println("x = " + x);
           } else {
              System.out.println("x = " + x);
           }
    }

II. void warm(int n) {
        for (int i = 0; i < 2 * n; ++i) {
           j = 0;
           while (j < n) {
              System.out.println("j = " + j);
              j = j + 5;
           }
        }
    }
```

Runtime analysis example

- Input size is n
- What are the running times of the following pseudocodes in Big-Oh notation?

```
 I.  void sunny(int n, int x) {
          for (int k = 0; k < n; ++k)
             if (x < 50) {
                for (int i = 0; i < n; ++i)
                   for (int j = 0; j < i; ++j)
                      System.out.println("x = " + x);
             } else {
                System.out.println("x = " + x);
             }
     }
```

$O(n^3)$

```
II.  void warm(int n) {
          for (int i = 0; i < 2 * n; ++i) {
             j = 0;
             while (j < n) {
                System.out.println("j = " + j);
                j = j + 5;
             }
          }
     }
```

Runtime analysis example

- Input size is n
- What are the running times of the following pseudocodes in Big-Oh notation?

```
 I. void sunny(int n, int x) {
        for (int k = 0; k < n; ++k)
           if (x < 50) {
              for (int i = 0; i < n; ++i)
                 for (int j = 0; j < i; ++j)
                    System.out.println("x = " + x);
           } else {
              System.out.println("x = " + x);
           }
    }
```

$O(n^3)$

```
II. void warm(int n) {
        for (int i = 0; i < 2 * n; ++i) {
           j = 0;
           while (j < n) {
              System.out.println("j = " + j);
              j = j + 5;
           }
        }
    }
```

$O(n^2)$

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 $a_1, a_2, \ldots, a_n$ 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:

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

Or: $$a_i = r^i a_0$$ ← 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:

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

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

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

- using the formula for geometric series:

  $\sum_{i=0..n} (½)^i = 1 + ½ + ¼ + \ldots (½)^n = 2$

- Application: algorithm for allocating dynamic memories

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

- Application: algorithm for selection sort using priority queue