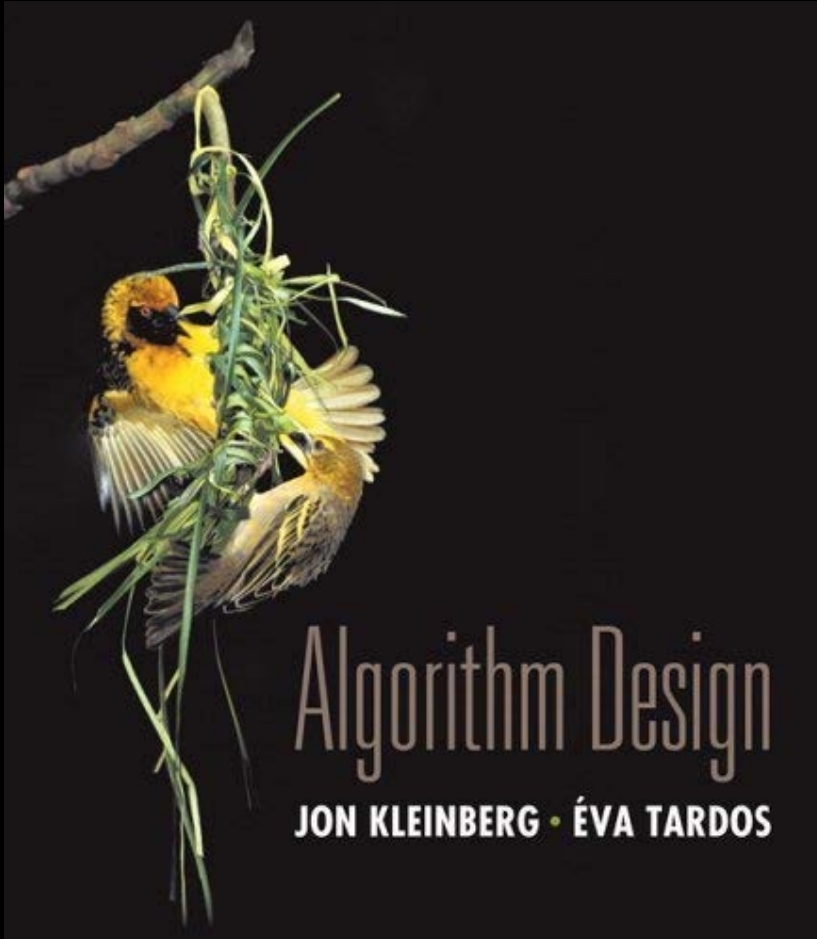


Chapter 2

Basics of Algorithm Analysis

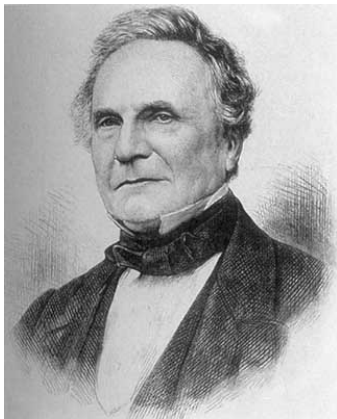


Slides by Kevin Wayne.
Copyright © 2005 Pearson-Addison Wesley.
All rights reserved.

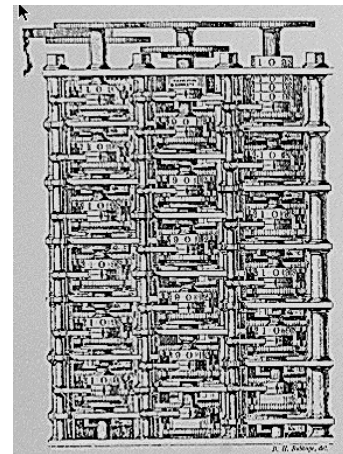
2.1 Computational Tractability

Computational Tractability

As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise - By what course of calculation can these results be arrived at by the machine in the shortest time? - *Charles Babbage*



Charles Babbage (1864)



Analytic Engine (schematic)

Polynomial-Time

Brute force. For many non-trivial problems, there is a natural brute force search algorithm that checks every possible solution.

- Typically takes 2^N time or worse for inputs of size N .
- Unacceptable in practice.

↖
 $n!$ for stable matching
with n men and n women

Desirable scaling property. When the input size doubles, the algorithm should only slow down by some constant factor C .

There exists constants $c > 0$ and $d > 0$ such that on every input of size N , its running time is bounded by $c N^d$ steps.

Def. An algorithm is **poly-time** if the above scaling property holds.

↖
choose $C = 2^d$

Worst-Case Analysis

Worst case running time. Obtain bound on **largest possible** running time of algorithm on input of a given size N .

- Generally captures efficiency in practice.
- Draconian view, but hard to find effective alternative.

Average case running time. Obtain bound on running time of algorithm on **random** input as a function of input size N .

- Hard (or impossible) to accurately model real instances by random distributions.
- Algorithm tuned for a certain distribution may perform poorly on other inputs.

Worst-Case Polynomial-Time

Def. An algorithm is **efficient** if its running time is polynomial.

Justification: **It really works in practice!**

- Although $6.02 \times 10^{23} \times N^{20}$ is technically poly-time, it would be useless in practice.
- In practice, the poly-time algorithms that people develop almost always have low constants and low exponents.
- Breaking through the exponential barrier of brute force typically exposes some crucial structure of the problem.

Exceptions.

- Some poly-time algorithms do have high constants and/or exponents, and are useless in practice.
- Some exponential-time (or worse) algorithms are widely used because the worst-case instances seem to be rare.

↖
simplex method
Unix grep

Why It Matters

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1,000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10,000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100,000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1,000,000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

2.2 Asymptotic Order of Growth

Permutations and combinations

- Factorial: “ n factorial”

$$n! = n \cdot (n - 1) \cdot \dots \cdot 2 \cdot 1$$

= number of permutations of $\{1, \dots, n\}$

- Combinations: “ n choose k ”

$$\binom{n}{k} = \frac{n \times (n - 1) \times \dots \times (n - k + 1)}{k \times (k - 1) \times \dots \times 2 \times 1} = \frac{n!}{k!(n - k)!}$$

= number of ways of choosing an unordered subset of k items in $\{1, \dots, n\}$ without repetition

Review Question

- In how many ways can we select two disjoint subsets of $\{1, \dots, n\}$, of size k and m , respectively?

- **Answer:**

$$\binom{n}{k} \binom{n-k}{m} = \binom{n}{m} \binom{n-m}{k} = \binom{n}{m+k} \binom{m+k}{k}$$

Asymptotic notation


O -notation (upper bounds):

$f(n) = O(g(n))$ means

there exist constants $c > 0$, $n_0 > 0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$.

EXAMPLE: $2n^2 = O(n^3)$ ($c = 1, n_0 = 2$)

*functions,
not values*



Asymptotic Notation

- **One-sided equality:** $T(n) = O(f(n))$.
 - Not transitive:
 - $f(n) = 5n^3$; $g(n) = 3n^2$
 - $f(n) = O(n^3) = g(n)$
 - but $f(n) \neq g(n)$.
 - Alternative notation: $T(n) \in O(f(n))$.

Set Definition

$O(g(n)) = \{ f(n) : \text{there exist constants } c > 0, n_0 > 0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \}$

EXAMPLE: $2n^2 \in O(n^3)$

Examples

- $10^6 n^3 + 2n^2 - n + 10 = O(n^3)$
- $n^{1/2} + \log n = O(n^{1/2})$
- $n (\log n + \sqrt{n}) = O(n^{3/2})$
- $n = O(n^2)$

Ω -notation (lower bounds)

O -notation is an *upper-bound* notation. It makes no sense to say $f(n)$ is at least $O(n^2)$.

$$\Omega(g(n)) = \{ f(n) : \text{there exist constants } c > 0, n_0 > 0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0 \}$$

EXAMPLE: $\sqrt{n} = \Omega(\log n) \quad (c = 1, n_0 = 16)$

Ω -notation (lower bounds)

- **Be careful:** “Any comparison-based sorting algorithm requires at least $O(n \log n)$ comparisons.”
 - Meaningless!
 - Use Ω for lower bounds.

Θ -notation (tight bounds)

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

EXAMPLE: $\frac{1}{2}n^2 - 2n = \Theta(n^2)$

Polynomials are simple:

$$a_d n^d + a_{d-1} n^{d-1} + \cdots + a_1 n + a_0 = \Theta(n^d)$$

O -notation and ω -notation

O -notation and Ω -notation are like \leq and \geq .
 o -notation and ω -notation are like $<$ and $>$.

$o(g(n)) = \{ f(n) : \text{for every constant } c > 0, \text{ there is a constant } n_0 > 0 \text{ such that } 0 \leq f(n) < cg(n) \text{ for all } n \geq n_0 \}$

EXAMPLE: $2n^2 = o(n^3) \quad (n_0 = 2/c)$

Overview of Asymptotic Notation

Notation	... means ...	Think...	E.g.	Lim $f(n)/g(n)$
$f(n)=O(n)$	$\exists c > 0, n_0 > 0$ $\forall n > n_0:$ $0 \leq f(n) < cg(n)$	Upper bound	$100n^2$ $= O(n^3)$	If it exists, it is $< \infty$
$f(n)=\Omega(g(n))$	$\exists c>0, n_0>0, \forall n > n_0 :$ $0 \leq cg(n) < f(n)$	Lower bound	2^n $= \Omega(n^{100})$	If it exists, it is > 0
$f(n)=\Theta(g(n))$	both of the above: $f=\Omega(g)$ and $f=O(g)$	Tight bound	$\log(n!)$ $= \Theta(n \log n)$	If it exists, it is > 0 and $< \infty$
$f(n)=o(g(n))$	$\forall c>0, \exists n_0>0, \forall n > n_0 :$ $0 \leq f(n) < cg(n)$	Strict upper bound	$n^2 = o(2^n)$	Limit exists, $=0$
$f(n)=\omega(g(n))$	$\forall c>0, \exists n_0>0, \forall n > n_0 :$ $0 \leq cg(n) < f(n)$	Strict lower bound	n^2 $= \omega(\log n)$	Limit exists, $=\infty$

Common Functions: Asymptotic Bounds

- **Polynomials.** $a_0 + a_1n + \dots + a_dn^d$ is $\Theta(n^d)$ if $a_d > 0$.
- **Polynomial time.** Running time is $O(n^d)$ for some constant d independent of the input size n .
- **Logarithms.** $\log_a n = \Theta(\log_b n)$ for all constants $a, b > 0$.

↑
can avoid specifying the base

log grows slower than every polynomial

↓
For every $x > 0$, $\log n = o(n^x)$.

Every polynomial grows slower than every exponential

- **Exponentials.** For all $r > 1$ and all $d > 0$, $n^d = o(r^n)$.
- **Factorial.** By Sterling's formula,

$$n! = (\sqrt{2\pi n}) \left(\frac{n}{e}\right)^n (1 + o(1)) = 2^{\Theta(n \log n)}$$

↖
grows faster than every exponential

Exercise: Show that $\log(n!) = \Theta(n \log n)$

- Upper bound:

$$\begin{aligned}\log(n!) &= \sum_{i=1}^n \log(i) \\ &\leq n \log(n)\end{aligned}$$

- Lower bound:

$$\begin{aligned}\log(n!) &= \sum_{i=1}^n \log(i) \\ &\geq \sum_{i=1}^n \log(i) \\ &\geq \frac{n}{2} \log\left(\frac{n}{2}\right) = \frac{n}{2} \log(n) - \frac{n}{2}\end{aligned}$$

Exercise: Show that $\log(n!) = \Theta(n \log n)$

- Stirling's formula:

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

$$\begin{aligned}\log(n!) &= \log(\sqrt{2\pi n}) + n \log(n) - n \log(e) + \underbrace{\log(1 + o(1))}_{\substack{=\log(1)+o(1) \\ \text{since log is continuous}}} \\ &= n \left(\underbrace{\log(n) - \log(e) + \frac{\log(2\pi n)}{n}}_{\text{constant} + o(1)} \right) + o(1) \\ &= n(\log(n) - O(1)) + o(1) \\ &= n \log n (1 - O(\frac{1}{\log n})) + o(1) \\ &= n \log n (1 \pm o(1)) = \Theta(n \log n)\end{aligned}$$

Sort by asymptotic order of growth

a) $n \log(n)$

b) \sqrt{n}

c) $\log(n)$

d) n^2

e) 2^n

f) n

g) $n!$

h) $n^{1,000,000}$

i) $n^{1/\log(n)}$

j) $\log(n!)$

k) $\binom{n}{2}$

l) $\binom{n}{n/2}$

Sort by asymptotic order of growth

a) $n \log(n)$

b) \sqrt{n}

c) $\log(n)$

d) n^2

e) 2^n

f) n

g) $n!$

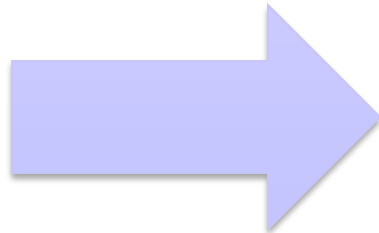
h) $n^{1,000,000}$

i) $n^{1/\log(n)}$

j) $\log(n!)$

k) $\binom{n}{2}$

l) $\binom{n}{n/2}$



1. $n^{1/\log n}$

2. $\log n$

3. \sqrt{n}

4. n

5. $n \log n = (\log(n!))$

6. $\binom{n}{2} = \Theta(n^2)$

7. $\binom{n}{2} = \Theta(n^2)$

8. $n^{1,000,000}$

9. $n^{1,000,000}$

10. $\binom{n}{n/2} = \Theta(2^n / \sqrt{n})$

11. 2^n

12. $n!$

Review question

- True or false?

1. $n^2 = O\left(\frac{n^2}{2}\right)$

2. $n^2 = \omega\left(\frac{n^2}{2}\right)$

3. $n^2 = \Omega\left(\frac{n^2}{2}\right)$

4. $n^2 = o(2^3 \log_2 n)$

Properties

- Transitivity.
 - If $f = O(g)$ and $g = O(h)$ then $f = O(h)$.
 - If $f = \Omega(g)$ and $g = \Omega(h)$ then $f = \Omega(h)$.
 - Similarly, for Θ -, o - and ω -notation.
- Additivity.
 - If $f = O(h)$ and $g = O(h)$ then $f + g = O(h)$.
 - If $f = \Omega(h)$ and $g = \Omega(h)$ then $f + g = \Omega(h)$.
 - Similarly, for Θ -, o - and ω -notation.

Question

- Let f, g be nonnegative functions.
- Consider the statement:
“either $f(n) = O(g(n))$ or $g(n) = O(f(n))$
(or both)”

Is this statement:

1. True for all functions f and g ?
2. True for some, but not all, functions f and g ?
3. False for all functions f and g ?

Conventions for formulas

Convention: A set in a formula represents an anonymous function in the set.

EXAMPLE: $f(n) = n^3 + O(n^2)$

(right-hand side)

means

$$f(n) = n^3 + h(n)$$

for some $h(n) \in O(n^2)$.

Convention for formulas

Convention: A set in a formula represents an anonymous function in the set.

EXAMPLE: $n^2 + O(n) = O(n^2)$

(left-hand side)

means

for any $f(n) \in O(n)$:

$$n^2 + f(n) = h(n)$$

for some $h(n) \in O(n^2)$.

Review question

- True or false?

1. $2 \binom{n}{2} = n^2(1 + o(1))$

2. $\log_2(100 n^2) = \log_2(n) + O(1)$

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

2.4 A Survey of Common Running Times

Linear Time: $O(n)$

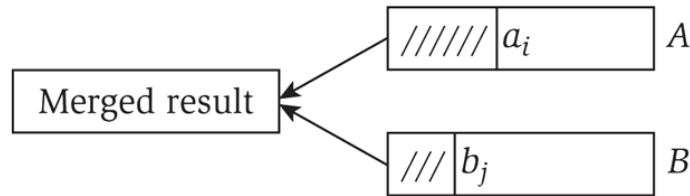
Linear time. Running time is at most a constant factor times the size of the input.

Computing the maximum. Compute maximum of n numbers a_1, \dots, a_n .

```
max ← a1
for i = 2 to n {
    if (ai > max)
        max ← ai
}
```


Linear Time: $O(n)$

Merge. Combine two sorted lists $A = a_1, a_2, \dots, a_n$ with $B = b_1, b_2, \dots, b_n$ into sorted whole.



```
i = 1, j = 1
while (both lists are nonempty) {
    if (ai ≤ bj) append ai to output list and increment i
    else          append bj to output list and increment j
}
append remainder of nonempty list to output list
```

Claim. Merging two lists of size n takes $O(n)$ time.

Pf. After each comparison, the length of output list increases by 1.

$O(n \log n)$ Time

$O(n \log n)$ time. Arises in divide-and-conquer algorithms.



also referred to as linearithmic time

Sorting. Mergesort and heapsort are sorting algorithms that perform $O(n \log n)$ comparisons.

Largest empty interval. Given n time-stamps x_1, \dots, x_n on which copies of a file arrive at a server, what is largest interval of time when no copies of the file arrive?

$O(n \log n)$ solution. Sort the time-stamps. Scan the sorted list in order, identifying the maximum gap between successive time-stamps.

Quadratic Time: $O(n^2)$

Quadratic time. Enumerate all pairs of elements.

Closest pair of points. Given a list of n points in the plane $(x_1, y_1), \dots, (x_n, y_n)$, find the pair that is closest.

$O(n^2)$ solution. Try all pairs of points.

```
min ←  $(x_1 - x_2)^2 + (y_1 - y_2)^2$ 
for i = 1 to n {
  for j = i+1 to n {
    d ←  $(x_i - x_j)^2 + (y_i - y_j)^2$ 
    if (d < min)
      min ← d
  }
}
```

← don't need to
take square roots

Remark. $\Omega(n^2)$ seems inevitable, but this is just an illusion. ← see chapter 5

Cubic Time: $O(n^3)$

Cubic time. Enumerate all triples of elements.

Set disjointness. Given n sets S_1, \dots, S_n each of which is a subset of $1, 2, \dots, n$, is there some pair of these which are disjoint?

$O(n^3)$ solution. For each pairs of sets, determine if they are disjoint.

```
foreach set  $S_i$  {  
    foreach other set  $S_j$  {  
        foreach element  $p$  of  $S_i$  {  
            determine whether  $p$  also belongs to  $S_j$   
        }  
        if (no element of  $S_i$  belongs to  $S_j$ )  
            report that  $S_i$  and  $S_j$  are disjoint  
    }  
}
```

Polynomial Time: $O(n^k)$ Time

Independent set of size k . Given a graph, are there k nodes such that no two are joined by an edge?

↖
 k is a constant

$O(n^k)$ solution. Enumerate all subsets of k nodes.

```
foreach subset S of k nodes {  
    check whether S is an independent set  
    if (S is an independent set)  
        report S is an independent set  
    }  
}
```

- Check whether S is an independent set = $O(k^2)$.
- Number of k element subsets = $\binom{n}{k} = \frac{n(n-1)(n-2)\cdots(n-k+1)}{k(k-1)(k-2)\cdots(2)(1)} \leq \frac{n^k}{k!}$
- $O(k^2 n^k / k!) = O(n^k)$.

↖
poly-time for $k=17$,
but not practical

Exponential Time

Independent set. Given a graph, what is maximum size of an independent set?

$O(n^2 2^n)$ solution. Enumerate all subsets.

```
s* ← ∅  
foreach subset S of nodes {  
    check whether S is an independent set  
    if (S is largest independent set seen so far)  
        update s* ← S  
}
```