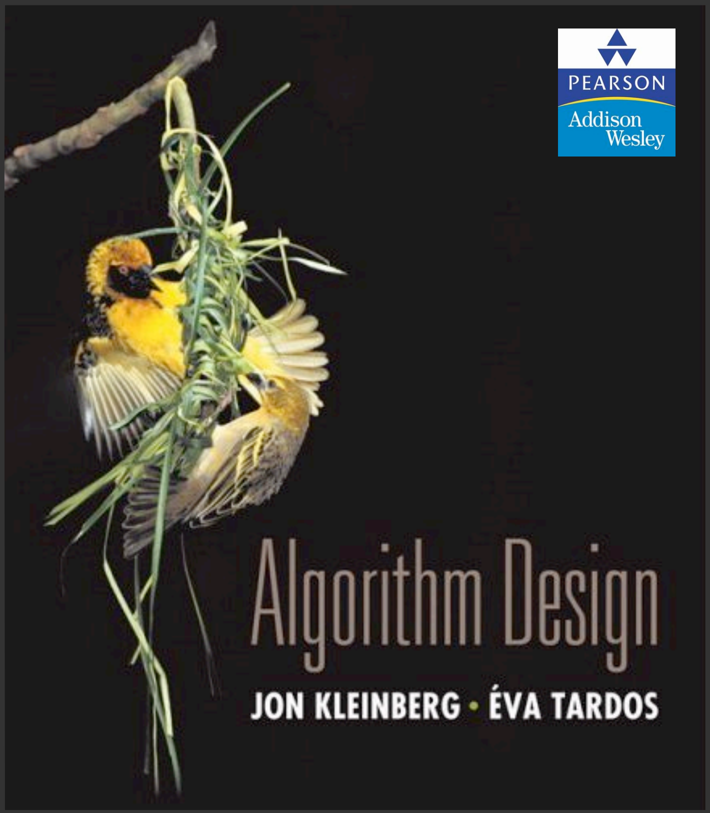


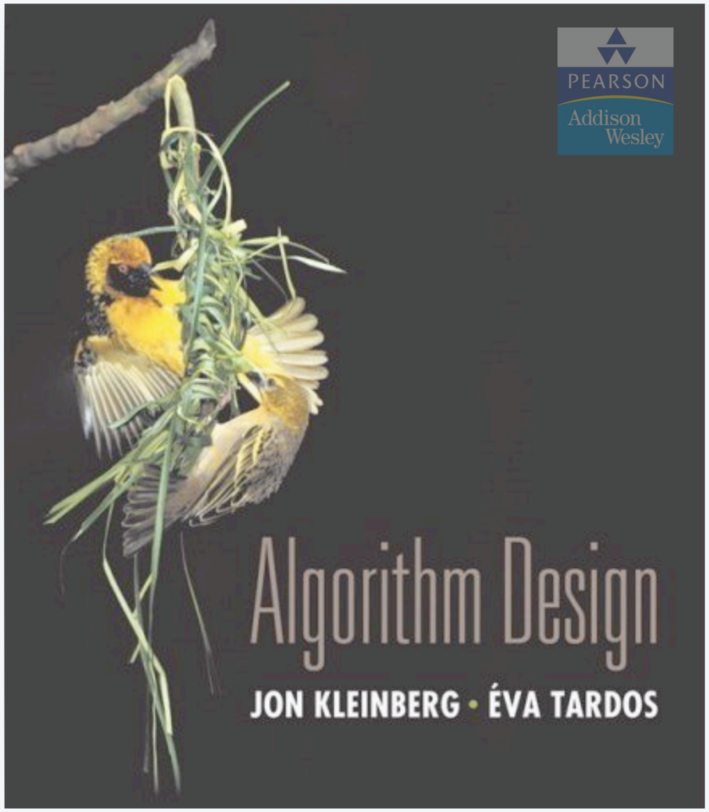

# 2. Algorithm Analysis

- *computational tractability*
- *asymptotic order of growth*
- *survey of common running times*

Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

Last updated on Sep 8, 2013 6:19 AM



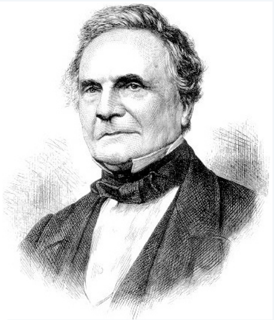

SECTION 2.1

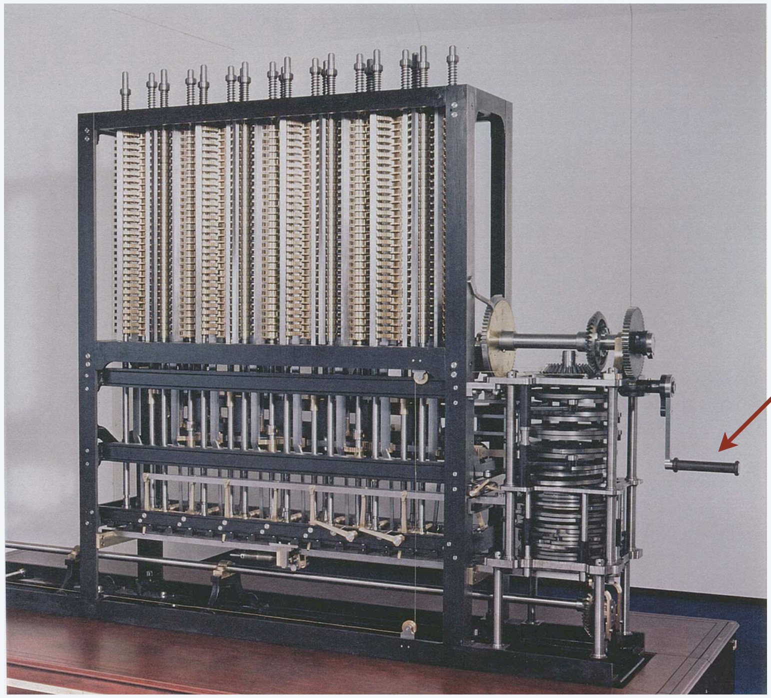
# 2. ALGORITHM ANALYSIS

- *computational tractability*
- *asymptotic order of growth*
- *survey of common running times*

# A strikingly modern thought

*" 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 (1864)*





how many times do you have to turn the crank?

**Analytic Engine**

# Brute force

Brute force. For many nontrivial 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.

# Polynomial running time

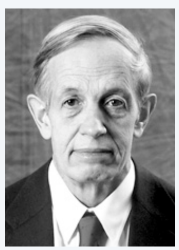
Desirable scaling property. When the input size doubles, the algorithm should only slow down by some constant factor $C$.

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

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

choose $C = 2^d$

**von Neumann (1953)**

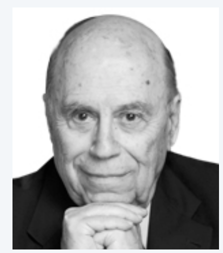


**Nash (1955)**

**Gödel (1956)**

**Cobham (1964)**

**Edmonds (1965)**



**Rabin (1966)**

# Polynomial running time

We say that an algorithm is efficient if has a polynomial running time.

Justification. It really works in practice!

- In practice, the poly-time algorithms that people develop 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/or are useless in practice.

Q. Which would you prefer $20\,n^{100}$ vs. $n^{1\,+\,0.02\,\ln n}$ ?

**Map graphs in polynomial time**

Mikkel Thorup*
Department of Computer Science, University of Copenhagen
Universitetsparken 1, DK-2100 Copenhagen East, Denmark
mthorup@diku.dk

**Abstract**

*Chen, Grigni, and Papadimitriou (WADS'97 and STOC'98) have introduced a modified notion of planarity, where two faces are considered adjacent if they share at least one point. The corresponding abstract graphs are called* map graphs. *Chen et.al. raised the question of whether map graphs can be recognized in polynomial time. They showed that the decision problem is in NP and presented a polynomial time algorithm for the special case where we allow at most 4 faces to intersect in any point — if only 3 are allowed to intersect in a point, we get the usual planar graphs.*

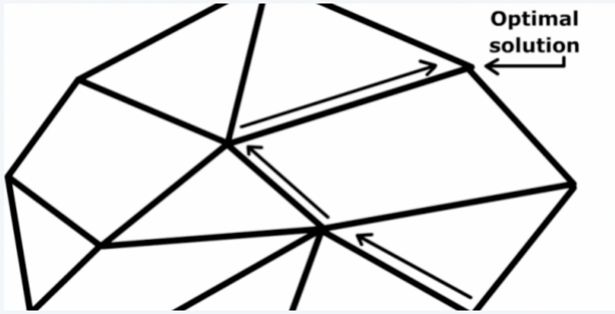
*Chen et.al. conjectured that map graphs can be recognized in polynomial time, and in this paper, their conjecture is settled affirmatively.*

# Worst-case analysis

Worst case. Running time guarantee for any input of size $n$.

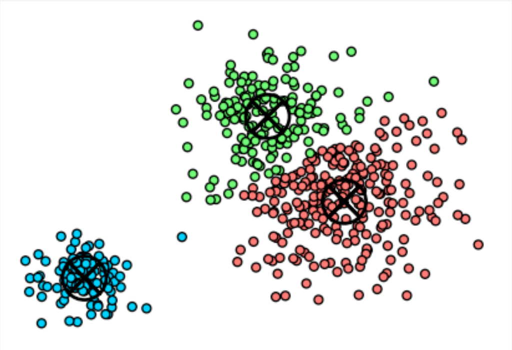
- Generally captures efficiency in practice.
- Draconian view, but hard to find effective alternative.

Exceptions. Some exponential-time algorithms are used widely in practice because the worst-case instances seem to be rare.




simplex algorithm


Linux grep



k-means algorithm

# Types of analyses

Worst case. Running time guarantee for any input of size $n$.
Ex. Heapsort requires at most $2\,n \log_2 n$ compares to sort $n$ elements.

Probabilistic. Expected running time of a randomized algorithm.
Ex. The expected number of compares to quicksort $n$ elements is $\sim 2n \ln n$.

Amortized. Worst-case running time for any sequence of $n$ operations.
Ex. Starting from an empty stack, any sequence of $n$ push and pop operations takes $O(n)$ operations using a resizing array.

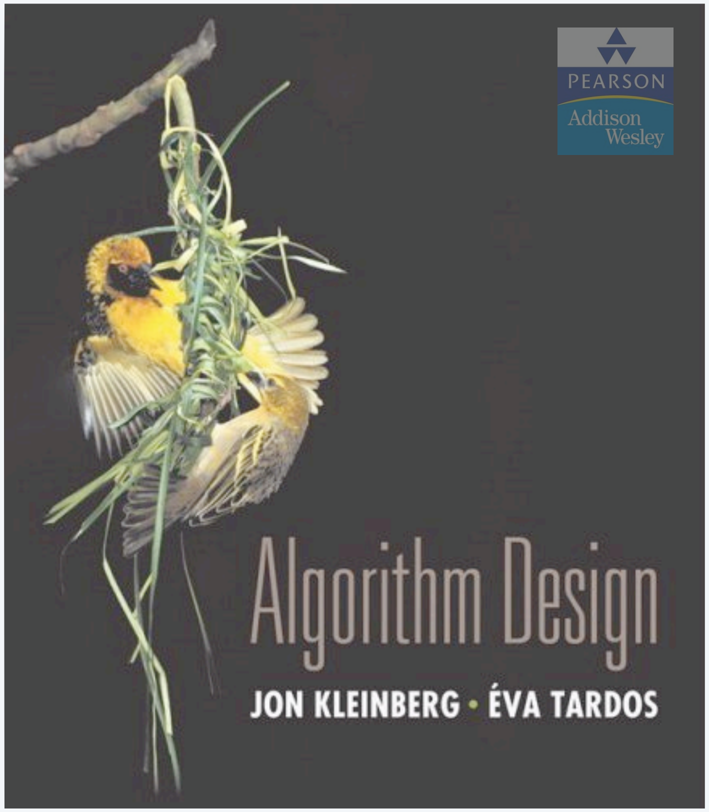
Average-case. Expected running time for a random input of size $n$.
Ex. The expected number of character compares performed by 3-way radix quicksort on $n$ uniformly random strings is $\sim 2n \ln n$.

Also. Smoothed analysis, competitive analysis, ...

# 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 |



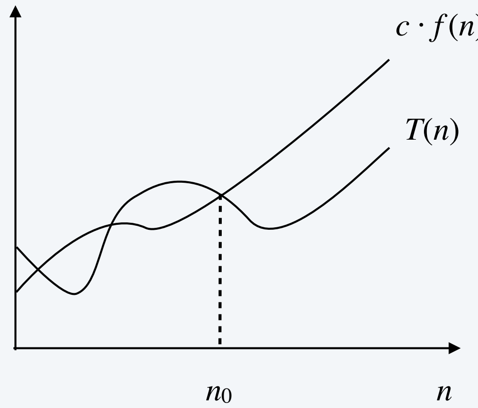

SECTION 2.2

# 2. ALGORITHM ANALYSIS

- computational tractability
- *asymptotic order of growth*
- survey of common running times

# Big-Oh notation

Upper bounds. $T(n)$ is $O(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$.




Ex. $T(n) = 32n^2 + 17n + 1$.

- $T(n)$ is $O(n^2)$. ⟵ choose c = 50, $n_0$ = 1
- $T(n)$ is also $O(n^3)$.
- $T(n)$ is neither $O(n)$ nor $O(n \log n)$.

Typical usage. Insertion makes $O(n^2)$ compares to sort $n$ elements.

Alternate definition. $T(n)$ is $O(f(n))$ if $\limsup_{n \to \infty} \frac{T(n)}{f(n)} < \infty$.

## Notational abuses

Equals sign. $O(f(n))$ is a set of functions, but computer scientists often write $T(n) = O(f(n))$ instead of $T(n) \in O(f(n))$.

Ex. Consider $f(n) = 5n^3$ and $g(n) = 3n^2$ .

- We have $f(n) = O(n^3) = g(n)$.
- Thus, $f(n) = g(n)$.

Domain. The domain of $f(n)$ is typically the natural numbers $\{ 0, 1, 2, \ldots \}$.

- Sometimes we restrict to a subset of the natural numbers.
  Other times we extend to the reals.

Nonnegative functions. When using big-Oh notation, we assume that the functions involved are (asymptotically) nonnegative.

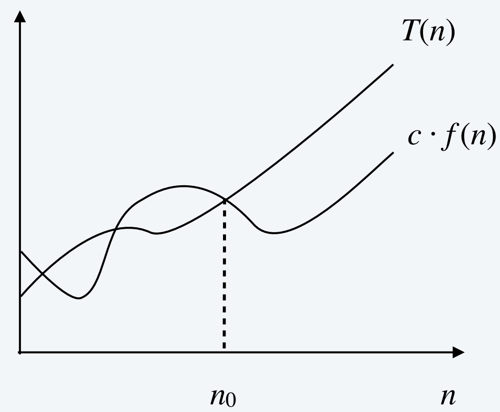
Bottom line. OK to abuse notation; not OK to misuse it.

# Big-Omega notation

Lower bounds. $T(n)$ is $\Omega(f(n))$ if there exist constants $c > 0$ and $n_0 \ge 0$ such that $T(n) \ge c \cdot f(n)$ for all $n \ge n_0$.

Ex. $T(n) = 32n^2 + 17n + 1$.

- $T(n)$ is both $\Omega(n^2)$ and $\Omega(n)$. ← choose c = 32, $n_0$ = 1
- $T(n)$ is neither $\Omega(n^3)$ nor $\Omega(n^3 \log n)$.

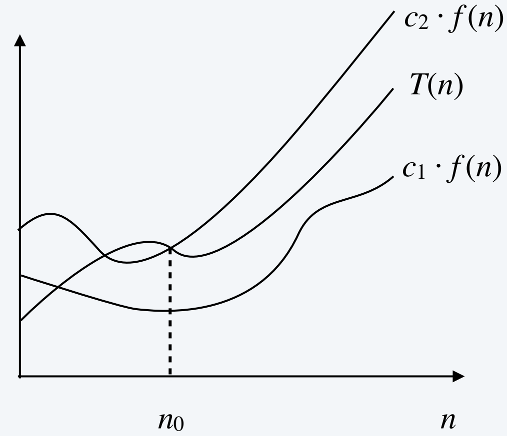



Typical usage. Any compare-based sorting algorithm requires $\Omega(n \log n)$ compares in the worst case.

Meaningless statement. Any compare-based sorting algorithm requires at least $O(n \log n)$ compares in the worst case.

# Big-Theta notation

**Tight bounds.** $T(n)$ is $\Theta(f(n))$ if there exist constants $c_1 > 0$, $c_2 > 0$, and $n_0 \geq 0$ such that $c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$ for all $n \geq n_0$.




**Ex.** $T(n) = 32n^2 + 17n + 1$.

- $T(n)$ is $\Theta(n^2)$. ← choose $c_1 = 32$, $c_2 = 50$, $n_0 = 1$
- $T(n)$ is neither $\Theta(n)$ nor $\Theta(n^3)$.

**Typical usage.** Mergesort makes $\Theta(n \log n)$ compares to sort $n$ elements.

# Useful facts

**Proposition.** If $\lim_{n \to \infty} \frac{f(n)}{g(n)} = c > 0$, then $f(n)$ is $\Theta(g(n))$.

**Pf.** By definition of the limit, there exists $n_0$ such such that for all $n \geq n_0$

$$\frac{1}{2} c < \frac{f(n)}{g(n)} < 2c$$

- Thus, $f(n) \leq 2 c\, g(n)$ for all $n \geq n_0$, which implies $f(n)$ is $O(g(n))$.
- Similarly, $f(n) \geq \frac{1}{2} c\, g(n)$ for all $n \geq n_0$, which implies $f(n)$ is $\Omega(g(n))$.

**Proposition.** If $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$, then $f(n)$ is $O(g(n))$.

# Asymptotic bounds for some common functions

Polynomials. Let $T(n) = a_0 + a_1 n + \ldots + a_d n^d$ with $a_d > 0$. Then, $T(n)$ is $\Theta(n^d)$.

Pf. $$\lim_{n \to \infty} \frac{a_0 + a_1 n + \ldots + a_d n^d}{n^d} = a_d > 0$$

Logarithms. $\Theta(\log_a n)$ is $\Theta(\log_b n)$ for any constants $a, b > 0$.

no need to specify base (assuming it is a constant)

Logarithms and polynomials. For every $d > 0$, $\log n$ is $O(n^d)$.

Exponentials and polynomials. For every $r > 1$ and every $d > 0$, $n^d$ is $O(r^n)$.

Pf. $$\lim_{n \to \infty} \frac{n^d}{r^n} = 0$$

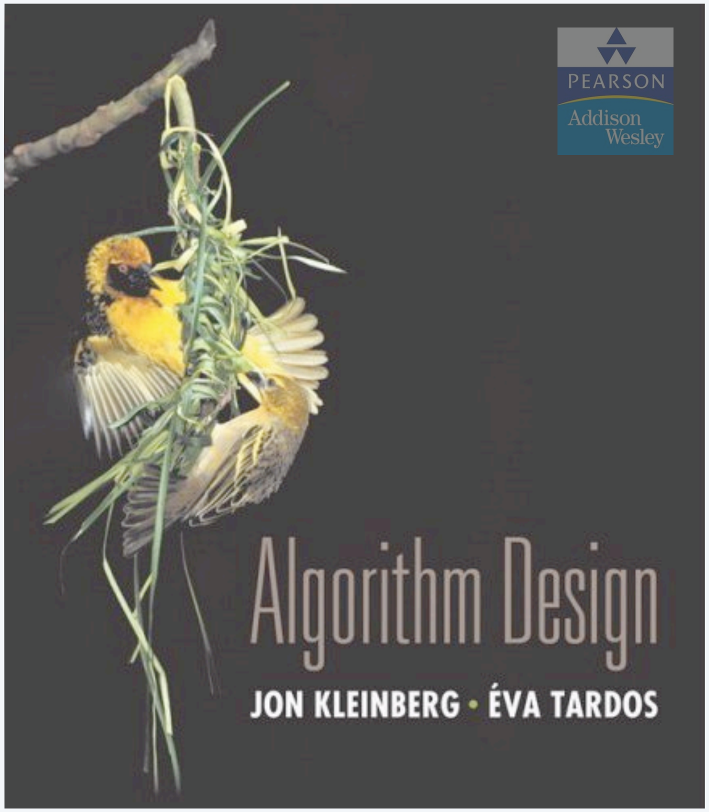
# Big-Oh notation with multiple variables

Upper bounds. $T(m, n)$ is $O(f(m, n))$ if there exist constants $c > 0$, $m_0 \geq 0$, and $n_0 \geq 0$ such that $T(m, n) \leq c \cdot f(m, n)$ for all $n \geq n_0$ and $m \geq m_0$.

Ex. $T(m, n) = 32mn^2 + 17mn + 32n^3$.

- $T(m, n)$ is both $O(mn^2 + n^3)$ and $O(mn^3)$.
- $T(m, n)$ is neither $O(n^3)$ nor $O(mn^2)$.

Typical usage. Breadth-first search takes $O(m + n)$ time to find the shortest path from $s$ to $t$ in a digraph.




SECTION 2.4

# 2. ALGORITHM ANALYSIS

- *computational tractability*
- *asymptotic order of growth*
- *survey of common running times*

# Linear time: O(n)

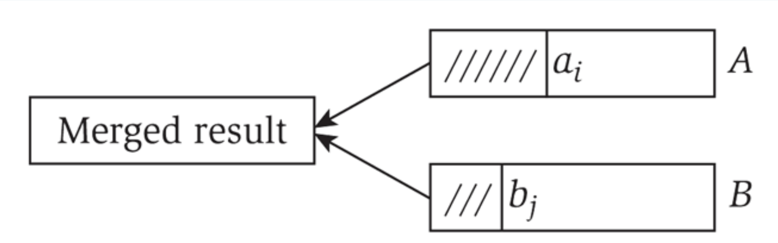
Linear time. Running time is proportional to input size.

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

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

# Linear time: O(n)

Merge. Combine two sorted lists $A = a_1, a_2, \ldots, a_n$ with $B = b_1, b_2, \ldots, b_n$ 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 compare, the length of output list increases by $1$.

# Linearithmic time: O(n log n)

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

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

Largest empty interval. Given $n$ time-stamps $x_1, \ldots, x_n$ on which copies of a file arrive at a server, what is largest interval when no copies of 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($n^2$)

Ex. Enumerate all pairs of elements.

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

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

```
min ← (x₁ - x₂)² + (y₁ - y₂)²
for i = 1 to n {
   for j = i+1 to n {
      d ← (xᵢ - xⱼ)² + (yᵢ - yⱼ)²
      if (d < min)
         min ← d
   }
}
```

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 $S_1, \ldots, S_n$ each of which is a subset of $1, 2, \ldots, n$, is there some pair of these which are disjoint?

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

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

# Polynomial time: O(n^k)

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 in an independent set
   if (S is an independent set)
      report S is an independent set
   }
}
```

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

poly-time for k=17,
but not practical

# Exponential time

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

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

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

## Sublinear time

Search in a sorted array. Given a sorted array $A$ of $n$ numbers, is a given number $x$ in the array?

O(log n) solution. Binary search.

```
lo ← 1, hi ← n
while (lo ≤ hi) {
   mid ← (lo + hi) / 2
   if      (x < A[mid]) hi ← mid - 1
   else if (x > A[mid]) lo ← mid + 1
   else return yes
}
return no
```