# Algorithm Analysis

## Part I

Tyler Moore

CS 2123, The University of Tulsa

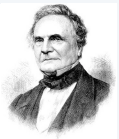
Some slides created by or adapted from Dr. Kevin Wayne. For more information see

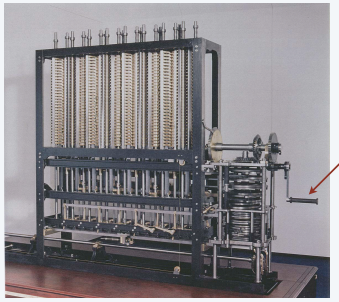
http://www.cs.princeton.edu/~wayne/kleinberg-tardos.

Some slides adapted from Dr. Steven Skiena. For more information see http://www.algorist.com

## 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

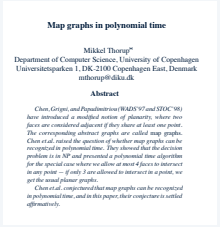
We say that an algorithm is efficient if has a polynomial running time.

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 $20n^{100}$ 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.*

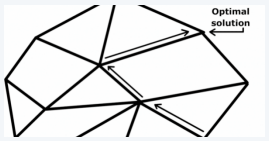


## Worst-case analysis

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

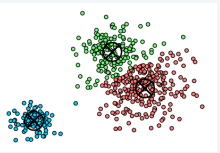
- Generally captures efficiency in practice.
- Draconian view, but hard to find effective alternative.

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 $2n\log_2 n$ 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, ...



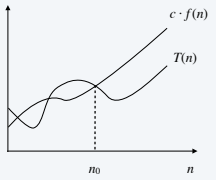
## Why it matters

**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 |

## 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.

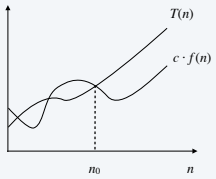
Nonnegative functions. When using big-Oh notation, we assume that the functions involved are (asymptotically) nonnegative.

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 $n_0 \geq 0$ such that $T(n) \geq c \cdot f(n)$ for all $n \geq n_0$.




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 $\Omega(n^3 \log n)$.

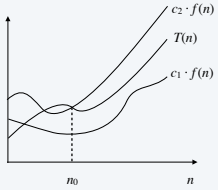
Typical usage. Any compare-based sorting algorithm requires $\Omega(n \log n)$ compares in the worst case.

Meaningless statement. Any compare-based sorting algorithm requires at least $\Omega(n \log n)$ 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.

## Big Oh Examples

**Definition**

$T(n)$ 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 $n \geq n_0$.

1. $3n^2 + 4n + 6 = O(n^2)$?
   - Yes, because for $c = 13$ and $n_o \geq 1$, $3n^2 + 4n + 6 \leq 3n^2 + 4n^2 + 6n^2 = 13n^2$
2. $3n^2 + 4n + 6 = O(n^3)$?
   - Yes, because for $c = 1$ and $n_o \geq 13$, $3n^2 + 4n + 6 \leq 3n^2 + 4n^2 + 6n^2 = 13n^2 \leq 13n^3$
3. $3n^2 + 4n + 6 = O(n)$?
   - No, because $c \cdot n < 3n^2 + 4n + 6$ when $n > c$



## Big Omega Examples

**Definition**

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

1. $3n^2 + 4n + 6 = \Omega(n^2)$?
   - Yes, because for $c = 2$ and $n_o \geq 1$, $3n^2 + 4n + 6 \geq 2n^2$
2. $3n^2 + 4n + 6 = \Omega(n^3)$?
   - No, because for $c = 13$ and $n_o \geq 1$, $3n^2 + 4n + 6 < 13n^3$
3. $3n^2 + 4n + 6 = \Omega(n)$?
   - Yes, because for $c = 2$ and $n_0 \geq 100$, $3n^2 + 4n + 6 > 2n$



## Big Theta Examples

**Definition**

$T(n)$ 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 $n \geq n_0$.

1. $3n^2 + 4n + 6 = \Theta(n^2)$?
   - Yes, because $O$ and $\Omega$ apply
2. $3n^2 + 4n + 6 = \Theta(n^3)$?
   - No, because only $O$ applies
3. $3n^2 + 4n + 6 = \Theta(n)$?
   - No, because only $\Omega$ applies



## Exercises

1. Is $3n + 4n = O(n^2)$? (yes or no)
2. Is $2^n + 10n = \Omega(n^3)$? (yes or no)
3. Pick a suitable $c$ and $n_0$ to show that $3n^2 + 2n = \Omega(n^2)$

## Big Oh Addition/Subtraction

- Suppose $f(n) = O(n^2)$ and $g(n) = O(n^2)$.
- What do we know about $g'(n) = f(n) + g(n)$?
  - Adding bounding constants shows $g'(n) = O(n^2)$
- What do we know about $g''(n) = f(n) - g(n)$?
  - Since bounding constants may not cancel, $g''(n) = O(n^2)$
- What about lower bounds? Does $g'(n) = \Omega(n^2)$
  - We know nothing about lower bounds on $g'$ and $g''$ because we don't know about the lower bounds on $f$ and $g$.



## Big Oh Multiplication

- Multiplication by a constant does not change the asymptotics
  - $O(c \cdot f(n)) \to O(f(n))$
  - $\Omega(c \cdot f(n)) \to \Omega(f(n))$
  - $\Theta(c \cdot f(n)) \to \Theta(f(n))$
- But when both functions in a product are increasing, both are important
  - $O(f(n)) \cdot O(g(n)) \to O(f(n) \cdot g(n))$
  - $\Omega(f(n)) \cdot \Omega(g(n)) \to \Omega(f(n) \cdot g(n))$
  - $\Theta(f(n)) \cdot \Theta(g(n)) \to \Theta(f(n) \cdot g(n))$



### 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.



## 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 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.

## Binary Search and Logarithms

- In binary search we throw away half the possible number of keys after each comparison.
- Thus twenty comparisons suffice to find any name in the million-name Manhattan phone book!
- **Question:** how many times can we halve $n$ before getting to 1?



## Logarithms and Binary Trees

- How tall a binary tree do we need until we have $n$ leaves?

$\rightarrow$ The number of potential leaves doubles with each level.

- How many times can we double 1 until we get to $n$?



## Logarithms and Bits

- How many bits do you need to represent the numbers from 0 to $2^i - 1$?



## Logarithms and Multiplication

- Recall that $\log_a(xy) = log_a(x) + log_a(y)$
- This is how people used to multiply before calculators, and remains useful for analysis.
- What if $x = a$?

## The Base is not Asymptotically Important

- Recall the definition, $c^{\log_c x} = x$ and that

$$\log_b a = \frac{\log_c a}{\log_c b}$$

So for $a = 2$ and $c = 100$:

$$\log_2 n = \frac{\log_{100} n}{\log_{100} 2}$$

- Since $\frac{1}{\log_{100} 2} = 6.643$ is a constant, we can ignore it when calculating Big Oh



## Federal Sentencing Guidelines

F1.1. Fraud and Deceit; Forgery; Offenses Involving Altered or Counterfeit Instruments other than Counterfeit Bearer Obligations of the United States. (a) Base offense Level: 6 (b) Specific offense Characteristics (1) If the loss exceeded $2,000, increase the offense level as follows:

| **Loss(Apply the Greatest)** | **Increase in Level** |
|---|---|
| (A) $2,000 or less | no increase |
| (B) More than $2,000 | add 1 |
| (C) More than $5,000 | add 2 |
| (D) More than $10,000 | add 3 |
| (E) More than $20,000 | add 4 |
| (F) More than $40,000 | add 5 |
| (G) More than $70,000 | add 6 |
| (H) More than $120,000 | add 7 |
| (I) More than $200,000 | add 8 |
| (J) More than $350,000 | add 9 |
| (K) More than $500,000 | add 10 |
| (L) More than $800,000 | add 11 |
| (M) More than $1,500,000 | add 12 |
| (N) More than $2,500,000 | add 13 |
| (O) More than $5,000,000 | add 14 |
| (P) More than $10,000,000 | add 15 |
| (Q) More than $20,000,000 | add 16 |
| (R) More than $40,000,000 | add 17 |
| (Q) More than $80,000,000 | add 18 |

- The increase in punishment level grows logarithmically in the amount of money stolen.
- Thus it pays to commit one big crime rather than many small crimes totaling the same amount.
- In other words, **Make the Crime Worth the Time**