http://algs4.cs.princeton.edu

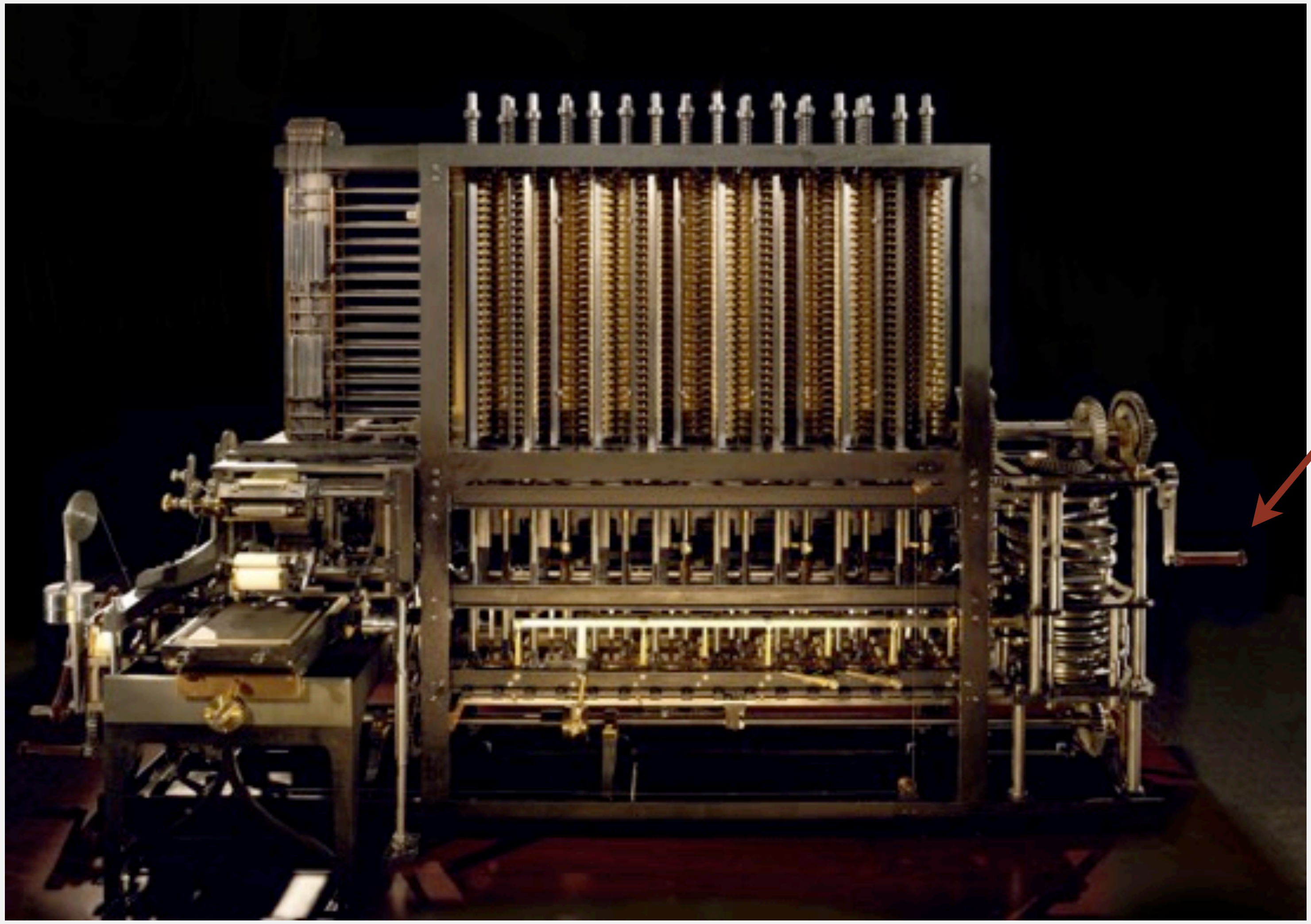
1.4 ANALYSIS OF ALGORITHMS

- *introduction*
- *observations*
- *mathematical models*
- *order-of-growth classifications*
- *memory*

Last updated on Sep 21, 2015, 5:44 PM

Running time

" *As soon as an Analytical Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will then 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?

Reasons to analyze algorithms

Predict performance.

Compare algorithms.

this course
(COS 226)

Provide guarantees.

theory of algorithms
(COS 423)

Understand theoretical basis.

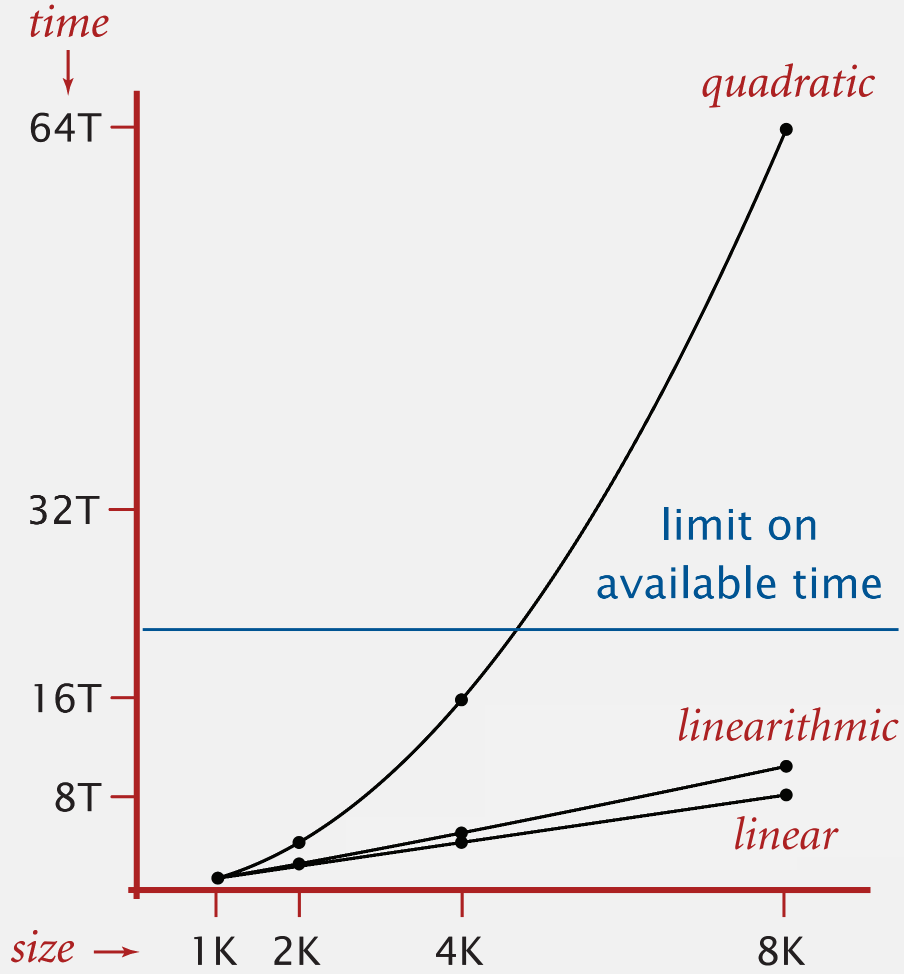
Primary practical reason: avoid performance bugs.

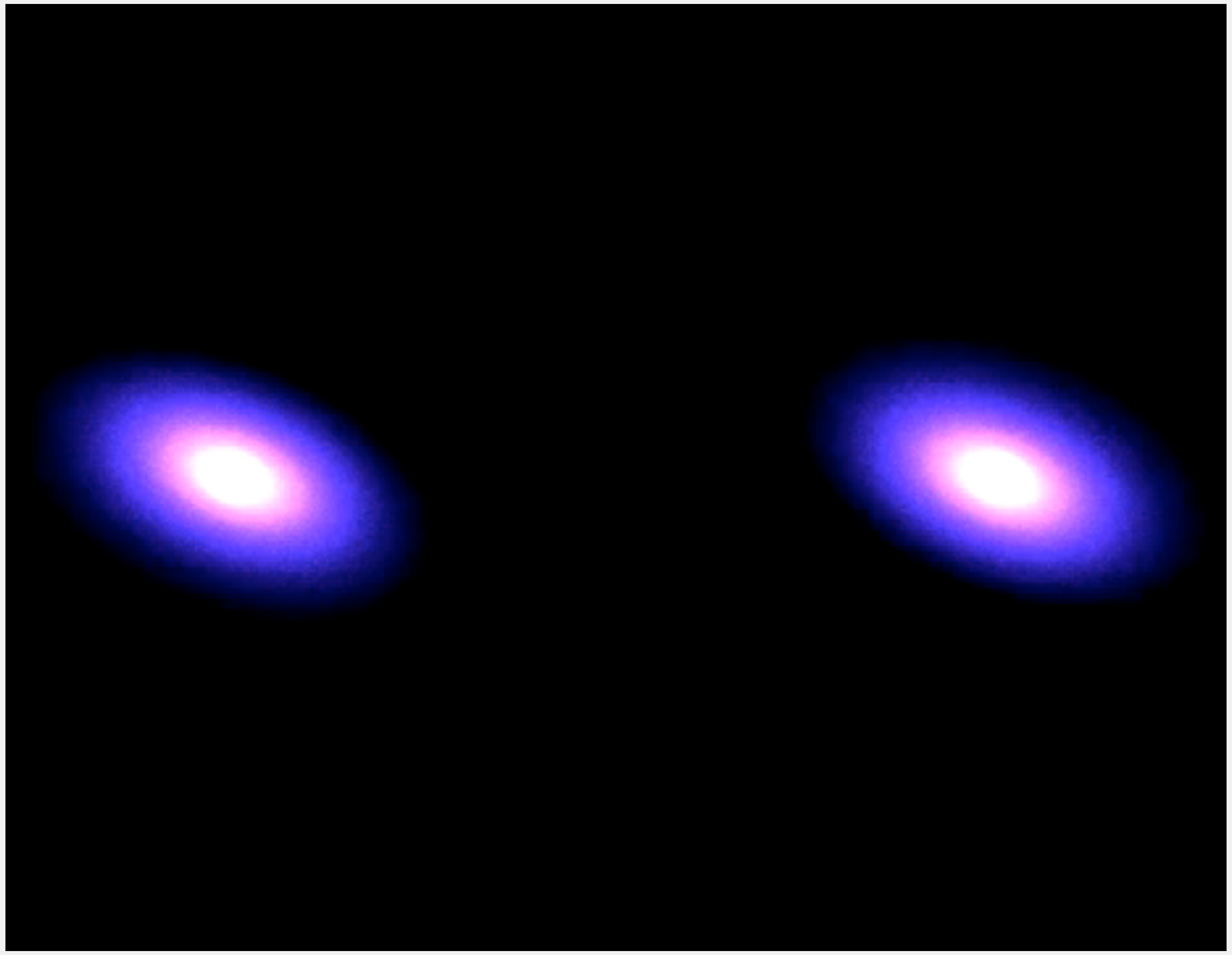
An algorithmic success story

N-body simulation.

- Simulate gravitational interactions among N bodies.
- Applications: cosmology, fluid dynamics, semiconductors, ...
- Brute force: N^2 steps.
- Barnes-Hut algorithm: $N \log N$ steps, enables new research.

Andrew Appel
PU '81





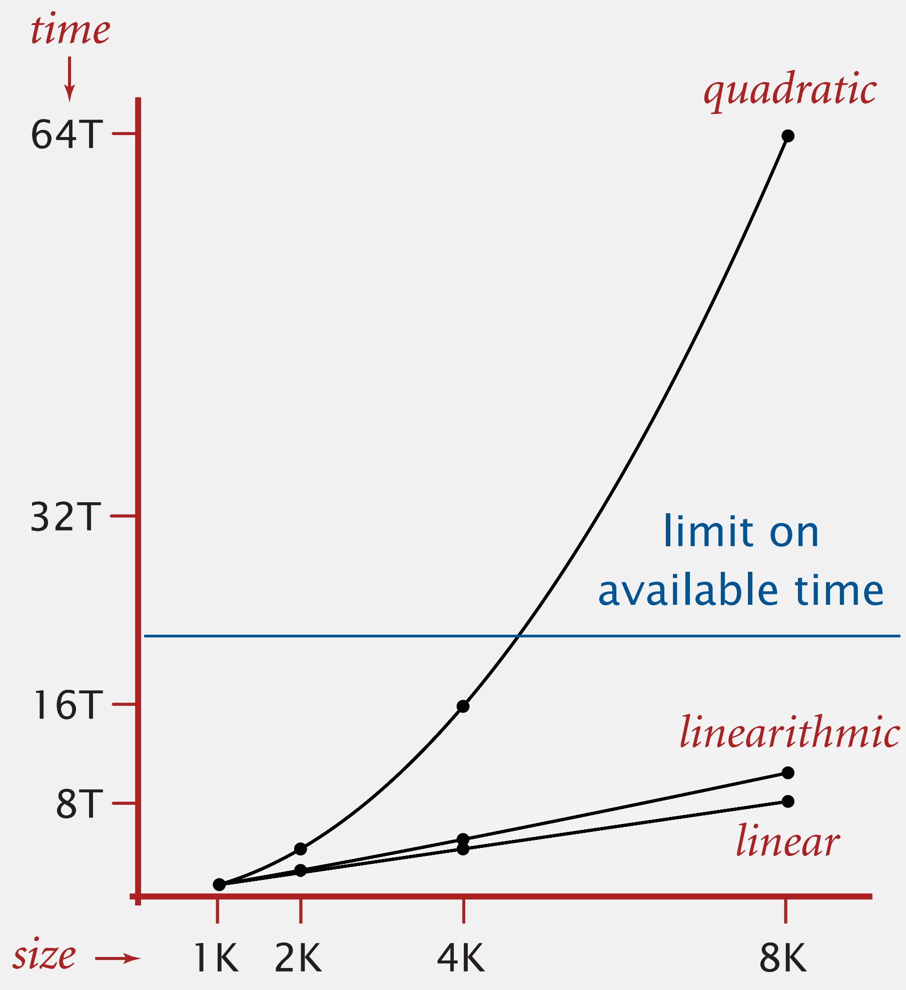
Another algorithmic success story

Discrete Fourier transform.

- Express signal as weighted sum of sines and cosines.
- Applications: DVD, JPEG, MRI, astrophysics, ...
- Brute force: N^2 steps.
- FFT algorithm: $N \log N$ steps, enables new technology.

James Cooley

John Tukey



http://algs4.cs.princeton.edu

1.4 Analysis of Algorithms

- introduction
- **observations**
- mathematical models
- order-of-growth classifications
- memory

The challenge

Q. Will my program be able to solve a large practical input?

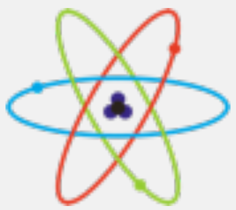
Insight. [Knuth 1970s] Use scientific method to understand performance.

Scientific method applied to the analysis of algorithms

A framework for predicting performance and comparing algorithms.

Scientific method.

- Observe some feature of the natural world.
- Hypothesize a model that is consistent with the observations.
- Predict events using the hypothesis.
- Verify the predictions by making further observations.
- Validate by repeating until the hypothesis and observations agree.

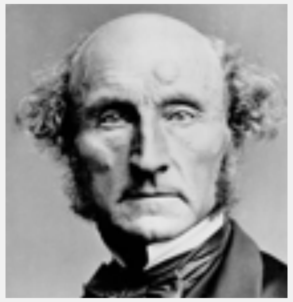


Principles.

- Experiments must be reproducible.
- Hypotheses must be falsifiable.

Francis Bacon

René Descartes



John Stuart Mills

Feature of the natural world. Computer itself.

Example: 3-SUM

3-SUM. Given N distinct integers, how many triples sum to exactly zero?

Context. Deeply related to problems in computational geometry.

```
% more 8ints.txt
8
30 -40 -20 -10 40 0 10 5

% java ThreeSum 8ints.txt
4
```

	a[i]	a[j]	a[k]	sum
1	30	-40	10	0
2	30	-20	-10	0
3	-40	40	0	0
4	-10	0	10	0

3-SUM: brute-force algorithm

```
public class ThreeSum
{
   public static int count(int[] a)
   {
      int N = a.length;
      int count = 0;
      for (int i = 0; i < N; i++)
         for (int j = i+1; j < N; j++)
            for (int k = j+1; k < N; k++)
               if (a[i] + a[j] + a[k] == 0)
                  count++;
      return count;
   }

   public static void main(String[] args)
   {
      In in = new In(args[0]);
      int[] a = in.readAllInts();
      StdOut.println(count(a));
   }
}
```

check each triple

for simplicity, ignore integer overflow

Measuring the running time

Q. How to time a program?

A. Manual.

`% java ThreeSum 1Kints.txt`

tick tick tick

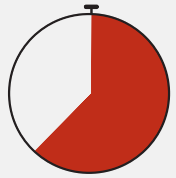
`70`

`% java ThreeSum 2Kints.txt`

tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick

`528`

`% java ThreeSum 4Kints.txt`



tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick
tick tick tick tick tick tick tick tick

`4039`

Measuring the running time

Q. How to time a program?

A. Automatic.

public class Stopwatch		(part of `stdlib.jar`)
	`Stopwatch()`	*create a new stopwatch*
`double`	`elapsedTime()`	*time since creation (in seconds)*

```
public static void main(String[] args)
{
   In in = new In(args[0]);
   int[] a = in.readAllInts();
   Stopwatch stopwatch = new Stopwatch();
   StdOut.println(ThreeSum.count(a));
   double time = stopwatch.elapsedTime();
   StdOut.println("elapsed time = " + time);
}
```

Empirical analysis

Run the program for various input sizes and measure running time.

Empirical analysis

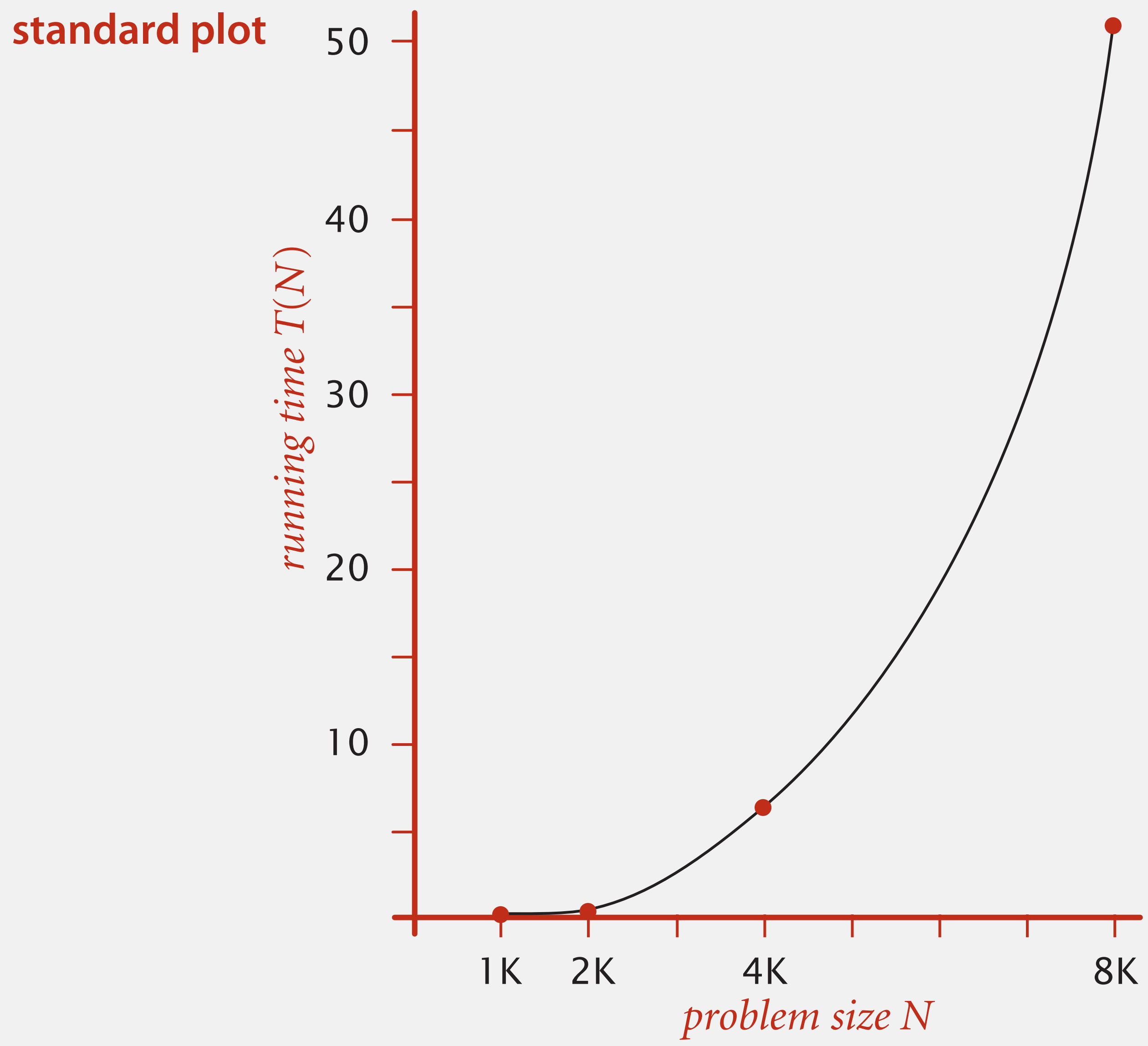
Run the program for various input sizes and measure running time.

N	time (seconds) †
250	0.0
500	0.0
1,000	0.1
2,000	0.8
4,000	6.4
8,000	51.1

† on a 2.8GHz Intel PU-226 with 64GB DDR E3 memory and 32MB L3 cache; running Oracle Java 1.7.0_45-b18 on Springdale Linux v. 6.5

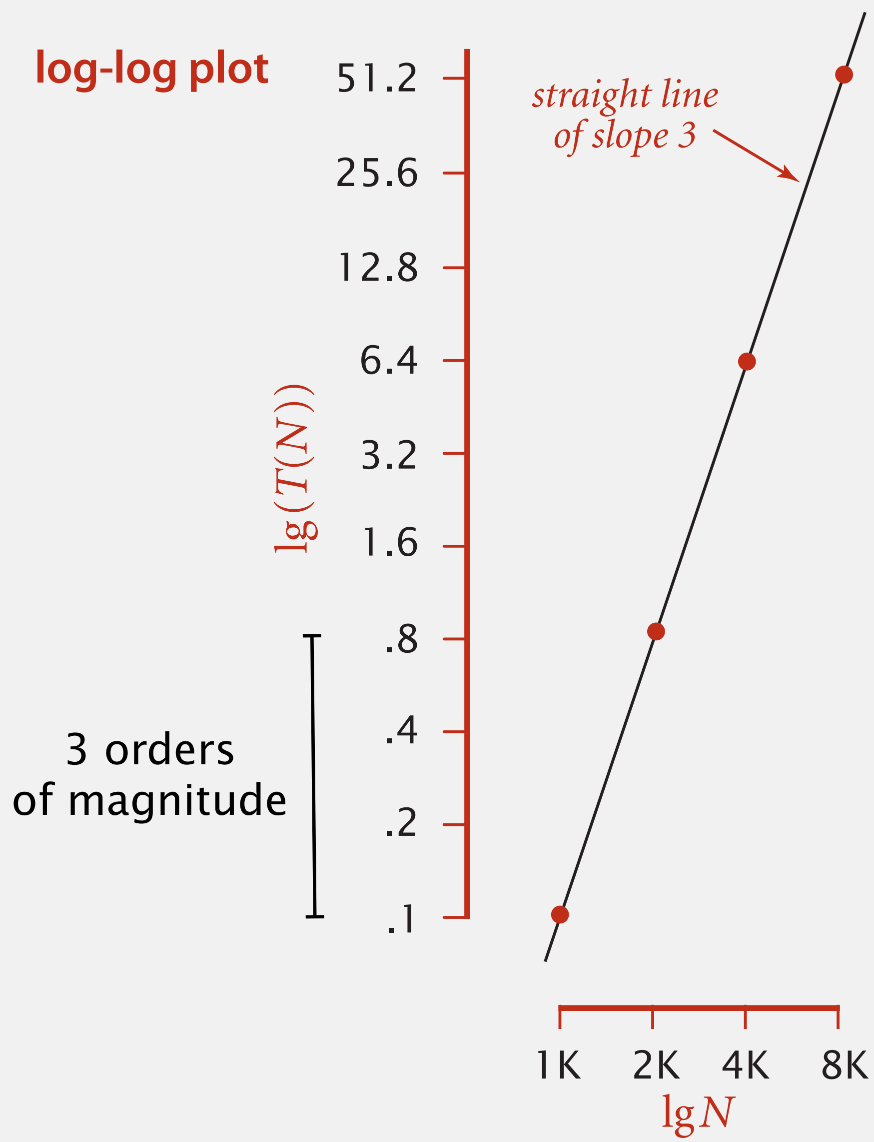
Data analysis

Standard plot. Plot running time $T(N)$ vs. input size N.



Data analysis

Log-log plot. Plot running time $T(N)$ vs. input size N using log-log scale.



$$\lg(T(N)) = b \lg N + c$$
$$b = 2.999$$
$$c = -33.2103$$

$$T(N) = a N^b, \text{ where } a = 2^c$$

power law

Regression. Fit straight line through data points: $a N^b$.

slope

Hypothesis. The running time is about $1.006 \times 10^{-10} \times N^{2.999}$ seconds.

Prediction and validation

Hypothesis. The running time is about $1.006 \times 10^{-10} \times N^{2.999}$ seconds.

Predictions.

- 51.0 seconds for $N = 8{,}000$.
- 408.1 seconds for $N = 16{,}000$.

Observations.

N	time (seconds) †
8,000	51.1
8,000	51.0
8,000	51.1
16,000	410.8

validates hypothesis!

Doubling hypothesis

Doubling hypothesis. Quick way to estimate b in a power-law relationship.

Run program, doubling the size of the input.

N	time (seconds) †	ratio	lg ratio
250	0.0		–
500	0.0	4.8	2.3
1,000	0.1	6.9	2.8
2,000	0.8	7.7	2.9
4,000	6.4	8.0	3.0
8,000	51.1	8.0	3.0

lg (6.4 / 0.8) = 3.0

seems to converge to a constant b ≈ 3

$$\frac{T(N)}{T(N/2)} = \frac{aN^b}{a(N/2)^b} = 2^b$$

Hypothesis. Running time is about $a N^b$ with b = lg ratio.

Caveat. Cannot identify logarithmic factors with doubling hypothesis.

Doubling hypothesis

Doubling hypothesis. Quick way to estimate b in a power-law relationship.

Q. How to estimate a (assuming we know b) ?

A. Run the program (for a sufficient large value of N) and solve for a.

N	time (seconds) †
8,000	51.1
8,000	51.0
8,000	51.1

$51.1 = a \times 8000^3$

$\Rightarrow \; a = 0.998 \times 10^{-10}$

Hypothesis. Running time is about $0.998 \times 10^{-10} \times N^3$ seconds.

almost identical hypothesis
to one obtained via regression

Analysis of algorithms quiz 1

Estimate the running time to solve a problem of size N = 96,000.

A. 39 *seconds*.

B. 52 *seconds*.

C. 117 *seconds*.

D. 350 *seconds*.

E. *I don't know.*

N	time (seconds) †
1000	0.02
2000	0.05
4,000	0.20
8,000	0.81
16,000	3.25
32,000	13.00

Experimental algorithmics

System independent effects.

- Algorithm.
- Input data.

determines exponent b in power law $a\,N^{\,b}$

System dependent effects.

- Hardware: CPU, memory, cache, ...
- Software: compiler, interpreter, garbage collector, ...
- System: operating system, network, other apps, ...

determines constant a in power law $a\,N^{\,b}$

Bad news. Sometimes difficult to get precise measurements.

Good news. Much easier and cheaper than other sciences.

An aside

Algorithmic experiments are virtually free by comparison with other sciences.

Chemistry
(1 experiment)

Biology
(1 experiment)

Physics
(1 experiment)

Computer Science
(1 million experiments)

Bottom line. No excuse for not running experiments to understand costs.

http://algs4.cs.princeton.edu

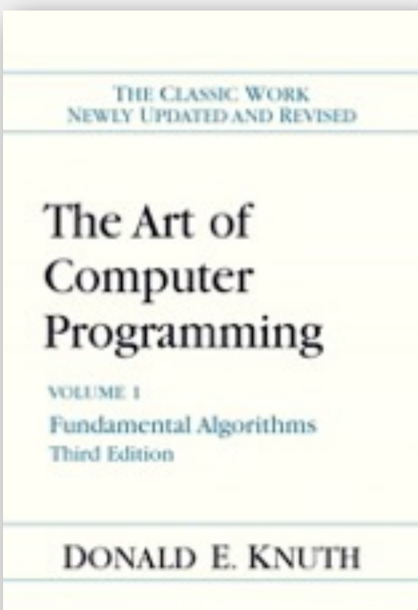
1.4 Analysis of Algorithms

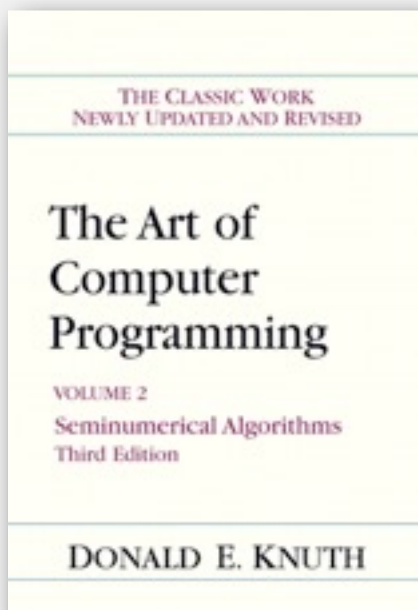
- *introduction*
- *observations*
- *mathematical models*
- *order-of-growth classifications*
- *memory*

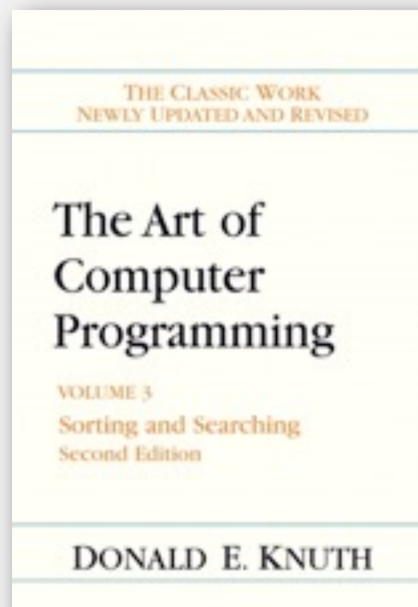
Mathematical models for running time

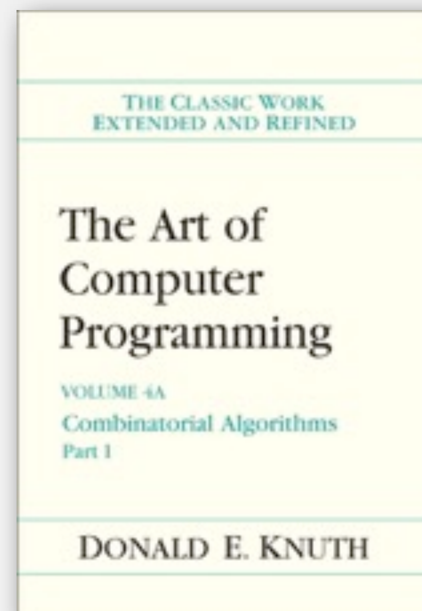
Total running time: sum of cost × frequency for all operations.

- Need to analyze program to determine set of operations.
- Cost depends on machine, compiler.
- Frequency depends on algorithm, input data.









Donald Knuth
1974 Turing Award

In principle, accurate mathematical models are available.

Example: 1-SUM

Q. How many instructions as a function of input size N ?

```
int count = 0;
for (int i = 0; i < N; i++)
   if (a[i] == 0)
      count++;
```

N array accesses

operation	cost (ns) †	frequency
variable declaration	2/5	2
assignment statement	1/5	2
less than compare	1/5	$N + 1$
equal to compare	1/10	N
array access	1/10	N
increment	1/10	N to $2\,N$

† representative estimates (with some poetic license)

Example: 2-SUM

Q. How many instructions as a function of input size N?

```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      if (a[i] + a[j] == 0)
         count++;
```

$$0 + 1 + 2 + \ldots + (N-1) = \frac{1}{2} N (N-1) = \binom{N}{2}$$

Pf. [Gauss]

$$\begin{array}{ccccccccccccc} & T(N) & = & 0 & + & 1 & + & \ldots & + & (N-2) & + & (N-1) \\ + & T(N) & = & (N-1) & + & (N-2) & + & \ldots & + & 1 & + & 0 \\ \hline & 2\,T(N) & = & (N-1) & + & (N-1) & + & \ldots & + & (N-1) & + & (N-1) \end{array}$$

$$\Rightarrow \quad T(N) = N(N-1)/2$$

Example: 2-SUM

Q. How many instructions as a function of input size N ?

```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      if (a[i] + a[j] == 0)
         count++;
```

$$0 + 1 + 2 + \ldots + (N-1) = \frac{1}{2} N(N-1) = \binom{N}{2}$$

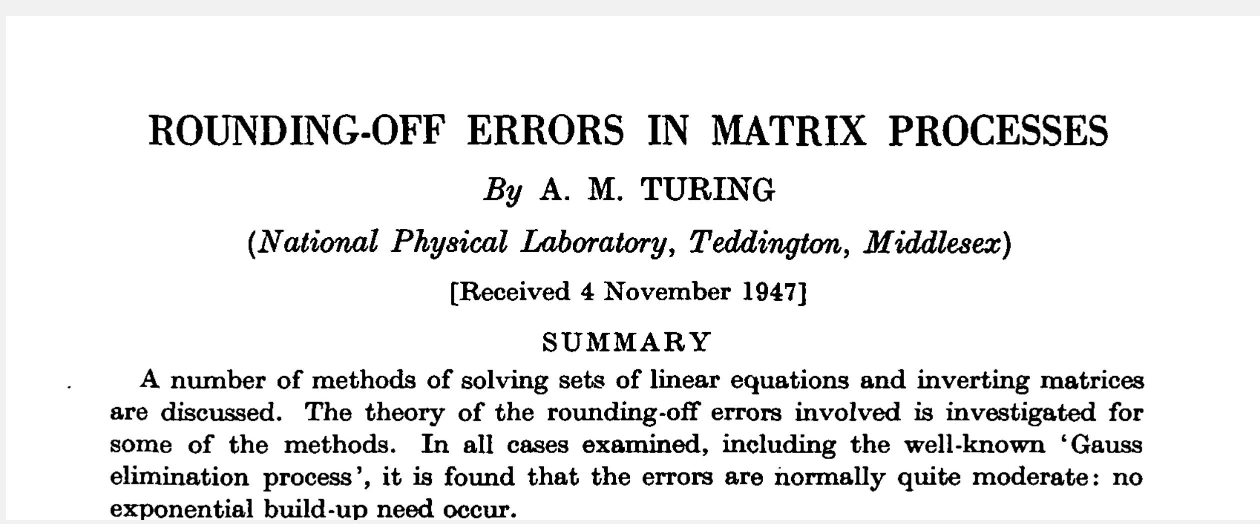
operation	cost (ns)	frequency
variable declaration	2/5	$N+2$
assignment statement	1/5	$N+2$
less than compare	1/5	$\frac{1}{2}(N+1)(N+2)$
equal to compare	1/10	$\frac{1}{2}N(N-1)$
array access	1/10	$N(N-1)$
increment	1/10	$\frac{1}{2}N(N+1)$ to N^2

1/4 N^2 + 13/20 N + 13/10 ns
to
3/10 N^2 + 3/5 N + 13/10 ns

(tedious to count exactly)

Simplifying the calculations

" It is convenient to have a measure of the amount of work involved in a computing process, even though it be a very crude one. We may count up the number of times that various elementary operations are applied in the whole process and then given them various weights. We might, for instance, count the number of additions, subtractions, multiplications, divisions, recording of numbers, and extractions of figures from tables. In the case of computing with matrices most of the work consists of multiplications and writing down numbers, and we shall therefore only attempt to count the number of multiplications and recordings. " — *Alan Turing*



ROUNDING-OFF ERRORS IN MATRIX PROCESSES

By A. M. TURING

(*National Physical Laboratory, Teddington, Middlesex*)

[Received 4 November 1947]

SUMMARY

A number of methods of solving sets of linear equations and inverting matrices are discussed. The theory of the rounding-off errors involved is investigated for some of the methods. In all cases examined, including the well-known 'Gauss elimination process', it is found that the errors are normally quite moderate: no exponential build-up need occur.

Simplification 1: cost model

Cost model. Use some basic operation as a proxy for running time.

```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      if (a[i] + a[j] == 0)
         count++;
```

$$0 + 1 + 2 + \ldots + (N-1) = \frac{1}{2} N (N-1) = \binom{N}{2}$$

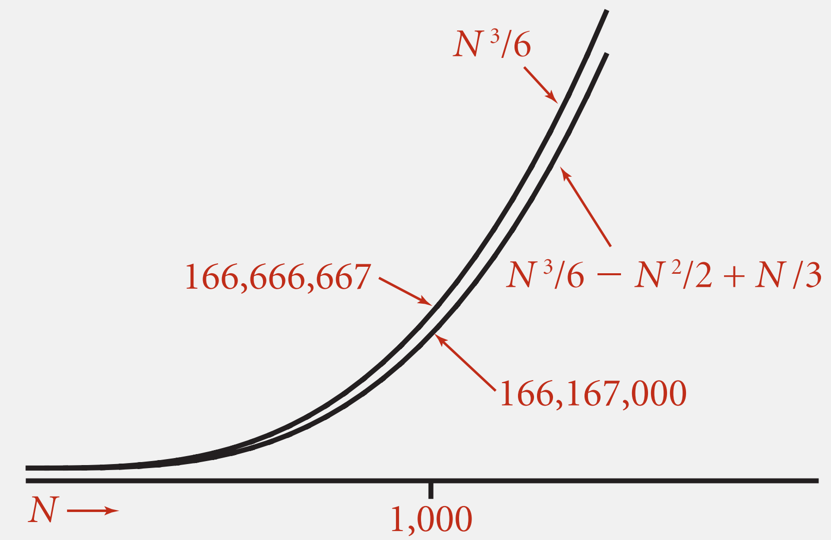
operation	cost (ns)	frequency
variable declaration	2/5	$N + 2$
assignment statement	1/5	$N + 2$
less than compare	1/5	$½ (N + 1) (N + 2)$
equal to compare	1/10	$½ N (N - 1)$
array access	1/10	$\boxed{N (N - 1)}$
increment	1/10	$½ N (N + 1)$ to N^2

cost model = array accesses

(we assume compiler/JVM do not optimize any array accesses away!)

Simplification 2: tilde notation

- Estimate running time (or memory) as a function of input size N.
- Ignore lower order terms.
  - when N is large, terms are negligible
  - when N is small, we don't care



Leading-term approximation

Ex 1. $\tfrac{1}{6} N^3 + 20 N + 16 \quad \sim \quad \tfrac{1}{6} N^3$

Ex 2. $\tfrac{1}{6} N^3 + 100 N^{4/3} + 56 \quad \sim \quad \tfrac{1}{6} N^3$

Ex 3. $\tfrac{1}{6} N^3 - \underbrace{\tfrac{1}{2} N^2 + \tfrac{1}{3} N} \quad \sim \quad \tfrac{1}{6} N^3$

discard lower-order terms
(e.g., N = 1000: 166.67 million vs. 166.17 million)

Technical definition. $f(N) \sim g(N)$ means $\displaystyle \lim_{N \to \infty} \frac{f(N)}{g(N)} = 1$

Simplification 2: tilde notation

- Estimate running time (or memory) as a function of input size N.
- Ignore lower order terms.
  - when N is large, terms are negligible
  - when N is small, we don't care

operation	frequency	tilde notation
variable declaration	$N + 2$	$\sim N$
assignment statement	$N + 2$	$\sim N$
less than compare	$\frac{1}{2} (N + 1) (N + 2)$	$\sim \frac{1}{2} N^2$
equal to compare	$\frac{1}{2} N (N - 1)$	$\sim \frac{1}{2} N^2$
array access	$N (N - 1)$	$\sim N^2$
increment	$\frac{1}{2} N (N + 1)$ to N^2	$\sim \frac{1}{2} N^2$ to $\sim N^2$

Example: 2-SUM

Q. Approximately how many array accesses as a function of input size N?

```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      if (a[i] + a[j] == 0)
         count++;
```

"inner loop"

$$0 + 1 + 2 + \ldots + (N-1) = \frac{1}{2} N (N-1) = \binom{N}{2}$$

A. $\sim N^2$ array accesses.

Bottom line. Use cost model and tilde notation to simplify counts.

Example: 3-SUM

Q. Approximately how many array accesses as a function of input size N?

```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      for (int k = j+1; k < N; k++)
         if (a[i] + a[j] + a[k] == 0)
            count++;
```

"inner loop"

$$\binom{N}{3} = \frac{N(N-1)(N-2)}{3!} \sim \frac{1}{6}N^3$$

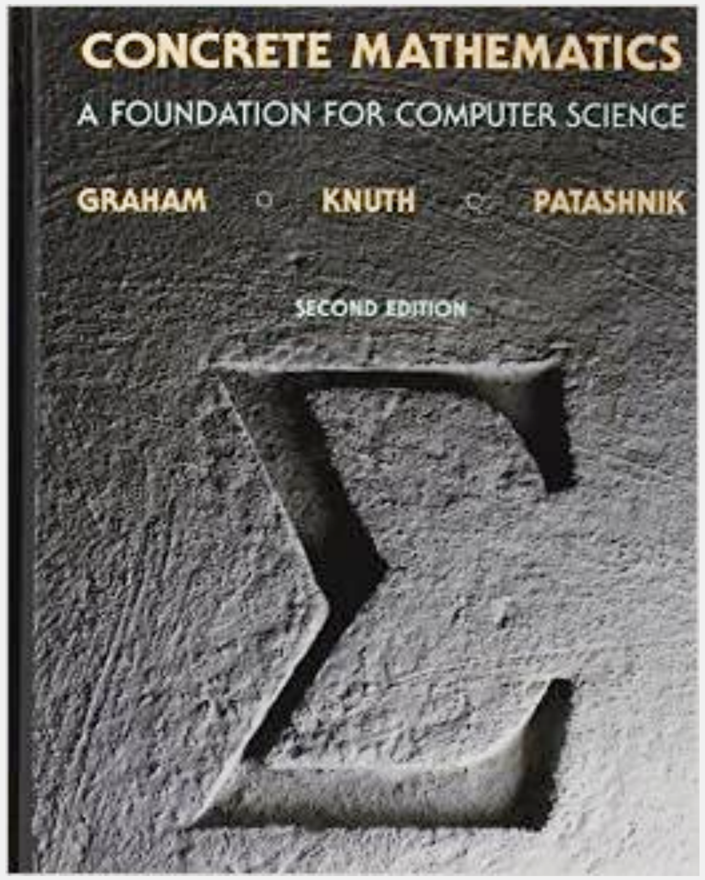
A. $\sim \frac{1}{2} N^3$ array accesses.

Bottom line. Use cost model and tilde notation to simplify counts.

Estimating a discrete sum

Q. How to estimate a discrete sum?

A1. Take a discrete mathematics course (COS 340).



Estimating a discrete sum

Q. How to estimate a discrete sum?

A2. Replace the sum with an integral, and use calculus!

Ex 1. $1 + 2 + \ldots + N.$

$$\sum_{i=1}^{N} i \;\sim\; \int_{x=1}^{N} x\,dx \;\sim\; \frac{1}{2}N^2$$

Ex 2. $1 + 1/2 + 1/3 + \ldots + 1/N.$

$$\sum_{i=1}^{N} \frac{1}{i} \;\sim\; \int_{x=1}^{N} \frac{1}{x}dx \;=\; \ln N$$

Ex 3. 3-sum triple loop.

$$\sum_{i=1}^{N}\sum_{j=i}^{N}\sum_{k=j}^{N} 1 \;\sim\; \int_{x=1}^{N}\int_{y=x}^{N}\int_{z=y}^{N} dz\,dy\,dx \;\sim\; \frac{1}{6}N^3$$

Ex 4. $1 + ½ + ¼ + ⅛ + \ldots$

$$\int_{x=0}^{\infty} \left(\frac{1}{2}\right)^x dx \;=\; \frac{1}{\ln 2} \;\approx\; 1.4427$$

$$\sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i \;=\; 2$$

integral trick doesn't always work!

Mathematical models for running time

In principle, accurate mathematical models are available.

In practice, formulas can be complicated.

costs (depend on machine, compiler)

$$T_N = c_1 A + c_2 B + c_3 C + c_4 D + c_5 E$$

$A =$ array access

$B =$ integer add

$C =$ integer compare

$D =$ increment

$E =$ variable assignment

frequencies
(depend on algorithm, input)

Bottom line. We use approximate models in this course: $T(N) \sim c\, N^3$.

Analysis of algorithms quiz 2

How many array accesses does the following code fragment make as a function of N ?

```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      for (int k = 1; k < N; k = k*2)
         if (a[i] + a[j] >= a[k])
            count++;
```

A. $\sim N^2 \lg N$

B. $\sim 3/2\ N^2 \lg N$

C. $\sim 1/2\ N^3$

D. $\sim 3/2\ N^3$

E. *I don't know.*

http://algs4.cs.princeton.edu

1.4 Analysis of Algorithms

- introduction
- observations
- mathematical models
- **order-of-growth classifications**
- memory

Common order-of-growth classifications

Definition. If $f(N) \sim c\, g(N)$ for some constant $c > 0$, then the order of growth of $f(N)$ is $g(N)$.

- Ignores leading coefficient.
- Ignores lower-order terms.

Ex. The order of growth of the running time of this code is N^3.

```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      for (int k = j+1; k < N; k++)
         if (a[i] + a[j] + a[k] == 0)
            count++;
```

Typical usage. Mathematical analysis of running times.

where leading coefficient depends on machine, compiler, JVM, ...

Commonly-used notations in the theory of algorithms

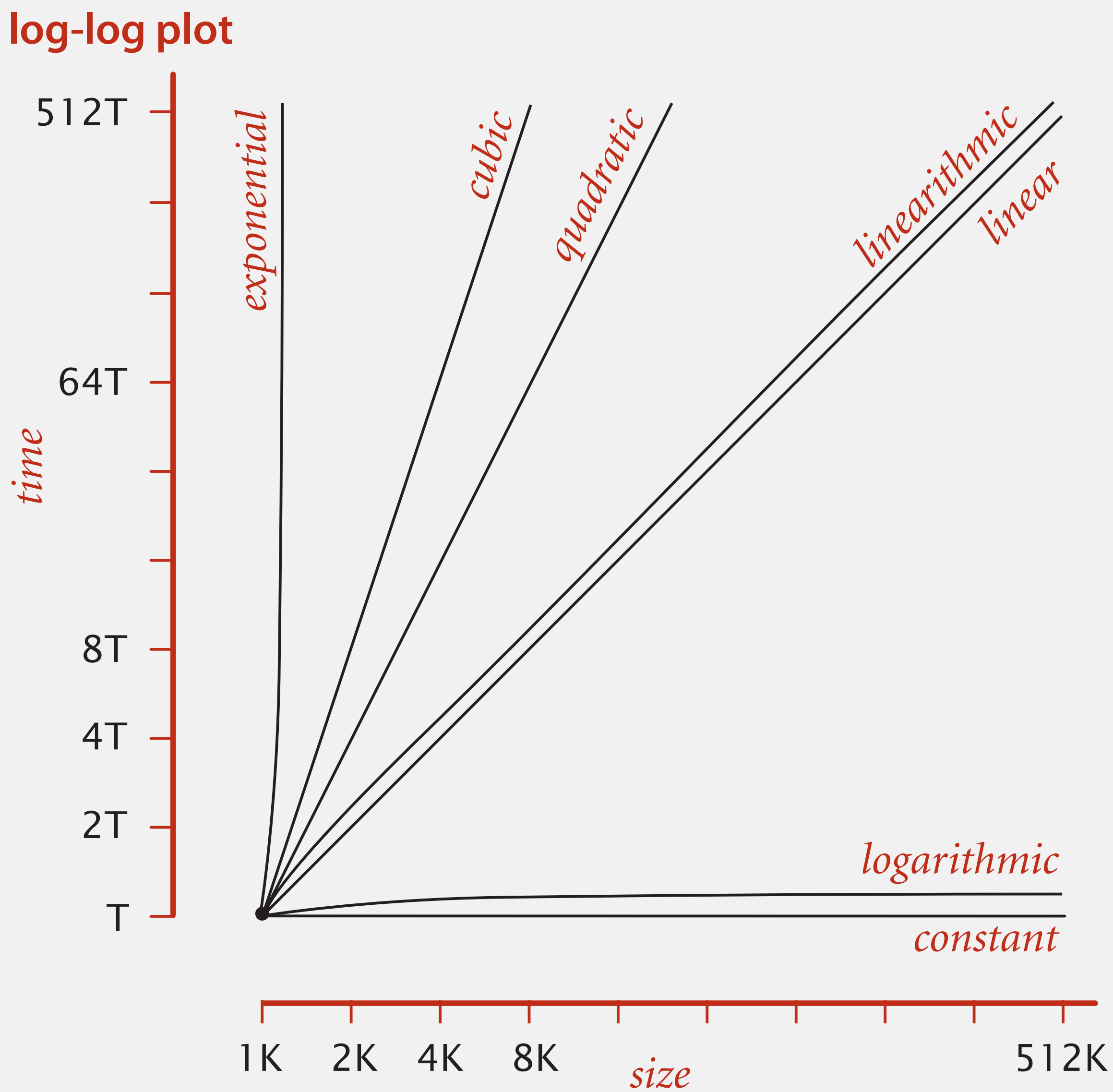
notation	provides	example	shorthand for	used to
Big Theta	asymptotic order of growth	$\Theta(N^2)$	$\frac{1}{2} N^2$ $10 N^2$ $5 N^2 + 22 N \log N + 3N$ $\vdots$	classify algorithms
Big O	$\Theta(N^2)$ and smaller	$O(N^2)$	$10 N^2$ $100 N$ $22 N \log N + 3 N$ $\vdots$	develop upper bounds
Big Omega	$\Theta(N^2)$ and larger	$\Omega(N^2)$	$\frac{1}{2} N^2$ N^5 $N^3 + 22 N \log N + 3 N$ $\vdots$	develop lower bounds

Common order-of-growth classifications

Good news. The set of functions

$$1,\ \log N,\ N,\ N\log N,\ N^2,\ N^3, \text{ and } 2^N$$

suffices to describe the order of growth of most common algorithms.



Typical orders of growth

Common order-of-growth classifications

order of growth	name	typical code framework	description	example	$T(2N) / T(N)$
1	**constant**	`a = b + c;`	statement	add two numbers	1
$\log N$	**logarithmic**	`while (N > 1)` `{  N = N/2;  ...  }`	divide in half	binary search	~ 1
N	**linear**	`for (int i = 0; i < N; i++)` `{  ...  }`	single loop	find the maximum	2
$N \log N$	**linearithmic**	*see mergesort lecture*	divide and conquer	mergesort	~ 2

N^2	**quadratic**	`for (int i = 0; i < N; i++)`
		` for (int j = 0; j < N; j++)`
		`  {  ...  }`
			double loop	check all pairs	4

N^3	**cubic**	`for (int i = 0; i < N; i++)`
		` for (int j = 0; j < N; j++)`
		`  for (int k = 0; k < N; k++)`
		`  {  ...  }`
			triple loop	check all triples	8

2^N	**exponential**	*see combinatorial search lecture*	exhaustive search	check all subsets	2^N

Binary search

Goal. Given a sorted array and a key, find index of the key in the array?

Binary search. Compare key against middle entry.

- Too small, go left.
- Too big, go right.
- Equal, found.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Binary search: implementation

Trivial to implement?

- First binary search published in 1946.
- First bug-free one in 1962.
- Bug in Java's `Arrays.binarySearch()` discovered in 2006.

JUN
2

Extra, Extra - Read All About It: Nearly All Binary Searches and Mergesorts are Broken

Posted by Joshua Bloch, Software Engineer

I remember vividly Jon Bentley's first Algorithms lecture at CMU, where he asked all of us incoming Ph.D. students to write a binary search, and then dissected one of our implementations in front of the class. Of course it was broken, as were most of our implementations. This made a real impression on me, as did the treatment of this material in his wonderful *Programming Pearls* (Addison-Wesley, 1986; Second Edition, 2000). The key lesson was to carefully consider the invariants in your programs.

http://googleresearch.blogspot.com/2006/06/extra-extra-read-all-about-it-nearly.html

Binary search: Java implementation

Invariant. If key appears in array a[], then a[lo] ≤ key ≤ a[hi].

```
public static int binarySearch(int[] a, int key)
{
   int lo = 0, hi = a.length - 1;
   while (lo <= hi)
   {
      int mid = lo + (hi - lo) / 2;
      if      (key < a[mid]) hi = mid - 1;
      else if (key > a[mid]) lo = mid + 1;
      else return mid;
   }
   return -1;
}
```

why not mid = (lo + hi) / 2 ?

one "3-way compare"

Binary search: mathematical analysis

Proposition. Binary search uses at most $1 + \lg N$ key compares to search in a sorted array of size N.

Def. $T(N)$ = # key compares to binary search a sorted subarray of size ≤ N.

Binary search recurrence. $T(N) \le T(N/2) + 1$ for $N > 1$, with $T(1) = 1$.

left or right half (floored division)

possible to implement with one 2-way compare (instead of 3-way)

Pf sketch. [assume N is a power of 2]

$$
\begin{aligned}
T(N) &\le T(N/2) + 1 && \text{[given]} \\
&\le T(N/4) + 1 + 1 && \text{[apply recurrence to first term]} \\
&\le T(N/8) + 1 + 1 + 1 && \text{[apply recurrence to first term]} \\
&\ \ \vdots \\
&\le T(N/N) + \underbrace{1 + 1 + \ldots + 1}_{\lg N} && \text{[stop applying, } T(1) = 1 \text{]} \\
&= 1 + \lg N
\end{aligned}
$$

http://algs4.cs.princeton.edu

1.4 Analysis of Algorithms

- introduction
- observations
- mathematical models
- order-of-growth classifications
- *memory*

Basics

Bit. 0 or 1.

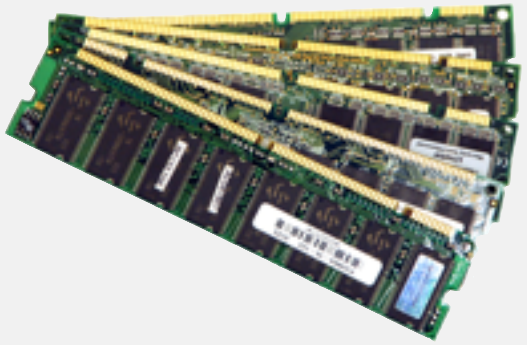
Byte. 8 bits.

NIST

most computer scientists

Megabyte (MB). 1 million or 2^{20} bytes.

Gigabyte (GB). 1 billion or 2^{30} bytes.



64-bit machine. We assume a 64-bit machine with 8-byte pointers.

some JVMs "compress" ordinary object pointers to 4 bytes to avoid this cost

Typical memory usage for primitive types and arrays

type	bytes
`boolean`	1
`byte`	1
`char`	2
`int`	4
`float`	4
`long`	8
`double`	8

primitive types

type	bytes
`char[]`	$2N + 24$
`int[]`	$4N + 24$
`double[]`	$8N + 24$

one-dimensional arrays

type	bytes
`char[][]`	$\sim 2\,M\,N$
`int[][]`	$\sim 4\,M\,N$
`double[][]`	$\sim 8\,M\,N$

two-dimensional arrays

Typical memory usage for objects in Java

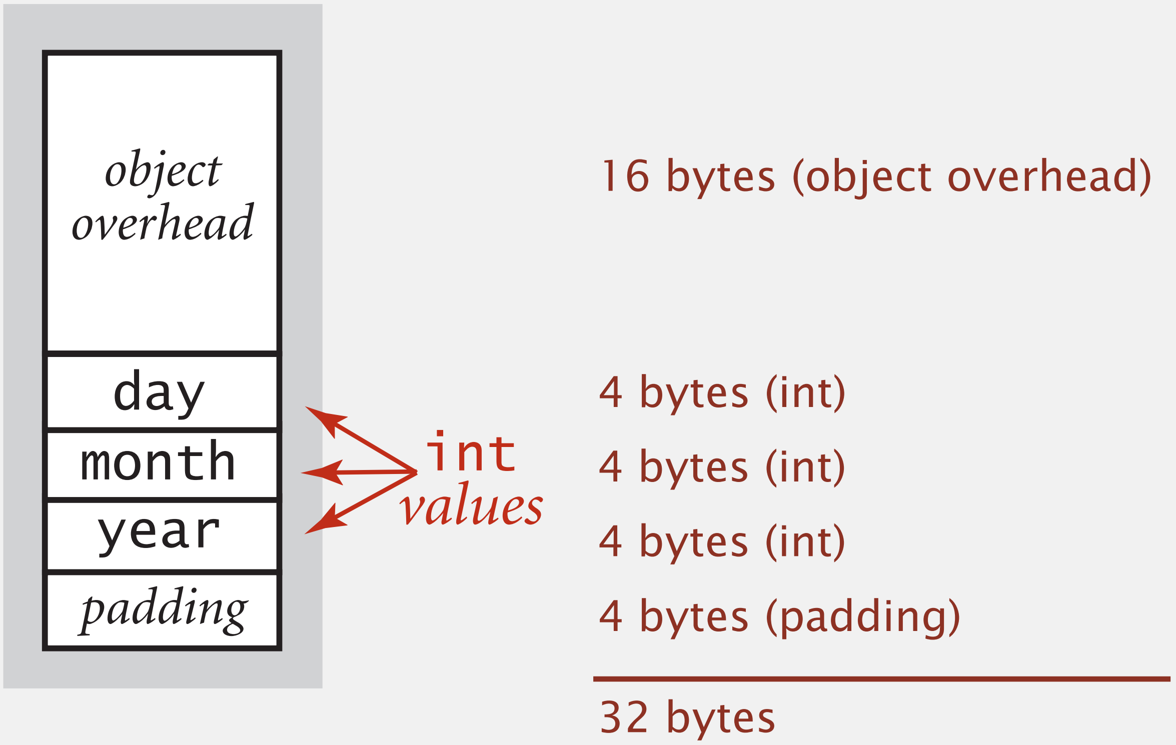
Object overhead. 16 bytes.

Reference. 8 bytes.

Padding. Each object uses a multiple of 8 bytes.

Ex 1. A `Date` object uses 32 bytes of memory.

```
public class Date
{
   private int day;
   private int month;
   private int year;
...
}
```



Typical memory usage summary

Total memory usage for a data type value:

- Primitive type: 4 bytes for `int`, 8 bytes for `double`, ...
- Object reference: 8 bytes.
- Array: 24 bytes + memory for each array entry.
- Object: 16 bytes + memory for each instance variable.
- Padding: round up to multiple of 8 bytes.

+ 8 extra bytes per inner class object
(for reference to enclosing class)

Note. Depending on application, we may want to count memory for any referenced objects (recursively).

Analysis of algorithms quiz 3

How much memory does a `WeightedQuickUnionUF` use as a function of N ?

A. $\sim 4\,N$ *bytes*

B. $\sim 8\,N$ *bytes*

C. $\sim 4\,N^2$ *bytes*

D. $\sim 8\,N^2$ *bytes*

E. *I don't know.*

```
public class WeightedQuickUnionUF
{
   private int[] parent;
   private int[] size;
   private int count;

   public WeightedQuickUnionUF(int N)
   {
      parent = new int[N];
      size = new int[N];
      count = 0;
      for (int i = 0; i < N; i++)
         parent[i] = i;
      for (int i = 0; i < N; i++)
         size[i] = 1;
   }
   ...
}
```

Technical Interview Questions

Apple Computer

facebook

Cisco Systems

Google

IBM

Nintendo

Jane Street

Morgan Stanley

NETFLIX

Adobe

RSA Security

D E Shaw & Co

ORACLE

PANDORA

Akamai

YAHOO!

amazon.com

Microsoft

PIXAR Animation Studios

The 3-sum problem

3-Sum. Given N distinct integers, find three such that $a + b + c = 0$.

Version 0. N^3 time, N space.

Version 1. $N^2 \log N$ time, N space.

Version 2. N^2 time, N space.

Note. For full credit, running time should be worst case.

*Fastest known algorithm (published in 2014): $N^2 / (\log N / \log \log N)^{2/3}$ time

THE 3-SUM PROBLEM: AN $N^2 \log N$ ALGORITHM

Algorithm.

- Step 1: Sort the N (distinct) numbers.
- Step 2: For each pair of numbers `a[i]` and `a[j]`, binary search for `-(a[i] + a[j])`.

Analysis. Order of growth is $N^2 \log N$.

- Step 1: N^2 with insertion sort
  (or $N \log N$ with mergesort).
- Step 2: $N^2 \log N$ with binary search.

input

```
30 -40 -20 -10 40 0 10 5
```

sort

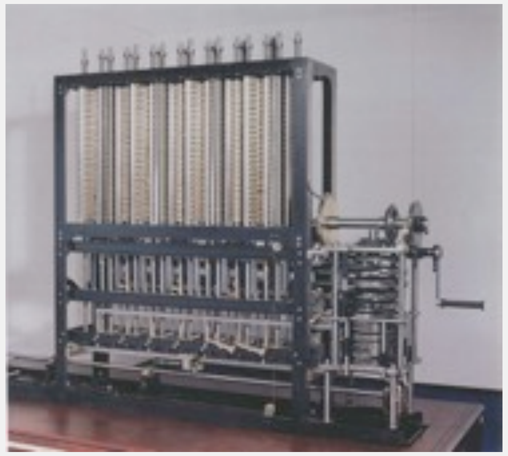
```
-40 -20 -10 0 5 10 30 40
```

binary search

(-40, -20)	60
(-40, -10)	50
(-40, 0)	40
(-40, 5)	35
(-40, 10)	30
⋮	⋮
(-20, -10)	30
⋮	⋮
(-10, 0)	10
⋮	⋮
(10, 30)	~~-40~~
(10, 40)	-50
(30, 40)	-70

only count if a[i] < a[j] < a[k] to avoid double counting

Turning the crank: summary



Empirical analysis.

- Execute program to perform experiments.
- Assume power law.
- Formulate a hypothesis for running time.
- Model enables us to make predictions.

Mathematical analysis.

- Analyze algorithm to count frequency of operations.
- Use tilde notation to simplify analysis.
- Model enables us to explain behavior.

$$\sum_{h=0}^{\lfloor \lg N \rfloor} \lceil N/2^{h+1} \rceil \, h \sim N$$

Scientific method.

- Mathematical model is independent of a particular system; applies to machines not yet built.
- Empirical analysis is necessary to validate mathematical models and to make predictions.