Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

Algorithms
FOURTH EDITION
ROBERT SEDGEWICK | KEVIN WAYNE

https://algs4.cs.princeton.edu

1.4 ANALYSIS OF ALGORITHMS

- *introduction*
- *running time (experimental analysis)*
- *running time (mathematical models)*
- *memory usage*

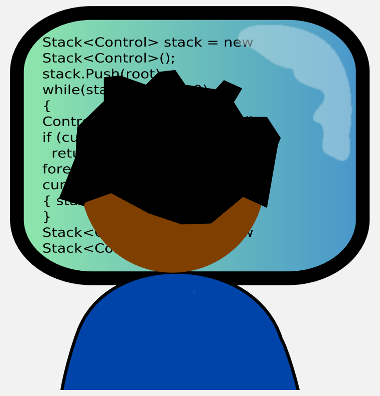
Last updated on 2/2/23 8:42 AM

https://algs4.cs.princeton.edu

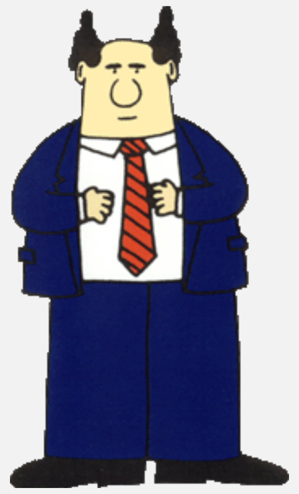
1.4 ANALYSIS OF ALGORITHMS

▸ *introduction*
▸ *running time (experimental analysis)*
▸ *running time (mathematical models)*
▸ *memory usage*

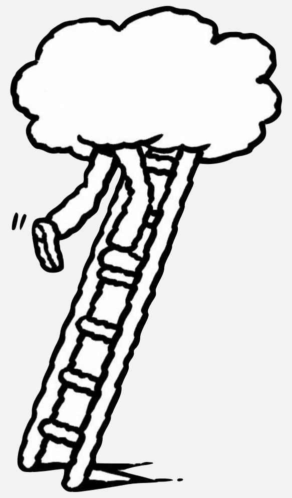
Different viewpoints



programmer needs to
develop a working solution



client wants to solve
problem efficiently

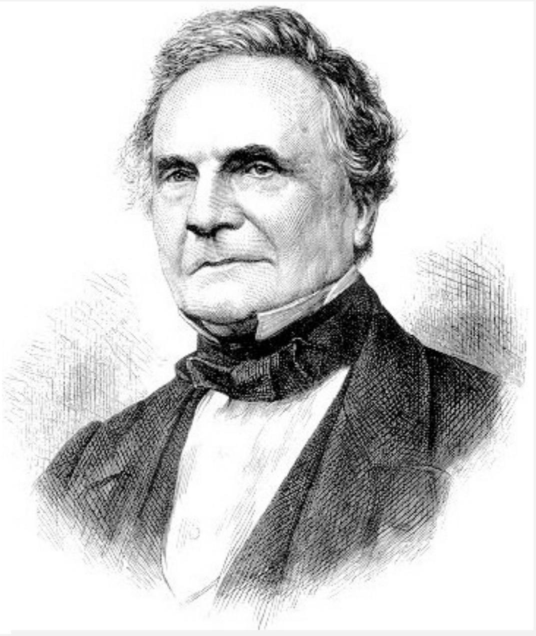


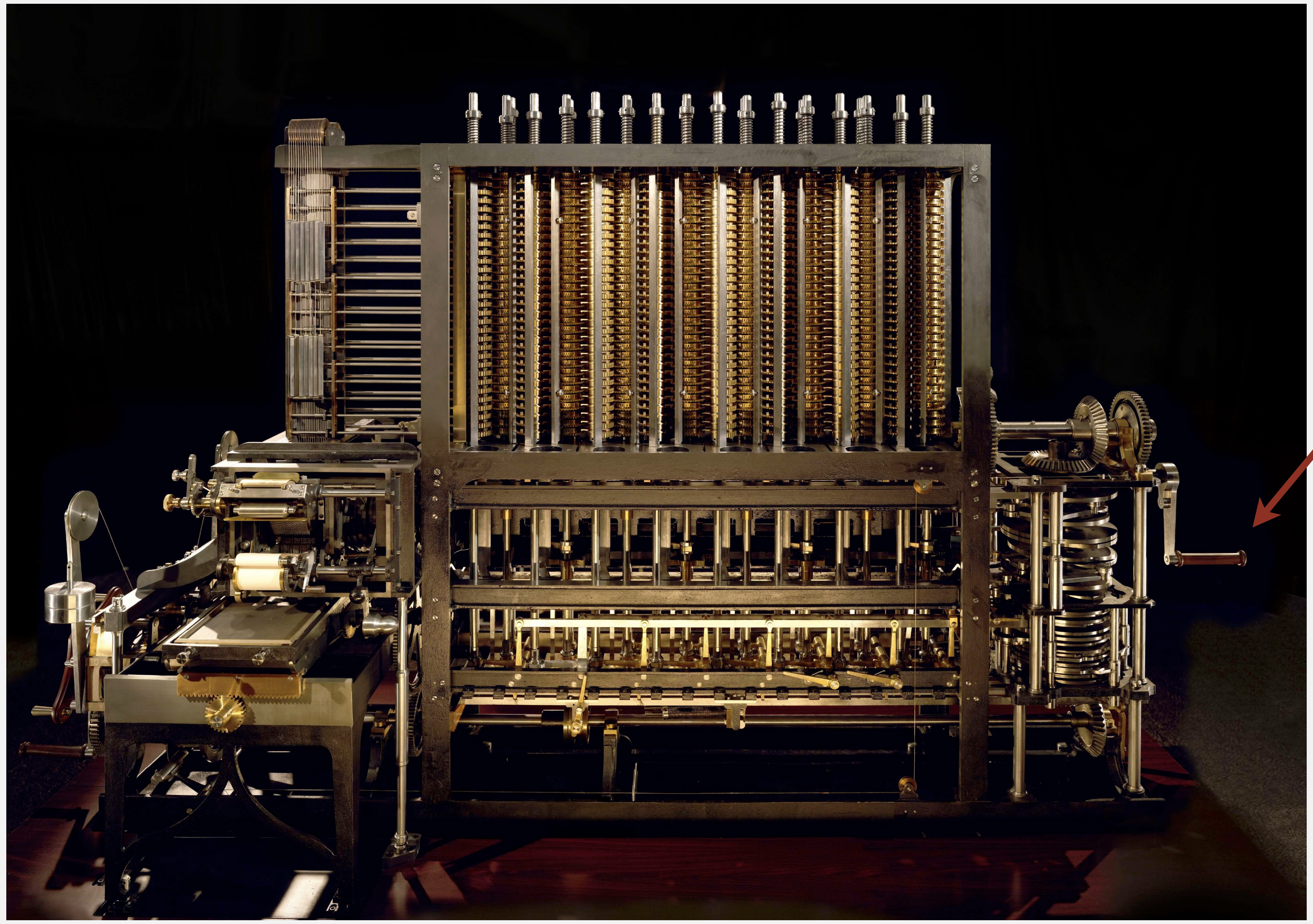
theoretician seeks
to understand

student (you)
will play all of
these roles in this course

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)*

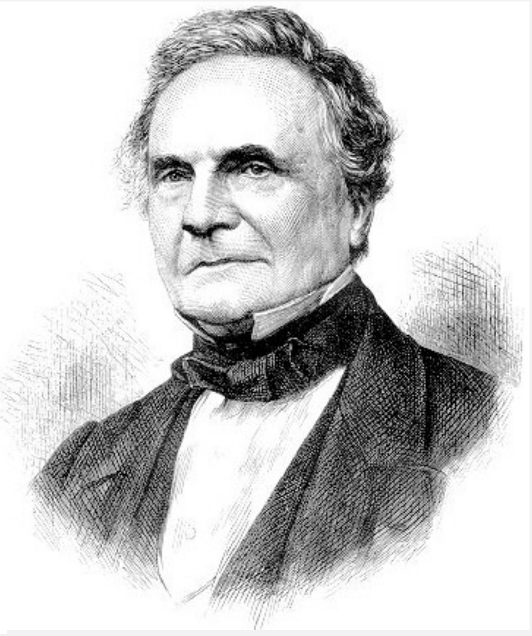




how many times
do you have to turn
the crank?

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)*



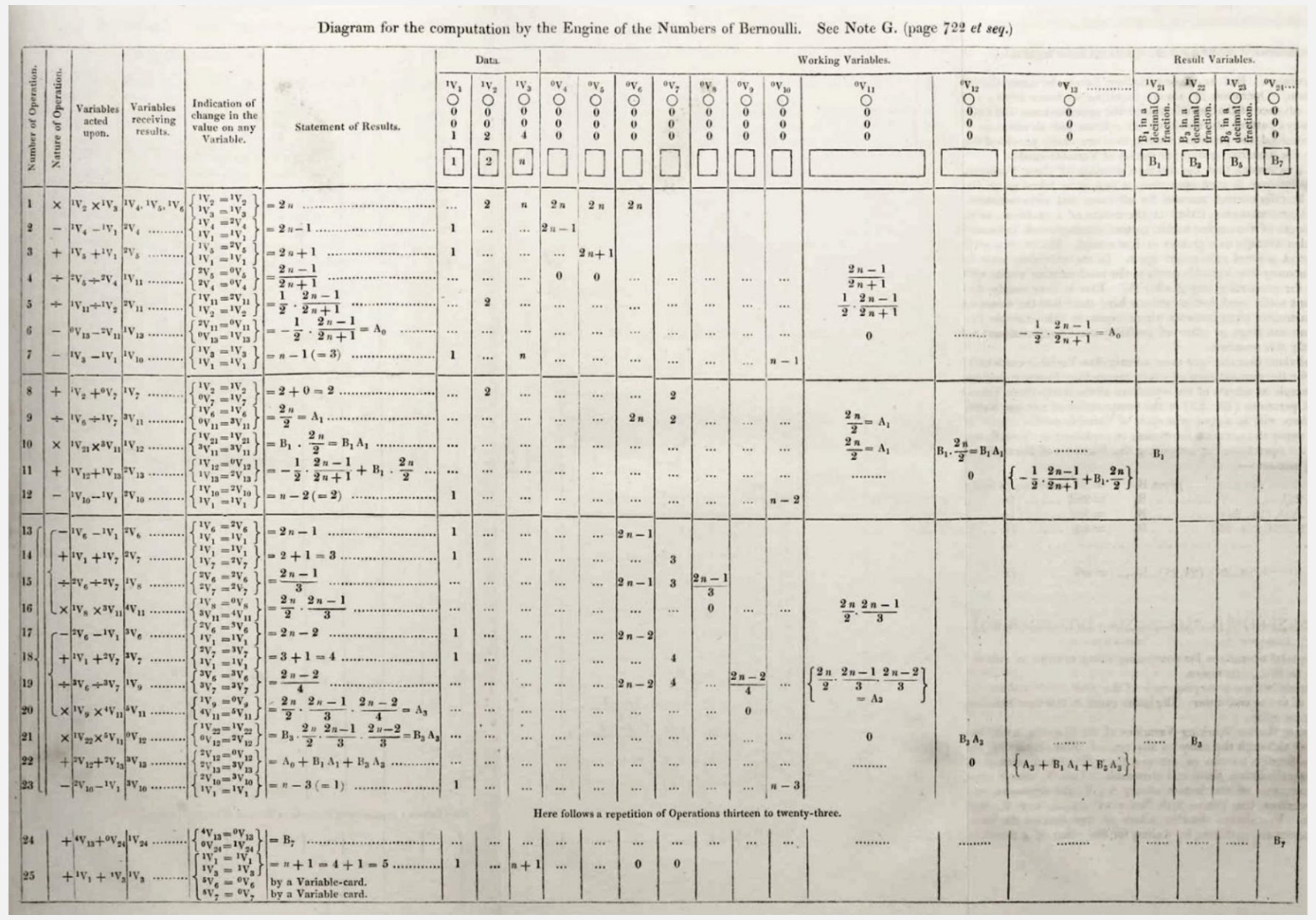


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

Ada Lovelace’s algorithm to compute Bernoulli numbers on Analytic Engine (1843)

Rare book containing the world’s first computer algorithm earns $125,000 at auction

By Matt Kennedy
July 25, 2018

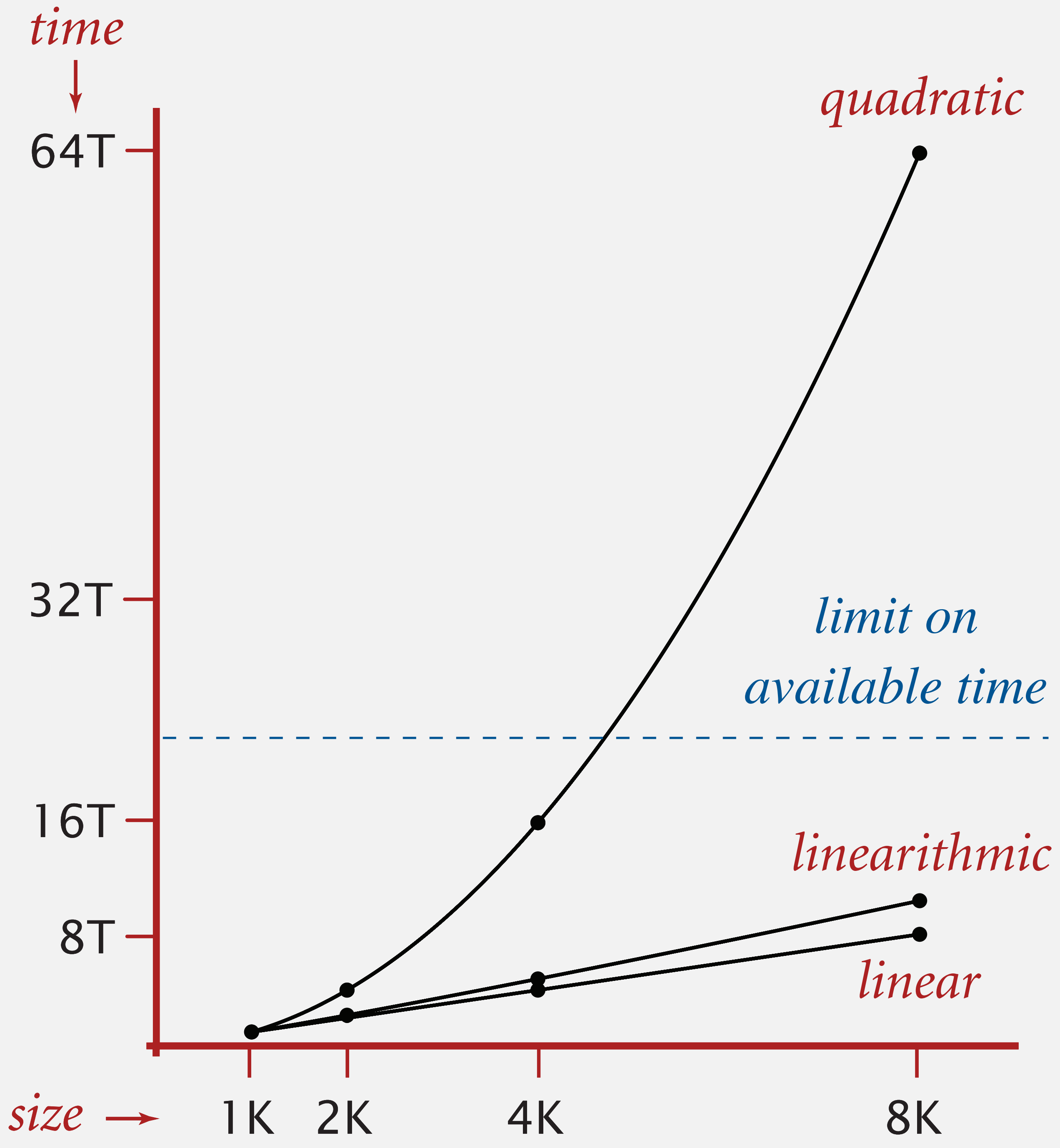
SKETCH
OF THE
ANALYTICAL ENGINE
INVENTED BY
CHARLES BABBAGE, Esq.
By L. F. MENABREA,
of Turin,
OFFICER OF THE MILITARY ENGINEERS.
WITH NOTES BY THE TRANSLATOR.
Lady Lovelace
[Extracted from the ‘Scientific Memoirs,’ vol. iii.]
LONDON:
PRINTED BY RICHARD AND JOHN E. TAYLOR,
RED LION COURT, FLEET STREET.
1843.

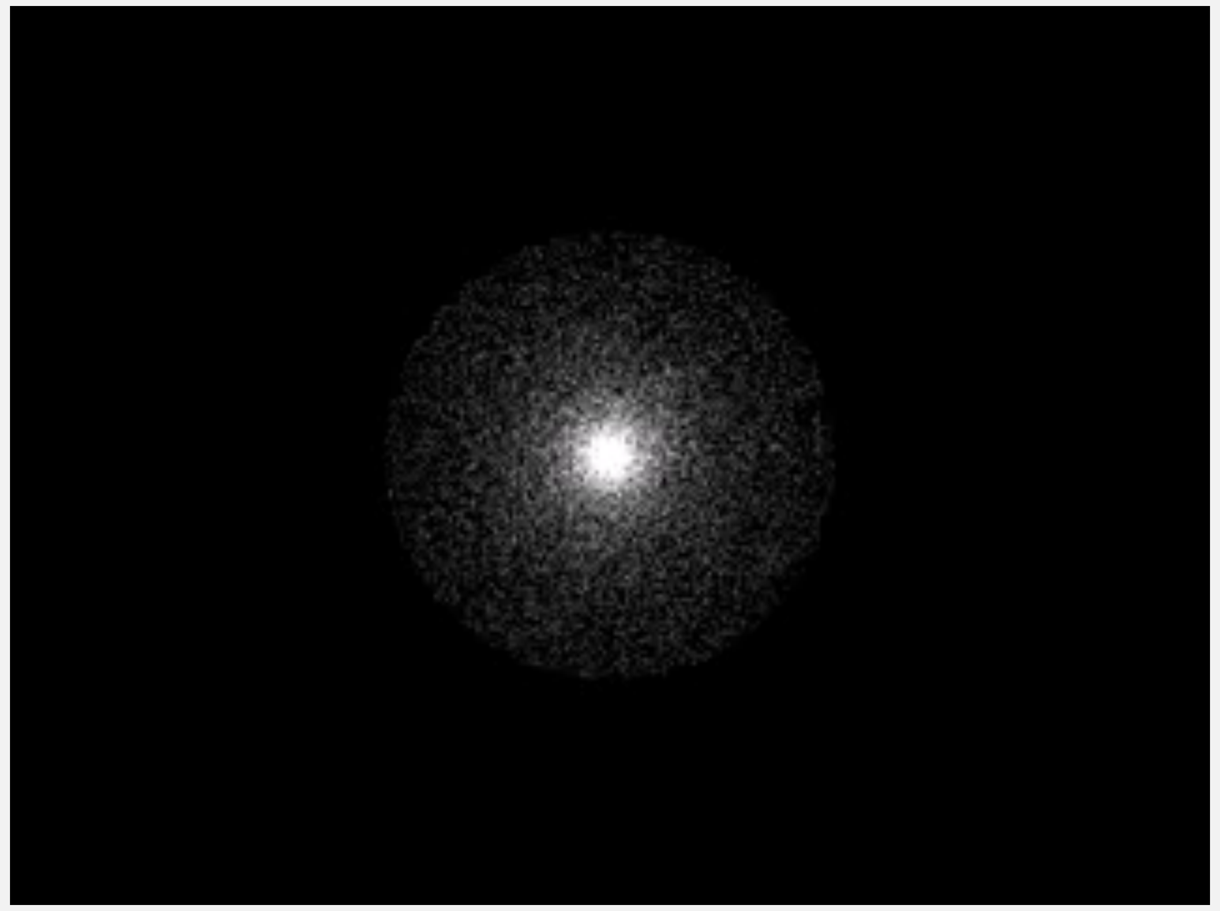
An algorithmic success story

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.

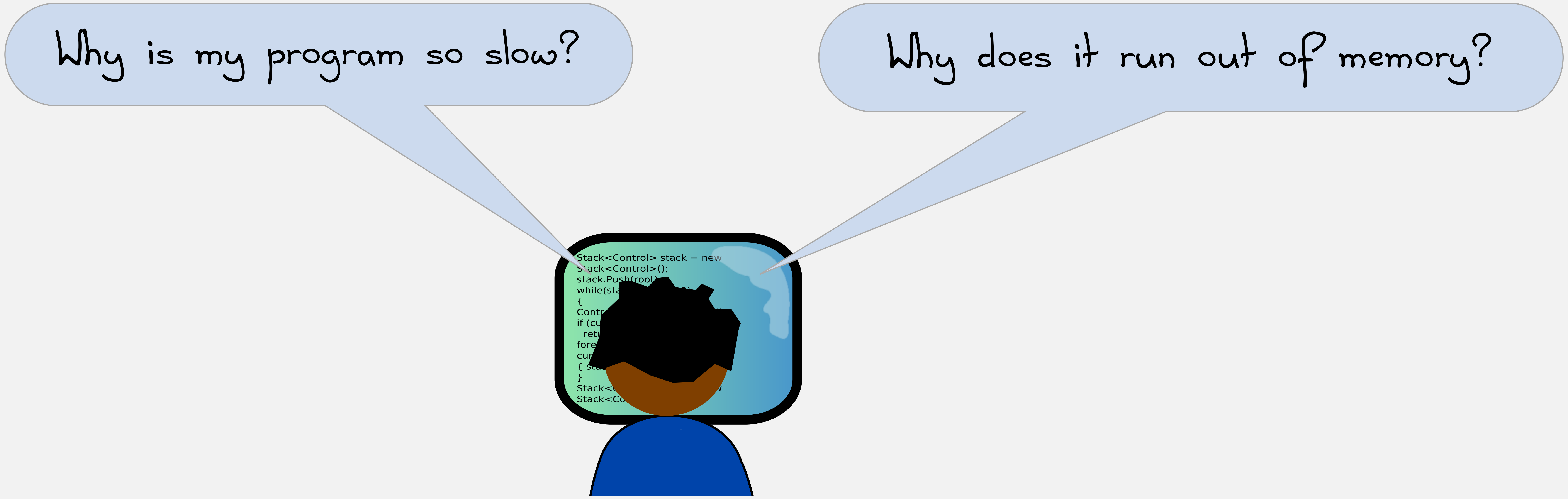
Andrew Appel
PU '81





The challenge

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



Our approach. Combination of experiments and mathematical modeling.

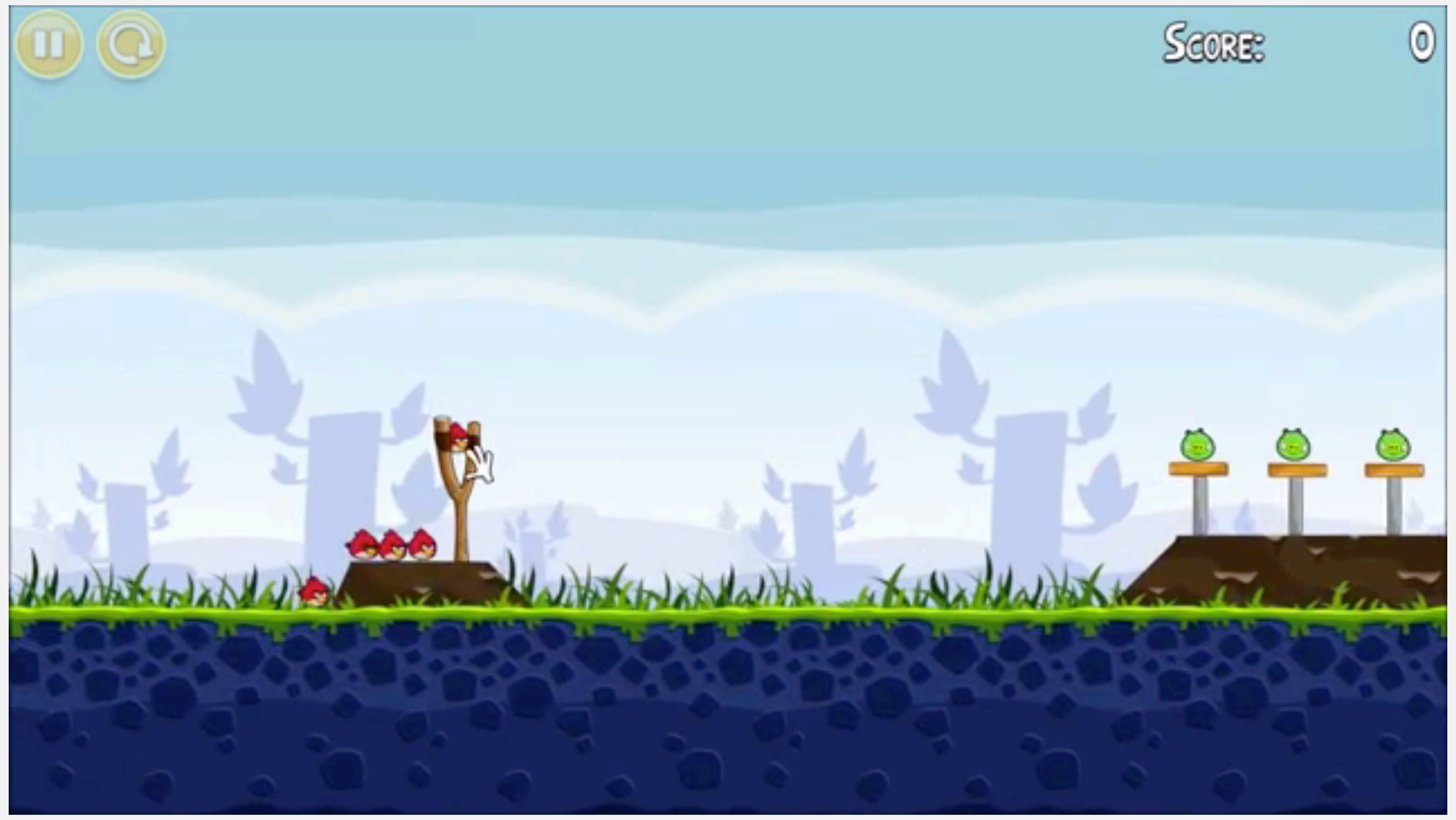
Example: 3-SUM

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

```
~/Desktop/3sum> more 8ints.txt
8
30 -40 -20 -10 40 0 10 5

~/Desktop/3sum> 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	✔

Context. Connected with problems in computational geometry.



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 distinct triples

for simplicity,
ignore integer overflow

Algorithms

Robert Sedgewick | Kevin Wayne

https://algs4.cs.princeton.edu

1.4 ANALYSIS OF ALGORITHMS

- ▸ *introduction*
- ▸ *running time (experimental analysis)*
- ▸ *running time (mathematical models)*
- ▸ *memory usage*

Empirical analysis

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

```
%
```

Empirical analysis

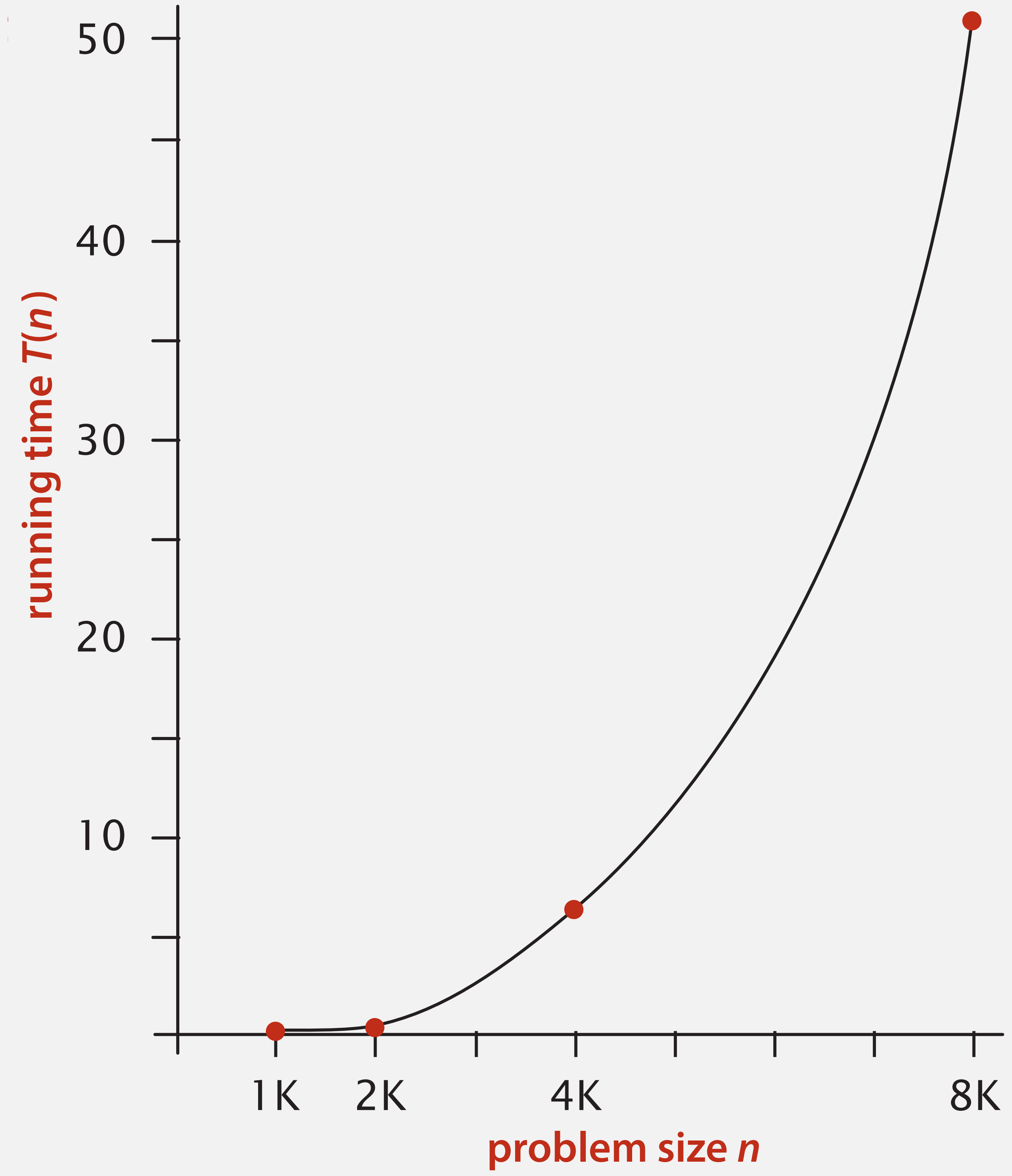
Run the program for various input sizes and measure running time.

n	time (seconds) †
250	0
500	0
1,000	0.1
2,000	0.8
4,000	6.4
8,000	51.1
16,000	?

† 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.

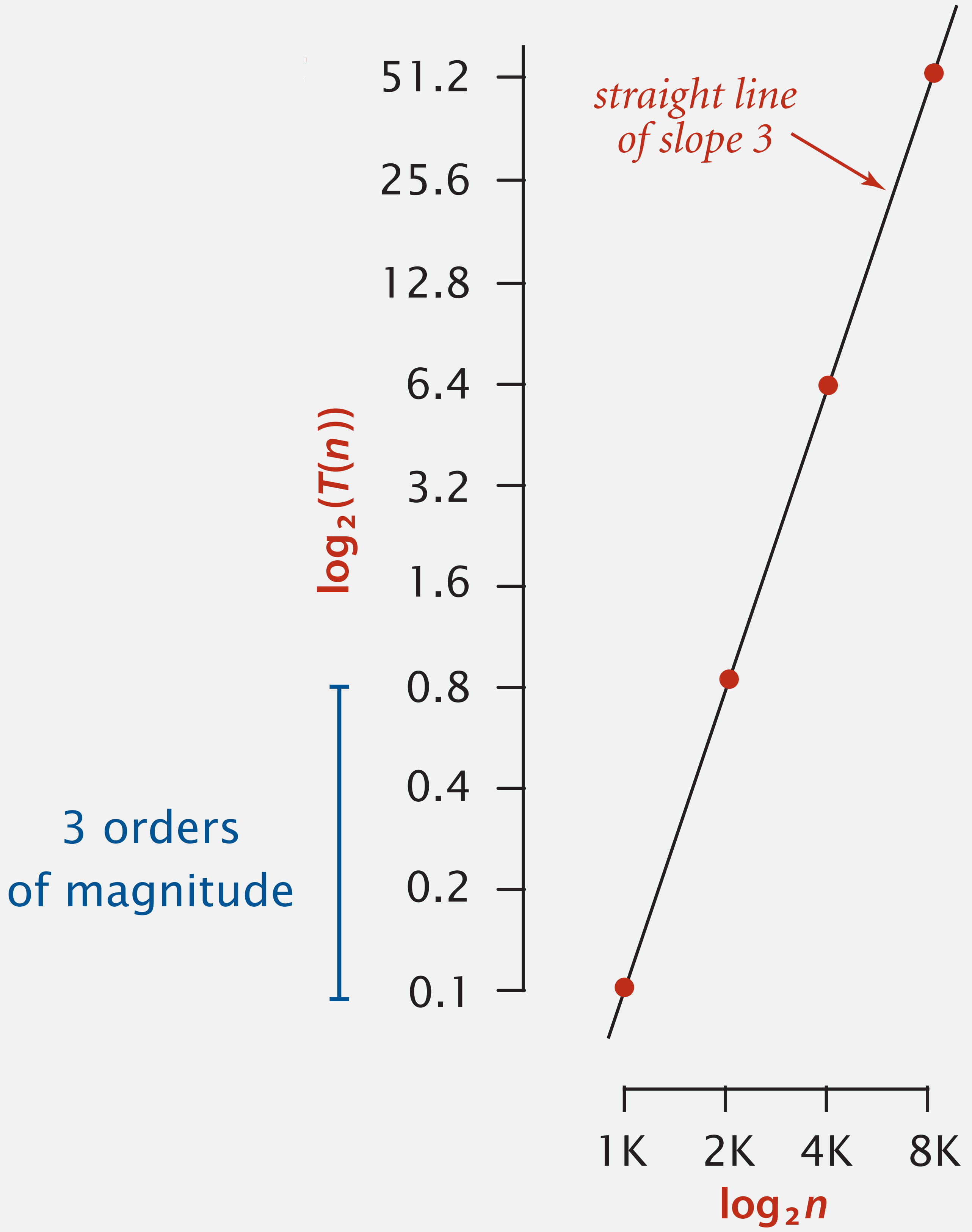


Hypothesis (power law). $T(n) = a\, n^b$.

Questions. How to validate hypothesis? How to estimate a and b ?

Data analysis

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



slope

$$\log_2(T(n)) = 2.999 \log_2 n + (-33.21)$$

$$T(n) = 2^{-33.21} \times n^{2.999}$$
$$= 1.006 \times 10^{-10} \times n^{2.999}$$

"order of growth" of running time is about n^3 [stay tuned]

Regression. Fit straight line through data points.

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

Doubling hypothesis

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

Run program, doubling the size of the input.

n	time (seconds) †	ratio	$\log_2$ ratio
250	0		–
500	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	3.0
8,000	51.1	8	3.0

$\log_2 (6.4 / 0.8) = 3.0$

seems to converge to a constant $b \approx 3$

$$\frac{T(n)}{T(n/2)} = \frac{an^b}{a(n/2)^b} = 2^b$$

$$\Longrightarrow \quad b = \log_2 \frac{T(n)}{T(n/2)}$$

Hypothesis. Running time is about $T(n) = a\,n^b$, with $b = \log_2$ ratio.

Doubling hypothesis

Doubling hypothesis. Quick way to estimate exponent 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.

$\uparrow$

almost identical hypothesis
to one obtained via regression
(but less work)

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*

n	time (seconds)
1,000	0.02
2,000	0.05
4,000	0.20
8,000	0.81
16,000	3.25
32,000	13.01

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 $a\,n^b$

Java 8
ORACLE

ubuntu
14.04 LTS

Bad news. Sometimes difficult to get accurate measurements.

Context: the scientific method

Experimental algorithmics is an example of the scientific method.

Chemistry
(1 experiment)

Biology
(1 experiment)

"Tsar Bomba" - Novaya Zemlya archipelago, USSR: Oct. 30, 1961

Physics
(1 experiment)

Computer Science
(1 million experiments)

Good news. Experiments are easier and cheaper than other sciences.

Algorithms

Robert Sedgewick | Kevin Wayne

https://algs4.cs.princeton.edu

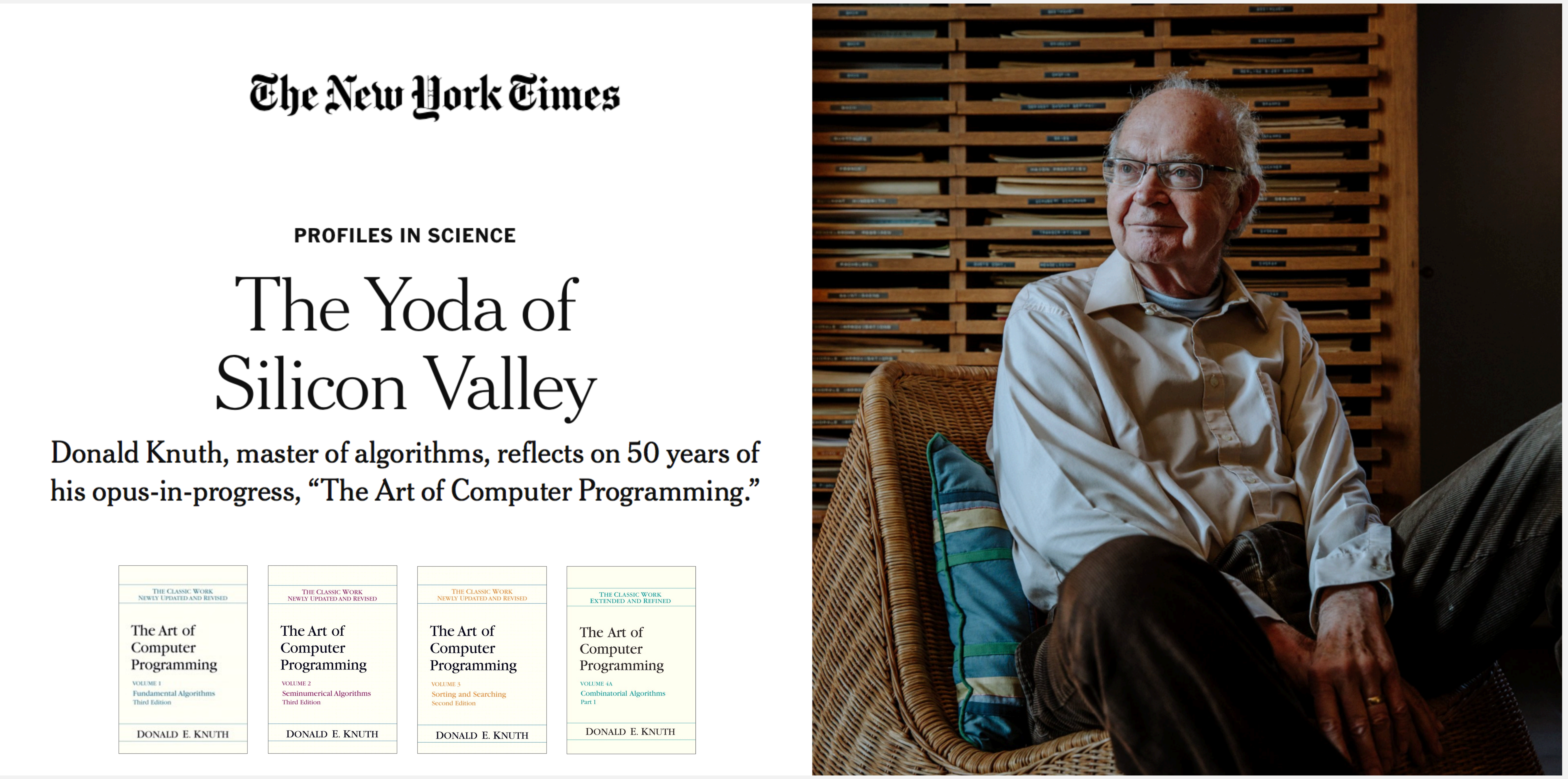
1.4 ANALYSIS OF ALGORITHMS

- ▸ *introduction*
- ▸ *running time (experimental analysis)*
- ▸ *running time (mathematical models)*
- ▸ *memory usage*

Mathematical models for running time

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

- Need to analyze program to determine set of operations.
- Frequency depends on algorithm and input data.
- Cost depends on CPU, compiler, operating system, ...



The New York Times

PROFILES IN SCIENCE

The Yoda of Silicon Valley

Donald Knuth, master of algorithms, reflects on 50 years of his opus-in-progress, “The Art of Computer Programming.”

The Art of Computer Programming
DONALD E. KNUTH

The Art of Computer Programming
DONALD E. KNUTH

The Art of Computer Programming
DONALD E. KNUTH

The Art of Computer Programming
DONALD E. KNUTH

Warning. No general-purpose method (e.g., halting problem).

Example: 1-SUM

Q. How many operations as a function of input size n ?

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

exactly 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 $2\,n$

in practice, depends on caching, bounds checking, …
(see COS 217)

† representative estimates (with some poetic license)

Analysis of algorithms: quiz 2

How many array accesses as a function of 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++;
```

A. $\frac{1}{2} n (n - 1)$

B. $n (n - 1)$

C. $2 n^2$

D. $2 n (n - 1)$

Example: 2-SUM

Q. How many operations 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)

Simplification 1: cost model

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

array accesses, compares, floating-point operations, disk accesses, API calls, ...

```
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++;
```

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**	$\boxed{n (n - 1)}$
increment	1/10	$\frac{1}{2} n (n + 1)$ to n^2

← cost model = array accesses

(we're assuming compiler/JVM does not optimize any array accesses away!)

Simplification 2: asymptotic notations

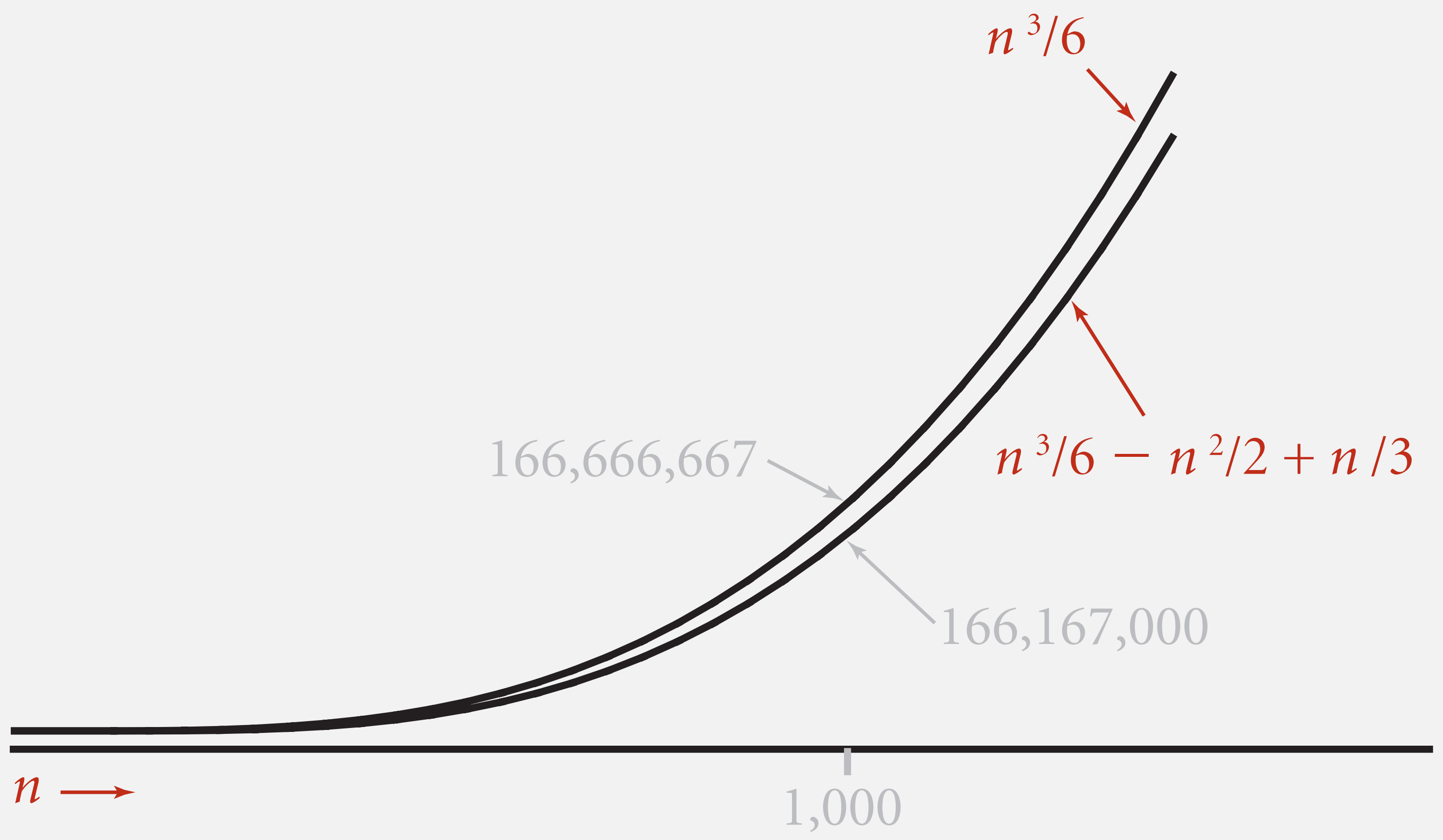
Tilde notation. Discard lower-order terms.

Big Theta notation. Also discard leading coefficient.

formal definitions involve limits

function	tilde	big Theta
$4n^5 + 20n + 16$	$\sim 4n^5$	$\Theta(n^5)$
$7n^2 + 100n^{4/3} + 56$	$\sim 7n^2$	$\Theta(n^2)$
$\frac{1}{6}n^3 \underbrace{- \frac{1}{2}n^2 + \frac{1}{3}n}$	$\sim \frac{1}{6}n^3$	$\Theta(n^3)$

discard lower-order terms
(e.g., n = 1,000: 166.67 million vs. 166.17 million)



Leading-term approximation

Rationale.

- When n is large, lower-order terms are negligible.
- When n is small, we don't care.

Common order-of-growth classifications

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

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

$\Theta(n^3)$	☹️	**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

$\Theta(2^n)$	😈	**exponential**	*see combinatorial search lecture*	exhaustive search	*check all subsets*	2^n

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.

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"

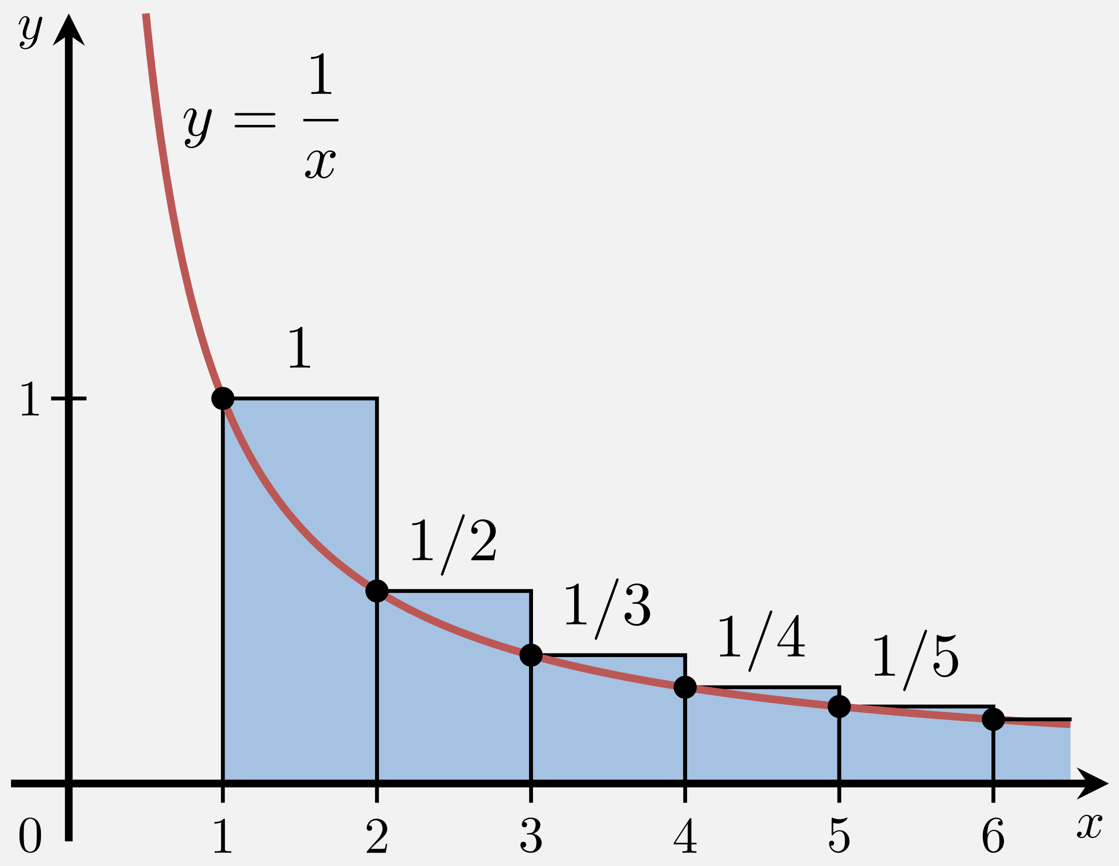
A. $\sim \frac{1}{2} n^3$ array accesses.

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

see COS 240

Bottom line. Use cost model and asymptotic notation to simplify analysis.

Some useful discrete sums and approximations



Triangular sum. $1 + 2 + 3 + \ldots + n \sim \frac{1}{2} n^2$

Harmonic sum. $1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} \sim \int_{x=1}^{n} \frac{1}{x}\, dx = \ln n$

Geometric sum. $1 + 2 + 4 + 8 + \ldots + n = 2n - 1$

↑

n a power of 2

Analysis of algorithms: quiz 3

How many array accesses 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. $\sim n^2 \log_2 n$

B. $\sim 3/2 \;\; n^2 \log_2 n$

C. $\sim 1/2 \, n^3$

D. $\sim 3/2 \, n^3$

Analysis of algorithms: quiz 4

What is order of growth of running time as a function of n ?

```
int count = 0;
for (int i = n; i >= 1; i = i/2)
   for (int j = 1; j <= i; j++)
      count++;
```

“inner loop”

A. $\Theta(n)$

B. $\Theta(n \log n)$

C. $\Theta(n^2)$

D. $\Theta(2^n)$

Algorithms

Robert Sedgewick | Kevin Wayne

https://algs4.cs.princeton.edu

1.4 ANALYSIS OF ALGORITHMS

- *introduction*
- *running time (experimental analysis)*
- *running time (mathematical models)*
- *memory usage*

Basics

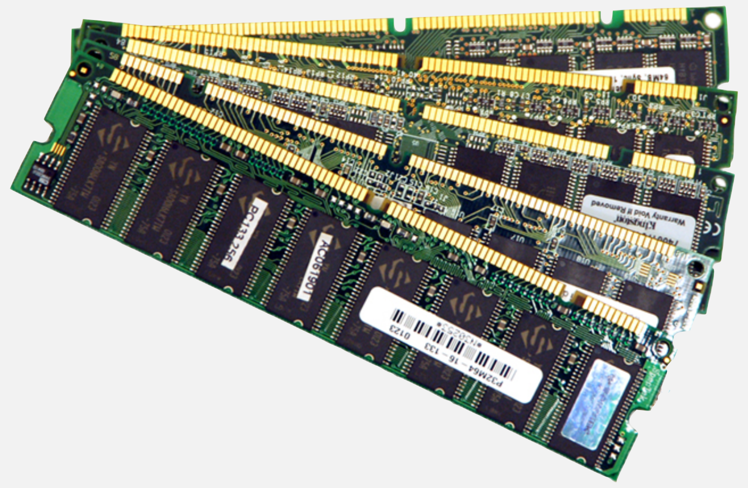
Bit. 0 or 1.

NIST ↓ most computer scientists ↓

Byte. 8 bits.

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
`boolean[]`	$\boxed{1n} + 24$
`int[]`	$4n + 24$
`double[]`	$8n + \boxed{24}$

wasteful
(but $\sim 36n$ in Python 3)

array overhead = 24 bytes

one-dimensional arrays (length n)

type	bytes
`boolean[][]`	$\sim 1\ n^2$
`int[][]`	$\sim 4\ n^2$
`double[][]`	$\sim 8\ n^2$

two-dimensional arrays (n-by-n)

Typical memory usage for objects in Java

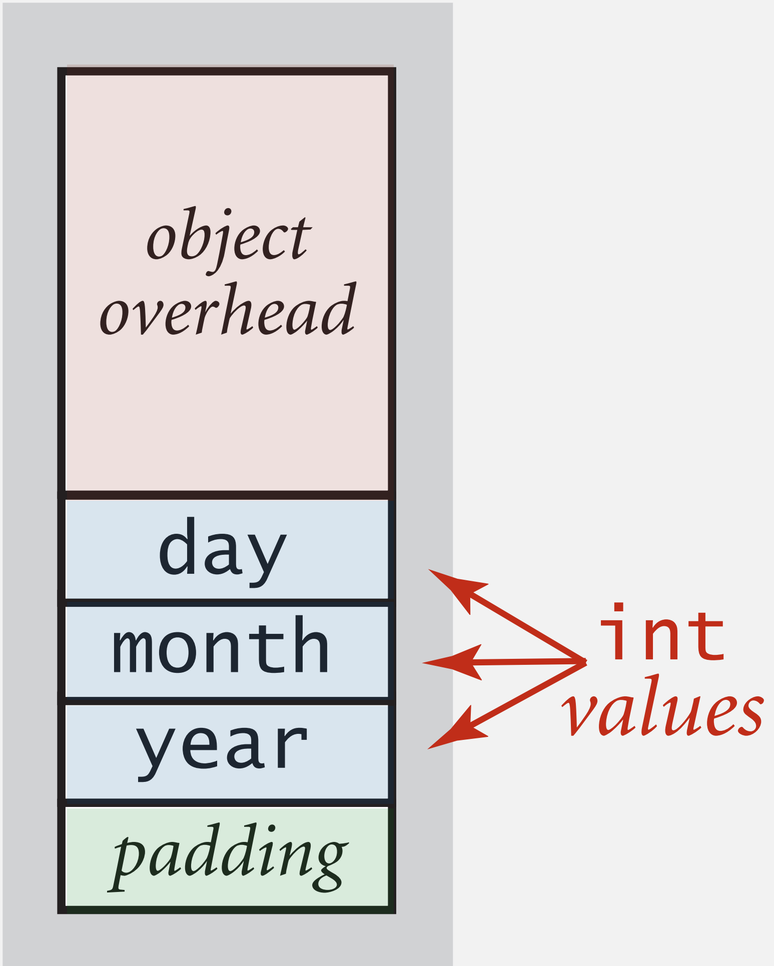
Object overhead. 16 bytes.

Reference. 8 bytes.

Padding. Round up memory of each object to be a multiple of 8 bytes.

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

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



16 bytes (object overhead)

4 bytes (int)

4 bytes (int)

4 bytes (int)

4 bytes (padding)

32 bytes

Analysis of algorithms: quiz 5

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

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

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

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

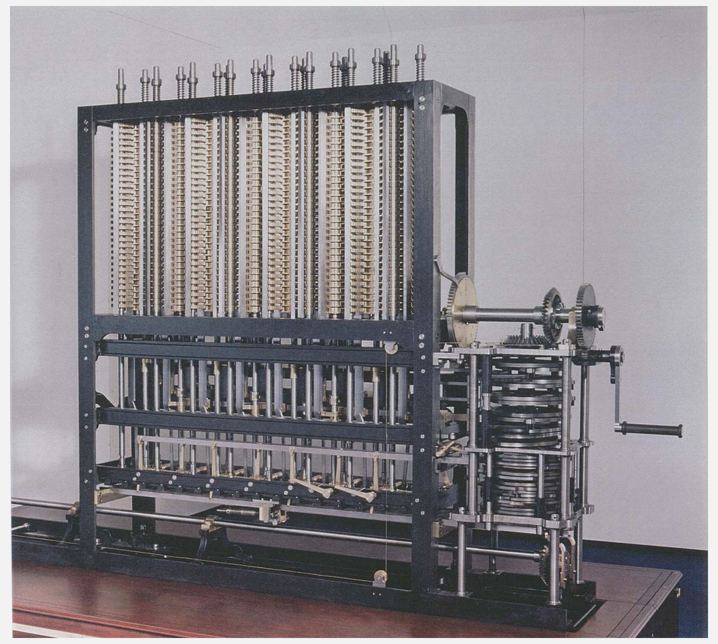
D. $\sim 8\,n^2$ *bytes*

```
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;
   }
   ...
}
```

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 and big-Theta notations to simplify analysis.
- Model enables us to explain behavior.

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

This course. Learn to use both.

> “ *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.”* — *Alan Turing (1947)*