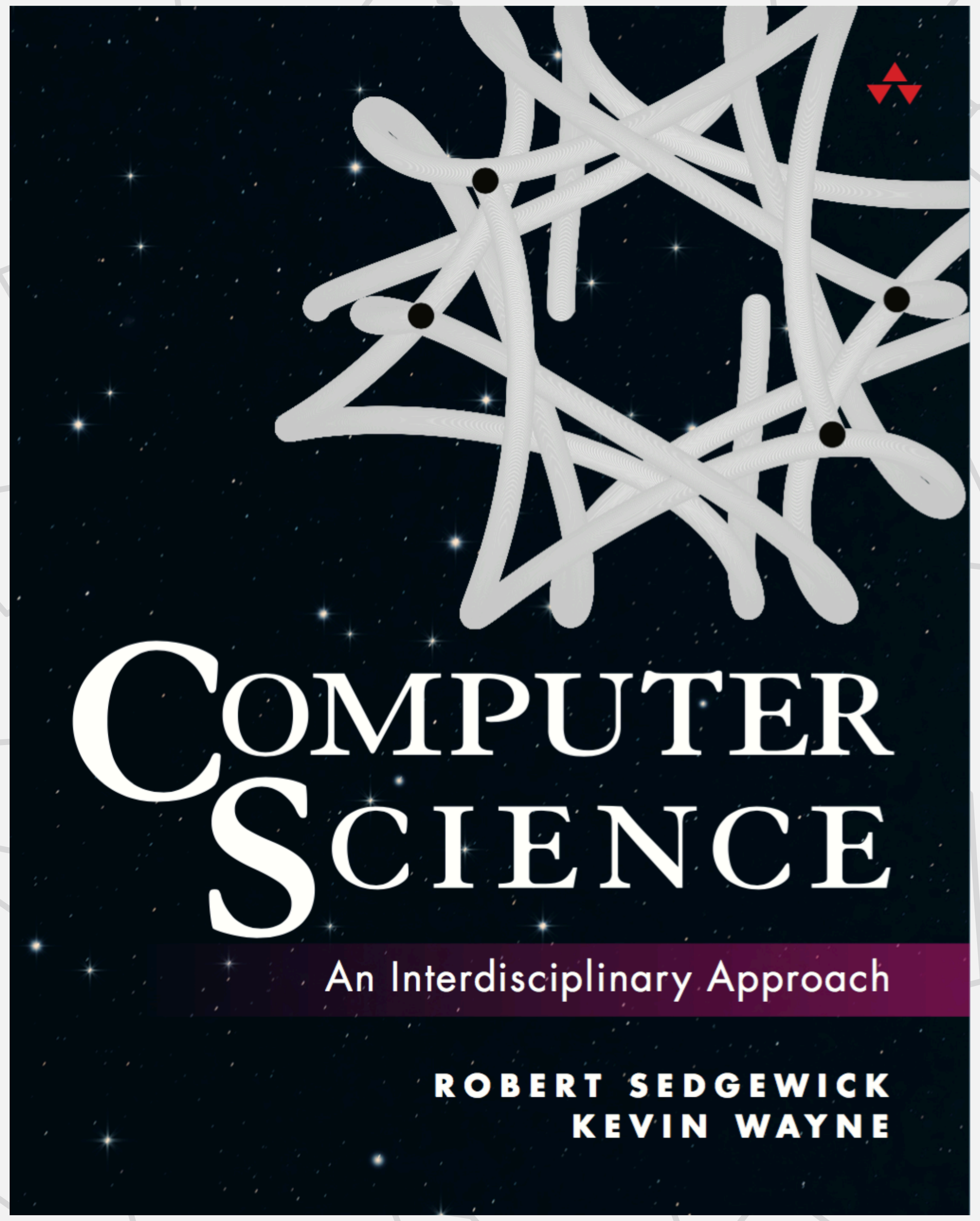


http://introcs.cs.princeton.edu

4.1 Performance

- *empirical analysis*
- *mathematical analysis*

Performance

Goal. Estimate running time (or memory) as a function of input size n.

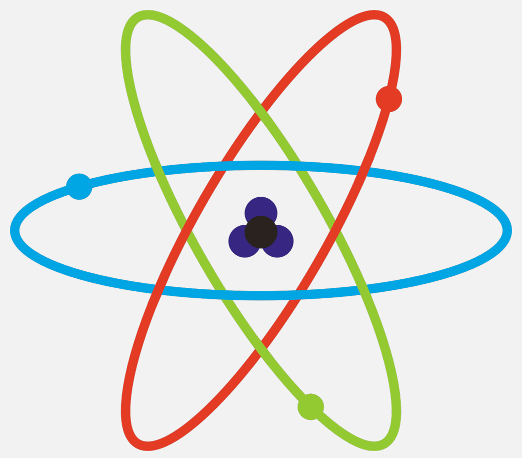
Q. Aren't computers fast enough that it doesn't matter.

A. No. Program could take 1 second or 50 years. You need to know which.

Empirical vs. mathematical analysis

Empirical analysis.

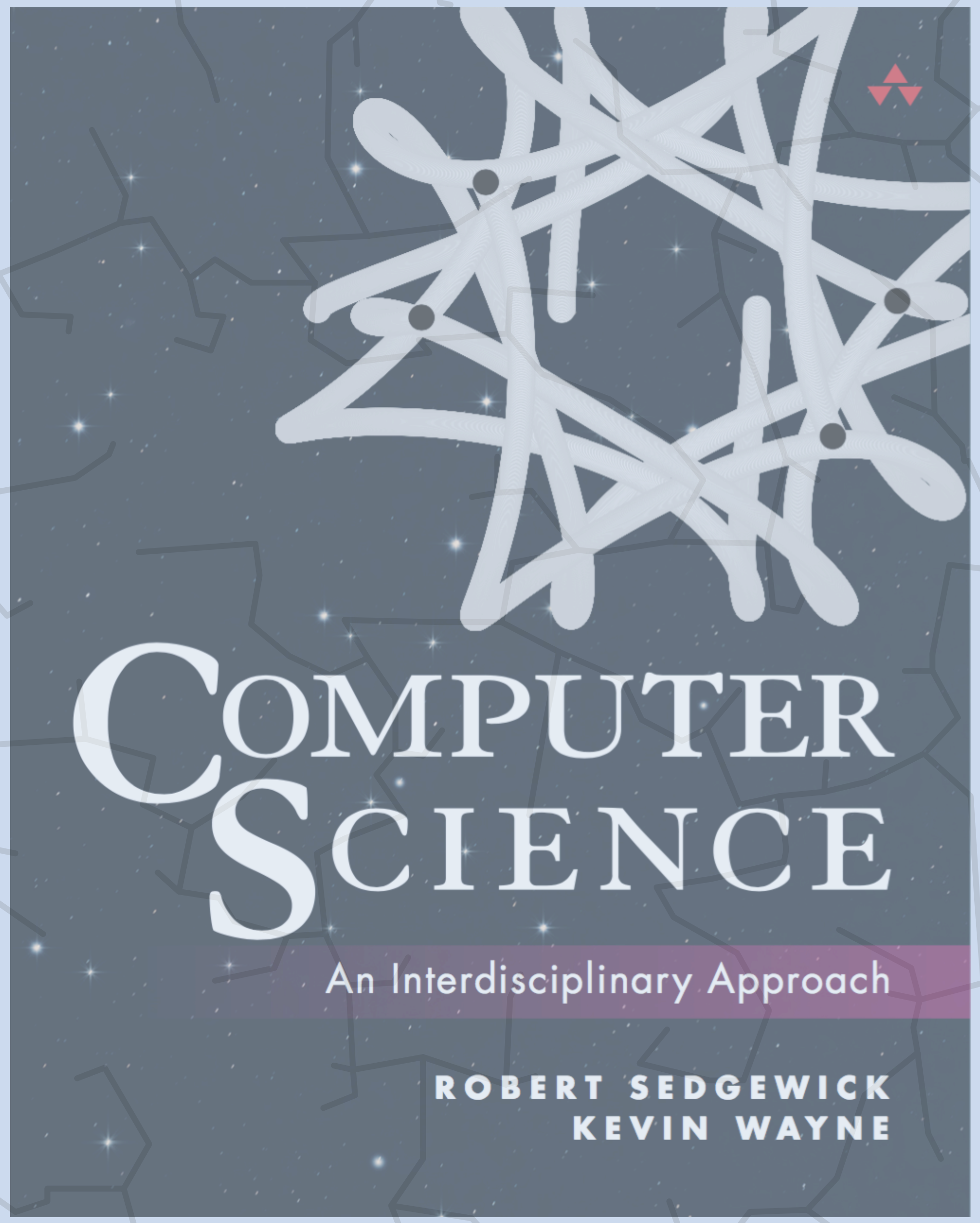
- Execute program to perform experiments.
- Formulate a hypotheses for running time.
- Model enables us to make predictions.



Mathematical analysis.

- Analyze code to count core operations.
- Simplify by discarding lower-order terms.
- Model enables us to explain behavior.

$$1 + 2 + \ldots + n = \tfrac{1}{2}n(n+1)$$
$$\sim \tfrac{1}{2}n^2$$



http://introcs.cs.princeton.edu

4.1 Performance

- *empirical analysis*
- *mathematical analysis*

Empirical analysis

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

Empirical analysis

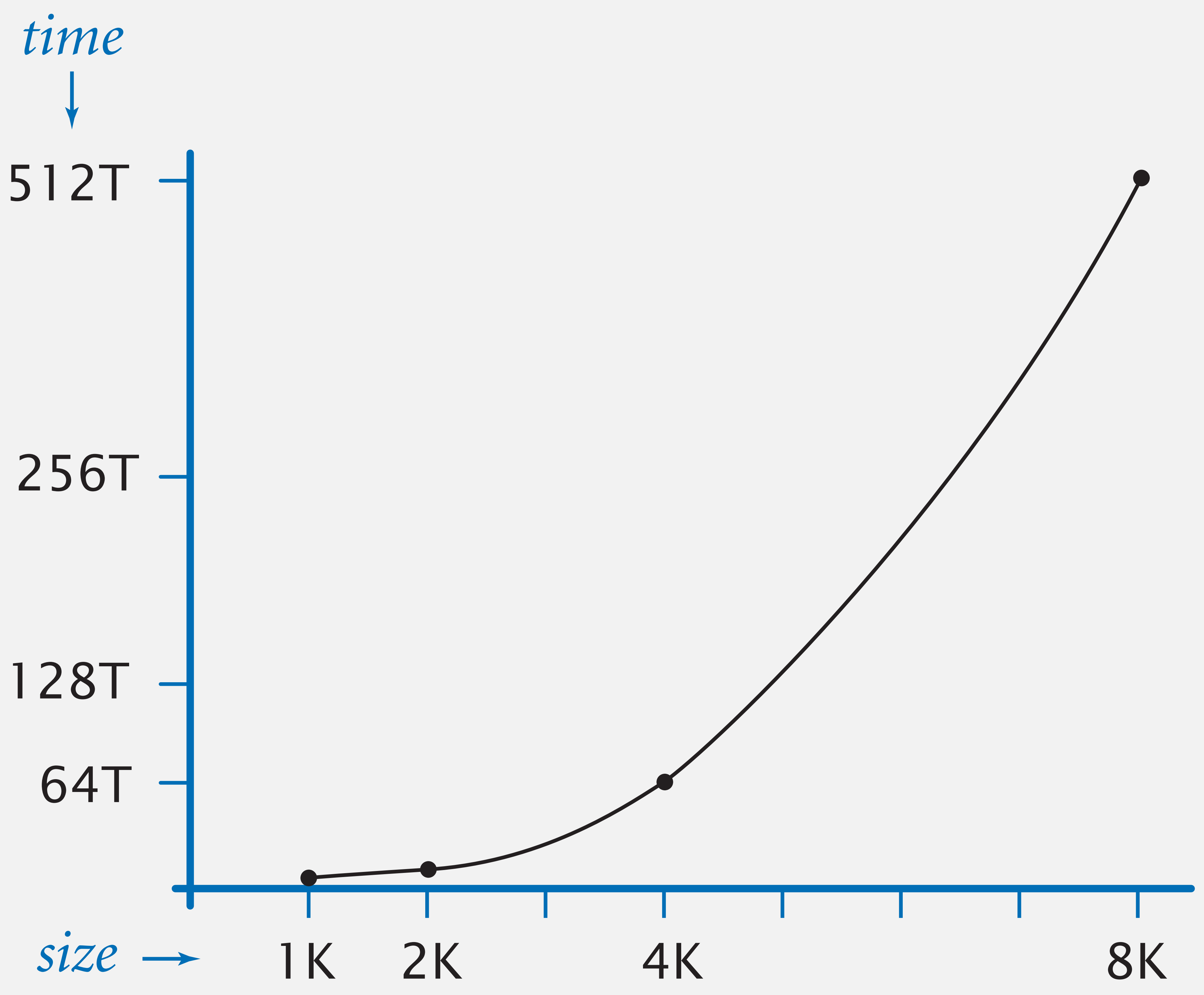
Measurements. Run the program for various input sizes and measure running time.

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-126 with 64GB DDR E3 memory and 32MB L3 cache;
running OpenJDK 1.8.0_71-b15 on Springdale Linux v. 7.2

Tip. To get cleaner data, use `-Xint` option: `java-introcs -Xint MyProgram`.

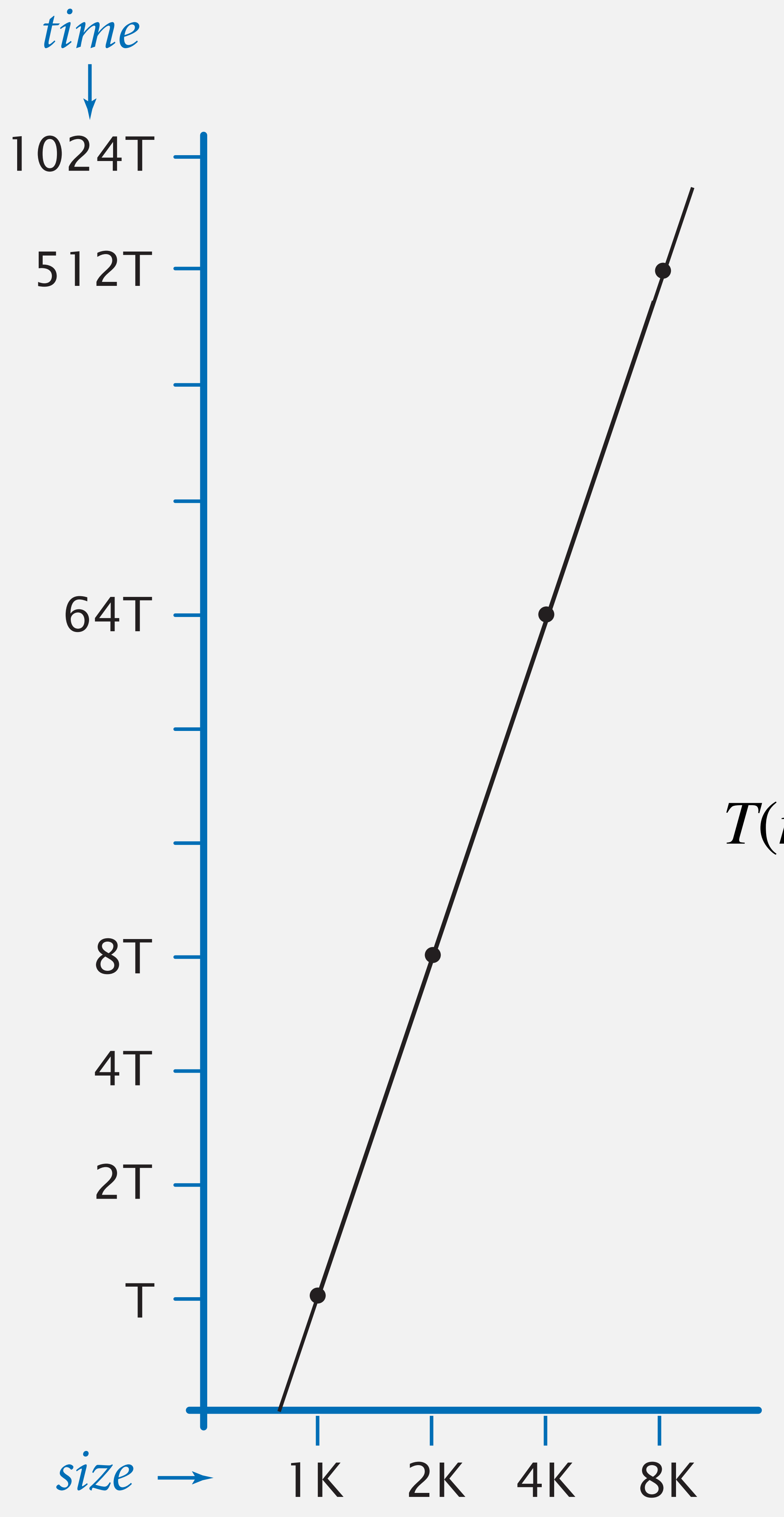
Approach 1. Plot running time $T(n)$ vs. input size n.



Standard plot

Data analysis

Approach 1. If running time obeys power law $T(n) = a\, n^{b}$, use log–log plot.



Log–log plot

slope

$$T(n) = 0.97 \times 10^{-10} \times n^{2.999} \text{ seconds}$$

Doubling hypothesis

Approach 2. Run program, doubling the size of the input.

n	time (seconds) †	ratio	$\log_2$ (ratio)
250	0.003	–	–
500	0.015	5.0	2.3
1,000	0.10	6.7	2.7
2,000	0.77	7.7	2.9
4,000	6.14	8.0	3.0
8,000	49.1	8.0	3.0

$\log_2 (49.1 / 6.14) = 3.0$

$49.1 / 6.14 = 8.0$

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

Hypothesis. Running time is approximately $T(n) = a\, n^b$ with $b = \log_2(\text{ratio})$.

Doubling hypothesis

Q. Why does $\log_2$ (ratio) give the exponent b ?

A. Assuming, $T(n) = an^b$

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

$$= \frac{2^b an^b}{an^b}$$

$$= 2^b$$

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

Doubling hypothesis

Approach 2. Run program, doubling the size of the input.

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	49.1
8,000	49
8,000	49.1

$$49.1 = a \times 8{,}000^3$$

$$\Rightarrow \quad a = 0.96 \times 10^{-10}$$

Hypothesis. Running time is about $0.96 \times 10^{-10} \times n^3$ seconds.

Performance quiz 1

Estimate the running time to solve a problem of size $n = 96{,}000$.

QuizSocket.com

4XFNCV Join Quiz

A. 39 *seconds*

B. 52 *seconds*

C. 117 *seconds*

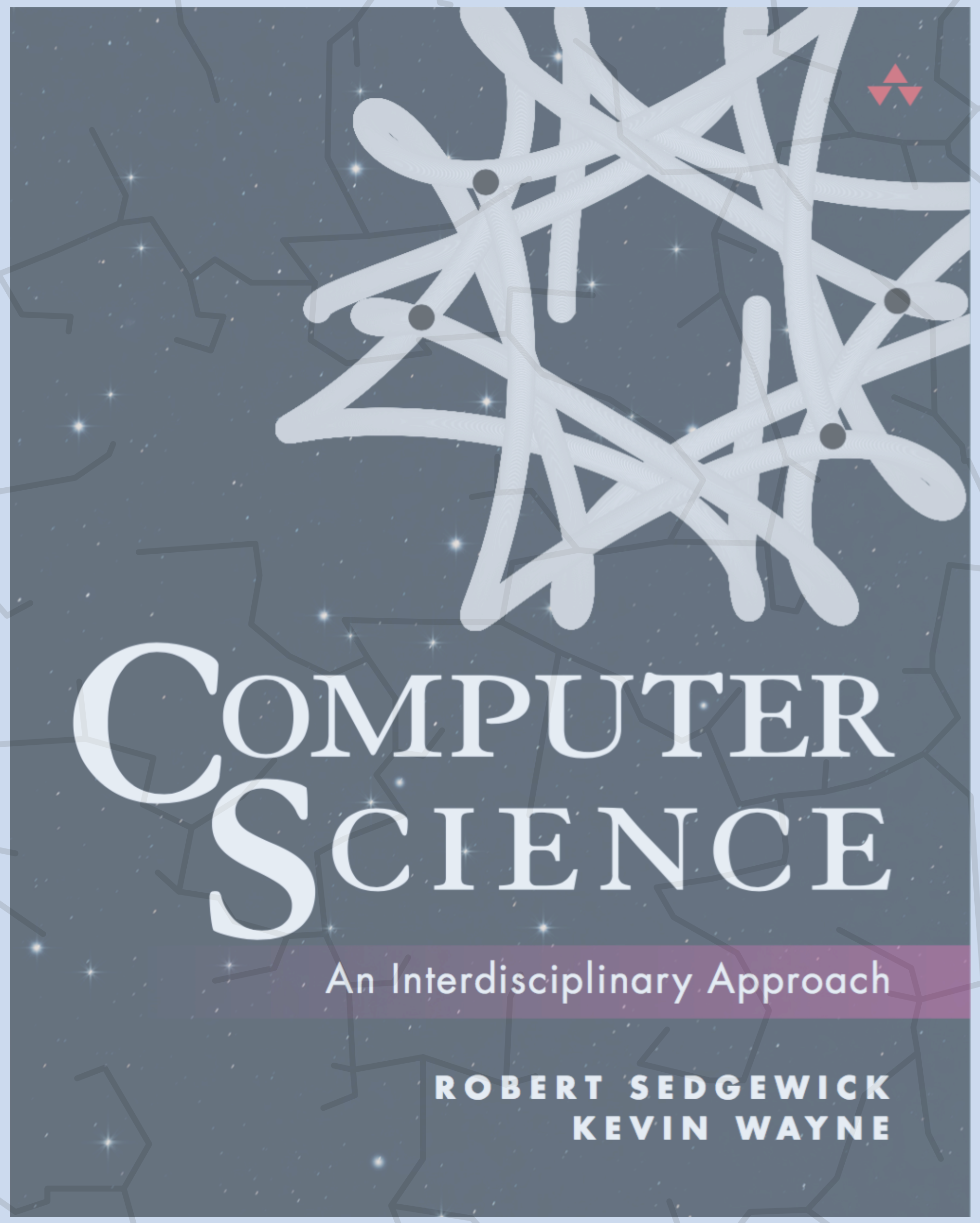
D. 350 *seconds*

n	time (seconds) †	ratio
1,000	0.02	–
2,000	0.05	2.5
4,000	0.20	4.0
8,000	0.81	4.1
16,000	3.25	4.0
32,000	13.01	4.0

$$T(n) = a\,n^2 \text{ seconds}$$

$$T(32{,}000) = 13 \text{ seconds}$$

$$\begin{aligned} T(3n) &= a\,(3n)^2 \\ &= 9\,a\,n^2 \\ &= 9\,T(n) \text{ seconds} \end{aligned}$$



http://introcs.cs.princeton.edu

4.1 PERFORMANCE

- *empirical analysis*
- *mathematical analysis*

Mathematical notations

Tilde notation. Simplify functions by ignoring lower-order terms.

Order-of-growth notation. Simplify functions by ignoring both lower-order terms and coefficient of leading term.

coefficient of leading term

lower-order terms

function	tilde notation	order of growth
$10\, n^3 + 50\, n^2 + 13$	$\sim 10\, n^3$	n^3
$\frac{1}{2}\, n^2 + 6n + 72$	$\sim \frac{1}{2}\, n^2$	n^2
$\frac{1}{2}\, n\, (n + 1)$	$\sim \frac{1}{2}\, n^2$	n^2
$(4n^2 + 26n) \cdot (3n^2 + 6n^{3/2} + \log_2 n)$	$\sim 12\, n^4$	n^4

Performance quiz 2

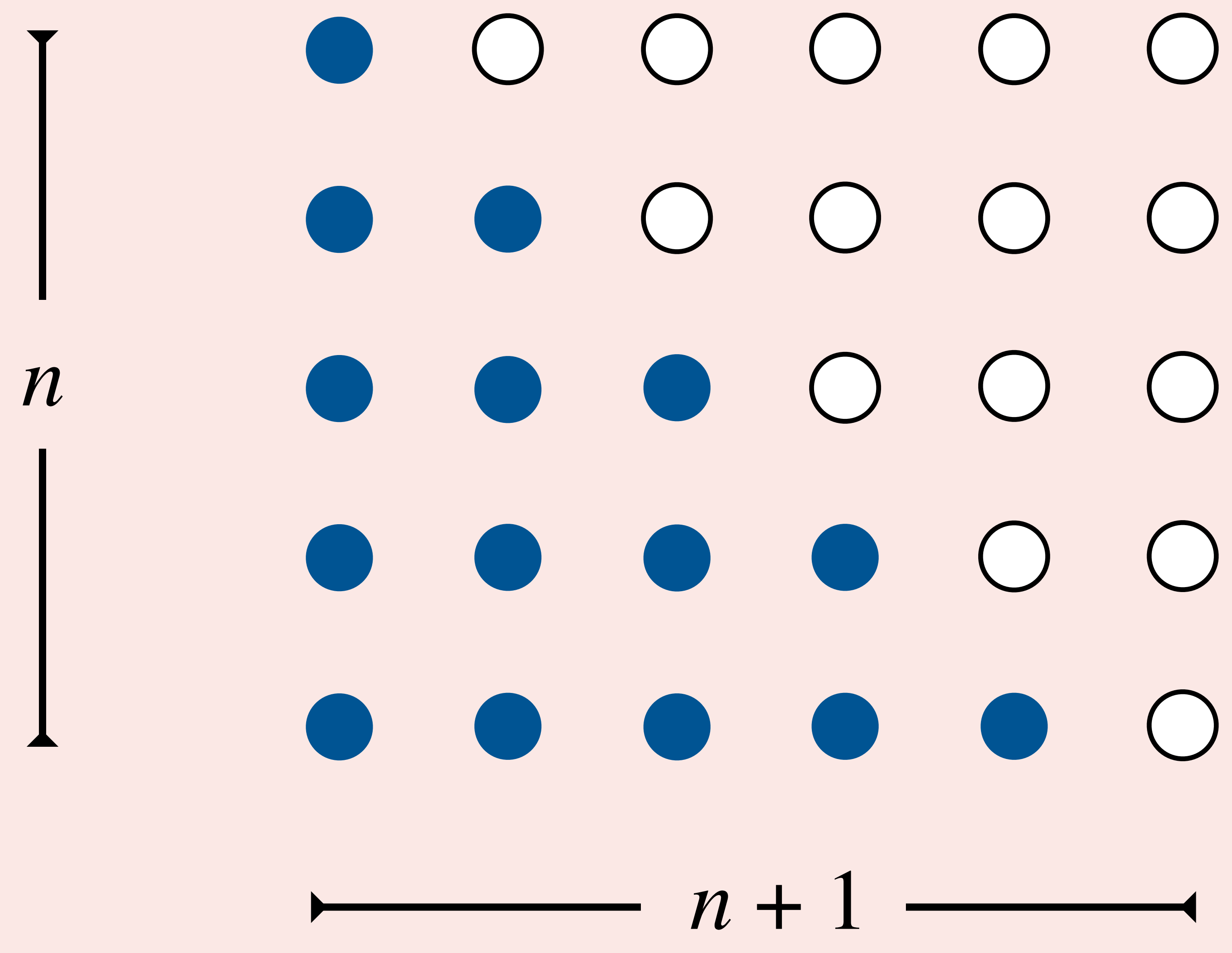
Which is the order of growth of the function $1 + 2 + 3 + \ldots + n$?

A. 1

B. n

C. $n \log n$

D. n^2



Constant order of growth

Number of machine instruction (e.g., TOY instructions) is a constant.

```
int sum = a + b;
```

← variable declaration; integer addition; and assignment

```
int product   = a * b;
int quotient  = a / b;
int xor       = a ^ b;
```

← constant number of constant-time operations is constant time

```
double root  = Math.sqrt(x);
double power = Math.pow(x, 0.5);
double sine  = Math.sin(x);
double log   = Math.log(x);
```

← constants for some operations are much larger than for others (and machine dependent)

Bottom line. Most primitive operations in Java take constant time.

Linear order of growth

Number of machine instruction is proportional to n.

```
double sum = 0.0;
for (int i = 0; i < n; i++)
   sum += x[i] * y[i];
```

← prototypical linear-time loop

```
int[] a = new int[n];
```

← implicit linear-time loop

```
int n = s.length();
String t = s.substring(0, n-1);
String u = s + "\n";
```

← substring extraction and string concatenation take time linear in the length of resulting string

Caveat. Non-primitive operations can take more than constant time.

Quadratic order of growth

Number of machine instruction is proportional n^2.

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

← prototypical quadratic-time loop

```
String s = "";
int n = a.length;
for (int i = 0; i < n; i++)
   s += a[i];
```

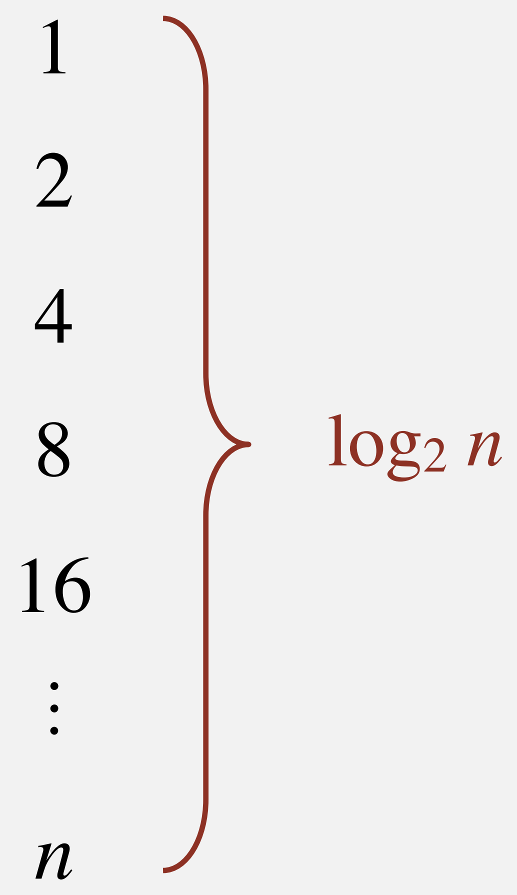
← not constant time! (annotating `s += a[i];`)

← performance gotcha: abusive string concatenation

Logarithmic order of growth

Number of machine instruction is proportional to $\log_2 n$.

```
for (int i = 1; i <= n; i = i*2) {
   count++;
}
```



Note. We use the notation $\log n$ with order of growth (because the base is irrelevant).

- Binary logarithm: $\lg n = \log_2 n$.
- Natural logarithm: $\ln n = \log_e n$.

Logarithmic order of growth

Number of machine instruction is proportional to $\log_2 n$.

http://www.20q.net

Note. Generalizes `TwentyQuestions.java` from book.

Performance quiz 3

Which is the order of growth of the following code fragment?

A. n^2

B. $n^2 \log n$

C. n^3

D. $n^3 \log n$

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

$n \times n \times \log n$

Performance quiz 4

Which is the order of growth of the following code fragment?

A. n^2

B. n^3

C. n^5

D. n^7

```
int count = 0;

for (int i = 0; i < n; i++)
   for (int j = 0; j < n; j++)
      count++;

for (int i = 0; i < n; i++)
   for (int j = 0; j < n; j++)
      for (int k = 0; k < n; k++)
         count++;

for (int i = 0; i < n; i++)
   for (int j = 0; j < n; j++)
      count++;
```

$$1 \; + n^2 \; + n^3 \; + n^2$$