



<https://algs4.cs.princeton.edu>

## 1.4 ANALYSIS OF ALGORITHMS

---

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



<https://algs4.cs.princeton.edu>

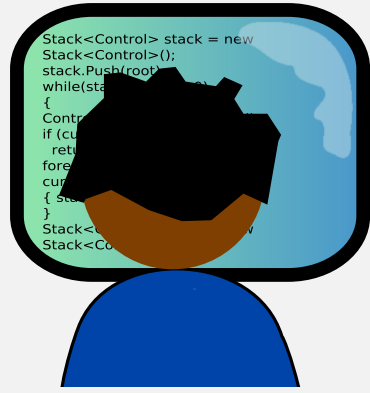
## 1.4 ANALYSIS OF ALGORITHMS

---

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

# Cast of characters

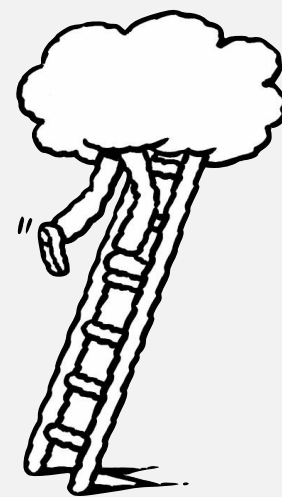
---



**programmer** needs to  
develop a working solution



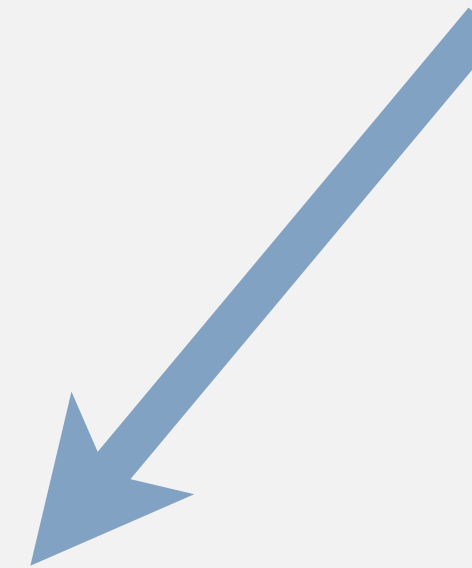
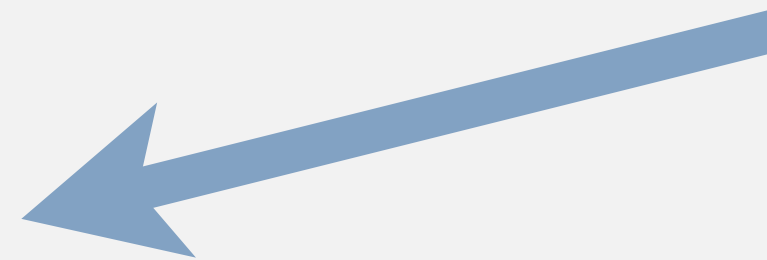
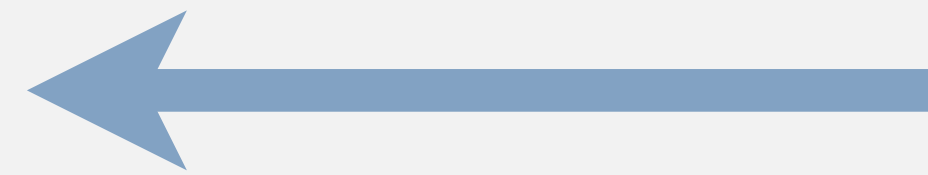
**client** wants to solve  
problem efficiently



**theoretician** seeks  
to understand



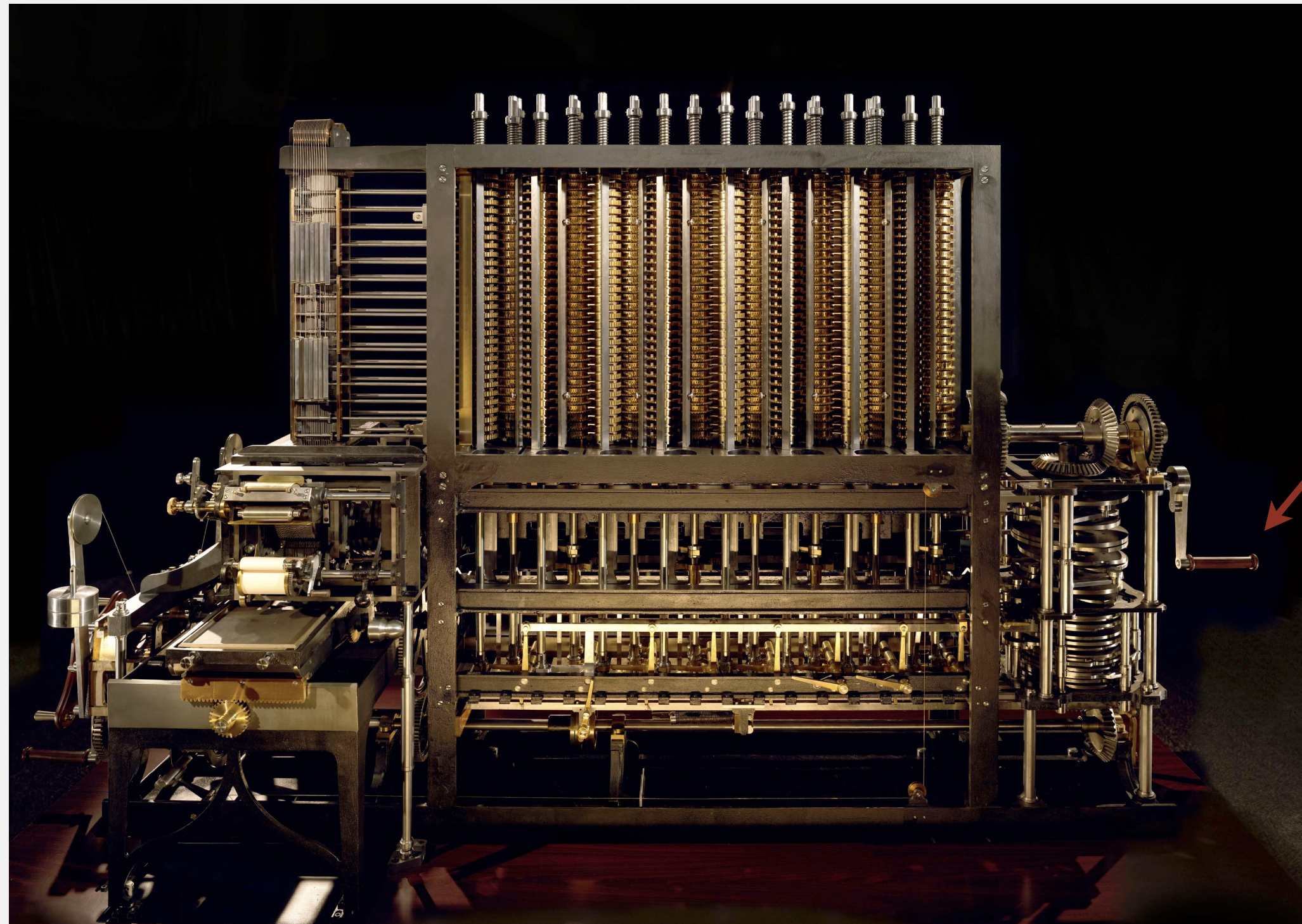
**student (you)**  
will play all of  
these roles in this course





# 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)*



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

By Matt Kennedy  
July 25, 2018

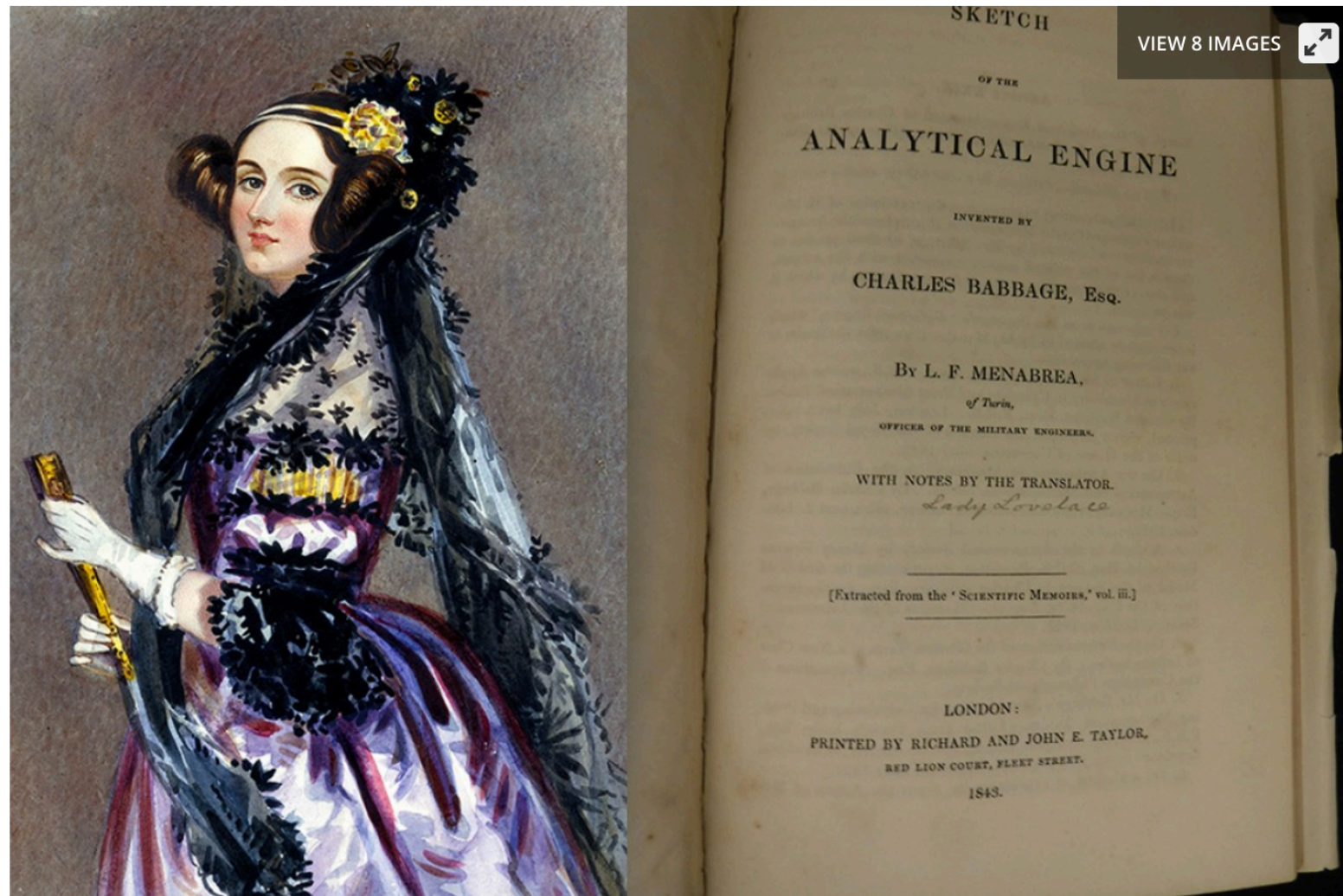


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 72<sup>d</sup> et seq.)

Number of Operation.	Nature of Operation.	Variables added upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.	Data.												Working Variables.												Result Variables.									
						V <sub>1</sub>	V <sub>2</sub>	V <sub>3</sub>	V <sub>4</sub>	V <sub>5</sub>	V <sub>6</sub>	V <sub>7</sub>	V <sub>8</sub>	V <sub>9</sub>	V <sub>10</sub>	V <sub>11</sub>	V <sub>12</sub>	V <sub>13</sub>	V <sub>14</sub>	V <sub>15</sub>	V <sub>16</sub>	V <sub>17</sub>	V <sub>18</sub>	V <sub>19</sub>	V <sub>20</sub>	V <sub>21</sub>	V <sub>22</sub>	V <sub>23</sub>	V <sub>24</sub>	V <sub>25</sub>	V <sub>26</sub>	V <sub>27</sub>	V <sub>28</sub>	V <sub>29</sub>	V <sub>30</sub>				
1	X	V <sub>2</sub> × V <sub>1</sub>	V <sub>6</sub> , V <sub>10</sub> , V <sub>14</sub>	{V <sub>6</sub> = V <sub>2</sub> V <sub>10</sub> = V <sub>2</sub> V <sub>14</sub> = V <sub>2</sub> }	= 2n	1	2	n																															
2	-	V <sub>4</sub> - V <sub>1</sub>	V <sub>4</sub>	{V <sub>4</sub> = V <sub>4</sub> V <sub>8</sub> = V <sub>4</sub> V <sub>12</sub> = V <sub>4</sub> }	= 2n-1	1			2n-1																														
3	+	V <sub>4</sub> + V <sub>1</sub>	V <sub>18</sub>	{V <sub>18</sub> = V <sub>4</sub> V <sub>22</sub> = V <sub>4</sub> V <sub>26</sub> = V <sub>4</sub> }	= 2n+1	1				2n+1																													
4	+	V <sub>4</sub> + V <sub>1</sub>	V <sub>12</sub>	{V <sub>12</sub> = V <sub>4</sub> V <sub>16</sub> = V <sub>4</sub> V <sub>20</sub> = V <sub>4</sub> }	= 2n-1	1			0																														
5	+	V <sub>10</sub> + V <sub>11</sub>	V <sub>11</sub>	{V <sub>11</sub> = V <sub>10</sub> V <sub>15</sub> = V <sub>10</sub> V <sub>19</sub> = V <sub>10</sub> }	= 1 2n-1	1																																	
6	-	V <sub>13</sub> - V <sub>10</sub>	V <sub>13</sub>	{V <sub>13</sub> = V <sub>2</sub> V <sub>17</sub> = V <sub>2</sub> V <sub>21</sub> = V <sub>2</sub> }	= 1 2n-1	1																																	
7	-	V <sub>13</sub> - V <sub>1</sub>	V <sub>13</sub>	{V <sub>13</sub> = V <sub>13</sub> V <sub>17</sub> = V <sub>13</sub> V <sub>21</sub> = V <sub>13</sub> }	= n-1 (=3)	1		n																															
8	+	V <sub>2</sub> + V <sub>1</sub>	V <sub>2</sub>	{V <sub>2</sub> = V <sub>2</sub> V <sub>6</sub> = V <sub>2</sub> V <sub>10</sub> = V <sub>2</sub> }	= 2+0=2	1																																	
9	+	V <sub>6</sub> + V <sub>1</sub>	V <sub>11</sub>	{V <sub>11</sub> = V <sub>6</sub> V <sub>15</sub> = V <sub>6</sub> V <sub>19</sub> = V <sub>6</sub> }	= 2 A <sub>1</sub>	1																																	
10	X	V <sub>10</sub> × V <sub>11</sub>	V <sub>12</sub>	{V <sub>12</sub> = B <sub>1</sub> V <sub>16</sub> = B <sub>1</sub> V <sub>20</sub> = B <sub>1</sub> }	= 1 2n-1	1																																	
11	+	V <sub>12</sub> + V <sub>13</sub>	V <sub>13</sub>	{V <sub>13</sub> = V <sub>12</sub> V <sub>17</sub> = V <sub>12</sub> V <sub>21</sub> = V <sub>12</sub> }	= 1 2n-1	1																																	
12	-	V <sub>10</sub> - V <sub>1</sub>	V <sub>18</sub>	{V <sub>18</sub> = V <sub>10</sub> V <sub>22</sub> = V <sub>10</sub> V <sub>26</sub> = V <sub>10</sub> }	= n-2 (=2)	1																																	
13	-	V <sub>4</sub> - V <sub>1</sub>	V <sub>4</sub>	{V <sub>4</sub> = V <sub>4</sub> V <sub>8</sub> = V <sub>4</sub> V <sub>12</sub> = V <sub>4</sub> }	= 2n-	1																																	
14	+	V <sub>4</sub> + V <sub>1</sub>	V <sub>12</sub>	{V <sub>12</sub> = V <sub>4</sub> V <sub>16</sub> = V <sub>4</sub> V <sub>20</sub> = V <sub>4</sub> }	= 2+1=3	1																																	
15	+	V <sub>4</sub> + V <sub>1</sub>	V <sub>18</sub>	{V <sub>18</sub> = V <sub>4</sub> V <sub>22</sub> = V <sub>4</sub> V <sub>26</sub> = V <sub>4</sub> }	= 2n-1	1																																	
16	X	V <sub>4</sub> × V <sub>10</sub>	V <sub>11</sub>	{V <sub>11</sub> = V <sub>4</sub> V <sub>15</sub> = V <sub>4</sub> V <sub>19</sub> = V <sub>4</sub> }	= 2 2n-1	1																																	
17	-	V <sub>4</sub> - V <sub>1</sub>	V <sub>4</sub>	{V <sub>4</sub> = V <sub>4</sub> V <sub>8</sub> = V <sub>4</sub> V <sub>12</sub> = V <sub>4</sub> }	= 2-2=0	1																																	
18	+	V <sub>4</sub> + V <sub>1</sub>	V <sub>12</sub>	{V <sub>12</sub> = V <sub>4</sub> V <sub>16</sub> = V <sub>4</sub> V <sub>20</sub> = V <sub>4</sub> }	= 3+1=4	1																																	
19	+	V <sub>6</sub> + V <sub>1</sub>	V <sub>11</sub>	{V <sub>11</sub> = V <sub>6</sub> V <sub>15</sub> = V <sub>6</sub> V <sub>19</sub> = V <sub>6</sub> }	= 2-2=0	1																																	
20	X	V <sub>4</sub> × V <sub>10</sub>	V <sub>11</sub>	{V <sub>11</sub> = V <sub>4</sub> V <sub>15</sub> = V <sub>4</sub> V <sub>19</sub> = V <sub>4</sub> }	= 2 2n-1	1																																	
21	X	V <sub>4</sub> × V <sub>10</sub>	V <sub>11</sub>	{V <sub>11</sub> = V <sub>4</sub> V <sub>15</sub> = V <sub>4</sub> V <sub>19</sub> = V <sub>4</sub> }	= 2 2n-1	1																																	
22	+	V <sub>12</sub> + V <sub>13</sub>	V <sub>13</sub>	{V <sub>13</sub> = V <sub>12</sub> V <sub>17</sub> = V <sub>12</sub> V <sub>21</sub> = V <sub>12</sub> }	= A <sub>2</sub> + B <sub>1</sub> A <sub>1</sub> + B <sub>2</sub> A <sub>2</sub>	1																																	
23	-	V <sub>13</sub> - V <sub>1</sub>	V <sub>13</sub>	{V <sub>13</sub> = V <sub>13</sub> V <sub>17</sub> = V <sub>13</sub> V <sub>21</sub> = V <sub>13</sub> }	= n-3 (=1)	1																																	

Here follows a repetition of Operations thirteen to twenty-three.

24	+	V <sub>13</sub> + V <sub>10</sub>	V <sub>14</sub>	{V <sub>14</sub> = V <sub>13</sub> V <sub>18</sub> = V <sub>13</sub> V <sub>22</sub> = V <sub>13</sub> }	= B <sub>1</sub>	1																												
25	+	V <sub>13</sub> + V <sub>10</sub>	V <sub>14</sub>	{V <sub>14</sub> = V <sub>13</sub> V <sub>18</sub> = V <sub>13</sub> V <sub>22</sub> = V <sub>13</sub> }	= +1=4+1=5	1		n+1			0																							

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



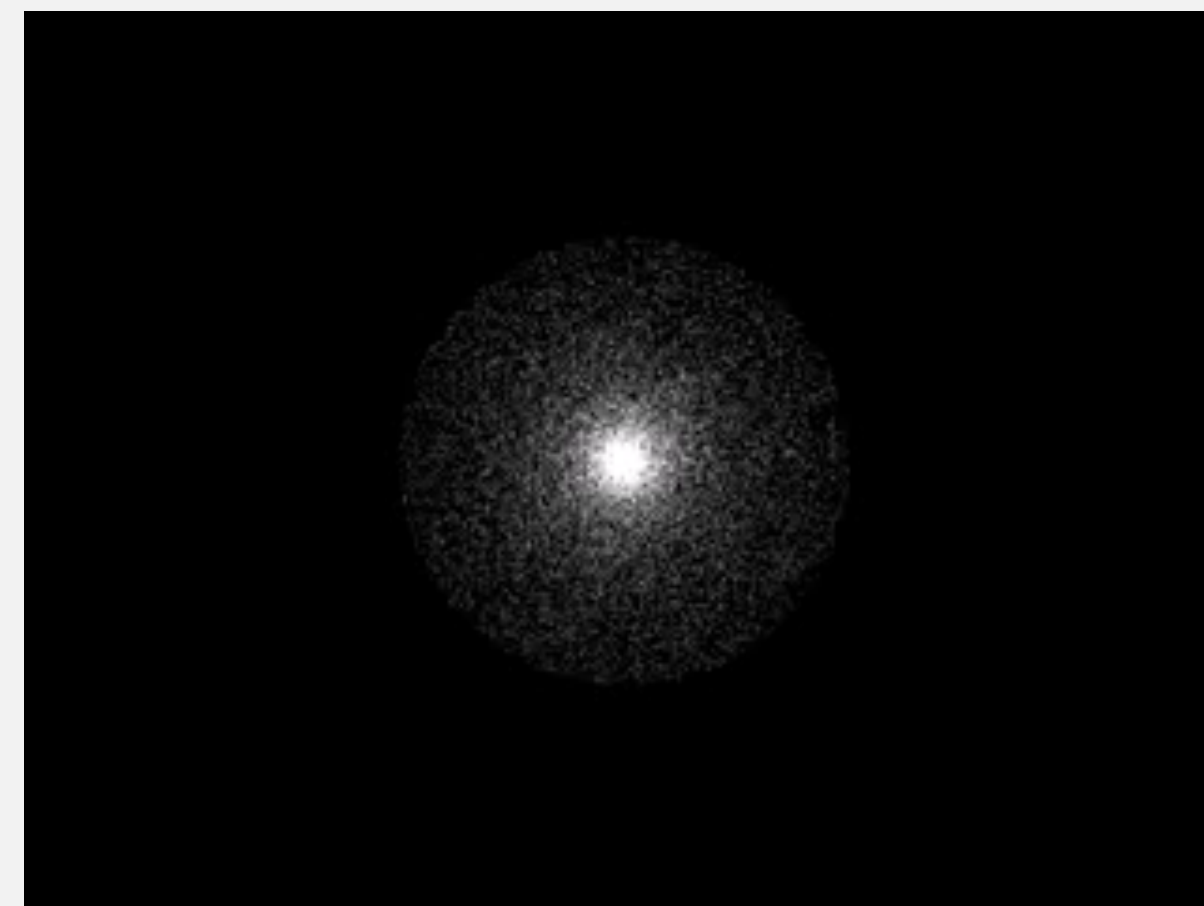
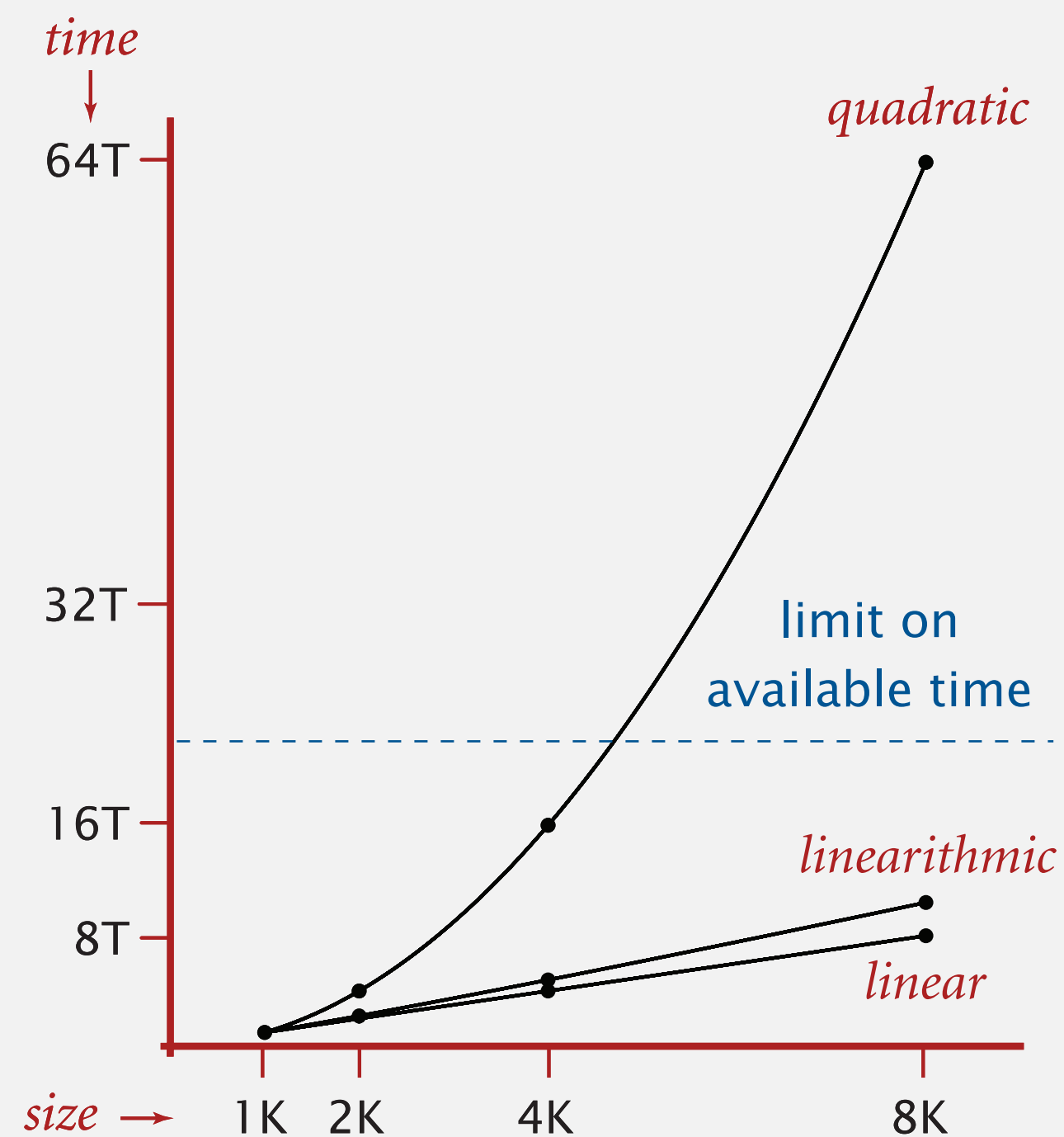
# An algorithmic success story

## 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





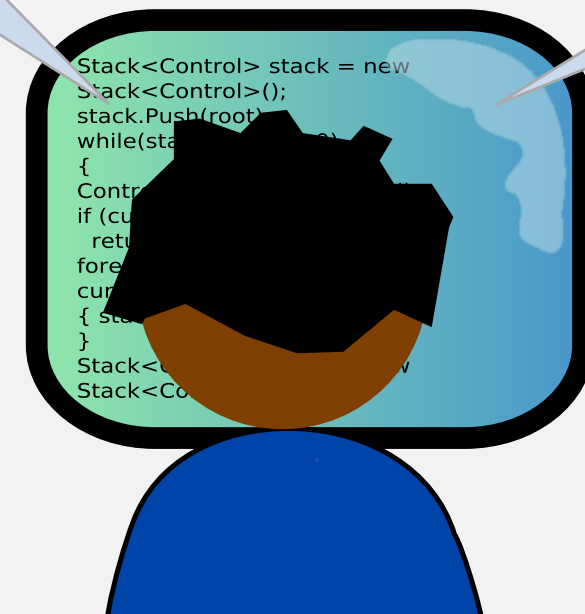
# The challenge

---

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

Why is my program so slow?

Why does it run out of memory?



Our approach. Combination of **experiments** and **mathematical modeling**.

# Example: 3-SUM



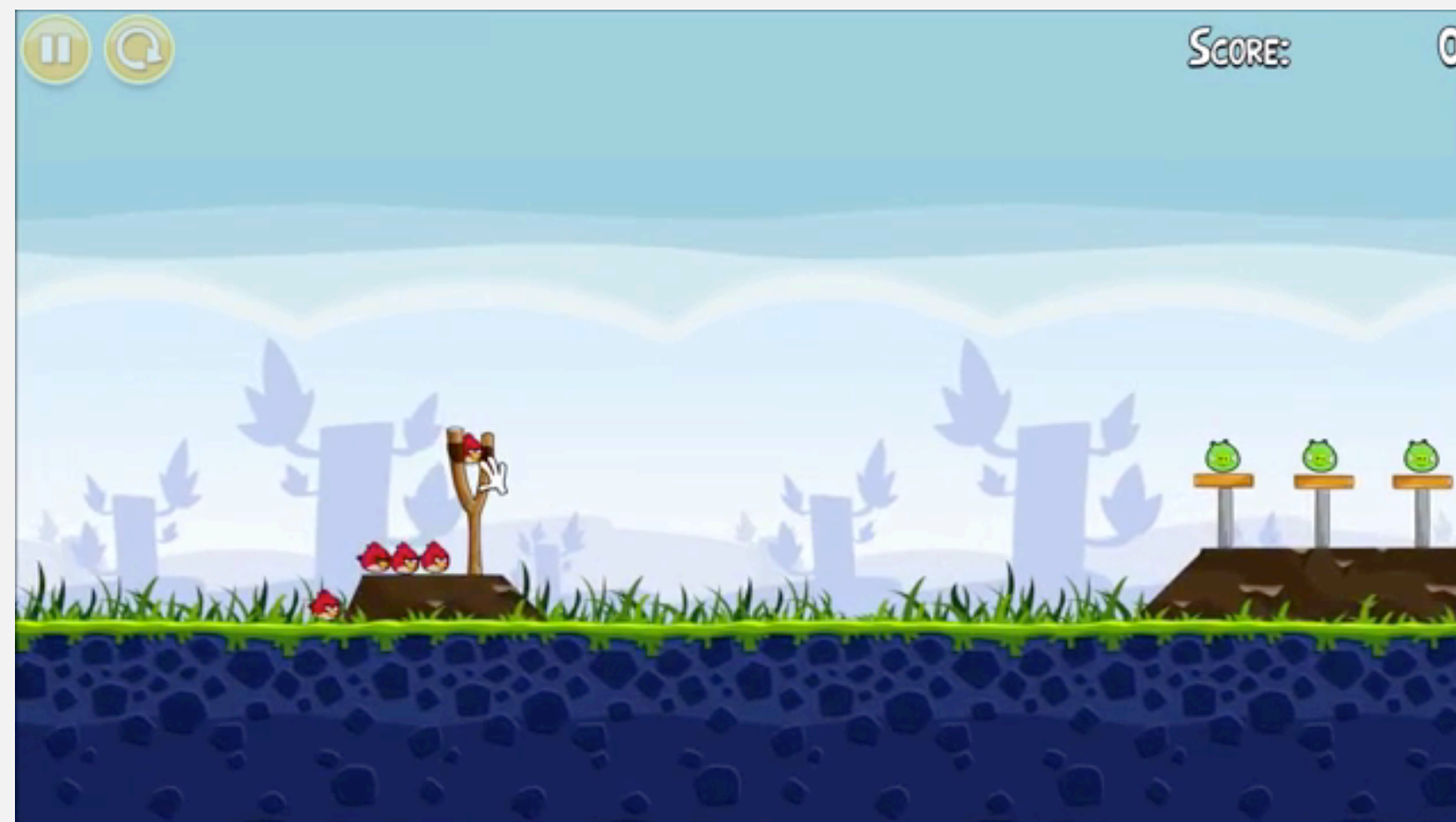
**3-SUM.** Given  $n$  distinct integers, how many triples sum to exactly zero?

```
~/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



<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	?

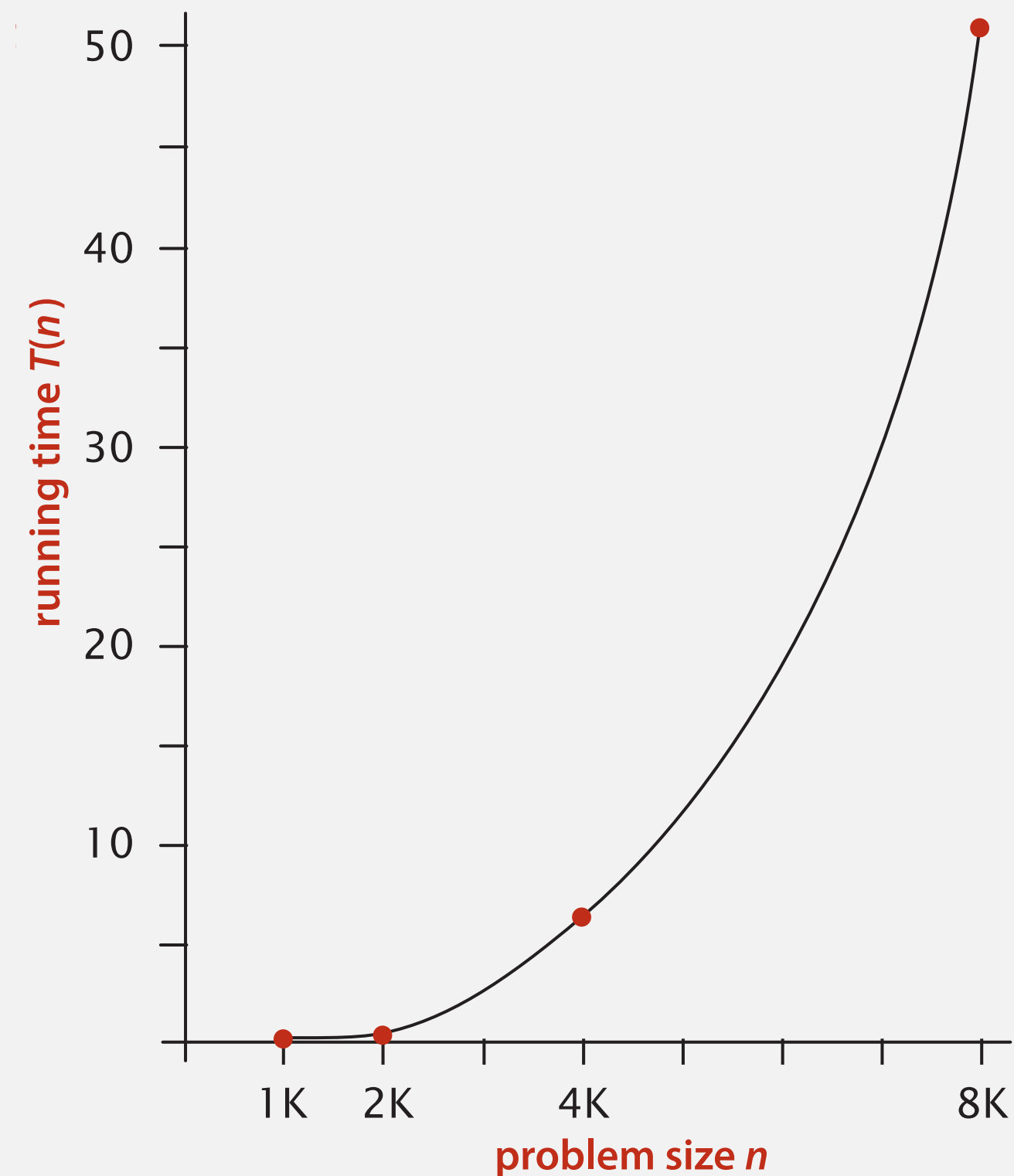
† on a 2.8GHz Intel PU-226 with 64GB  
DDR E3 memory and 32MB L3 cache;  
running Oracle Java 1.7.0\_45-b18 on  
Springdale Linux v. 6.5



# Data analysis

---

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

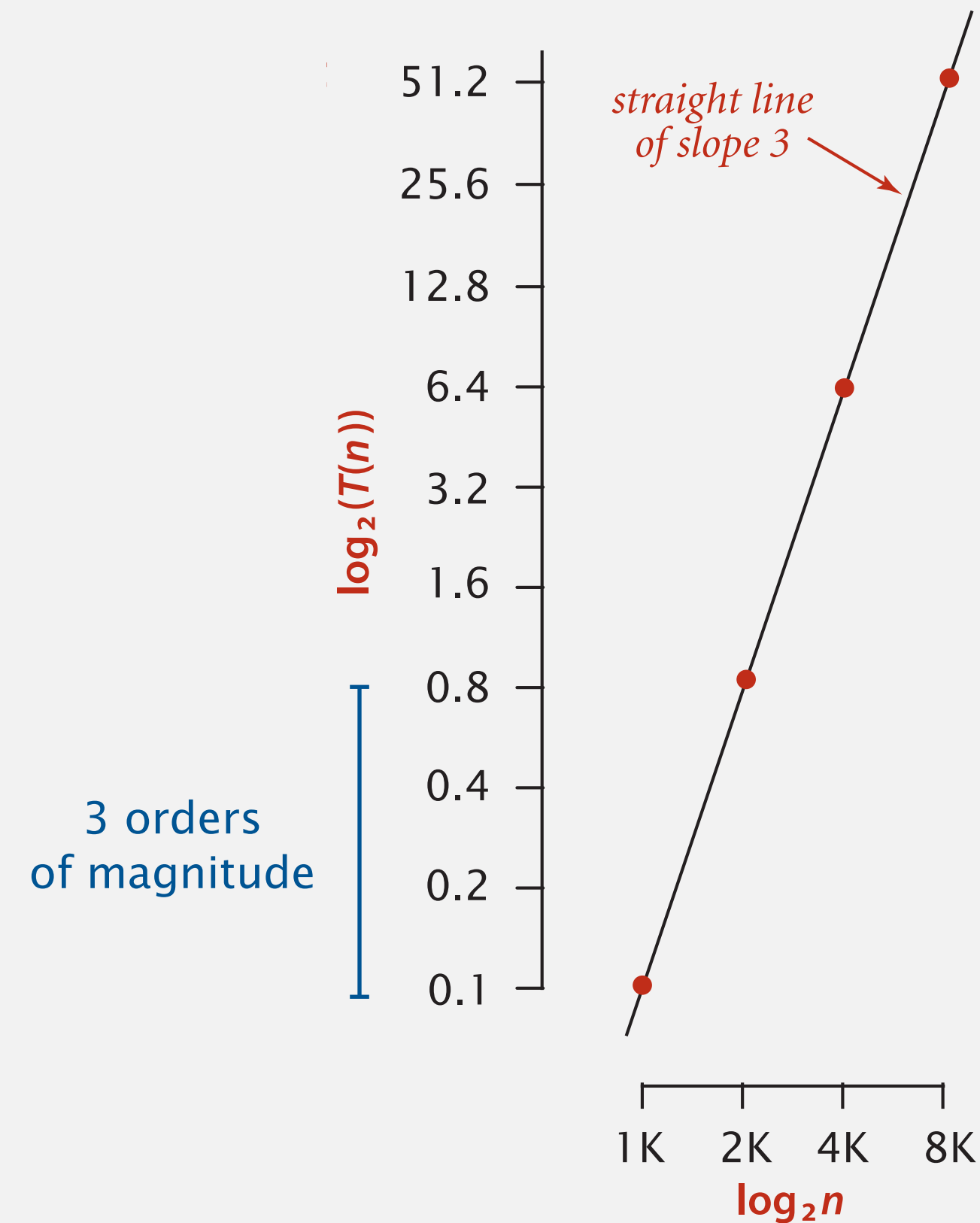


Hypothesis (power law).  $T(n) = a n^b$ .

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**.



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

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

“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

$$\frac{T(n)}{T(n/2)} = \frac{an^b}{a(n/2)^b} = 2^b$$
$$\implies b = \log_2 \frac{T(n)}{T(n/2)}$$

$\log_2 (6.4 / 0.8) = 3.0$

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

**Hypothesis.** Running time is about  $a n^b$  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.



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





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  $a n^b$



**Bad news.** Sometimes difficult to get accurate measurements.





Experimental algorithmics is an example of the **scientific method**.



**Chemistry**  
(1 experiment)



**Biology**  
(1 experiment)



**Computer Science**  
(1 million experiments)



**Physics**  
(1 experiment)

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



<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  $\times$  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, ....



**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 $2n$

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

† representative estimates (with some poetic license)



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++;
```

$0 + 1 + 2 + \dots + (n - 1) = \frac{1}{2} n(n - 1)$   
 $= \binom{n}{2}$

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 $n^2$

$\left\{ \begin{array}{l} 1/4 n^2 + 13/20 n + 13/10 \text{ ns} \\ \text{to} \\ 3/10 n^2 + 3/5 n + 13/10 \text{ ns} \end{array} \right.$   
(tedious to count exactly)

# Simplification 1: cost model

Cost model. Use some elementary 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++;
```

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 $n^2$

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

# Simplification 2: asymptotic notations

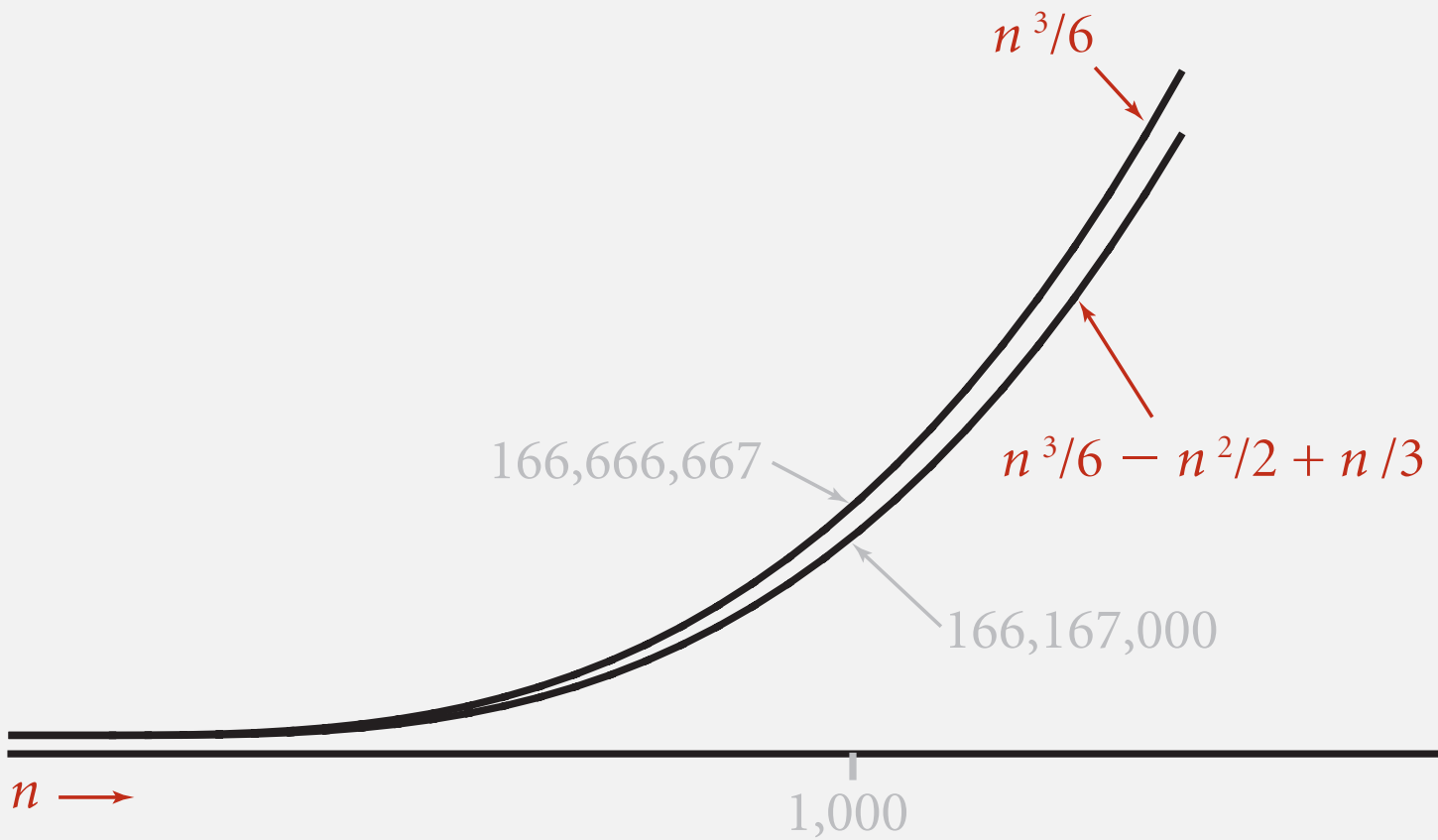
Tilde notation. Discard lower-order terms.

Big Theta notation. Also discard leading coefficient.

← formal definitions  
involve limits

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

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	<code>a = b + c;</code>	statement	<i>add two numbers</i>	1
$\Theta(\log n)$	😎	logarithmic	<code>for (int i = n; i &gt; 0; i /= 2) { ... }</code>	divide in half	<i>binary search</i>	$\sim 1$
$\Theta(n)$	😄	linear	<code>for (int i = 0; i &lt; n; i++) { ... }</code>	single loop	<i>find the maximum</i>	2
$\Theta(n \log n)$	😁	linearithmic	<i>see mergesort lecture</i>	divide and conquer	<i>mergesort</i>	$\sim 2$
$\Theta(n^2)$	😞	quadratic	<code>for (int i = 0; i &lt; n; i++)   for (int j = 0; j &lt; n; j++)   { ... }</code>	double loop	<i>check all pairs</i>	4
$\Theta(n^3)$	😓	cubic	<code>for (int i = 0; i &lt; n; i++)   for (int j = 0; j &lt; n; j++)     for (int k = 0; k &lt; n; k++)     { ... }</code>	triple loop	<i>check all triples</i>	8
$\Theta(2^n)$	😈	exponential	<i>see combinatorial search lecture</i>	exhaustive search	<i>check all subsets</i>	$2^n$

## 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”

$$\begin{aligned} 0 + 1 + 2 + \dots + (n-1) &= \frac{1}{2}n(n-1) \\ &= \binom{n}{2} \end{aligned}$$

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

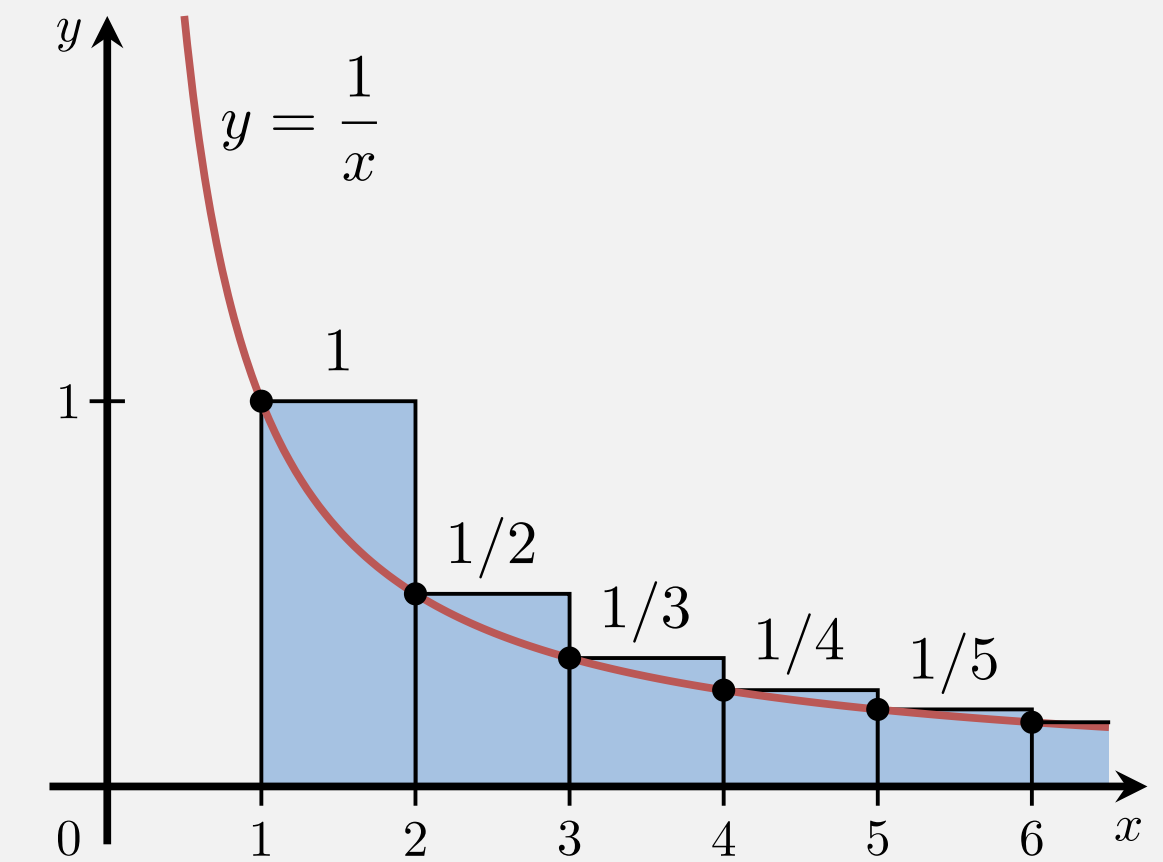
---

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

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

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

$\uparrow$   
 $n$  a power of 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++)
        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.  $\sim 3/2 n^3$



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.  $\Theta(2^n)$





<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.

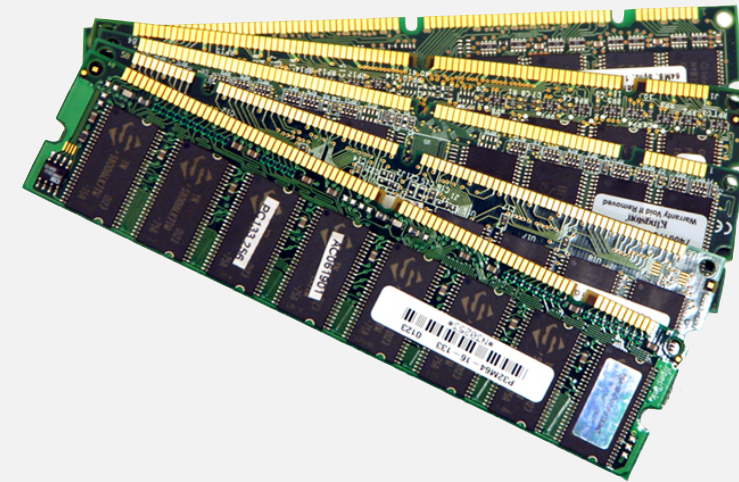
Byte. 8 bits.

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

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

NIST

most computer scientists



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[]	$1n + 24$
int[]	$4n + 24$
double[]	$8n + 24$

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)

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

array overhead = 24 byte

# Typical memory usage for objects in Java

---

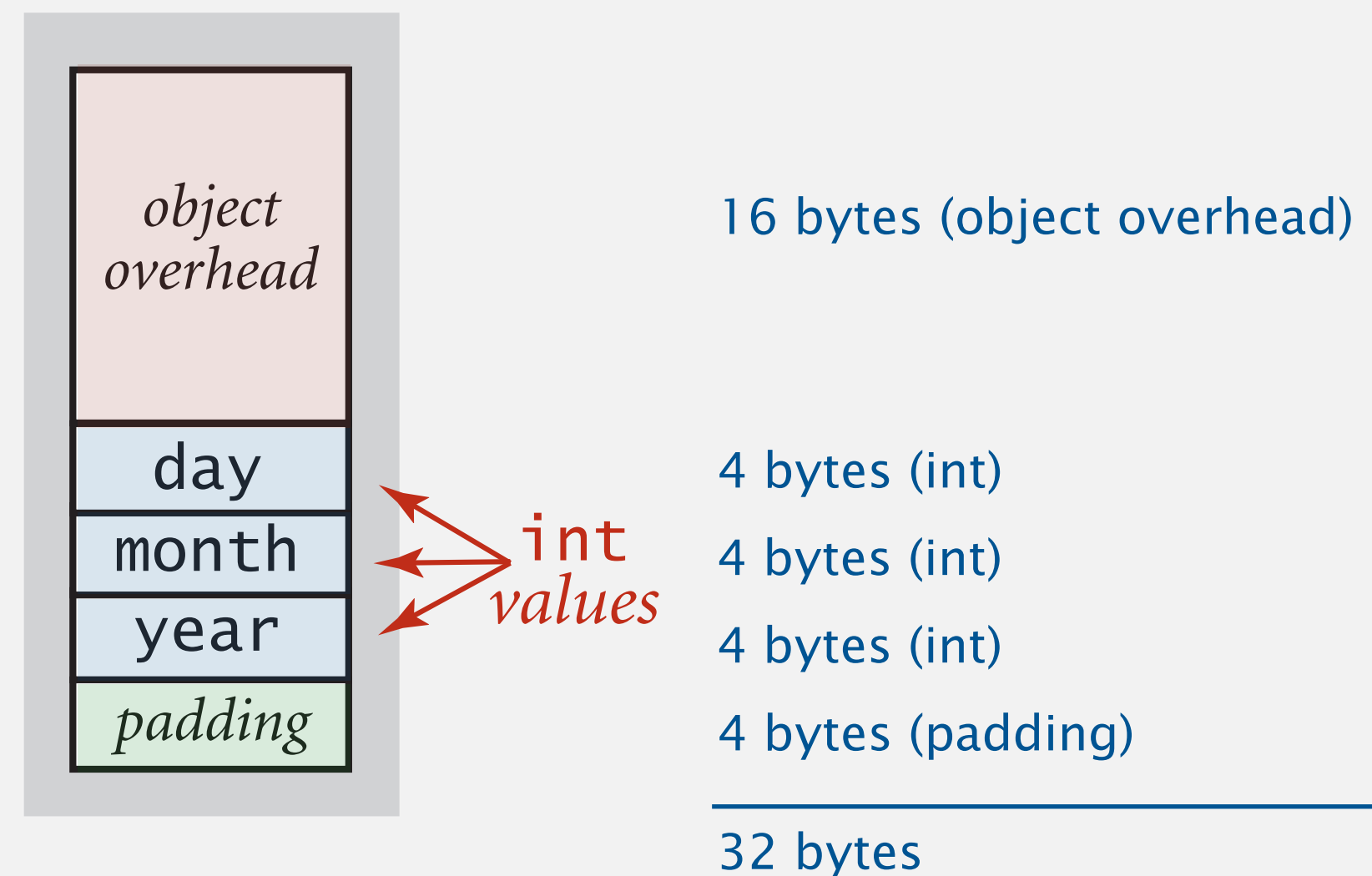
Object overhead. 16 bytes.

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;
    ...
}
```







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

- A.  $\sim 4n$  bytes
- B.  $\sim 8n$  bytes
- C.  $\sim 4n^2$  bytes
- D.  $\sim 8n^2$  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];

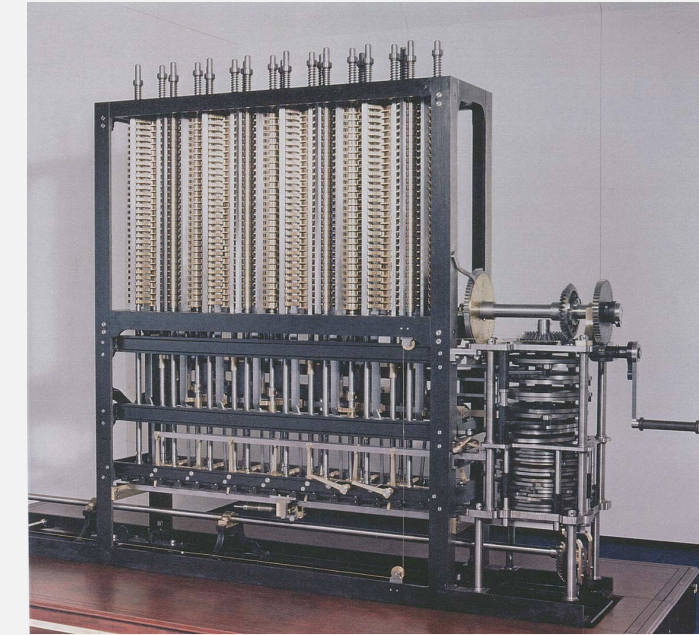
        count = 0;
        for (int i = 0; i < n; i++)
            parent[i] = i;
        for (int i = 0; i < n; i++)
            size[i] = 1;
    }
    ...
}
```

# 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**.

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

**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)*

