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

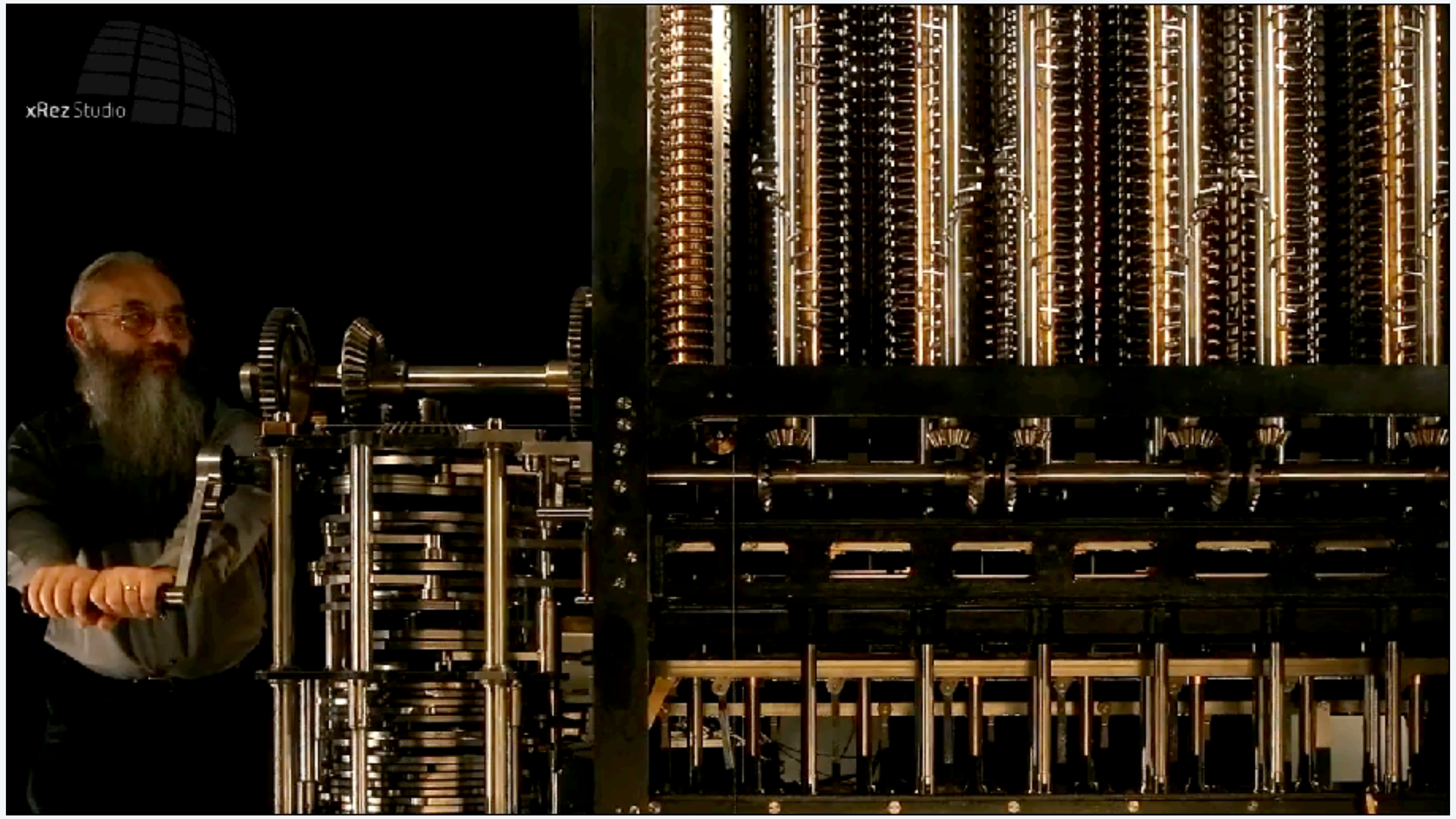
1.4 Analysis of Algorithms

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

Running time

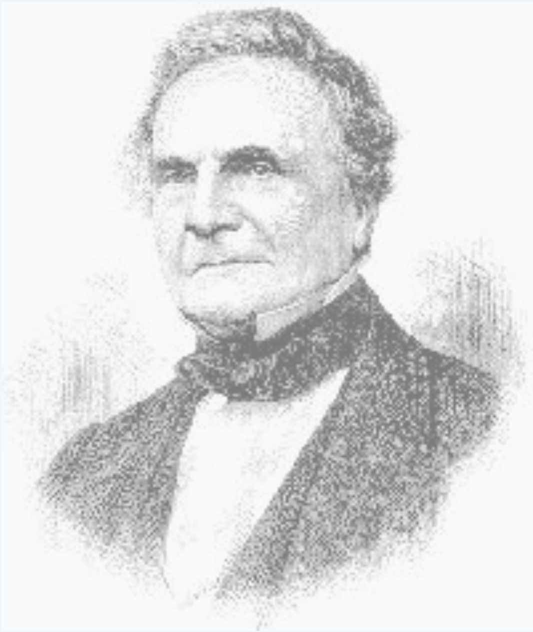
“ *As soon as an Analytical Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will then arise—By what course of calculation can these results be arrived at by the machine in the shortest time* ? ” — Charles Babbage (1864)

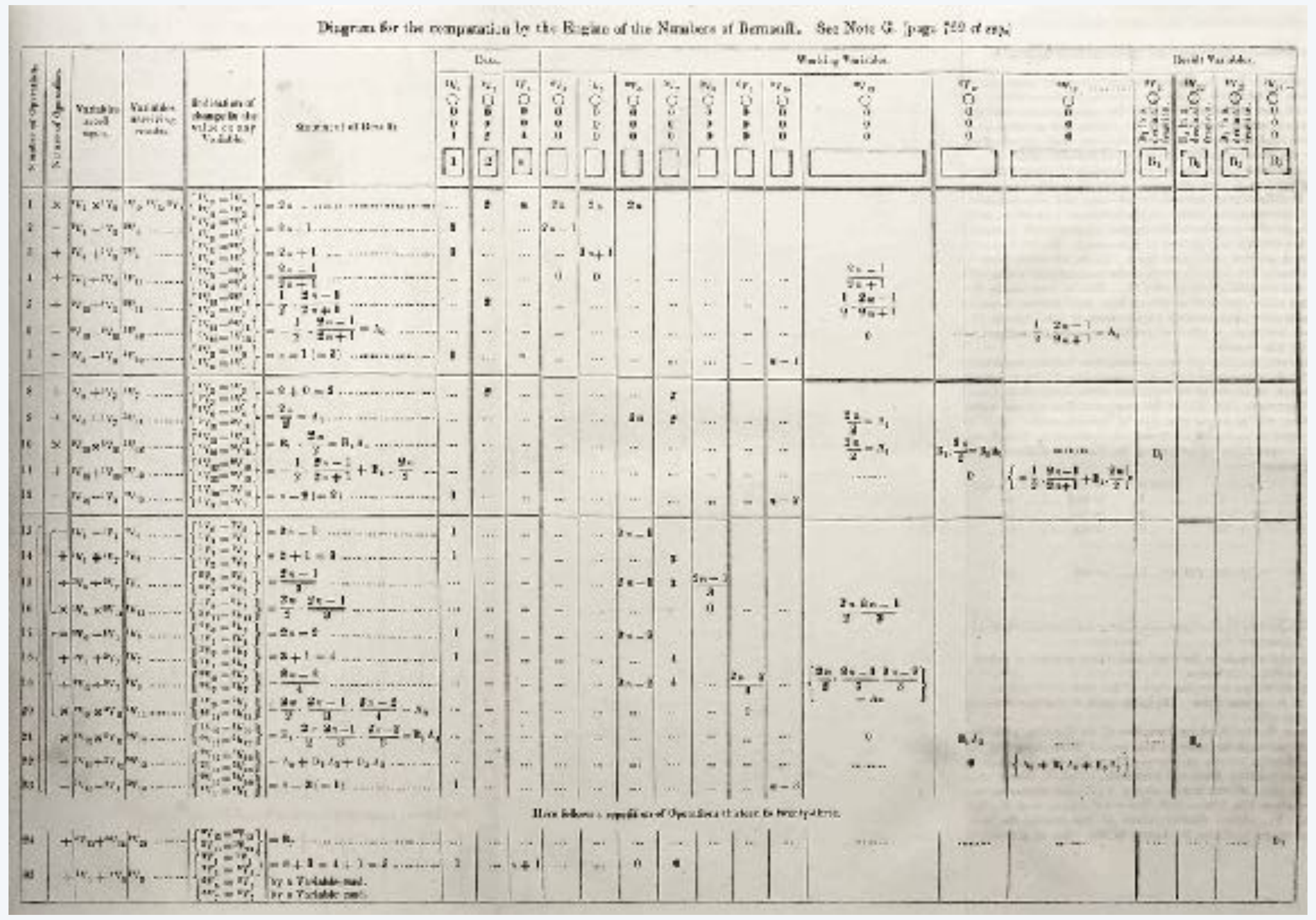
how many times do you have to turn the crank?



https://vimeo.com/49080293

Running time



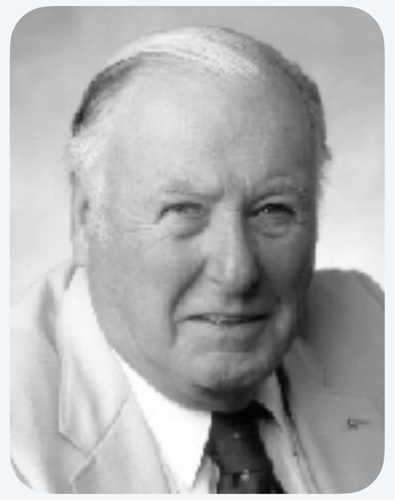


Ada Lovelace's algorithm
to compute Bernoulli numbers
on Analytical Engine (1843)

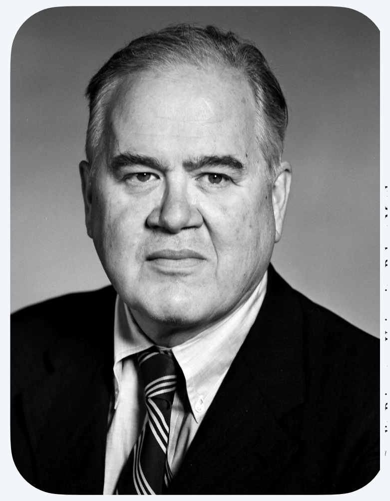
An algorithmic success story

Goal. Multiply two polynomials of degree n.

$$(x^3 + x^2 - 2x + 1) \cdot (3x^3 - x^2 + 2x + 1) \;=\; 3x^6 + 2x^5 - 5x^4 + 8x^3 - 4x^2 + 1$$

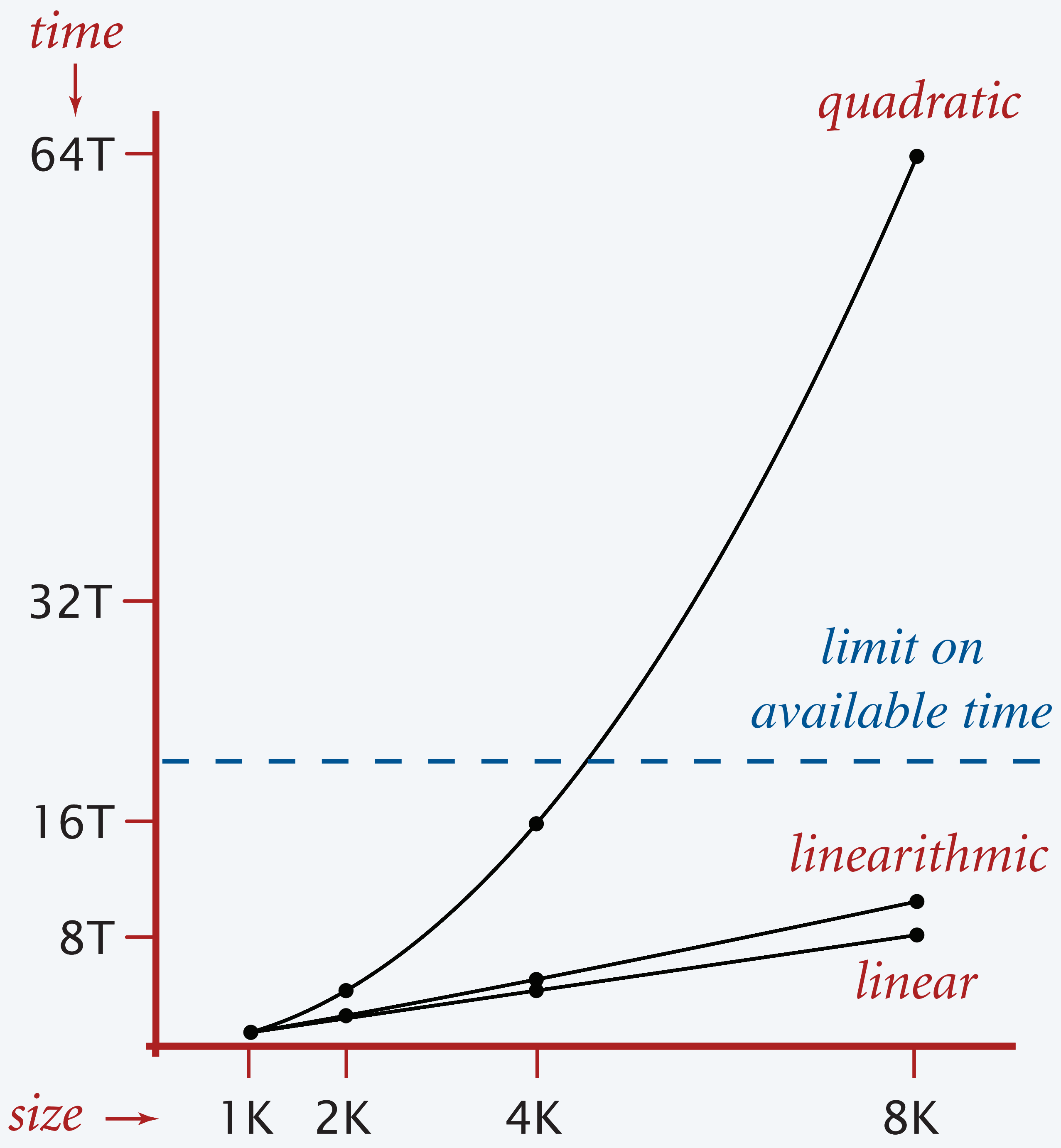


James Cooley



John Tukey

- Applications: JPEG compression, MRI, astrophysics, and more.
- Grade-school algorithm: $\Theta(n^2)$ operations.
- FFT algorithm: $\Theta(n \log n)$ operations, enabling modern technology.



Another algorithmic success story?

Why DeepSeek Could Change What Silicon Valley Believes About A.I.

A new A.I. model, released by a scrappy Chinese upstart, has rocked Silicon Valley and upended several fundamental assumptions about A.I. progress.

Listen to this article · 9:13 min Learn more

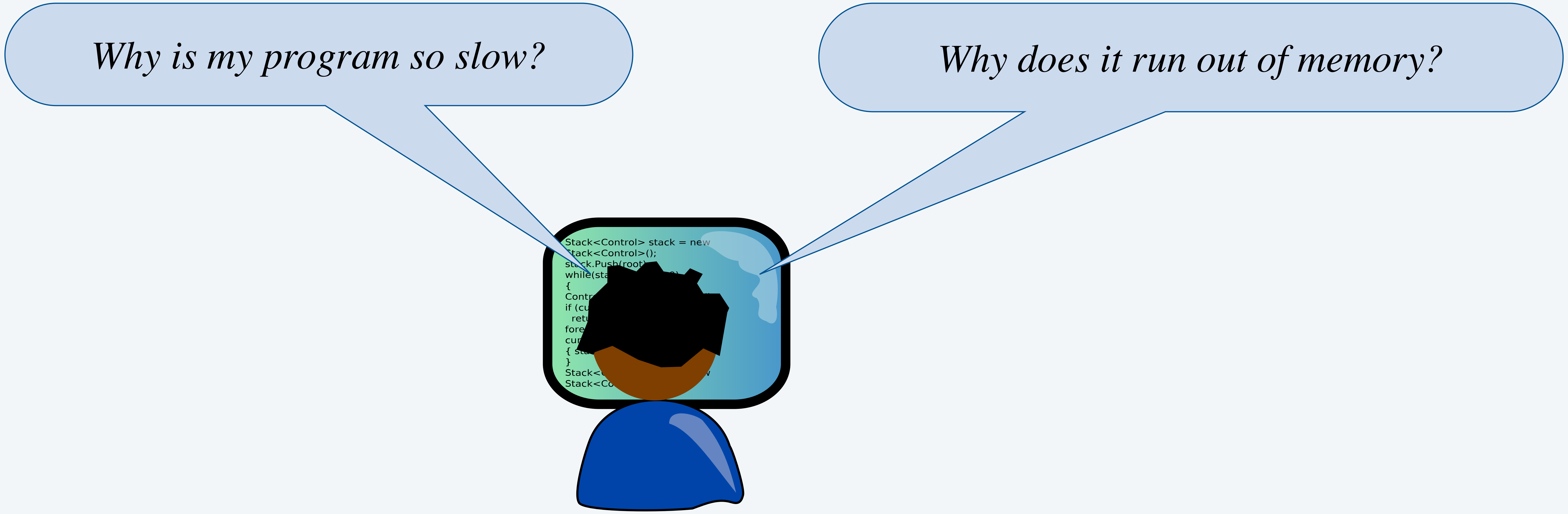
Share full article

457

The core challenge

Q1. Will my program handle on large, real-world inputs?

Q2. If not, how can I analyze and improve its performance?



Our approach: a combination of experiments and mathematical modeling.

Example: 3-SUM problem

Goal. Given an array of n distinct integers, count triples $i < j < k$ such that $a[i] + a[j] + a[k] = 0$.

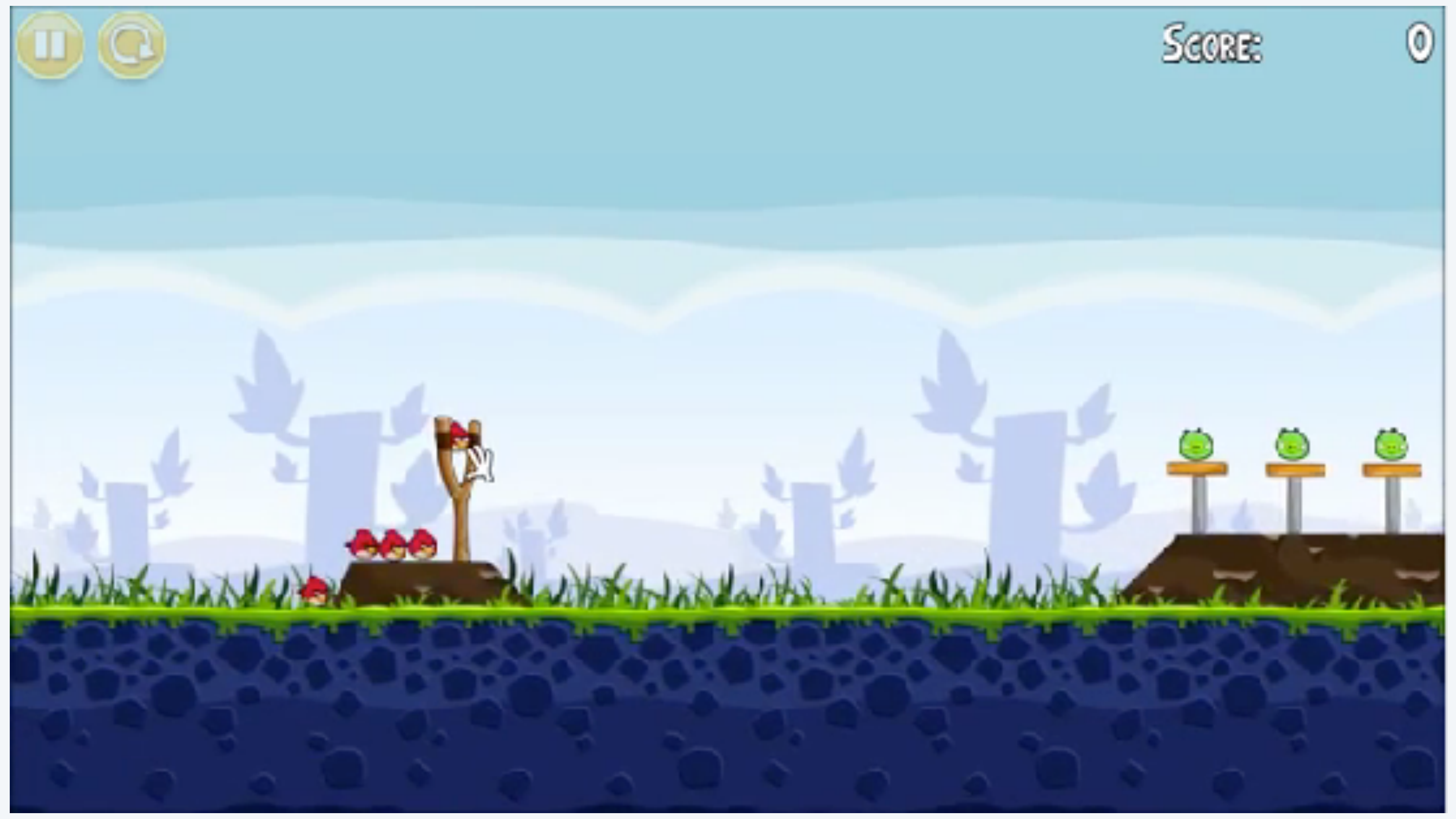
```
~/cos226/analysis> more 8ints.txt
8
30 -40 -20 -10 40 0 10 5

~/cos226/analysis> 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. Arises in in computational geometry (and even in computer games!)

Open problem. What is the optimal running time for solving 3-SUM ?



3-SUM problem: 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));
   }
}
```

count distinct triples that sum to 0

assume no integer overflow

https://algs4.cs.princeton.edu

1.4 Analysis of Algorithms

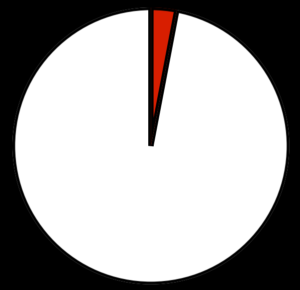
- *introduction*
- *running time (experimental analysis)*
- *running time (mathematical models)*
- *memory usage*

Measuring running time

Experiment. Measure the program's running time on inputs of different sizes.

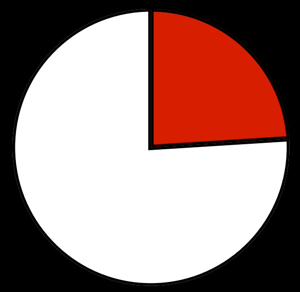
Observation. The running time $T(n)$ increases with the input size n.

```
~/cos226/analysis> java ThreeSum 1Kints.txt
70
```



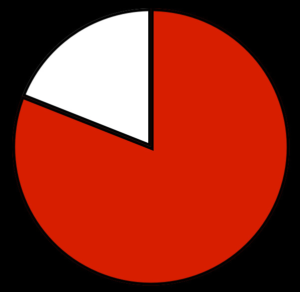
tick tick

```
~/cos226/analysis> java ThreeSum 2Kints.txt
528
```



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

```
~/cos226/analysis> java ThreeSum 3Kints.txt
1670
```



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

Measuring running time

Experiment. Measure the program's running time on inputs of different sizes.

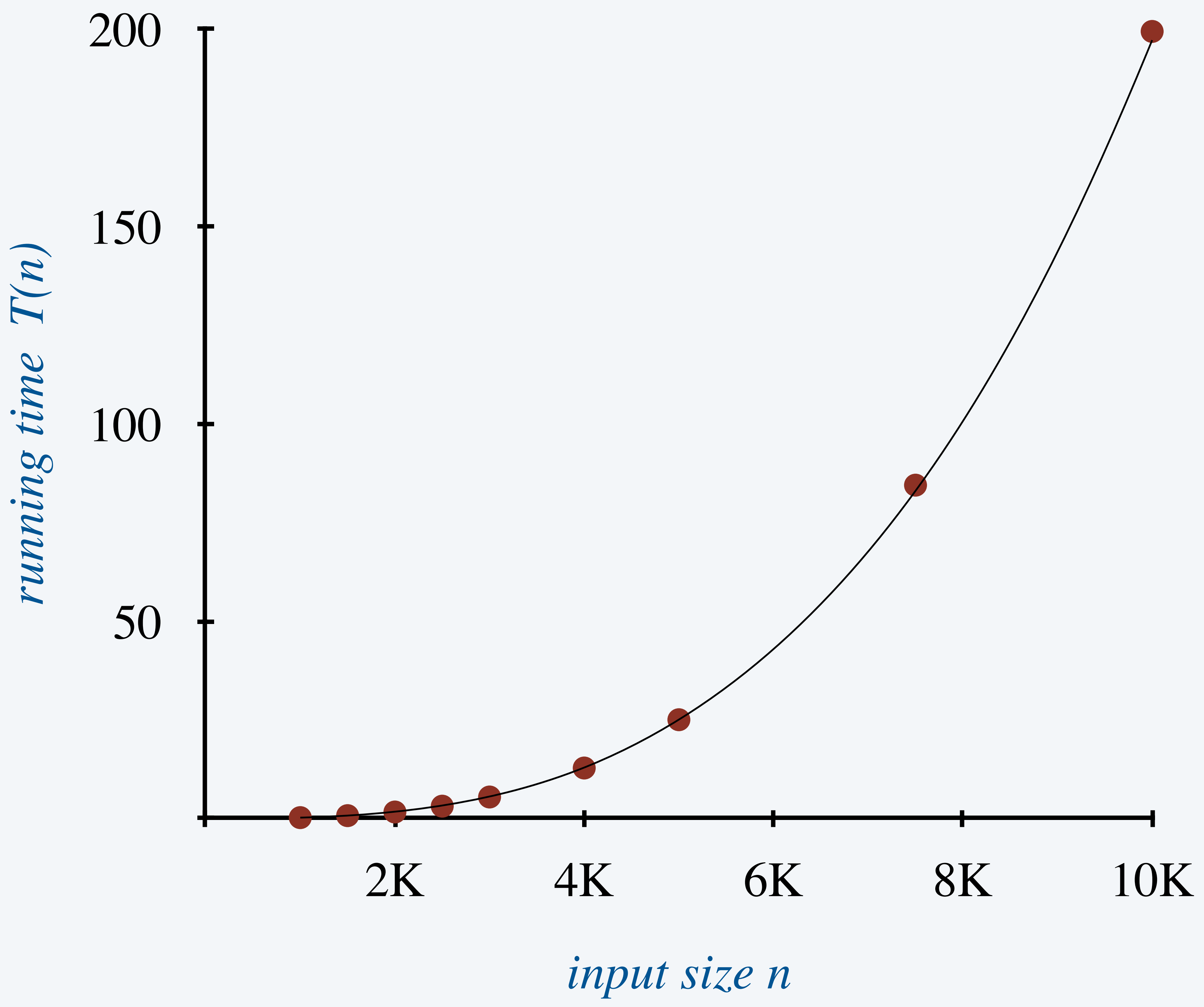
Observation. The running time $T(n)$ increases with the input size n.

n	time (seconds) †
1,000	0.21
1,500	0.71
2,000	1.63
2,500	3.11
3,000	5.43
4,000	12.8
5,000	25.0
7,500	84.4
10,000	199.3

† *Apple M2 Pro with 32 GB memory running OpenJDK 11 on MacOS Ventura*

Data analysis: running time vs. input size

Visualization. Plot the running time $T(n)$ versus the input size n.



Hypothesis. The running time follows a power law: $T(n) = a \times n^b$ seconds.

Questions. How can we test this hypothesis? How can we estimate a and b ?

Answer. Doubling test, $\frac{T(n)}{T(n/2)} = 2^b$.

Machine invariance

Hypothesis. For a fixed algorithm, the running times on different computers are the same up to a multiplicative constant factor.

Note. That constant factor can be large, sometimes several orders of magnitude.

1970s
(VAX-11/780)

10,000× *faster*

2020s
(Macbook Pro M2)

What affects the running time?

System independent effects.

- Algorithm.
- Input data.

determines exponent b in power law $T(n) = a \times n^b$

System dependent effects.

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

determines leading coefficient a in power law $T(n) = a \times n^b$

Bad news. Getting accurate timing measurements can be difficult.

system-dependent effects can introduce noise

https://algs4.cs.princeton.edu

1.4 Analysis of Algorithms

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

Mathematical model of running time

Model. The running time = $\sum$ (frequency of operation) $\times$ (cost of operation).

- Frequency of operation: depends on the algorithm and the specific input.
- Cost of operation: depends on hardware, software, system, and low-level implementation details.


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."

Warning. For arbitrary programs, frequencies may be impossible to determine. ⟵ *halting problem*

Example: one-sum problem

Q. How many operations does this code perform as a function of the input size n ?

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

operation	cost (ns) †	frequency
variable declaration	2 / 5	2
assignment statement	1 / 5	2
less than compare	1 / 5	$n + 1$
equality test	1 / 10	n
array read	1 / 10	n
increment	1 / 10	n to $2n$

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

painful to count exactly

† *representative estimates* (*with a bit of poetic license*)

Simplification 1: cost model

Cost model. Pick one elementary operation as a proxy for running time. ← *array accesses, compares, API calls, floating-point operations, …*

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

← *"inner loop"*

operation	cost (ns) †	frequency
variable declaration	2 / 5	2
assignment statement	1 / 5	2
less than compare	1 / 5	$n + 1$
equality test	1 / 10	n
array read	1 / 10	n
increment	1 / 10	n to $2n$

← *cost model = array accesses*

Simplification 2: asymptotic notation

Tilde notation. Ignore lower-order terms.

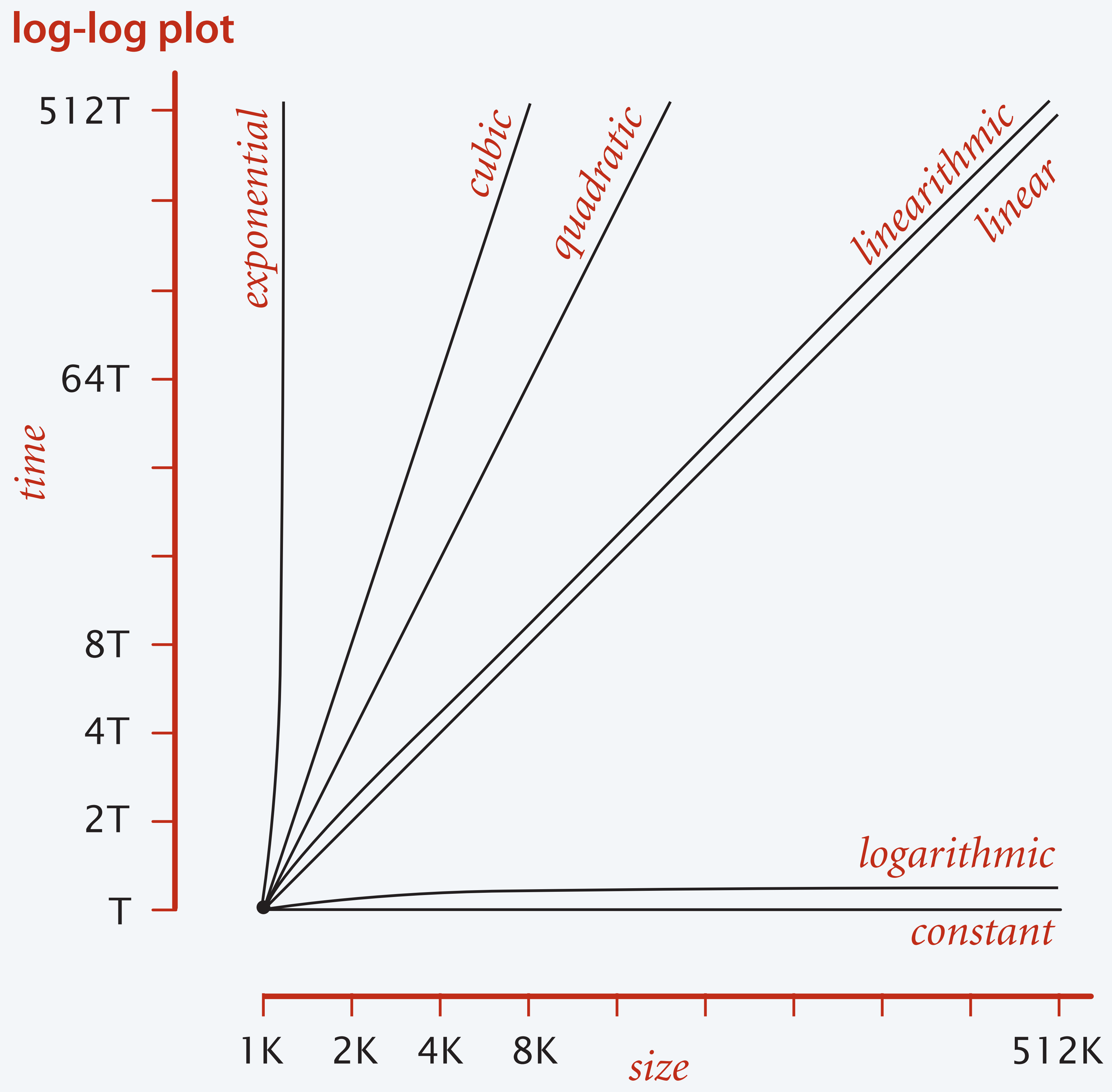
Big Theta notation. Ignore both lower-order terms and the leading constant.

rigorous definitions involve limits

"order of growth"

function	tilde notation	big Theta
$4n^5 + 20n^3 + 16$	$\sim 4n^5$	$\Theta(n^5)$
$0.01n^2 + 10n^{4/3} + 100\log^8 n$	$\sim 0.01n^2$	$\Theta(n^2)$
$2^n + n^5$	$\sim 2^n$	$\Theta(2^n)$
$\frac{1}{6}n^3 \underbrace{- \frac{1}{2}n^2 + \frac{1}{3}n}_{\text{discard lower-order terms}}$	$\sim \frac{1}{6}n^3$	$\Theta(n^3)$

(*e.g.*, n = 1,000: 166.667 *million* vs. 166.167 *million*)



Typical orders of growth

Rationale.

- For large n, lower-order terms have negligible effect.
- For small n, the value is so small that we don't care.

Analysis of algorithms: poll 1

Which of the following correctly describes the function $f(n) = n \log_2 n + 3n^2 + 10n$ **?**

A. $\sim 10\, n$

B. $\sim n \log_2 n$

C. $\sim n^2$

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

E. $\Theta(n^2)$

Example: 2-SUM analysis

Q. Approximately 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++;
```

"inner loop"

$$\underbrace{\overset{\substack{i=0\\ j=1,\ldots,n-1}}{\downarrow}(n-1) + \overset{\substack{i=1\\ j=2,\ldots,n-1}}{\downarrow}(n-2) + \overset{\substack{i=2\\ j=3,\ldots,n-1}}{\downarrow}(n-3) + \ldots + \overset{\substack{i=n-2\\ j=n-1}}{\downarrow}1 + \overset{\substack{i=n-1\\ \text{no } j}}{\downarrow}0}_{=\frac{n(n-1)}{2}}$$

Step 1. Pick a cost model: array accesses.

Step 2. Count array accesses: $2 \times \dfrac{n(n-1)}{2} \sim 1\, n^2$.

body inner loop makes 2 array accesses

Nested loops.

- Independent loops: analyze separately and multiply.
- Dependent loops: write a sum (and simplify).

Triangular sum

Claim. $0 + 1 + \ldots + (n-2) + (n-1) = \frac{1}{2}\, n\,(n-1)$.

Proof.

$n-1$

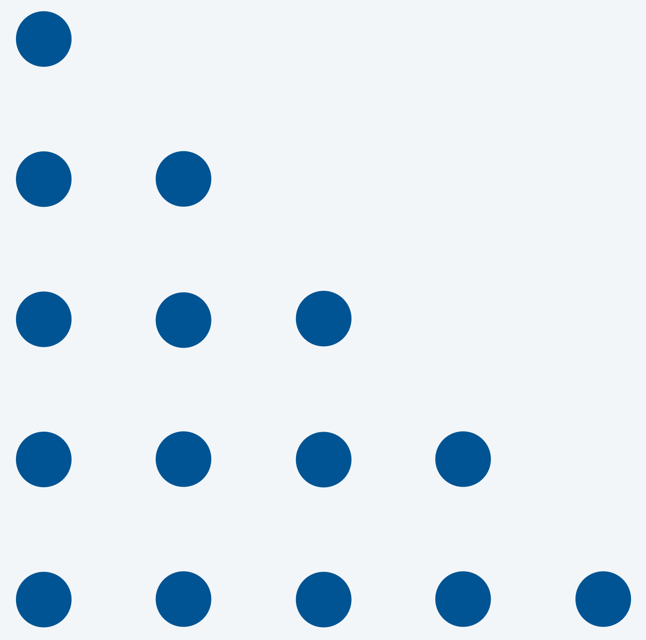
$n-1$

$n-1$

$$(n-1) + (n-2) + (n-3) + \ldots + 2 + 1 + 0 = (n-1) \times (n/2)$$

sum of each pair

number of pairs (assume n is even)



$1 + 2 + 3 + \ldots + n - 1$

Example: 3-SUM analysis

Q. Approximately 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++)
      for (int k = j+1; k < n; k++)
         if (a[i] + a[j] + a[k] == 0)
            count++;
```

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

see COS 240

Step 1. Pick a cost model: array accesses.

Step 2. Count the number of array accesses: $\Theta(n^3)$.

Bottom line. Using a cost model and asymptotic notation makes analysis manageable.

Common orders of growth

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

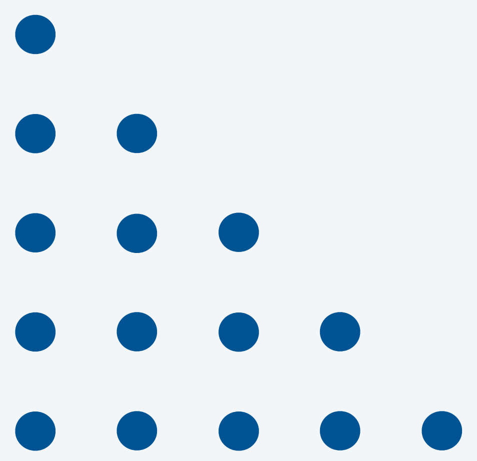
$\Theta(n^2)$	😕	**quadratic**	`for (int i = 0; i < n; i++)`
			`for (int j = 0; j < n; j++)`
			`{ ... }`
				double loop	*check all pairs*

$\Theta(n^3)$	☹️	**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*

$\Theta(2^n)$	👿	**exponential**	*towers of Hanoi*	brute-force search	*check all subsets*

Useful discrete sums and identities

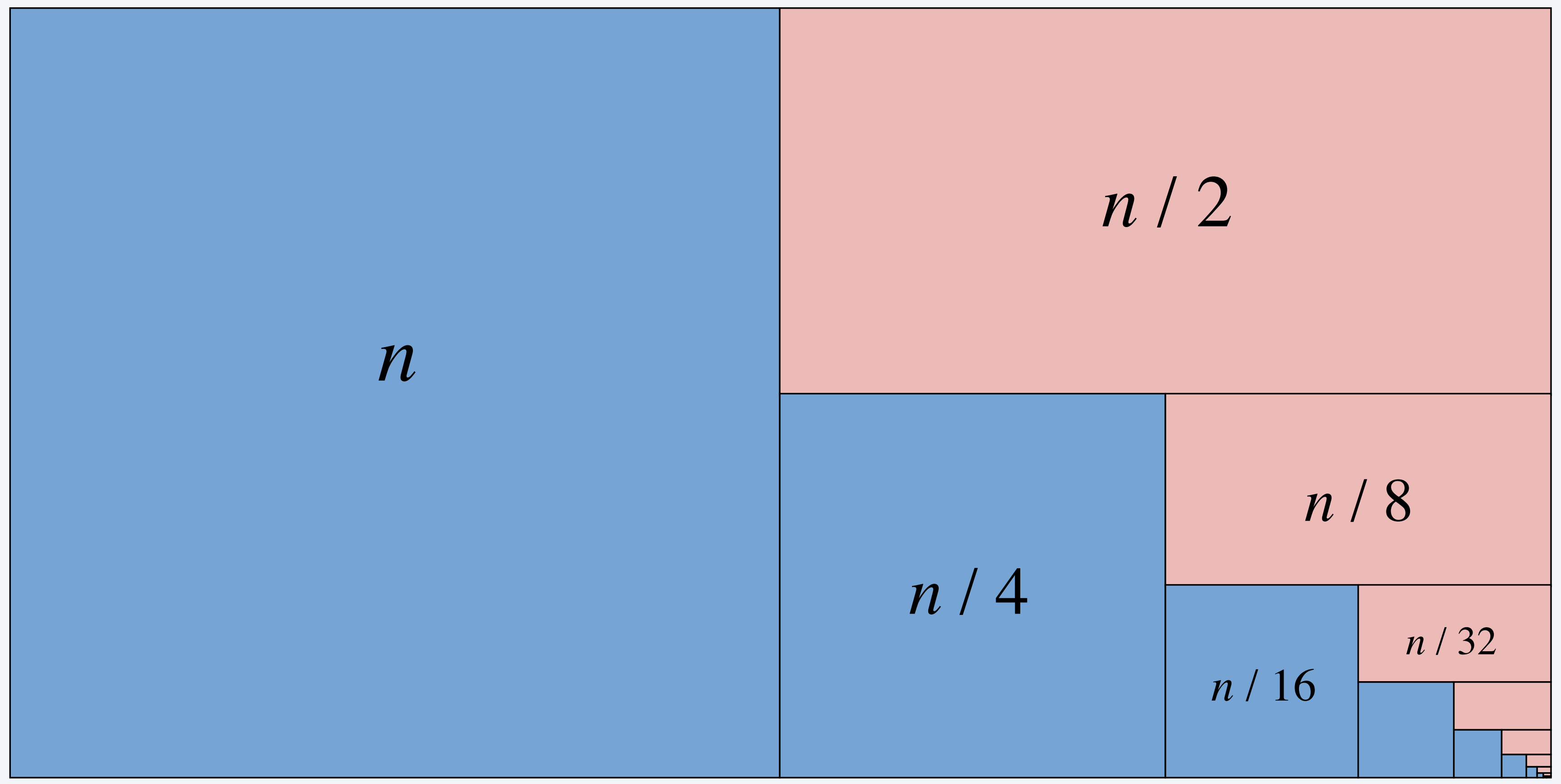
Triangular sum. $1 + 2 + 3 + \ldots + n \;\sim\; \frac{1}{2} n^2$



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

n is a power of 2

Geometric sum′. $n + \dfrac{n}{2} + \dfrac{n}{4} + \ldots + 1 \;=\; 2n - 1$



Logarithmic identities. $\log_2 x + \log_2 y \;=\; \log_2(xy)$

$$\log_b x \;=\; \frac{\log_2 x}{\log_2 b}$$ ← *change of base*

Analysis of algorithms: poll 2 (midterm f25)

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

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

*how would the answer change if k = k * 4 ?*

A. $\sim \frac{1}{2} n^2 \log_2 n$

B. $\sim \frac{1}{2} n^4 \log_2 n$

C. $\sim n^4 \log_2 n$

D. $\sim \frac{1}{2} n^6$

E. $\sim 2 n^6$

What is the order of growth of the 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++;
```

A. $\Theta(n)$

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

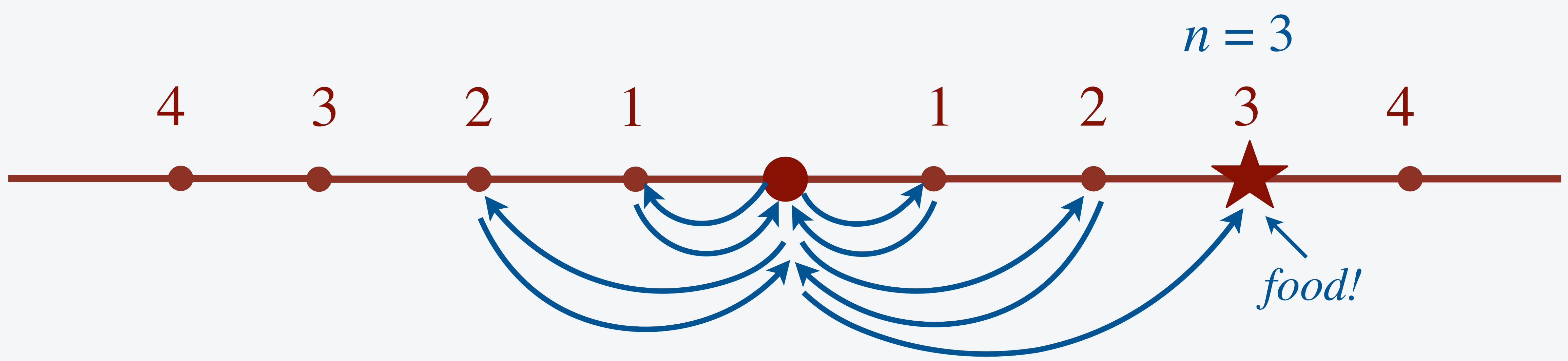
C. $\Theta(n^2)$

D. $\Theta(2^n)$

Example: hungry rat (midterm f22)

A rat in a sewer pipe is searching for food. If the nearest food source is n steps to the right of its starting location, how many steps will it take to reach it using the given strategy?

Strategy 1: Take 1 step right, return to start, take 1 step left, return to start.
Repeat with 2, 3, 4, 5… steps until food found.



A rat in a sewer pipe is searching for food. If the nearest food source is n steps to the right of its starting location, how many steps will it take to reach it using the given strategy?

assume n is a power of 2

Strategy 2: Take 1 step right, return to start, take 1 step left, return to start.
Repeat with 2, 4, 8, 16… steps until food found.

A. $\Theta(\log n)$

B. $\Theta(n)$

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

D. $\Theta(n^2)$

E. $\Theta(2^n)$

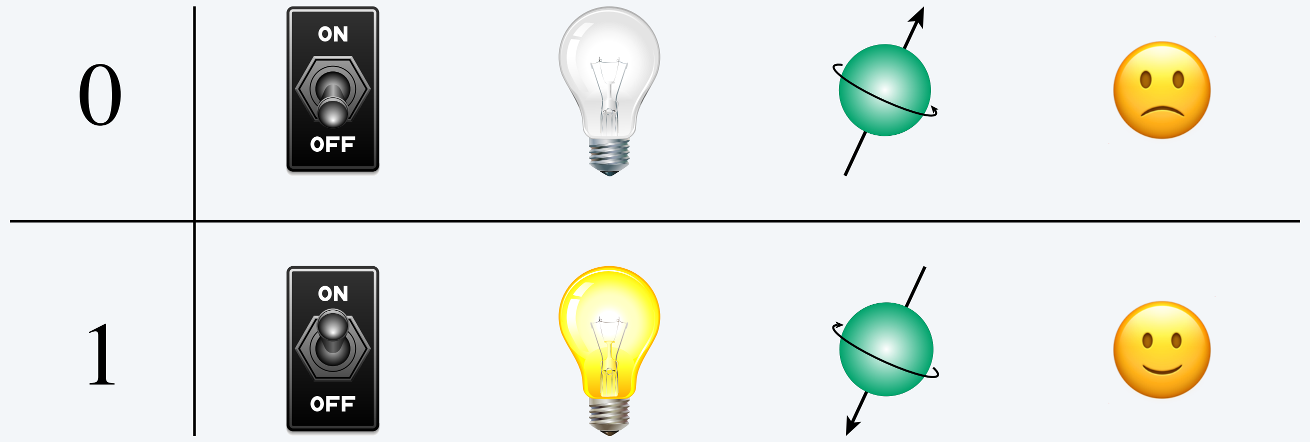
https://algs4.cs.princeton.edu

1.4 Analysis of Algorithms

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

Memory basics: bits, bytes, and pointers

Bit. A single binary digits (0 or 1).



term	symbol	size
byte	B	8 bits
kilobyte	KB	10^3 bytes
megabyte	MB	10^6 bytes
gigabyte	GB	10^9 bytes
terabyte	TB	10^{12} bytes

some systems use powers of 2
(*e.g.*, 1 MB = 2^{20} *bytes*)

Assumption. Running on a 64-bit machine with 8-byte pointers.

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

64-bit

Typical memory usage of primitive types and arrays in Java

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

primitive types

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

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 array of arrays)

type	bytes
object reference	8

64-bit machine

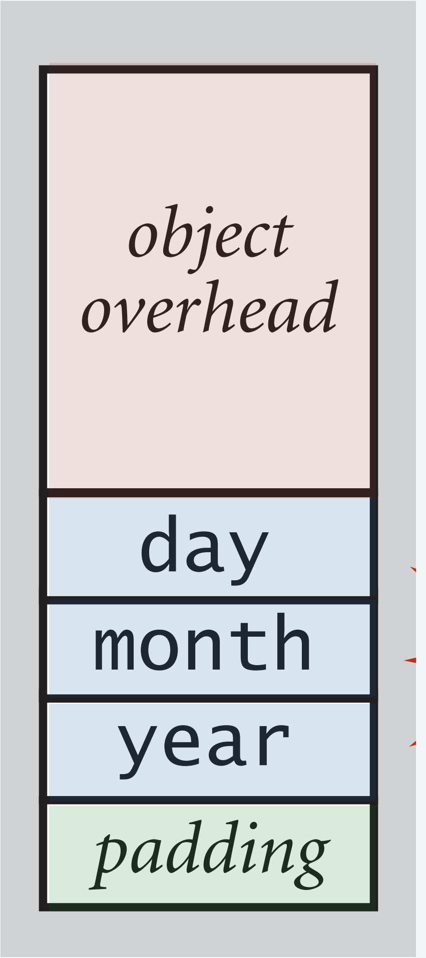
Typical memory usage for objects in Java

Objects memory = sum of memory for instance variables + overheads

Ex. Each *Date* object uses 32 bytes of memory.

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

   ...
}
```



16 bytes (object overhead) ← *class pointer, garbage collector bits, lock state, hash code, …*

4 *bytes* (`int`)

4 *bytes* (`int`)

4 *bytes* (`int`)

4 *bytes (padding, round to a multiply of 8 bytes)*

32 *bytes*

Array declaration is 8 bytes.

```
Date[] dates;
```
← *reference*

When *dates* contains n elements, it uses $\Theta(n)$ bytes.

Analysis of algorithms: poll 5

How much memory does a `WeightedQuickUnionUF` object 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;
   }
   ...
}
```

Credits

image	source	license
Charles Babbage	The Illustrated London News	public domain
Babbage Enginine in Operation	xRez Studio	
Algorithm for the Analytical Engine	Ada Lovelace	public domain
Ada Lovelace and Book	Moore Allen & Innocent	
Galaxies Colliding	SaltyMikan	
James Cooley	IEEE	
John Tukey	Princeton University	
Programmer Icon	Jaime Botero	public domain
Head in the Clouds	Ellis Nadler	education
Student Raising Hand	classroomclipart.com	educational use
Running Time	pano.si	
Analog Stopwatch	Adobe Stock	education license

Credits

image	source	license
Analog Stopwatch	Adobe Stock	education license
Apple M4 Chip	Apple	
Macbook Pro M2	Apple	
Scientific Method	Sue Cahalane	by author
Laboratory Apparatus	pixabay.com	public domain
Dissected Rat	Allen Lew	CC BY 2.0
Harmonic Integral	Wikimedia	public domain
Geometric Series	Wikimedia	CC BY-SA 3.0
Recursive Load	Marek Bennett	
The Yoda of Silicon Valley	New York Times	
Babbage's Analytical Engine	Science Museum, London	CC BY-SA 2.0
Alan Turing	Science Museum, London	

A final thought

“ *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 give them various weights.*” — Alan Turing (1947)