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

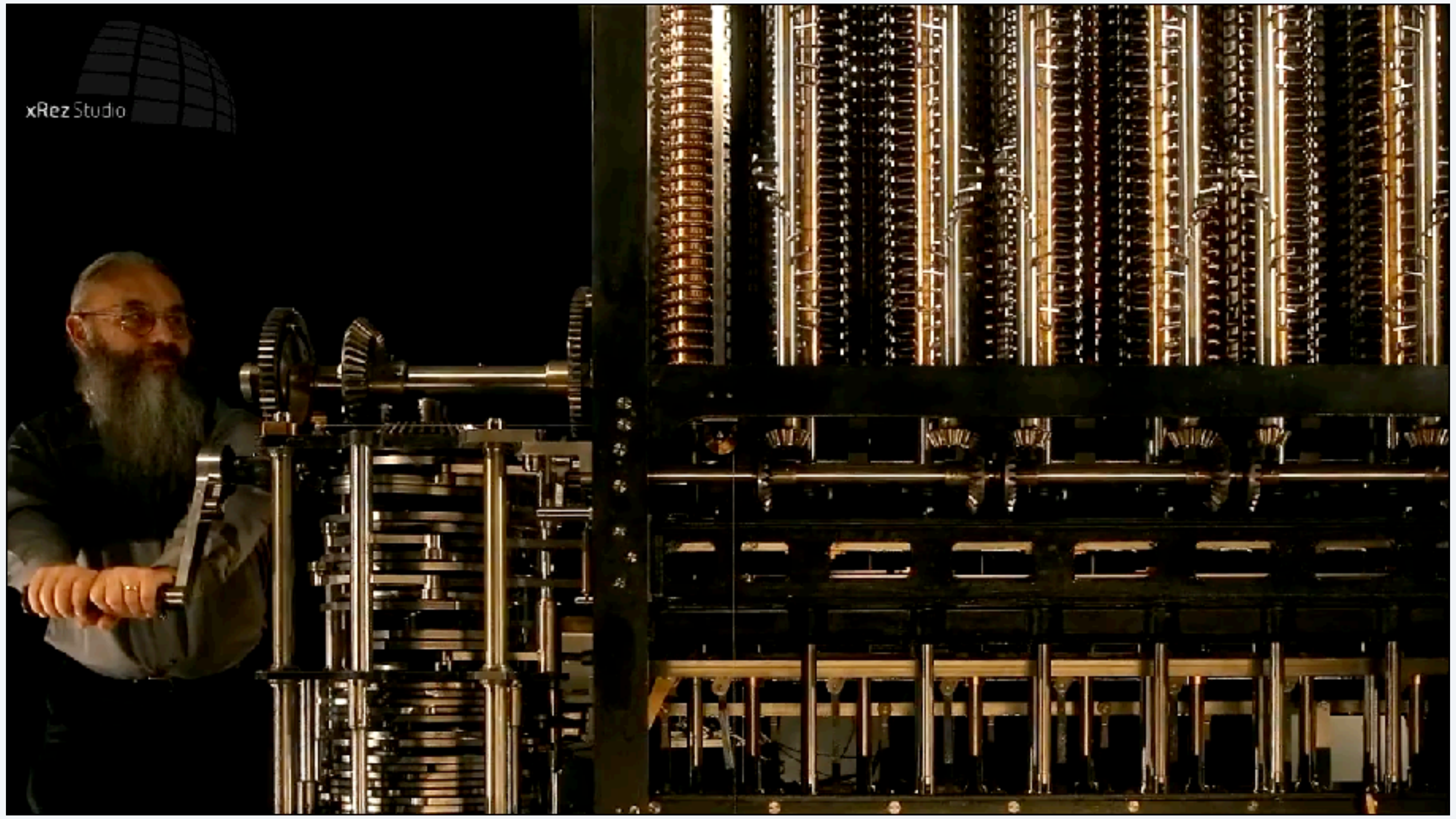
1.4 Analysis of Algorithms

- ‣ *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)

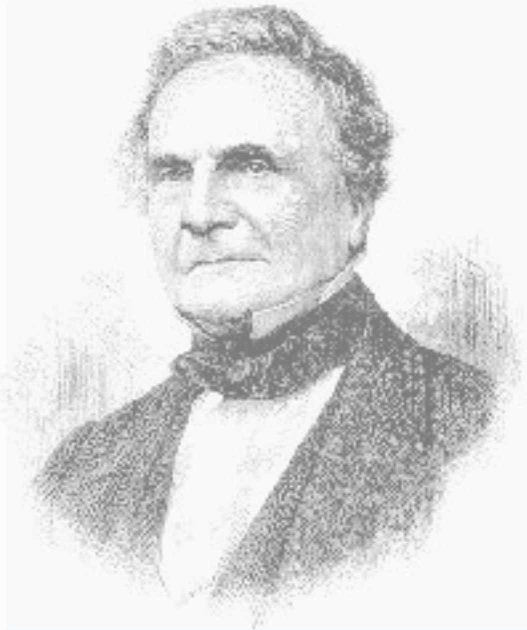
how many times do you have to turn the crank?

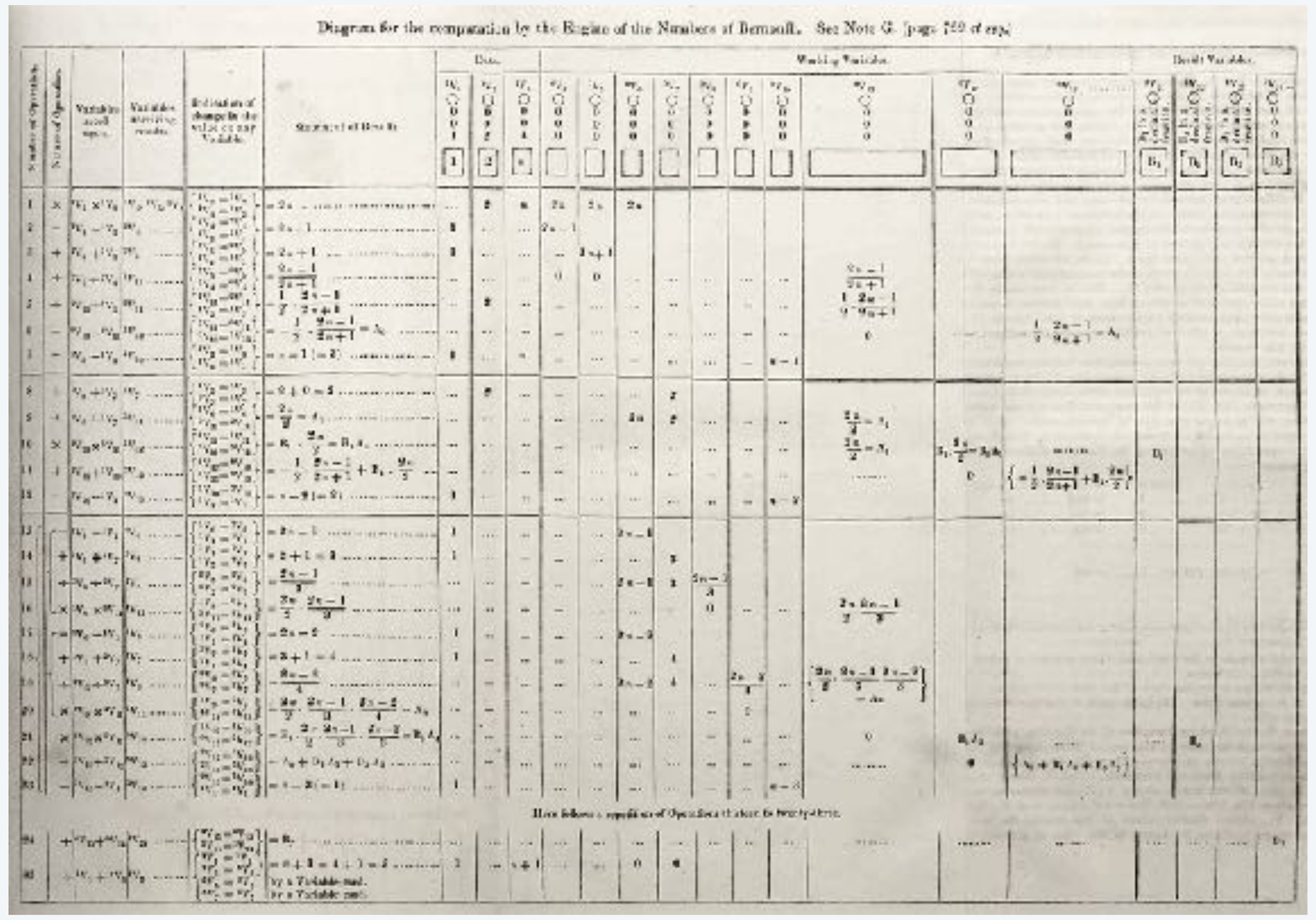


https://vimeo.com/49080293

Running time

> “ *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)





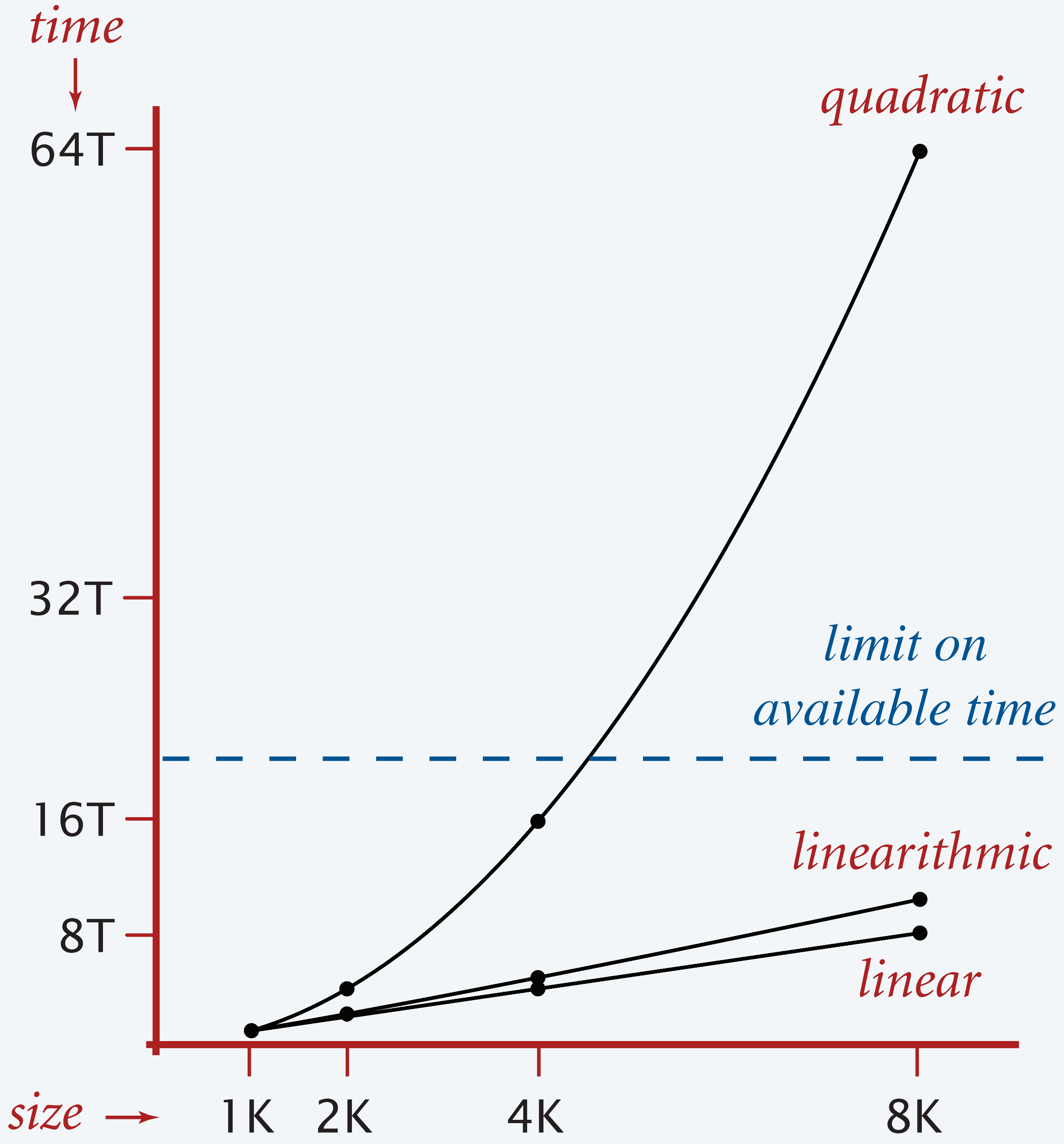
Ada Lovelace's algorithm
to compute Bernoulli numbers
on Analytic Engine (1843)

An algorithmic success story

N-body simulation.

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

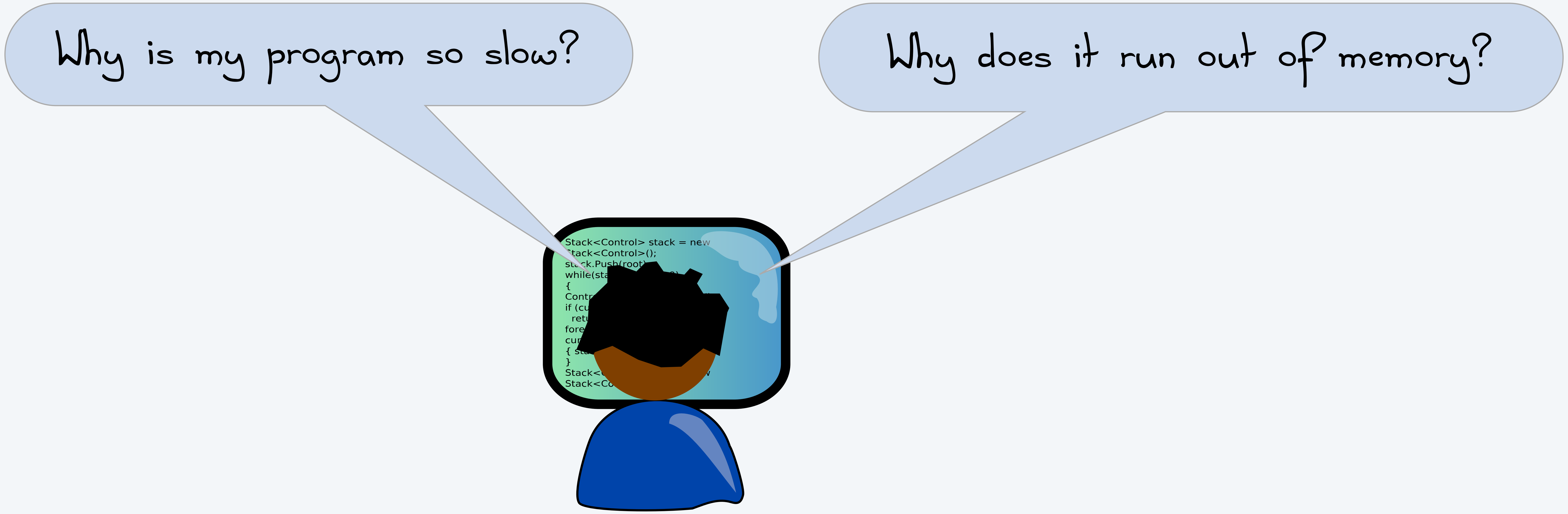
Andrew Appel
PU '81



The challenge

Q1. Will my program be able to solve a large practical input?

Q2. If not, how might I understand its performance characteristics so as to improve it?



Our approach. Combination of experiments and mathematical modeling.

Example: 3-SUM

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

```
~/cos226/3sum> more 8ints.txt
8
30 -40 -20 -10 40 0 10 5

~/cos226/3sum> 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

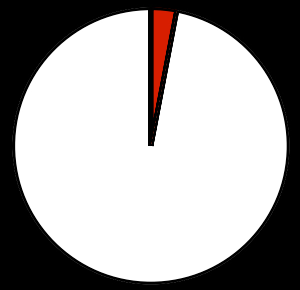
- *introduction*
- *running time (experimental analysis)*
- *running time (mathematical models)*
- *memory usage*

Measuring the running time

Running time. Run the program for inputs of varying size; measure running time.

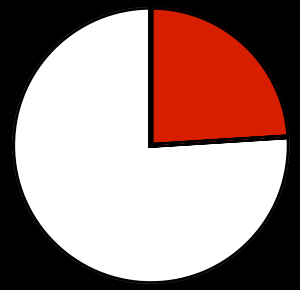
Observation. The running time $T(n)$ grows as a function of the input size n.

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



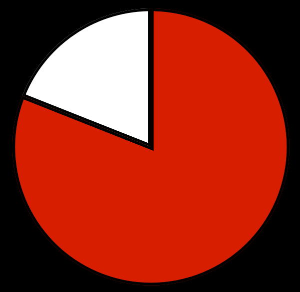
tick tick

```
~/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 the running time

Running time. Run the program for inputs of varying size; measure running time.

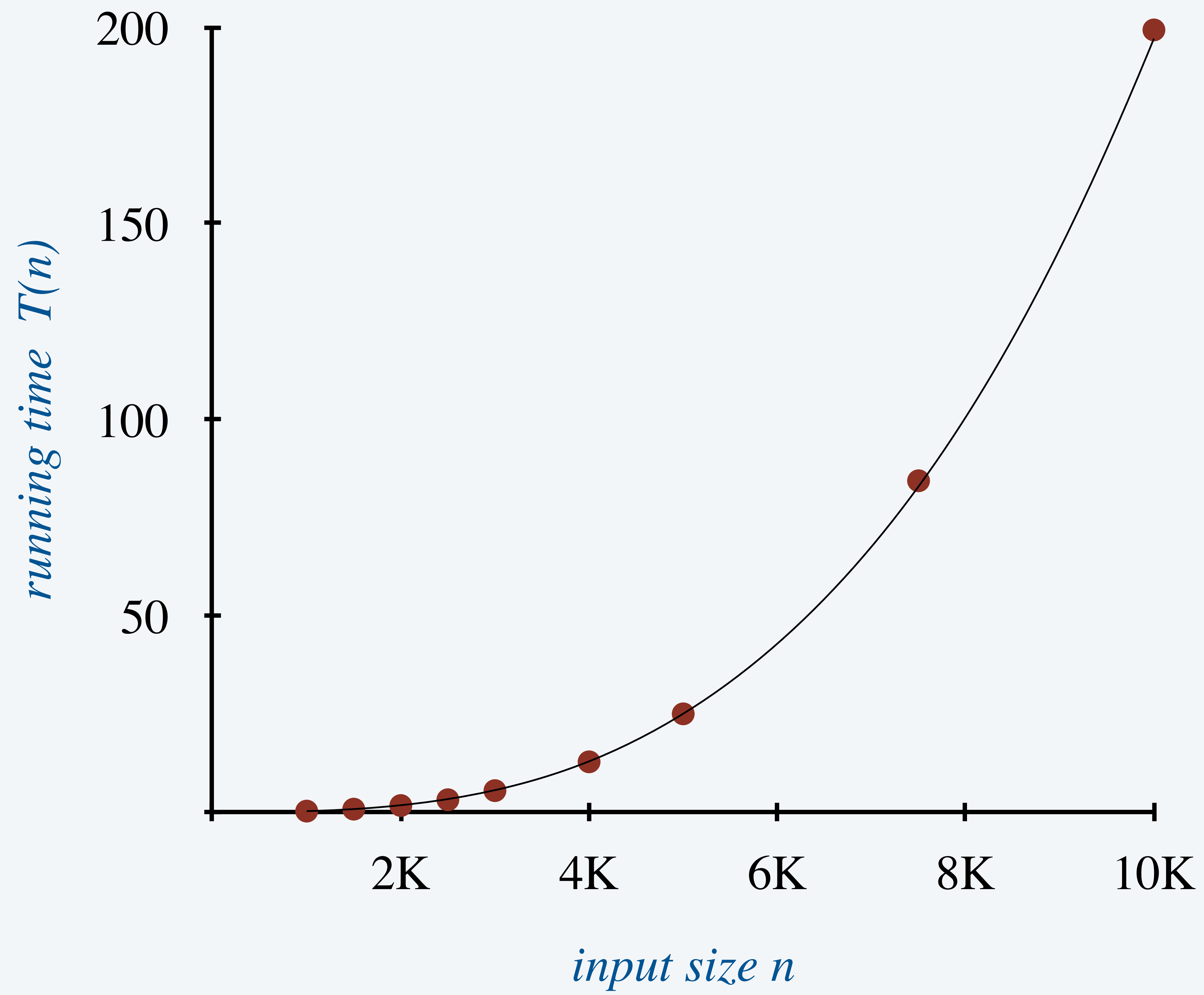
n	time (seconds) †
1,000	0.21
1,500	0.71
2,000	1.63
2,500	3.11
3,000	5.43
4,000	12.8
5,000	25.0
7,500	84.4
10,000	199.3

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

Data analysis: standard plot

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



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

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

Doubling test: estimating the exponent b

Doubling test. Run program, doubling the size of the input.

- Assume running time satisfies the "power law" $T(n) = a \times n^b$.
- Estimate $b = \log_2$ ratio.

n	time (seconds) †	ratio	$\log_2$ ratio
500	0.05	–	–
1,000	0.21	4.20	2.07
2,000	1.63	7.76	2.96
4,000	12.8	7.85	2.97
8,000	103.1	8.05	3.01
16,000	819.0	7.94	2.99

$\log_2 (103.1 / 12.8) = 3.01$

seems to converge to a constant $b \approx 3.0$

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

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

why the $\log_2$ ratio works

Doubling test: estimating the leading coefficient a

Doubling test. Run program, doubling the size of the input.

- Assume running time satisfies $T(n) = a \times n^b$.
- Estimate $b = \log_2$ ratio.
- Estimate a by solving $T(n) = a \times n^b$ for a sufficiently large value of n.

n	time (seconds) †	ratio	$\log_2$ ratio	
500	0.05	–	–	
1,000	0.21	4.20	2.07	
2,000	1.63	7.76	2.96	
4,000	12.8	7.85	2.97	
8,000	103.1	8.05	3.01	
16,000	819.0	7.94	2.99	$819.0 = a \times 16{,}000^3 \Rightarrow a = 2.00 \times 10^{-10}$

Hypothesis. Running time is about $2.00 \times 10^{-10} \times n^3$ seconds.

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

Order of growth

Hypothesis. Running times on different computers differ by a constant factor.

Note. That factor can be several orders of magnitude.

10,000× *faster*

1970s
(VAX-11/780)

2020s
(Macbook Pro M2)

Experimental algorithmics

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. 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)

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 frequency × cost for all 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++;
```

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)

tedious to count exactly

† *representative estimates (with some poetic license)*

Simplification 1: cost model

Cost model. Use some 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	$n + 1$
equal to compare	1/10	n
array access	1/10	n
increment	1/10	n to $2\ n$

← *cost model = array accesses*

Simplification 2: asymptotic notations

Tilde notation. Discard lower-order terms.

Big Theta notation. Discard lower-order terms and leading coefficient.

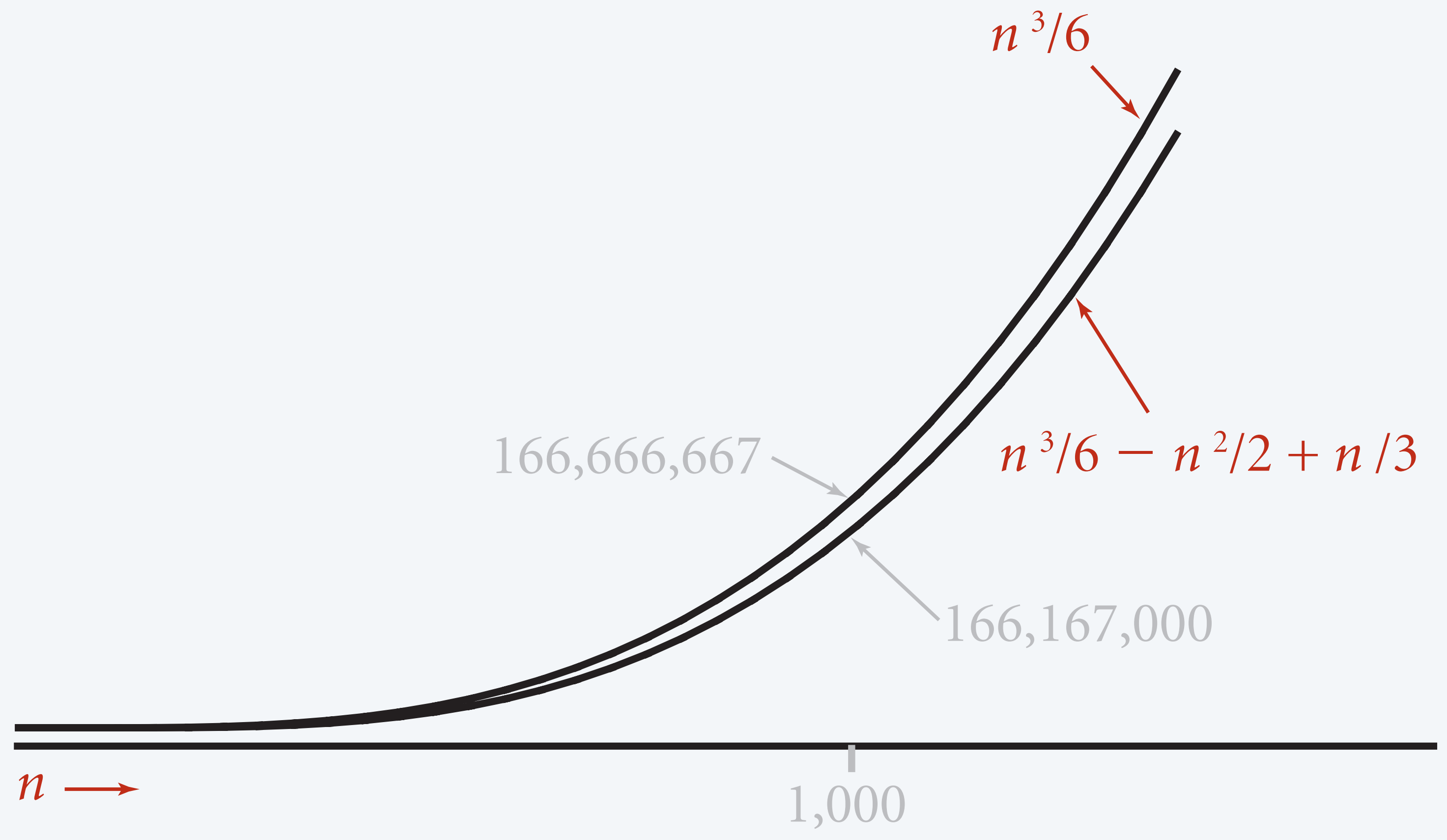
rigorous definitions involve limits

"order of growth"

function	tilde notation	big Theta
$4\,n^5 + 20\,n^3 + 16$	$\sim 4\,n^5$	$\Theta(n^5)$
$0.01\,n^2 + 100\,n^{4/3} - 56$	$\sim 0.01\,n^2$	$\Theta(n^2)$
$8\log^2(n) + 7\,n$	$\sim 7\,n$	$\Theta(n)$
$10\,n + 3\,n\log n$	$\sim 3\,n\log n$	$\Theta(n\log n)$
$2^n + n^5$	$\sim 2^n$	$\Theta(2^n)$

$$\tfrac{1}{6}\,n^3 \underbrace{-\ \tfrac{1}{2}\,n^2 + \tfrac{1}{3}\,n}_{\text{discard lower-order terms}}$$

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



Leading-term approximation

Rationale.

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

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++;
```

"inner loop"

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

B. $n\,(n-1)$

C. $2\,n^2$

D. $2\,n\,(n-1)$

Example: two-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++;
```

$$\underbrace{(n-1) + (n-2) + (n-3) \ldots + 2 + 1 + 0}_{\frac{1}{2}\, n\, (n-1)}$$

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

tedious to count exactly

$1/4\, n^2 + 13/20\, n + 13/10$ ns

to

$3/10\, n^2 + 3/5\, n + 13/10$ ns

Example: 2-SUM

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++;
```

$$\underbrace{(n-1) + (n-2) + (n-3) \ldots + 2 + 1 + 0}_{\frac{1}{2}\, n\,(n-1)}$$

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

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++;
```

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

see COS 240

A1. $\sim \frac{1}{2}n^3$ array accesses.

A2. $\Theta(n^3)$ array accesses.

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

Common order-of-growth classifications

order of growth	emoji	name	typical code framework	description	example
$\Theta(1)$	😍	**constant**	`a = b + c;`	statement	*add two numbers*
$\Theta(\log n)$	😎	**logarithmic**	`for (int i = n; i > 0; i /= 2)` `{ ... }`	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*

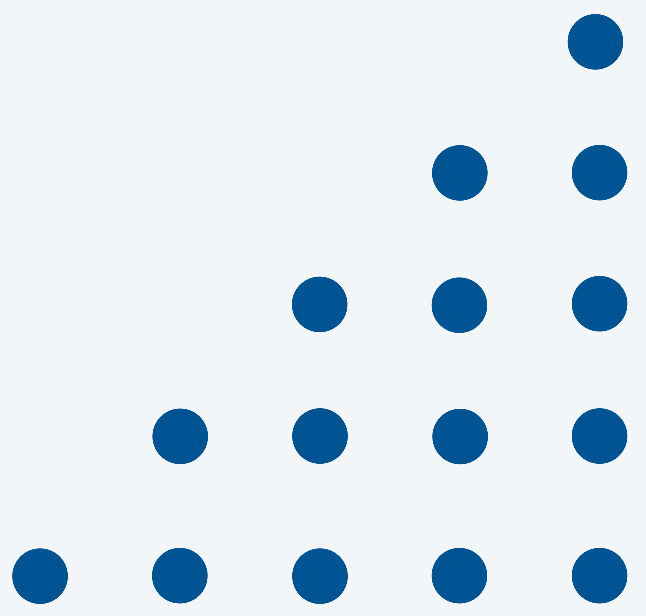
$\Theta(n^2)$	😕	**quadratic**	`for (int i = 0; i < n; i++)`
			`for (int j = 0; j < n; j++)`
			`{ ... }`
				double loop	*check all pairs*

$\Theta(n^3)$	☹️	**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*

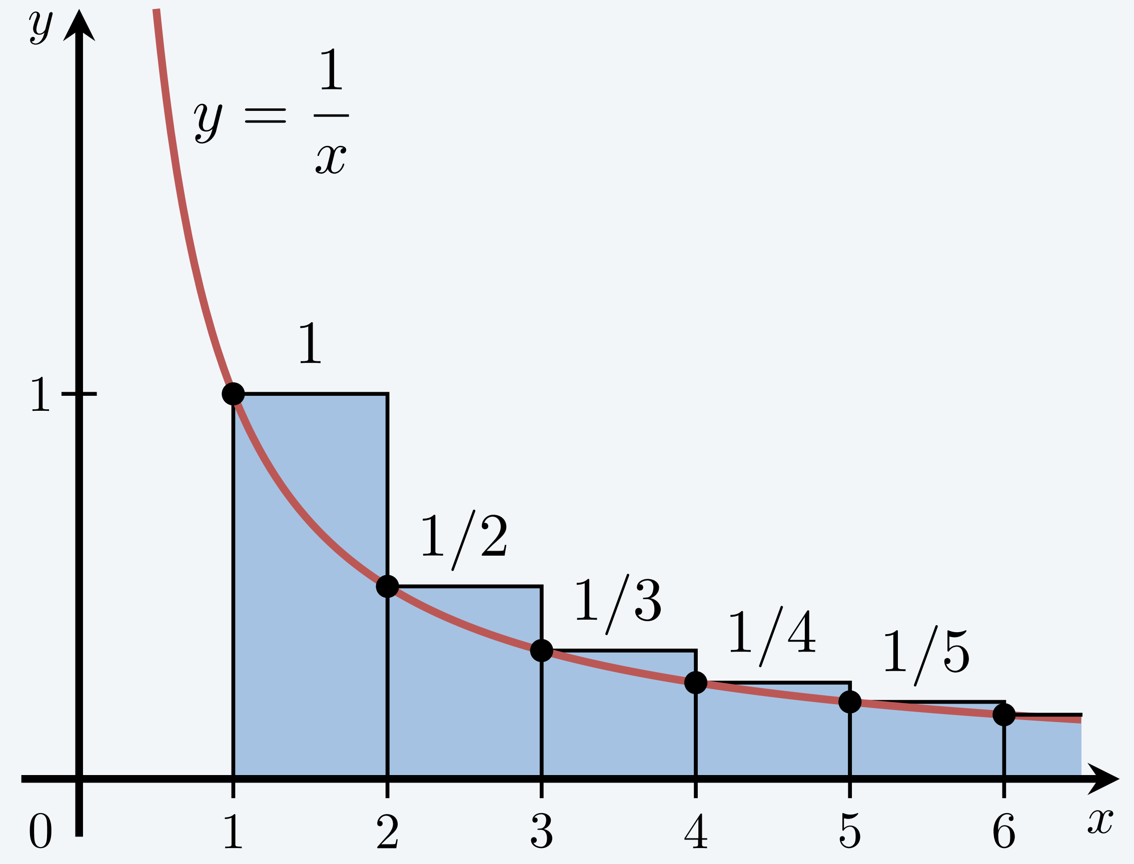
$\Theta(2^n)$	👿	**exponential**	*towers of Hanoi*	exhaustive search	*check all subsets*

Some useful discrete sums and approximations

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



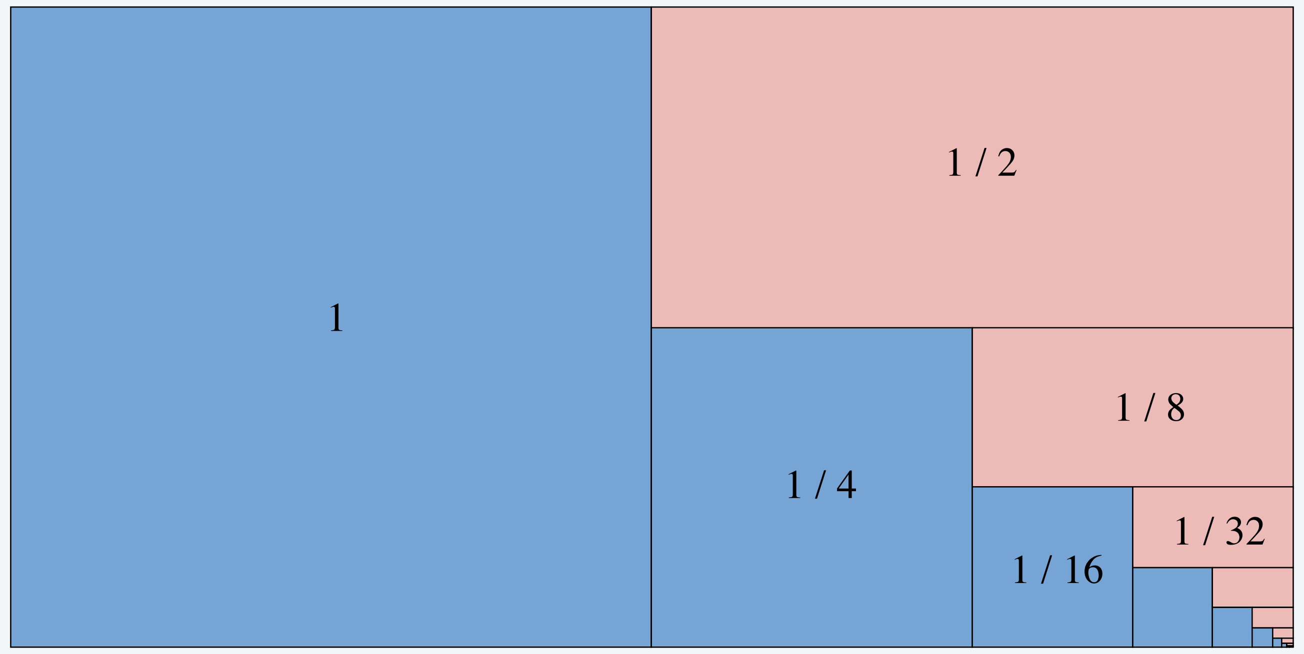
Harmonic sum. $1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} \sim \int_{x=1}^{n} \frac{1}{x}\, dx = \ln n$



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

n a power of 2

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



Analysis of algorithms: quiz 3

Approximately 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 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. $\Theta(2^n)$

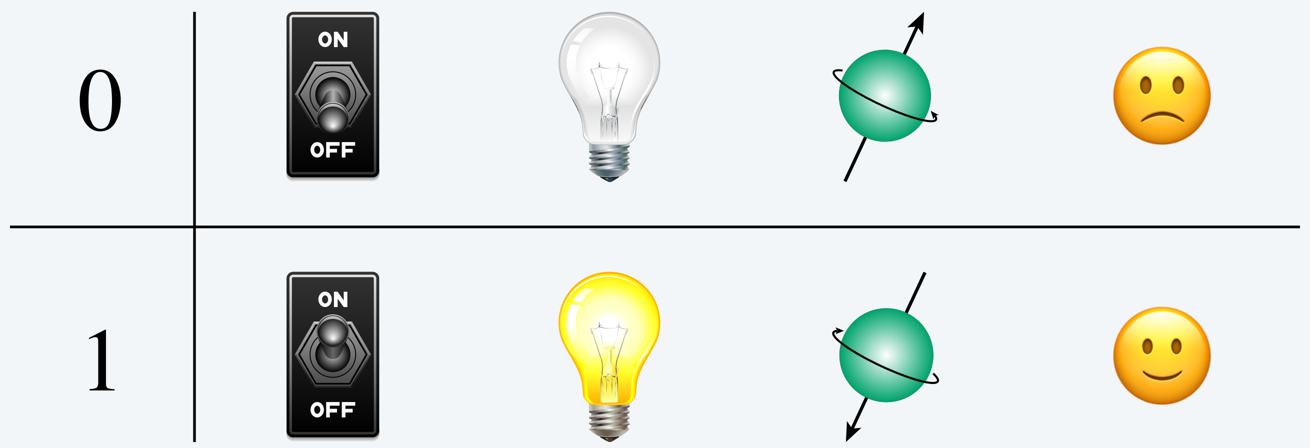
https://algs4.cs.princeton.edu

1.4 Analysis of Algorithms

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

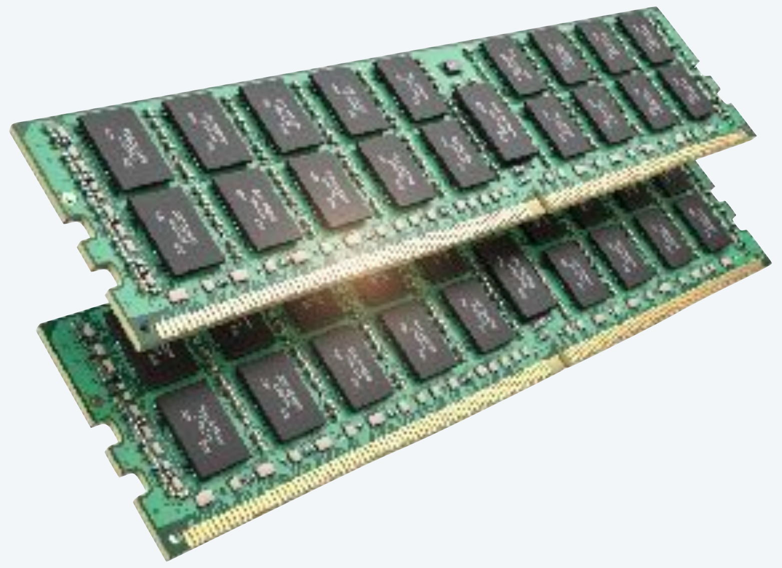
Memory basics

Bit. 0 or 1.



term	symbol	quantity
byte	B	8 bits
kilobyte	KB	1000 bytes
megabyte	MB	1000^2 bytes
gigabyte	GB	1000^3 bytes
terabyte	TB	1000^4 bytes

some define using powers of 2
(MB = 2^{10} bytes)



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

some JVMs "compress" 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[]`	$8n + 24$

wasteful
(but ~ 36n bytes 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

Reference. 8 bytes.

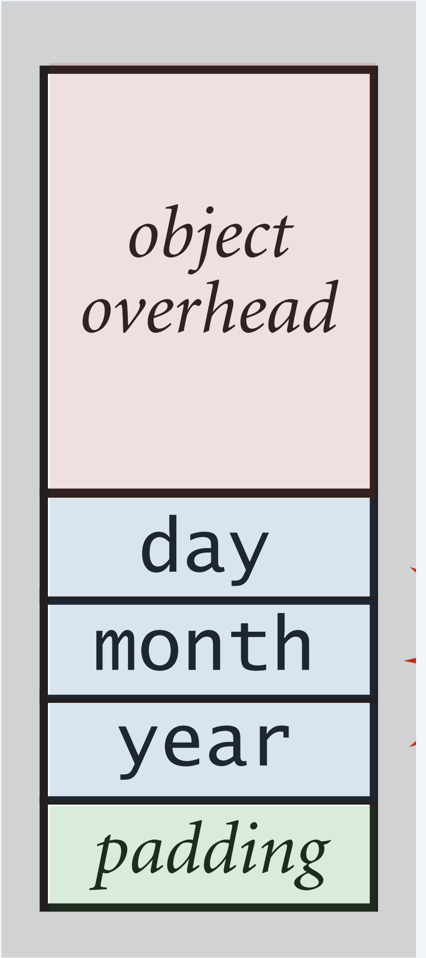
Object overhead. 16 bytes.

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

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)

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;
   }
   ...
}
```

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

16 bytes (object overhead)

$8 + (4n + 24)$ bytes each (reference + `int[]` array)

4 bytes (`int`)

4 bytes (padding)

$8n + 88 \sim 8n$ bytes

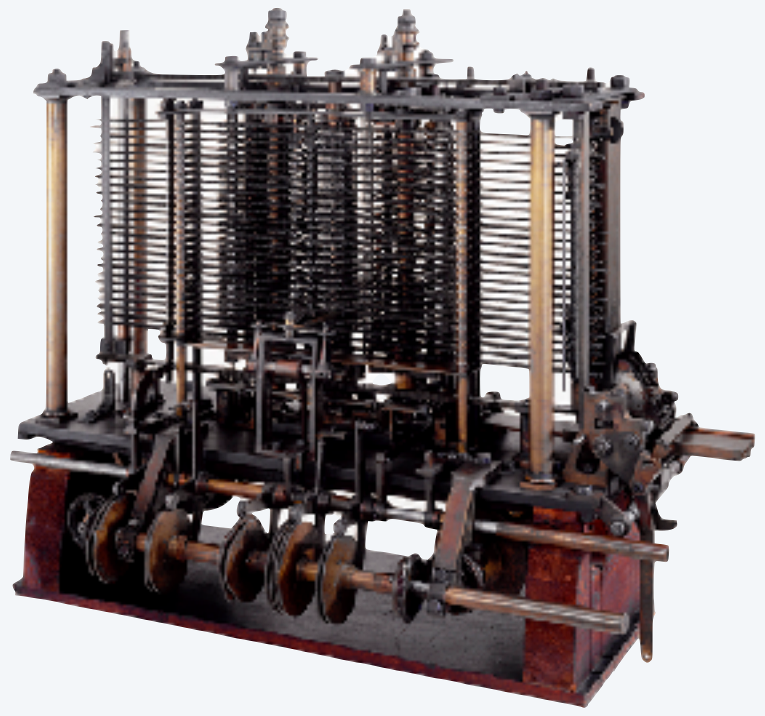
```
public class WeightedQuickUnionUF {
   private int[] parent;
   private int[] size;
   private int count;

   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 cost model and asymptotic 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.

Credits

image	source	license
Charles Babbage	The Illustrated London News	public domain
Babbage Enginine in Operation	xRez Studio	
Algorithm for the Analytic Engine	Ada Lovelace	public domain
Ada Lovelace and Book	Moore Allen & Innocent	
Galaxies Colliding	SaltyMikan	
Andrew Appel	Andrew Appel	
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
Apple M2 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 Analytic 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)