https://algs4.cs.princeton.edu

1.4 ANALYSIS OF ALGORITHMS

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

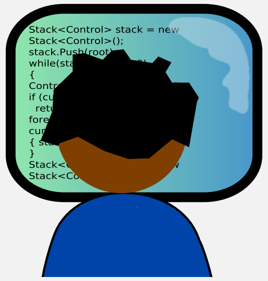
Last updated on 2/6/20 6:10 AM

https://algs4.cs.princeton.edu

1.4 Analysis of Algorithms

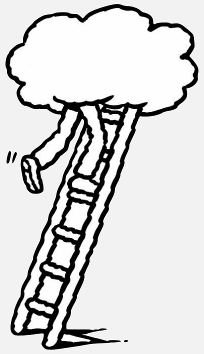
- *introduction*
- *running time (experimental analysis)*
- *running time (mathematical models)*
- *memory usage*

Cast of characters



Programmer needs to develop a working solution.

Client wants to solve problem efficiently.

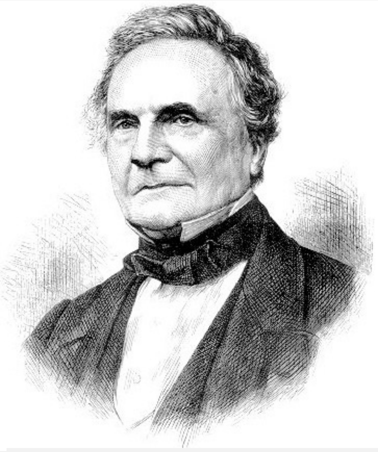


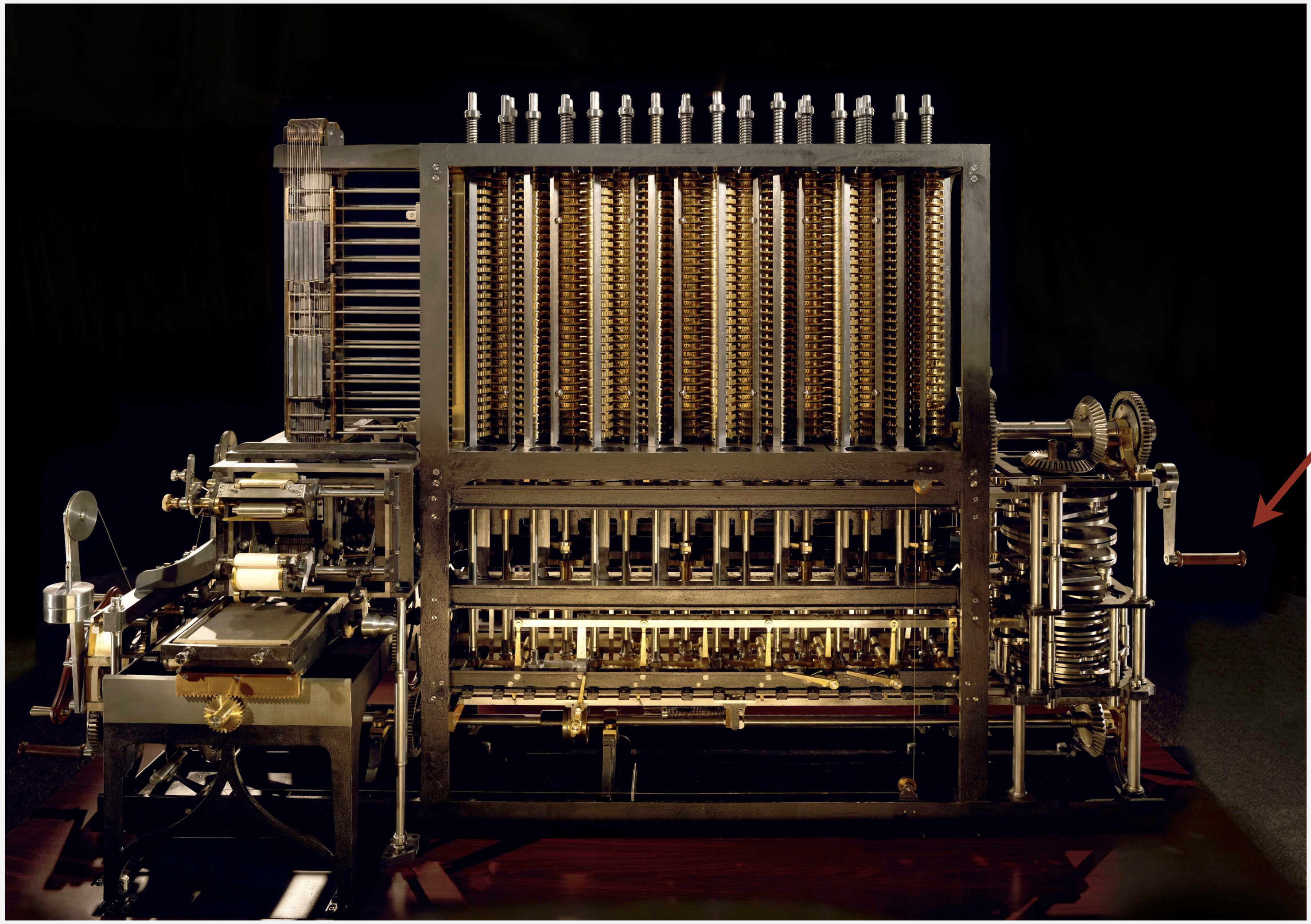
Theoretician seeks to understand.

Student (you) might play all of these roles someday.

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)

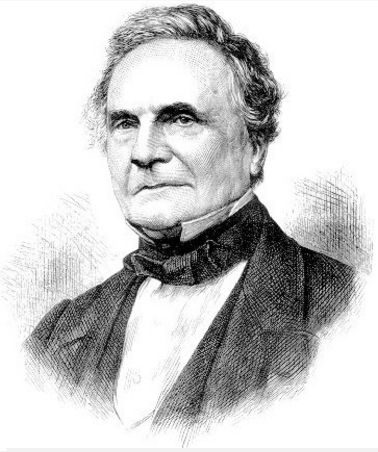


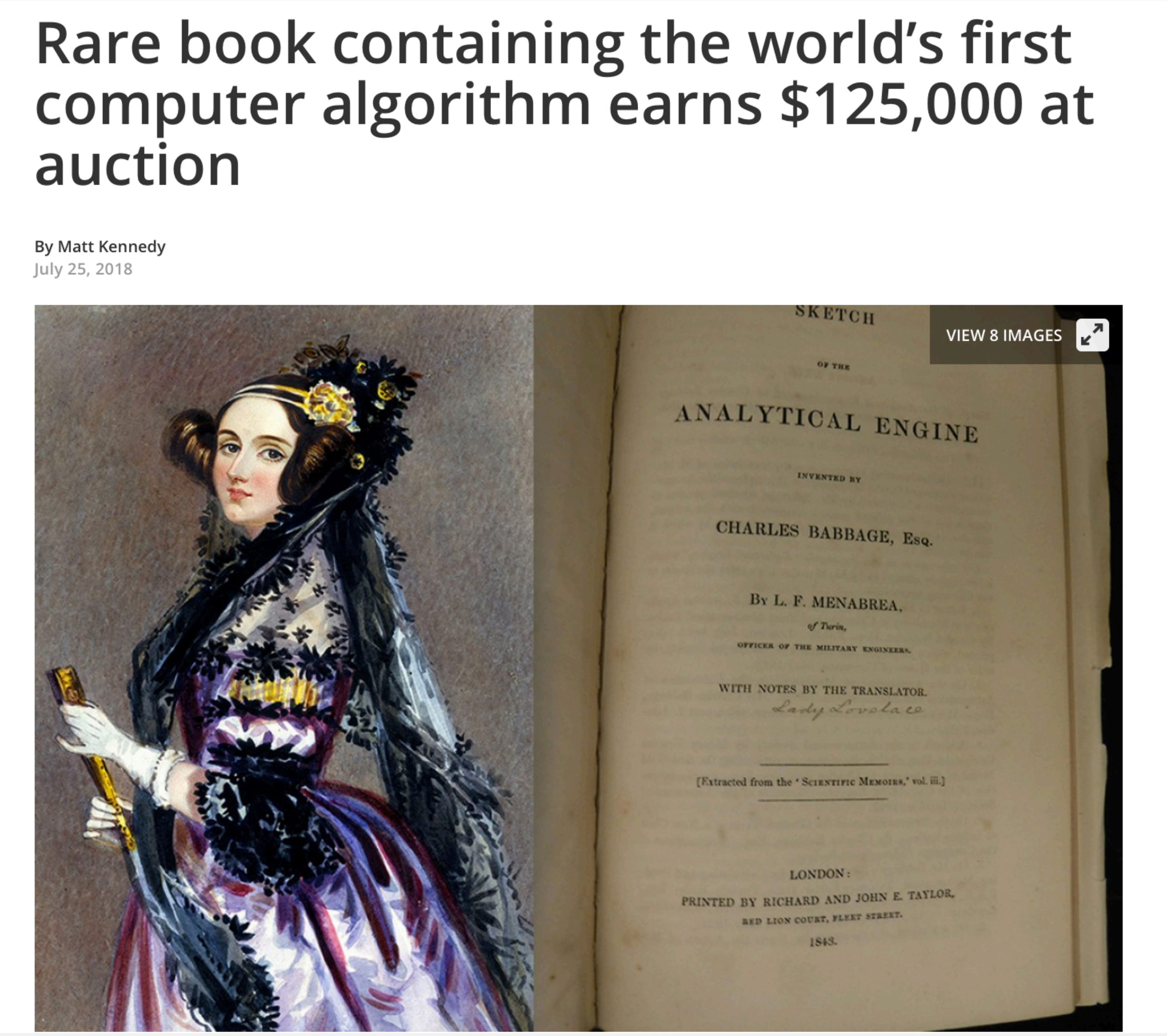


how many times do you have to turn the crank?

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

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.

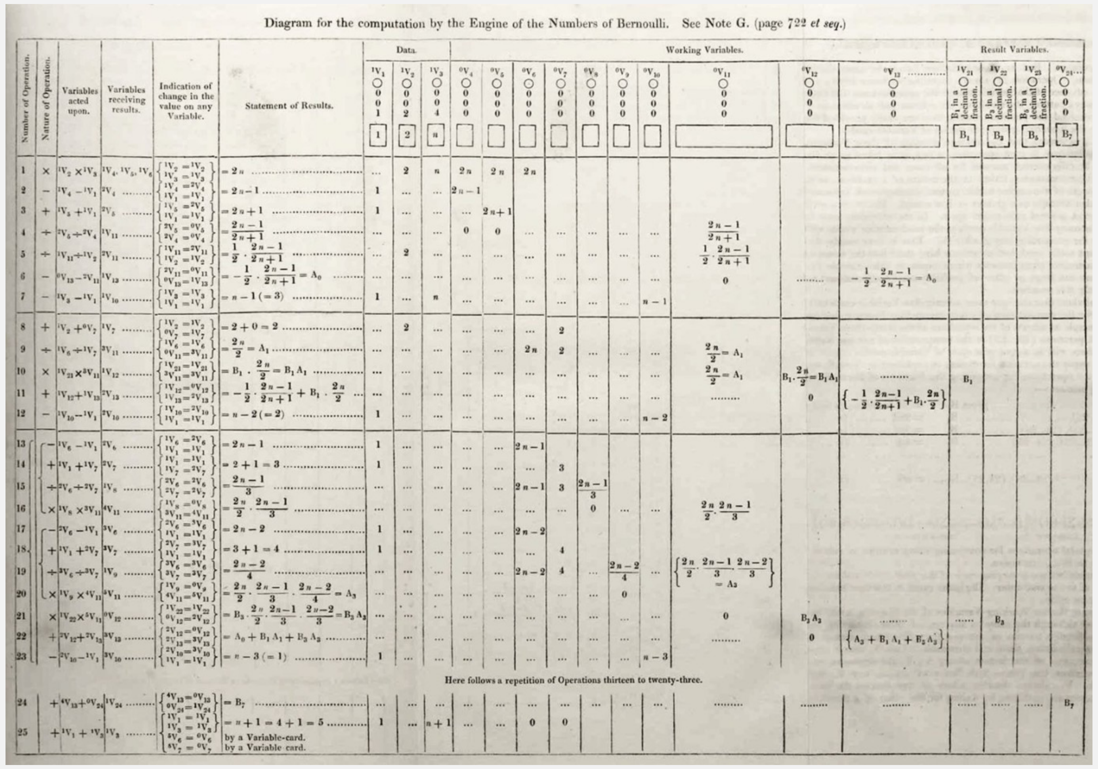


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.	Data.			Working Variables.										Result Variables.			
						$^1V_1$ ○ 0 0 0 1	$^1V_2$ ○ 0 0 0 2	$^1V_3$ ○ 0 0 0 4	$^0V_4$ ○ 0 0 0 0	$^0V_5$ ○ 0 0 0 0	$^0V_6$ ○ 0 0 0 0	$^0V_7$ ○ 0 0 0 0	$^0V_8$ ○ 0 0 0 0	$^0V_9$ ○ 0 0 0 0	$^0V_{10}$ ○ 0 0 0 0	$^0V_{11}$ ○ 0 0 0 0	$^0V_{12}$ ○ 0 0 0 0	$^0V_{13}$ ○ 0 0 0 0	$^1V_{21}$ ○ 0 0 0 0 B_1 in a decimal fraction.	$^1V_{22}$ ○ 0 0 0 0 B_3 in a decimal fraction.	$^1V_{23}$ ○ 0 0 0 0 B_5 in a decimal fraction.	$^0V_{24}$ ○ 0 0 0 0 ...
						1	2	n											B_1	B_3	B_5	B_7
1	×	$^1V_2 \times {}^1V_3$	$^1V_4, {}^1V_5, {}^1V_6$	$\left\{\begin{matrix}{}^1V_2={}^1V_2\\{}^1V_3={}^1V_3\end{matrix}\right\}$	$= 2n$	...	2	n	$2n$	$2n$	$2n$											
2	−	$^1V_4 - {}^1V_1$	$^2V_4$	$\left\{\begin{matrix}{}^1V_4={}^2V_4\\{}^1V_1={}^1V_1\end{matrix}\right\}$	$= 2n-1$	1	...	...	$2n-1$													
3	+	$^1V_5 + {}^1V_1$	$^2V_5$	$\left\{\begin{matrix}{}^1V_5={}^2V_5\\{}^1V_1={}^1V_1\end{matrix}\right\}$	$= 2n+1$	1	...	...	...	$2n+1$												
4	÷	$^2V_5 \div {}^2V_4$	$^1V_{11}$	$\left\{\begin{matrix}{}^2V_5={}^0V_5\\{}^2V_4={}^0V_4\end{matrix}\right\}$	$= \frac{2n-1}{2n+1}$	...	...	...	0	0	...	...	...	...	...	$\frac{2n-1}{2n+1}$						
5	÷	$^1V_{11} \div {}^1V_2$	$^2V_{11}$	$\left\{\begin{matrix}{}^1V_{11}={}^2V_{11}\\{}^1V_2={}^1V_2\end{matrix}\right\}$	$= \frac{1}{2} \cdot \frac{2n-1}{2n+1}$	...	2	...	...	...	...	...	...	...	...	$\frac{1}{2} \cdot \frac{2n-1}{2n+1}$						
6	−	$^0V_{13} - {}^2V_{11}$	$^1V_{13}$	$\left\{\begin{matrix}{}^2V_{11}={}^0V_{11}\\{}^0V_{13}={}^1V_{13}\end{matrix}\right\}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$	...	...	...	...	...	...	...	...	...	...	0	....	$-\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$				
7	−	$^1V_3 - {}^1V_1$	$^1V_{10}$	$\left\{\begin{matrix}{}^1V_3={}^1V_3\\{}^1V_1={}^1V_1\end{matrix}\right\}$	$= n - 1 \ (=3)$	1	...	n	...	...	...	...	...	...	$n-1$							
8	+	$^1V_2 + {}^0V_7$	$^1V_7$	$\left\{\begin{matrix}{}^1V_2={}^1V_2\\{}^0V_7={}^1V_7\end{matrix}\right\}$	$= 2 + 0 = 2$	...	2	...	...	...	...	2										
9	÷	$^1V_6 \div {}^1V_7$	$^3V_{11}$	$\left\{\begin{matrix}{}^1V_6={}^1V_6\\{}^0V_{11}={}^3V_{11}\end{matrix}\right\}$	$= \frac{2n}{2} = A_1$	...	...	...	...	...	$2n$	2	...	...	...	$\frac{2n}{2} = A_1$						
10	×	$^1V_{21} \times {}^3V_{11}$	$^1V_{12}$	$\left\{\begin{matrix}{}^1V_{21}={}^1V_{21}\\{}^3V_{11}={}^3V_{11}\end{matrix}\right\}$	$= B_1 \cdot \frac{2n}{2} = B_1 A_1$	...	...	...	...	...	...	...	...	...	...	$\frac{2n}{2} = A_1$	$B_1 \cdot \frac{2n}{2} = B_1 A_1$	....	B_1			
11	+	$^1V_{12} + {}^1V_{13}$	$^2V_{13}$	$\left\{\begin{matrix}{}^1V_{12}={}^0V_{12}\\{}^1V_{13}={}^2V_{13}\end{matrix}\right\}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}$	...	...	...	...	...	...	...	...	...	...	....	0	$\left\{-\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}\right\}$				
12	−	$^1V_{10} - {}^1V_1$	$^2V_{10}$	$\left\{\begin{matrix}{}^1V_{10}={}^2V_{10}\\{}^1V_1={}^1V_1\end{matrix}\right\}$	$= n - 2 \ (=2)$	1	...	...	...	...	...	...	...	...	$n-2$							
13	−	$^1V_6 - {}^1V_1$	$^2V_6$	$\left\{\begin{matrix}{}^1V_6={}^2V_6\\{}^1V_1={}^1V_1\end{matrix}\right\}$	$= 2n - 1$	1	...	...	...	...	$2n-1$											
14	+	$^1V_1 + {}^1V_7$	$^2V_7$	$\left\{\begin{matrix}{}^1V_1={}^1V_1\\{}^1V_7={}^2V_7\end{matrix}\right\}$	$= 2 + 1 = 3$	1	...	...	...	...	...	3										
15	÷	$^2V_6 \div {}^2V_7$	$^1V_8$	$\left\{\begin{matrix}{}^2V_6={}^2V_6\\{}^2V_7={}^2V_7\end{matrix}\right\}$	$= \frac{2n-1}{3}$	...	...	...	...	...	$2n-1$	3	$\frac{2n-1}{3}$									
16	×	$^1V_8 \times {}^3V_{11}$	$^4V_{11}$	$\left\{\begin{matrix}{}^1V_8={}^0V_8\\{}^3V_{11}={}^4V_{11}\end{matrix}\right\}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3}$	...	...	...	...	...	...	...	0	...	...	$\frac{2n}{2} \cdot \frac{2n-1}{3}$						
17	−	$^2V_6 - {}^1V_1$	$^3V_6$	$\left\{\begin{matrix}{}^2V_6={}^3V_6\\{}^1V_1={}^1V_1\end{matrix}\right\}$	$= 2n - 2$	1	...	...	...	...	$2n-2$											
18	+	$^1V_1 + {}^2V_7$	$^3V_7$	$\left\{\begin{matrix}{}^2V_7={}^3V_7\\{}^1V_1={}^1V_1\end{matrix}\right\}$	$= 3 + 1 = 4$	1	...	...	...	...	...	4										
19	÷	$^3V_6 \div {}^3V_7$	$^1V_9$	$\left\{\begin{matrix}{}^3V_6={}^3V_6\\{}^3V_7={}^3V_7\end{matrix}\right\}$	$= \frac{2n-2}{4}$	...	...	...	...	...	$2n-2$	4	...	$\frac{2n-2}{4}$	...	$\left\{\begin{matrix}\frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3}\\ = A_3\end{matrix}\right\}$						
20	×	$^1V_9 \times {}^4V_{11}$	$^5V_{11}$	$\left\{\begin{matrix}{}^1V_9={}^0V_9\\{}^4V_{11}={}^5V_{11}\end{matrix}\right\}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{4} = A_3$	...	...	...	...	...	...	...	...	0								
21	×	$^1V_{22} \times {}^5V_{11}$	$^0V_{12}$	$\left\{\begin{matrix}{}^1V_{22}={}^1V_{22}\\{}^0V_{12}={}^2V_{12}\end{matrix}\right\}$	$= B_3 \cdot \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3} = B_3 A_3$	...	...	...	...	...	...	...	...	...	...	0	$B_3 A_3$	....	......	B_3		
22	+	$^2V_{12} + {}^2V_{13}$	$^3V_{13}$	$\left\{\begin{matrix}{}^2V_{12}={}^0V_{12}\\{}^2V_{13}={}^3V_{13}\end{matrix}\right\}$	$= A_0 + B_1 A_1 + B_3 A_3$	...	...	...	...	...	...	...	...	...	...	....	0	$\left\{A_3 + B_1 A_1 + B_3 A_3\right\}$				
23	−	$^2V_{10} - {}^1V_1$	$^3V_{10}$	$\left\{\begin{matrix}{}^2V_{10}={}^3V_{10}\\{}^1V_1={}^1V_1\end{matrix}\right\}$	$= n - 3 \ (=1)$	1	...	...	...	...	...	...	...	...	$n-3$							
					Here follows a repetition of Operations thirteen to twenty-three.																	
24	+	$^4V_{13} + {}^0V_{24}$	$^1V_{24}$	$\left\{\begin{matrix}{}^4V_{13}={}^0V_{13}\\{}^0V_{24}={}^1V_{24}\end{matrix}\right\}$	$= B_7$	...	...	...	...	...	...	...	...	...	...	....	....	....	......	......	......	B_7
25	+	$^1V_1 + {}^1V_3$	$^1V_3$	$\left\{\begin{matrix}{}^1V_1={}^1V_1\\{}^1V_3={}^1V_3\\{}^5V_6={}^0V_6\\{}^5V_7={}^0V_7\end{matrix}\right\}$	$= n + 1 = 4 + 1 = 5$ by a Variable-card. by a Variable card.	1	...	$n+1$	...	...	0	0										

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

Reasons to analyze algorithms

Predict performance.

Compare algorithms.

Provide guarantees.

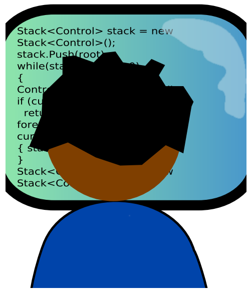
Understand theoretical basis.

this course (COS 226)

theory of algorithms (COS 423)

Primary practical reason: avoid performance bugs.

client gets poor performance because programmer did not understand performance characteristics

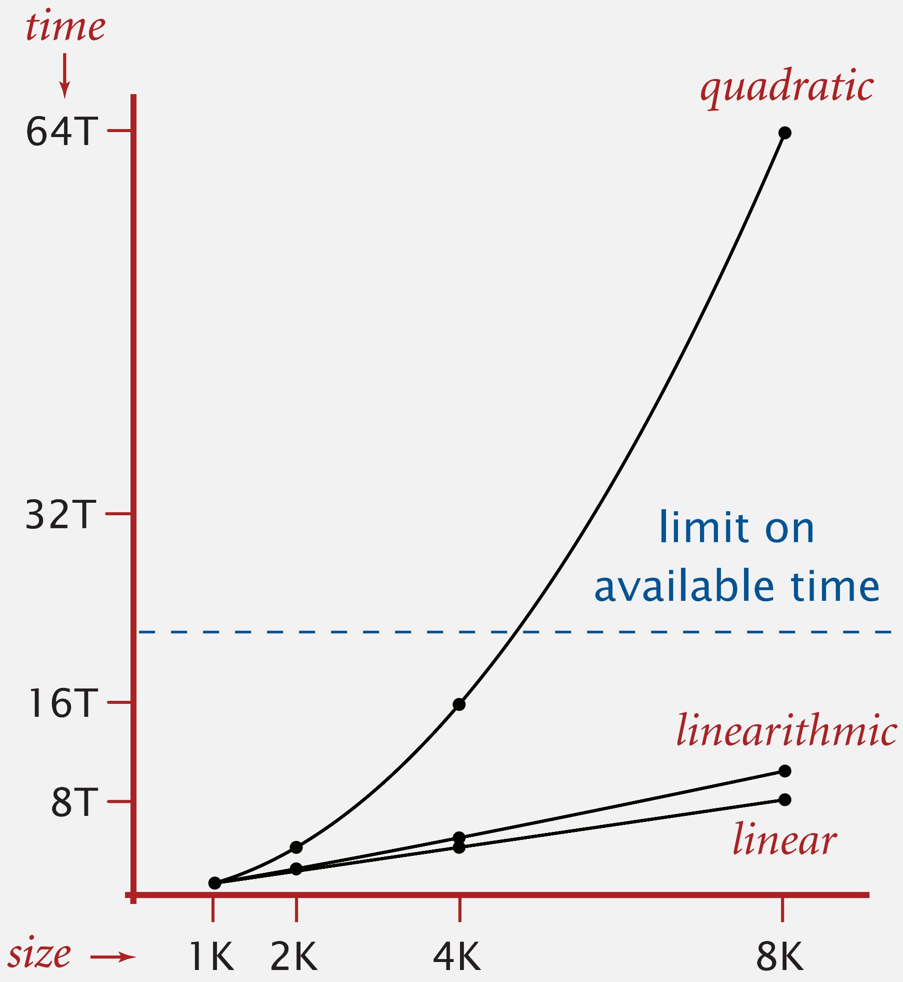


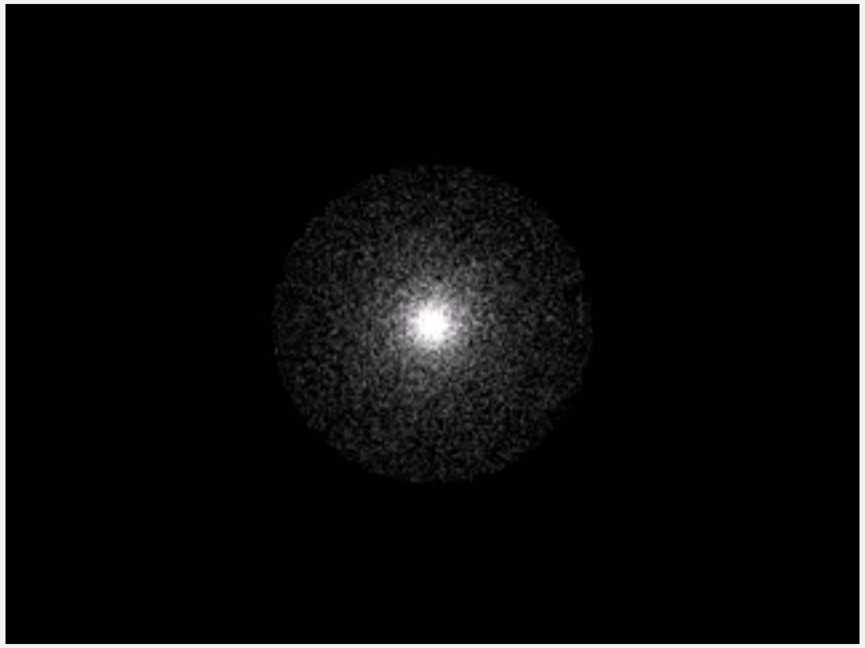
An algorithmic success story

Andrew Appel
PU ’81

N-body simulation.

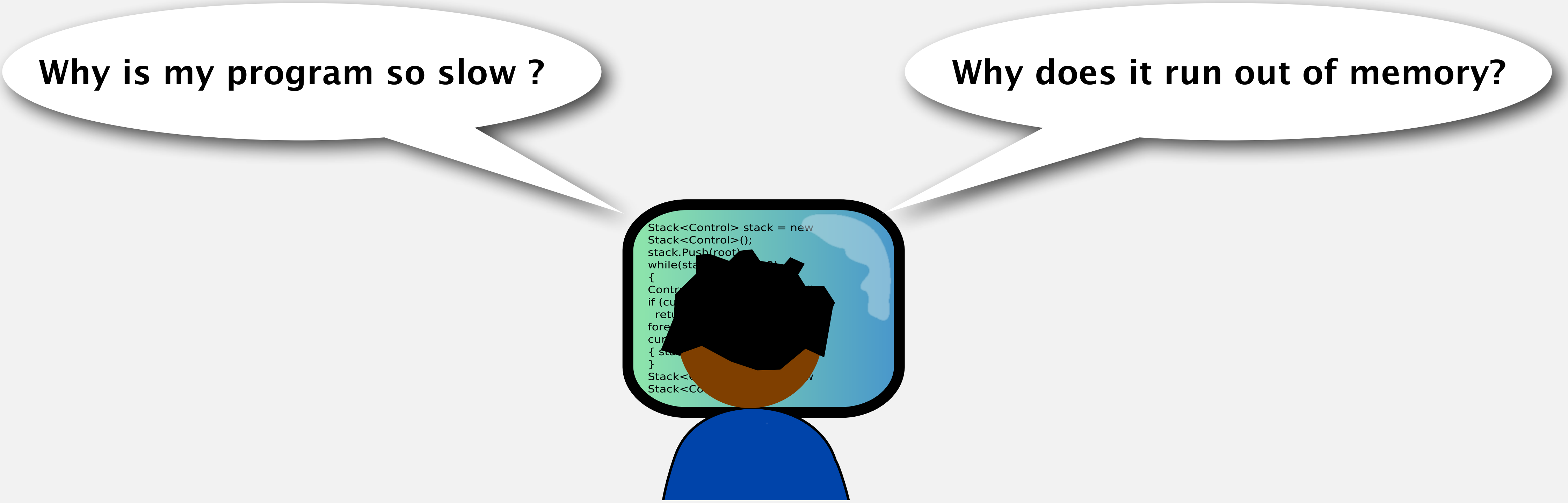
- Simulate gravitational interactions among n bodies.
- Applications: cosmology, fluid dynamics, semiconductors, ...
- Brute force: n^2 steps.
- Barnes–Hut algorithm: $n \log n$ steps, enables new research.





The challenge

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



```
Stack<Control> stack = new
Stack<Control>();
```

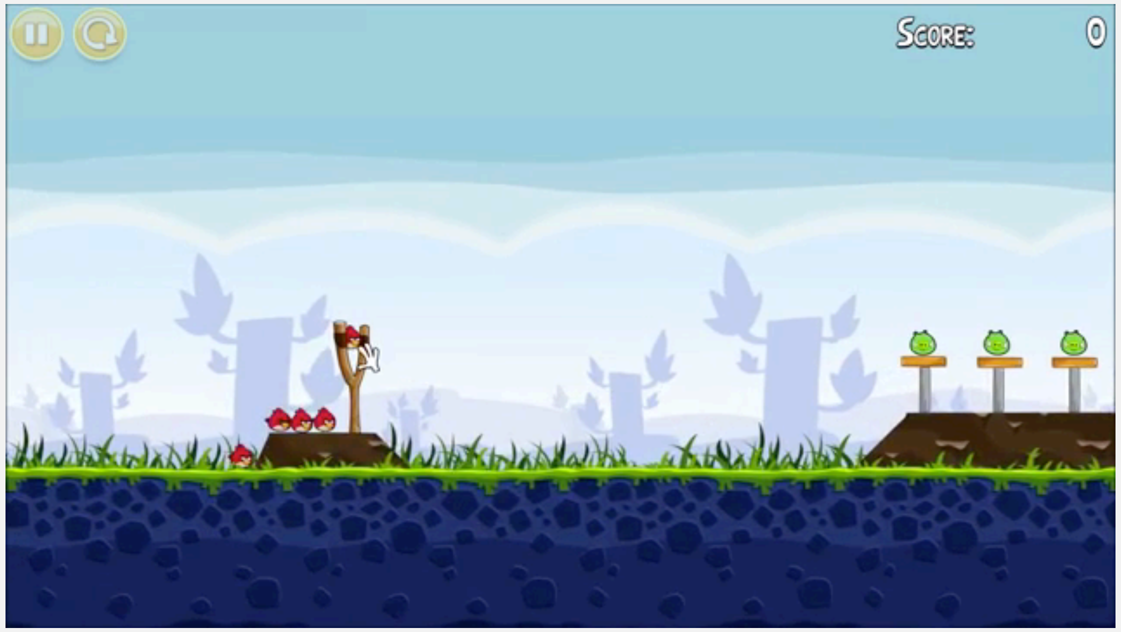
Our approach. Combination of experiments and mathematical modeling.

Example: 3-SUM

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

```
% more 8ints.txt
8
30 -40 -20 -10 40 0 10 5

% 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. Related to 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 each triple

for simplicity, ignore integer overflow

https://algs4.cs.princeton.edu

1.4 Analysis of Algorithms

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

Measuring the running time

Q. How to time a program?

A. Manual.

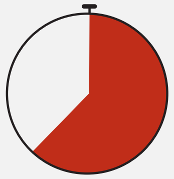
```
% java ThreeSum 1Kints.txt
```

tick tick tick

```
70
% java ThreeSum 2Kints.txt
```

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

```
528
% java ThreeSum 4Kints.txt
```



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 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 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 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

```
4039
```

Measuring the running time

Q. How to time a program?

A. Automatic.

```
import edu.princeton.cs.algs4.Stopwatch;

public static void main(String[] args)
{
   In in = new In(args[0]);
   int[] a = in.readAllInts();
   Stopwatch stopwatch = new Stopwatch();
   StdOut.println(ThreeSum.count(a));
   double time = stopwatch.elapsedTime();
   StdOut.println("elapsed time = " + time);
}
```

Empirical analysis

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

```
% █
```

Empirical analysis

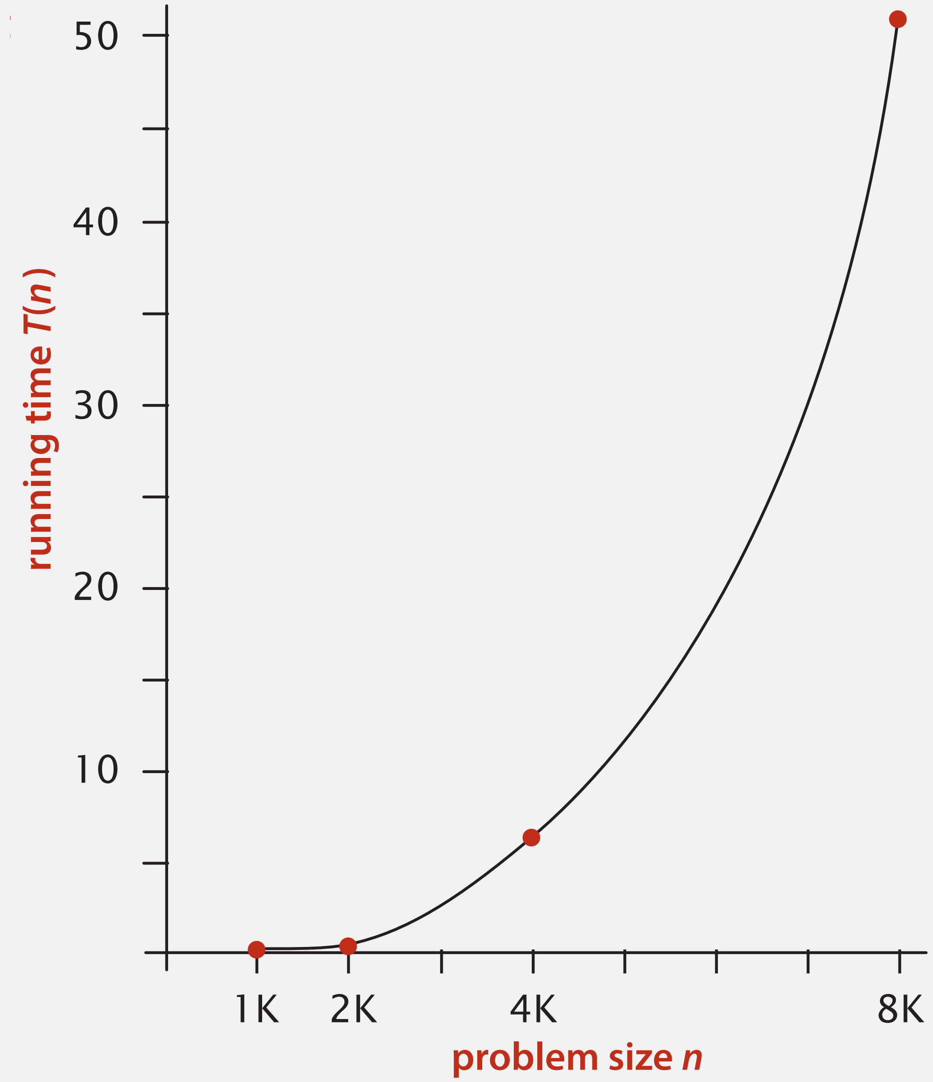
Run the program for various input sizes and measure running time.

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.

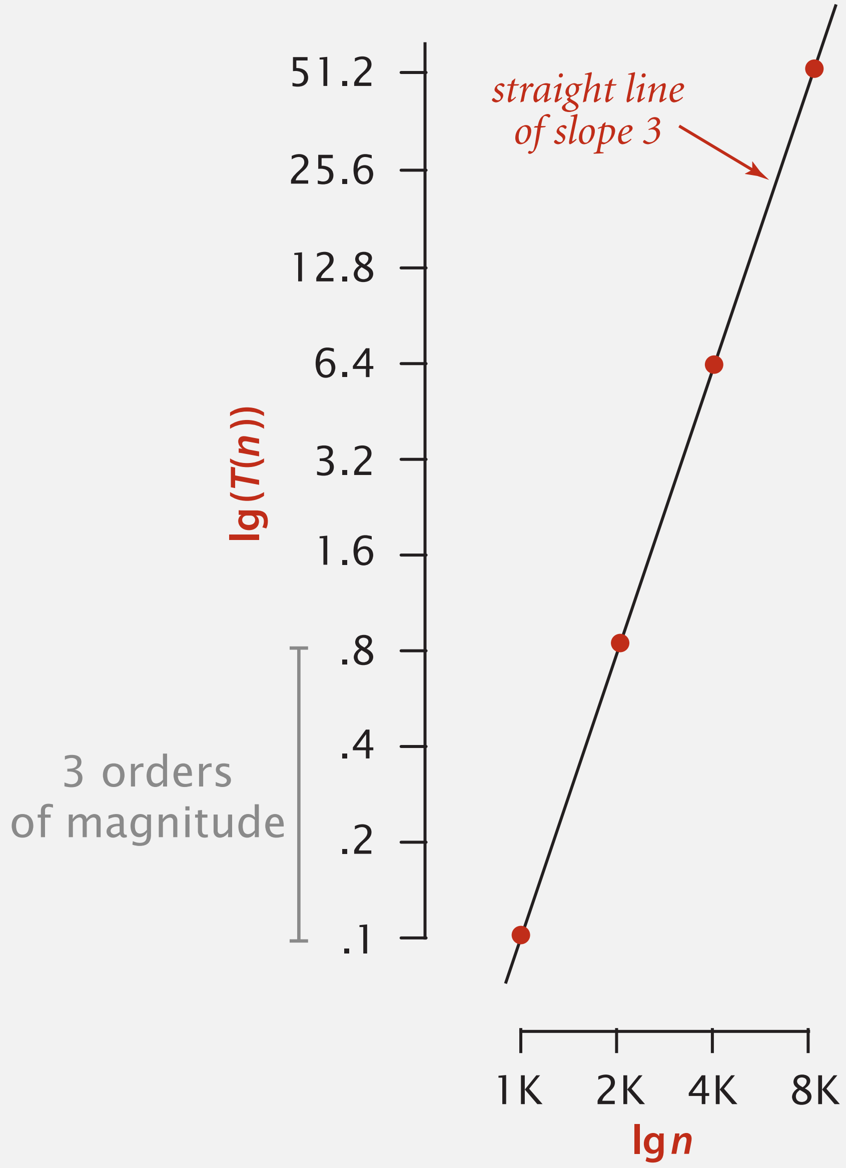


Hypothesis (power law). $T(n) = a\, n^b$.

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

$$\log_2(T(n)) = \underbrace{2.999}_{\text{slope}} \log_2 n + (-33.2103)$$

$$T(n) = 2^{-33.2103} \times n^{2.999}$$

Regression. Fit straight line through data points.

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

Prediction and validation

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

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

Predictions.

- 51.0 seconds for $n = 8{,}000$.
- 408.1 seconds for $n = 16{,}000$.

Observations.

n	time (seconds) †
8,000	51.1
8,000	51.0
8,000	51.1
16,000	410.8

validates hypothesis!

Doubling hypothesis

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

Run program, doubling the size of the input.

n	time (seconds) †	ratio	lg 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 $b \approx 3$

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

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

Caveat. Cannot identify logarithmic factors with doubling hypothesis.

Doubling hypothesis

Doubling hypothesis. Quick way to estimate 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

$$51.1 = a \times 8000^3$$

$$\Rightarrow \quad a = 0.998 \times 10^{-10}$$

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)

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 $a\, n^b$

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)*
- *binary search*
- *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.
- Cost depends on machine, compiler.
- Frequency depends on algorithm, input data.

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 CLASSIC WORK NEWLY UPDATED AND REVISED
The Art of Computer Programming
VOLUME 1
Fundamental Algorithms
Third Edition
DONALD E. KNUTH

THE CLASSIC WORK NEWLY UPDATED AND REVISED
The Art of Computer Programming
VOLUME 2
Seminumerical Algorithms
Third Edition
DONALD E. KNUTH

THE CLASSIC WORK NEWLY UPDATED AND REVISED
The Art of Computer Programming
VOLUME 3
Sorting and Searching
Second Edition
DONALD E. KNUTH

THE CLASSIC WORK EXTENDED AND REFINED
The Art of Computer Programming
VOLUME 4A
Combinatorial Algorithms
Part 1
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. *No idea.*

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	$½\ (n + 1)\ (n + 2)$
equal to compare	1/10	$½\ n\ (n - 1)$
array access	1/10	$n\ (n - 1)$
increment	1/10	$½\ n\ (n + 1)$ 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.

```
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!)

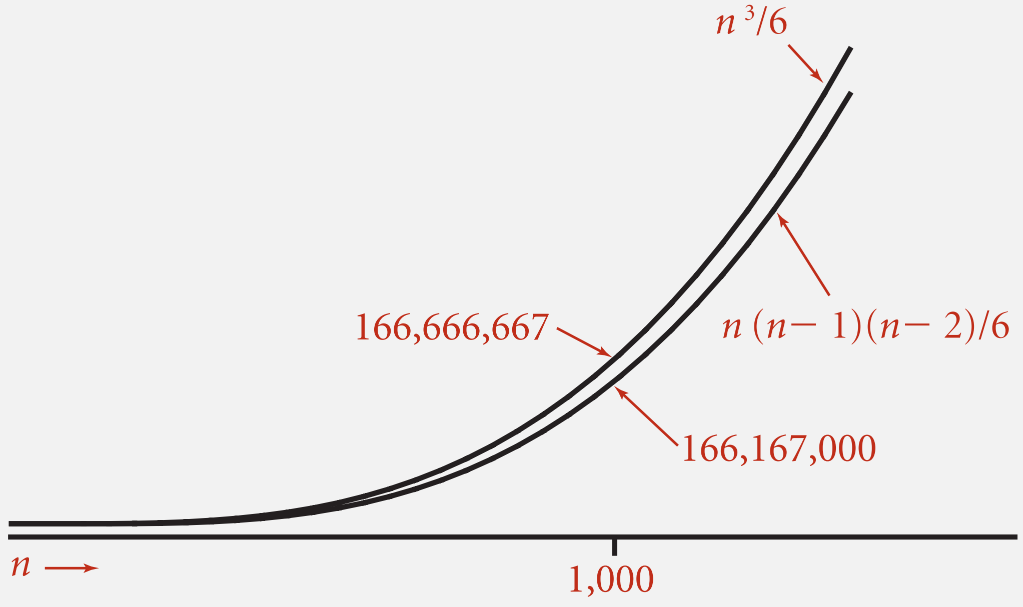
Simplification 2: asymptotic notations

Tilde notation. Discard lower-order terms.

Big Theta notation. Also discard leading coefficient.

function	tilde	big Theta
$4\,n^5 + 20\,n + 16$	$\sim 4\,n^5$	$\Theta(n^5)$
$7\,n^2 + 100\,n^{4/3} + 56$	$\sim 7\,n^2$	$\Theta(n^2)$
$⅙\,n^3 \underbrace{-\,½\,n^2 + ⅓\,n}$	$\sim ⅙\,n^3$	$\Theta(n^3)$

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	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**	`while (n > 1)`
		`{  n = n/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"

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

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"

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

A. ~ ½ n^3 array accesses.

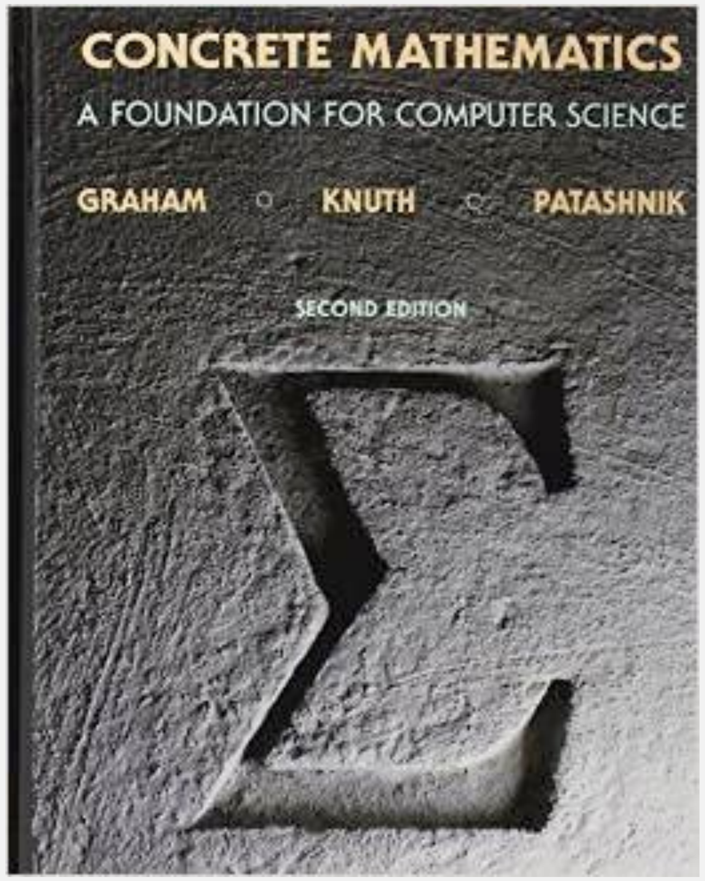
see COS 340

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

Estimating a discrete sum

Q. How to estimate a discrete sum?

A1. Take a discrete mathematics course (COS 340).



Estimating a discrete sum

Q. How to estimate a discrete sum?

A2. Replace the sum with an integral; use calculus!

Ex 1. $1 + 2 + \ldots + n$.

$$\sum_{i=1}^{n} i \;\sim\; \int_{x=1}^{n} x \, dx \;\sim\; \frac{1}{2} n^2$$

Ex 2. $1 + 1/2 + 1/3 + \ldots + 1/n$.

$$\sum_{i=1}^{n} \frac{1}{i} \;\sim\; \int_{x=1}^{n} \frac{1}{x} \, dx \;\sim\; \ln n$$

Ex 3. 3-sum triple loop.

$$\sum_{i=1}^{n} \sum_{j=i}^{n} \sum_{k=j}^{n} 1 \;\sim\; \int_{x=1}^{n} \int_{y=x}^{n} \int_{z=y}^{n} dz \, dy \, dx \;\sim\; \frac{1}{6} n^3$$

Ex 4. 1 + ½ + ¼ + ⅛ + …

$$\int_{x=0}^{\infty} \left(\frac{1}{2}\right)^{x} dx \;=\; \frac{1}{\ln 2} \;\approx\; 1.4427$$

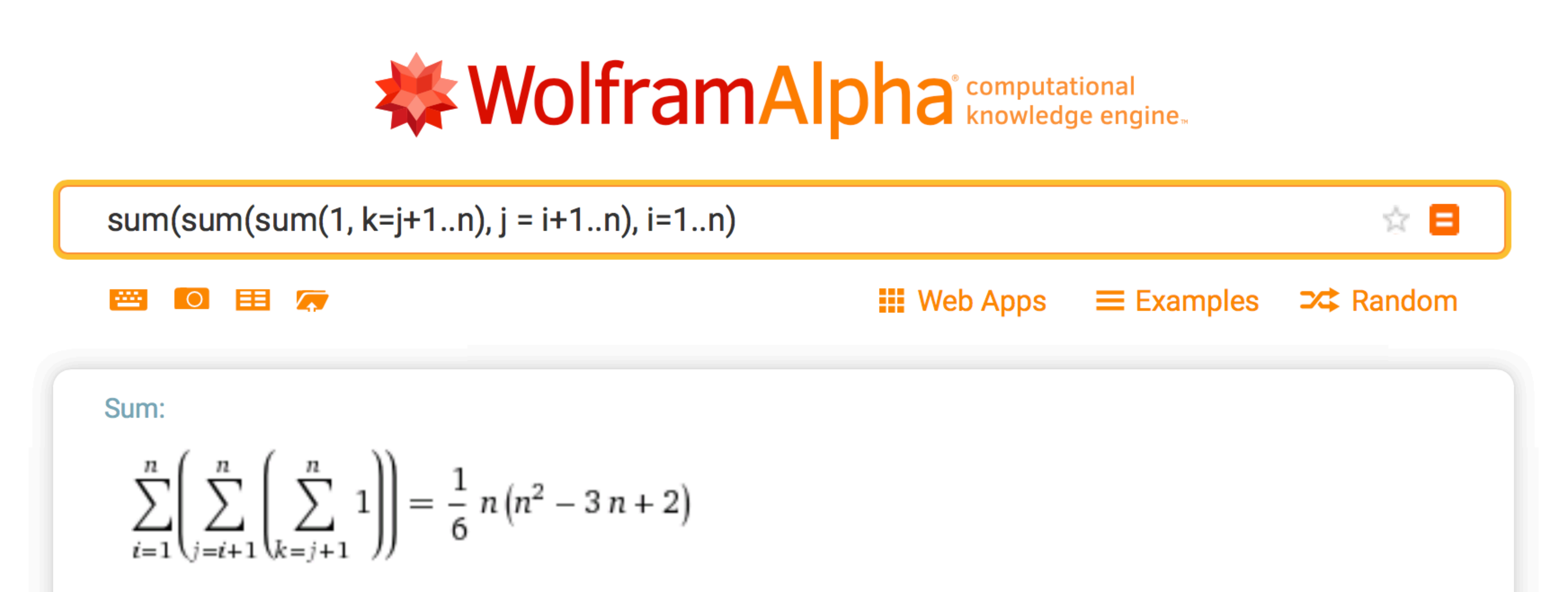
$$\sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^{i} \;=\; 2$$

integral trick
doesn't always work!

Estimating a discrete sum

Q. How to estimate a discrete sum?

A3. Use Maple or Wolfram Alpha.



sum(sum(sum(1, k=j+1..n), j = i+1..n), i=1..n)

Sum:

$$\sum_{i=1}^{n}\left(\sum_{j=i+1}^{n}\left(\sum_{k=j+1}^{n} 1\right)\right) = \frac{1}{6} n \left(n^2 - 3 n + 2\right)$$

https://www.wolframalpha.com

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 = 1; i <= n; 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)*
- ***binary search***
- *memory usage*

Binary search

Goal. Given a sorted array and a key, find index of the key in the array?

Binary search. Compare key against middle entry.

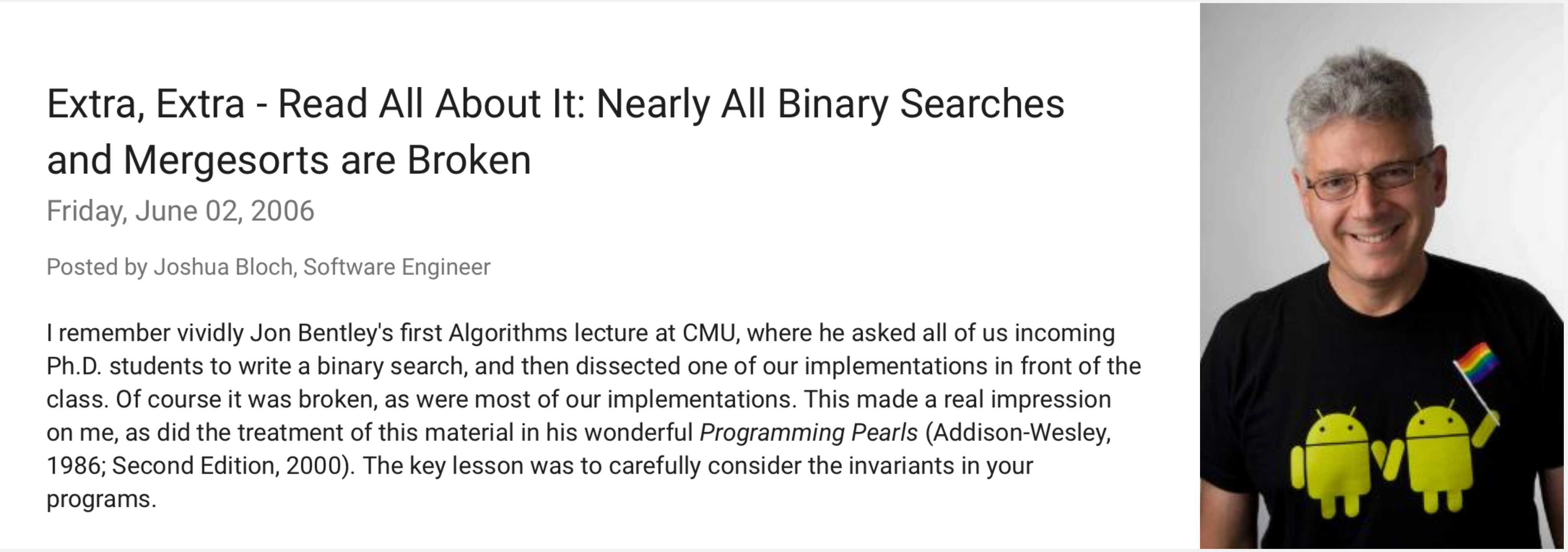
- Too small, go left.
- Too big, go right.
- Equal, found.

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Binary search: implementation

Trivial to implement?

- First binary search published in 1946.
- First bug-free one in 1962.
- Bug in Java's `Arrays.binarySearch()` discovered in 2006.



Extra, Extra - Read All About It: Nearly All Binary Searches and Mergesorts are Broken

Friday, June 02, 2006

Posted by Joshua Bloch, Software Engineer

I remember vividly Jon Bentley's first Algorithms lecture at CMU, where he asked all of us incoming Ph.D. students to write a binary search, and then dissected one of our implementations in front of the class. Of course it was broken, as were most of our implementations. This made a real impression on me, as did the treatment of this material in his wonderful *Programming Pearls* (Addison-Wesley, 1986; Second Edition, 2000). The key lesson was to carefully consider the invariants in your programs.

https://ai.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html

Binary search: Java implementation

Invariant. If `key` appears in array `a[]`, then `a[lo] ≤ key ≤ a[hi]`.

```
public static int binarySearch(int[] a, int key)
{
   int lo = 0, hi = a.length - 1;
   while (lo <= hi)
   {
      int mid = lo + (hi - lo) / 2;
      if      (key < a[mid]) hi = mid - 1;
      else if (key > a[mid]) lo = mid + 1;
      else return mid;
   }
   return -1;
}
```

why not `mid = (lo + hi) / 2` ?

one “3-way compare”

Binary search: analysis

Proposition. Binary search uses at most $1 + \log_2 n$ 3-way compares to search in a sorted array of length n.

Pf sketch.

slightly better, due to rounding and eliminating `a[mid]` from subarray

- Each iteration of `while` loop:
  - performs one 3-way compare
  - decreases the length of subarray remaining by a factor of 2
- Initial array length = n.
- The `while` loop terminates after length = 1.
- At most $1 + \log_2 n$ iterations. Why?

may terminate earlier (if search key is found)

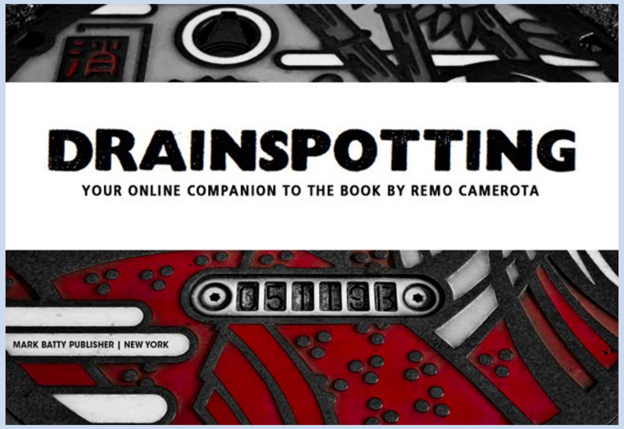
$$\underbrace{n \to n/2 \to n/4 \to n/8 \to \cdots \to 2 \to 1}_{1 + \log_2 n}$$

WHY ARE SEWER ACCESS COVERS ROUND?

New York, New York

Okayama, Japan

Zermatt, Switzerland


DRAINSPOTTING
YOUR ONLINE COMPANION TO THE BOOK BY REMO CAMEROTA
MARK BATTY PUBLISHER | NEW YORK

3-SUM

3-SUM. Given an array of n distinct integers, find three s.t. $a + b + c = 0$.

Version 0. $\Theta(n^3)$ time.

Version 1. $\Theta(n^2 \log n)$ time.

Version 2. $\Theta(n^2)$ time.

Note. For full credit, the running time should be in the worst case and use only constant extra space.

3-Sum: an N^2 log N algorithm

Algorithm.

- Step 1: Sort the n (distinct) numbers.
- Step 2: For each pair `a[i]` and `a[j]`:
  binary search for `-(a[i] + a[j])`.

Analysis. Running time is $\Theta(n^2 \log n)$.

- Step 1: $\Theta(n^2)$ with insertion sort.
  [or $\Theta(n \log n)$ with mergesort]
- Step 2: $\Theta(n^2 \log n)$ with binary search.

input

`30 -40 -20 -10 40 0 10 5`

sort

`-40 -20 -10 0 5 10 30 40`

binary search

(-40, -20)	60
(-40, -10)	50
(-40, 0)	40
(-40, 5)	35
(-40, 10)	30
⋮	⋮
(-20, -10)	30
⋮	⋮
(-10, 0)	10
⋮	⋮
(10, 30)	~~-40~~ ← only count if a[i] < a[j] < a[k] to avoid double counting
(10, 40)	-50
(30, 40)	-70

Algorithms

Robert Sedgewick | Kevin Wayne

https://algs4.cs.princeton.edu

1.4 Analysis of Algorithms

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

Basics

Bit. 0 or 1.

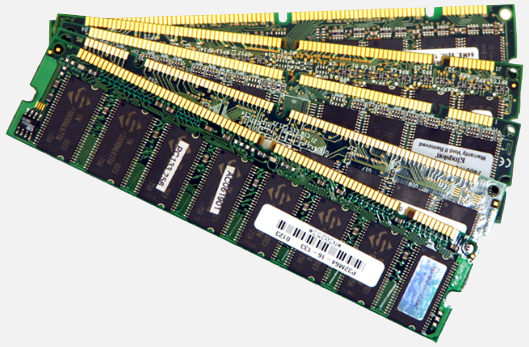
NIST

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

one-dimensional array (length n)

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

two-dimensional array (m-by-n)

Typical memory usage for objects in Java

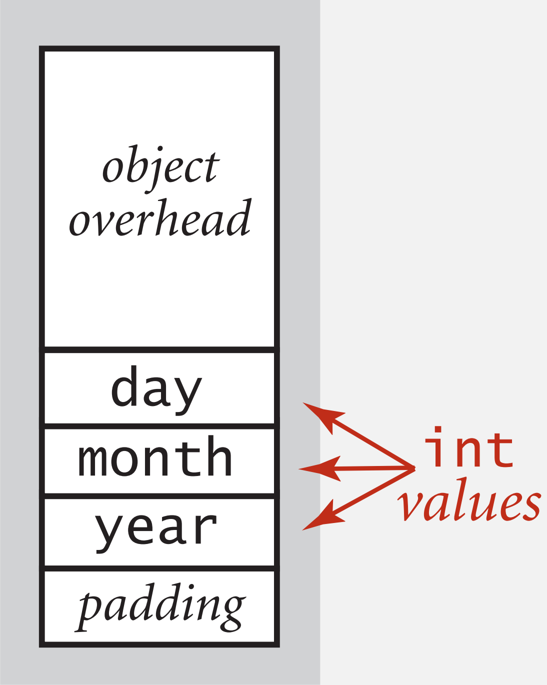
Object overhead. 16 bytes.

Reference. 8 bytes.

Padding. Memory of each object rounded up to use a multiple of 8 bytes.

Ex 1. A `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

Typical memory usage summary

Total memory usage for a data type value in Java:

- Primitive type: 4 bytes for `int`, 8 bytes for `double`, ...
- Object reference: 8 bytes.
- Array: 24 bytes + memory for each array entry.
- Object: 16 bytes + memory for each instance variable.
- Padding: round up to multiple of 8 bytes.

Note. Depending on application, we often count the memory for any referenced objects (recursively).

"deep memory"

Analysis of algorithms: quiz 5

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

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

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

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

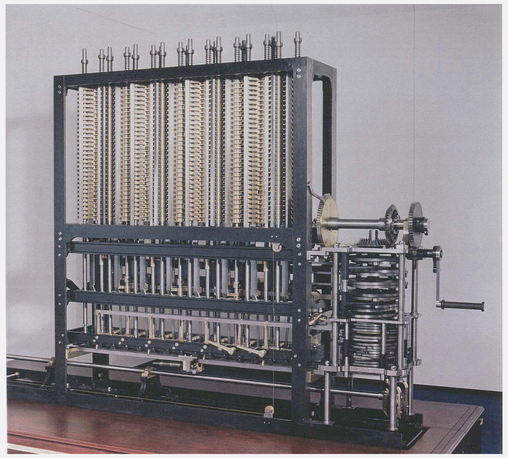
D. $\sim 8\,n^2$ *bytes*

```
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.