



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

1.4 ANALYSIS OF ALGORITHMS

- ▶ *introduction*
- ▶ *experimental analysis of running time*
- ▶ *mathematical models of running time*
- ▶ *order-of-growth classifications*
- ▶ *analysis of memory consumption*

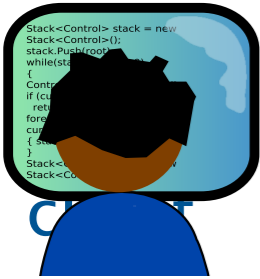


<https://algs4.cs.princeton.edu>

1.4 ANALYSIS OF ALGORITHMS

- ▶ *introduction*
- ▶ *experimental analysis of running time*
- ▶ *mathematical models of running time*
- ▶ *order-of-growth classifications*
- ▶ *analysis of memory consumption*

Cast of characters



Programmer needs to develop a working solution.



Client wants to solve problem efficiently.



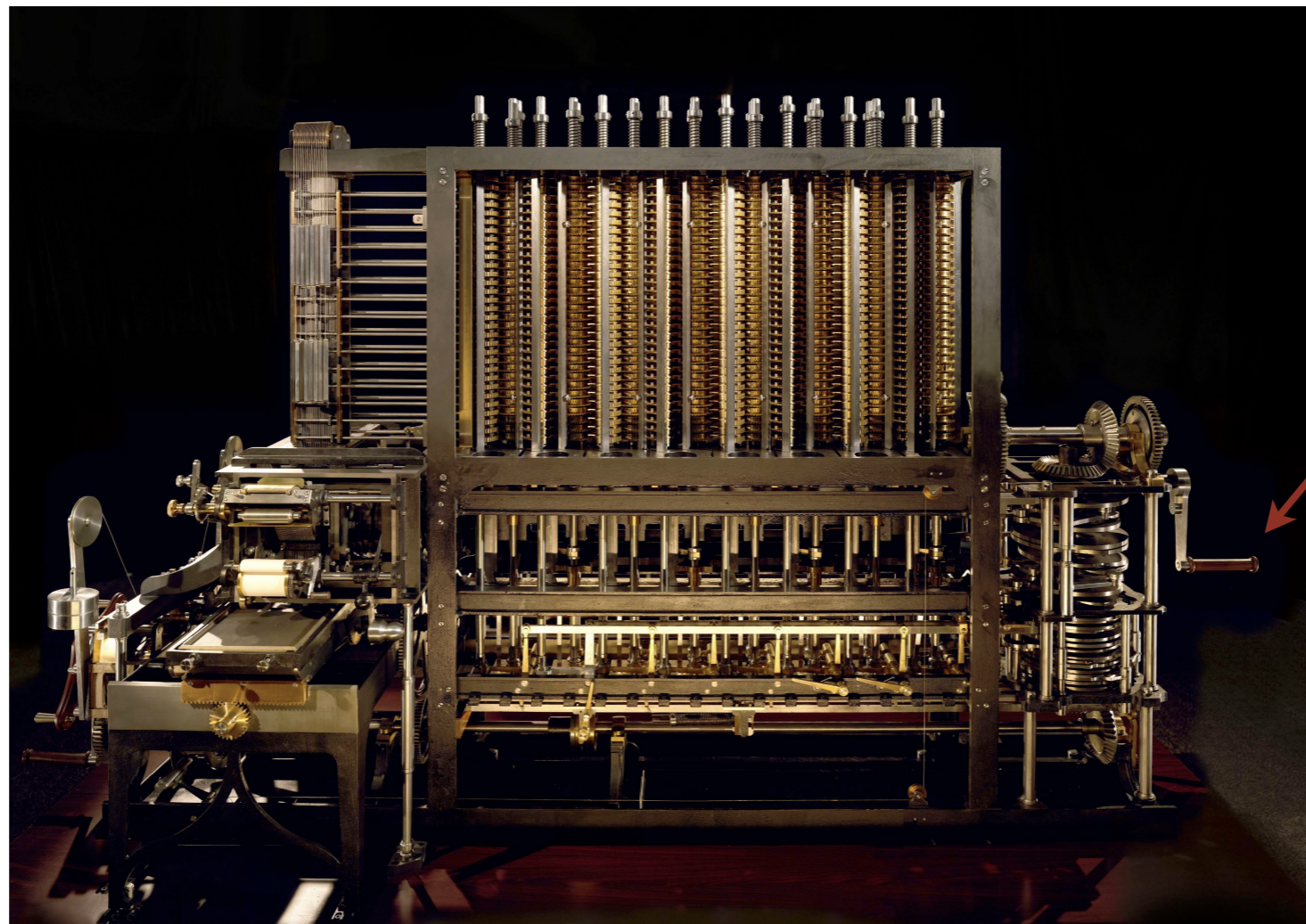
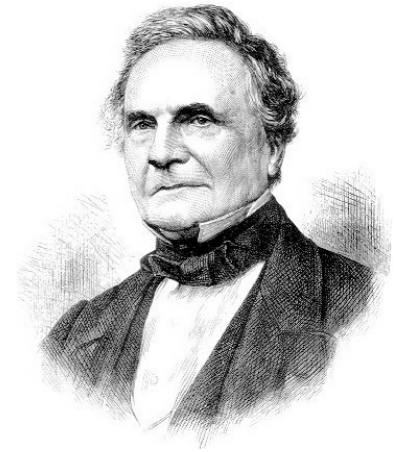
Theoretician seeks to understand.



Student (you) might play any or 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?



Reasons to analyze algorithms

Predict performance.

Compare algorithms.

Provide guarantees.

Understand theoretical basis.

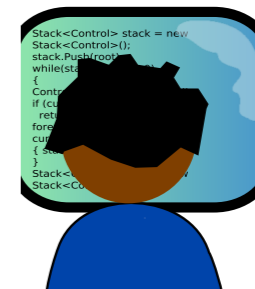
this course
(COS 226)

theory of algorithms
(COS 423)

Primary practical reason: avoid performance bugs.



client gets poor performance because programmer did not understand performance characteristics



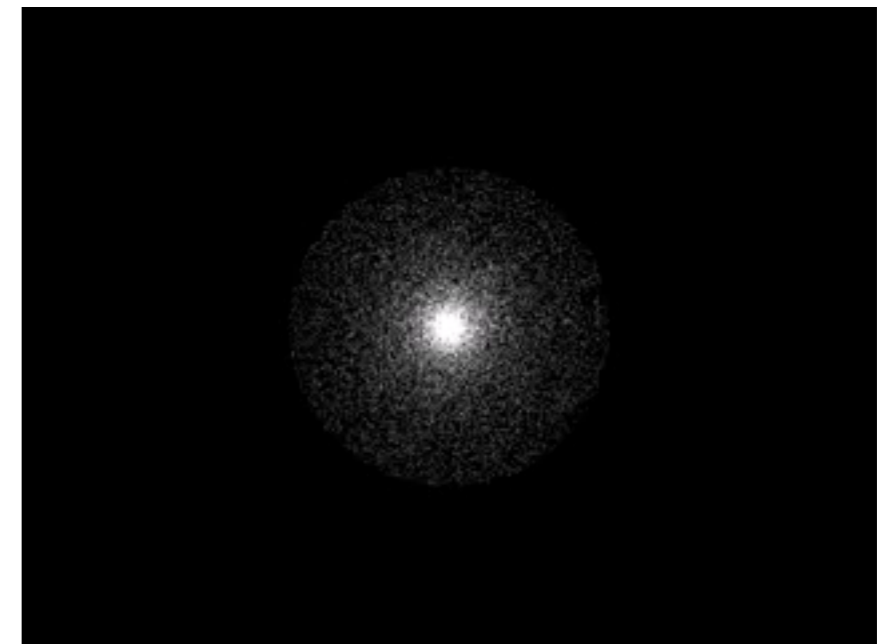
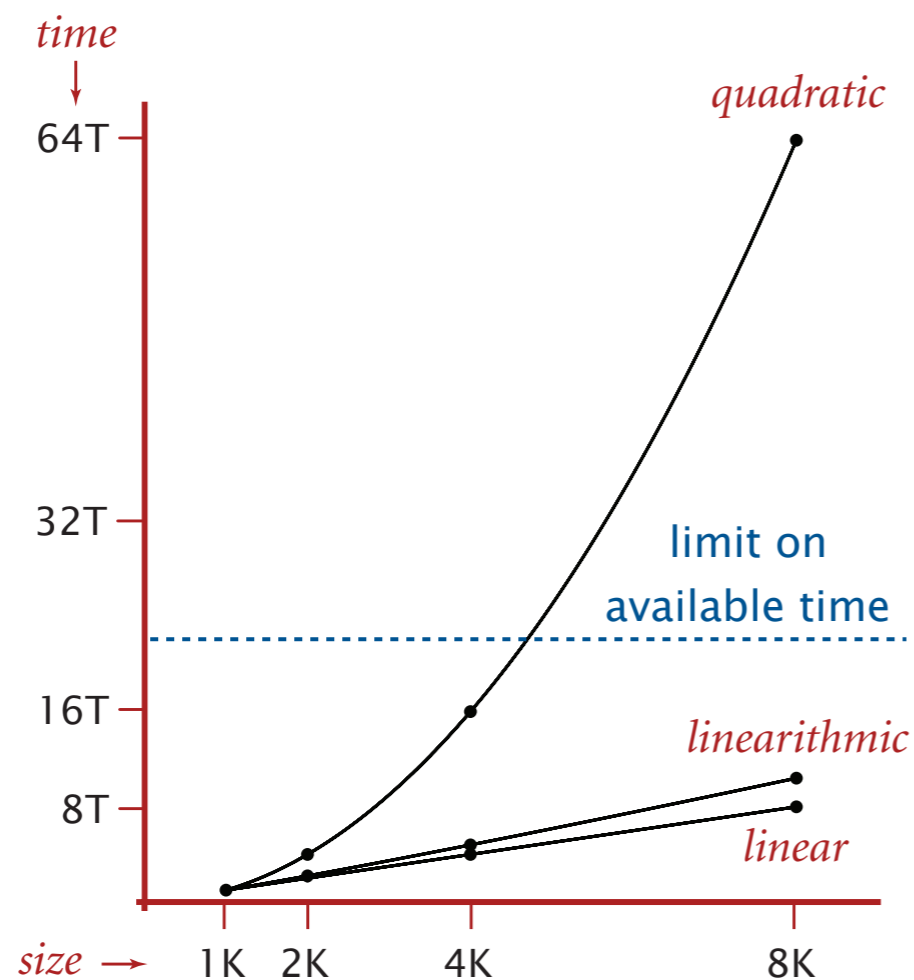
An algorithmic success story

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

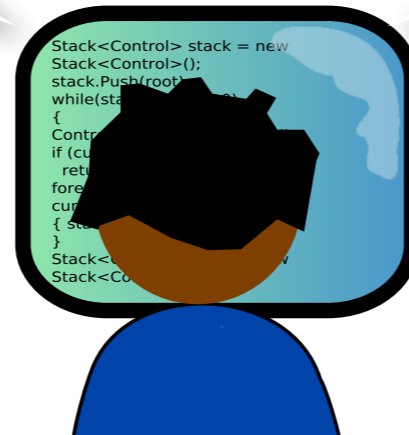


The challenge

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?



Approach: combination of experiments and mathematical modeling.

Observation: experiments and theory complement each other throughout computer science.

What are some computationally intensive tasks that your smartphone or laptop performs in the course of everyday use?



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

1.4 ANALYSIS OF ALGORITHMS

- ▶ *introduction*
- ▶ *experimental analysis of running time*
- ▶ *mathematical models of running time*
- ▶ *order-of-growth classifications*
- ▶ *analysis of memory consumption*

Example: 3-SUM

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

Input
a[] 30 -40 -20 -10 40 0 10 5

Output
4

a[i]	a[j]	a[k]	sum
30	-40	10	0
30	-20	-10	0
-40	40	0	0
-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

Measuring the running time

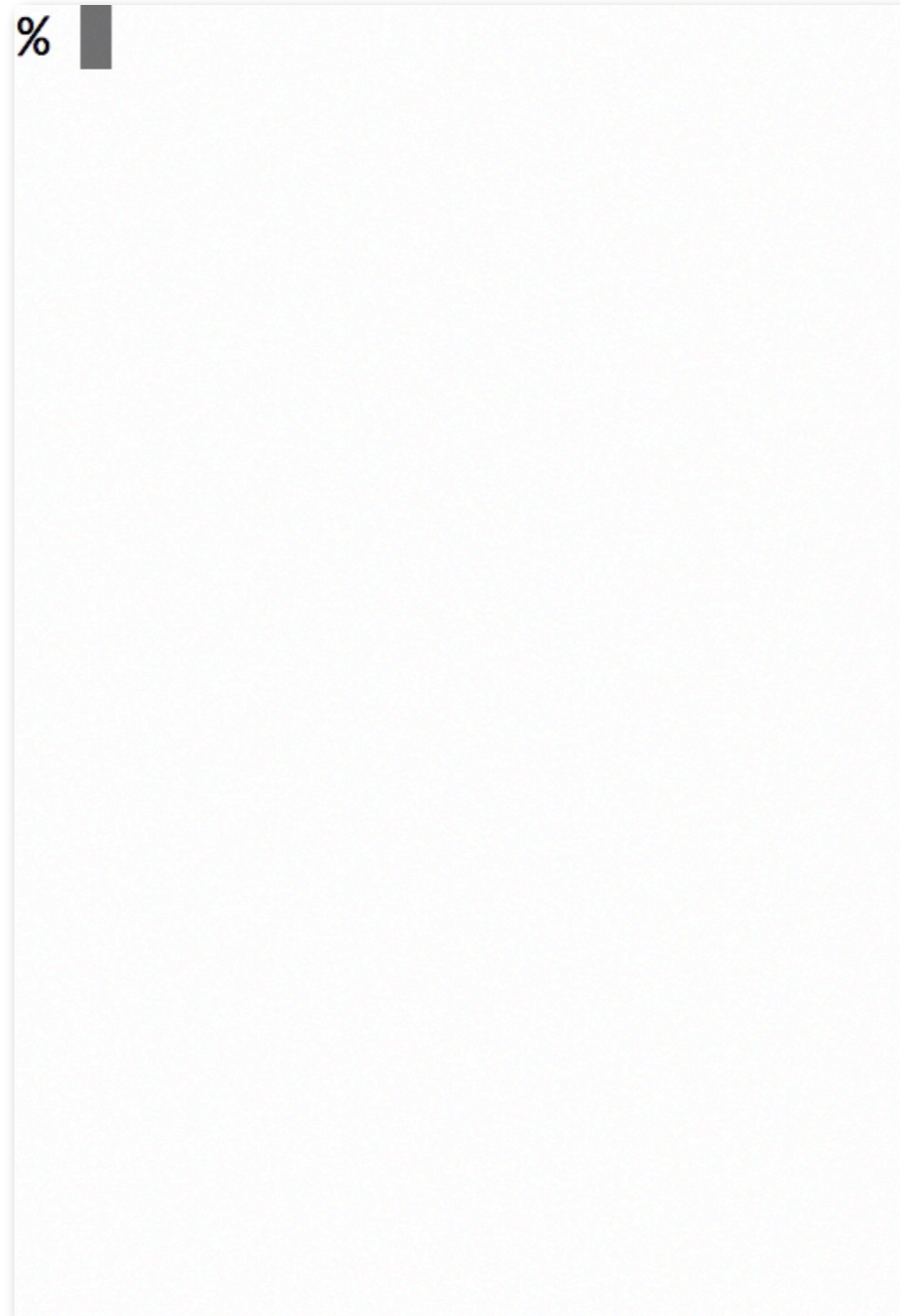
Q. How to time a program?

A. Automatic.

```
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

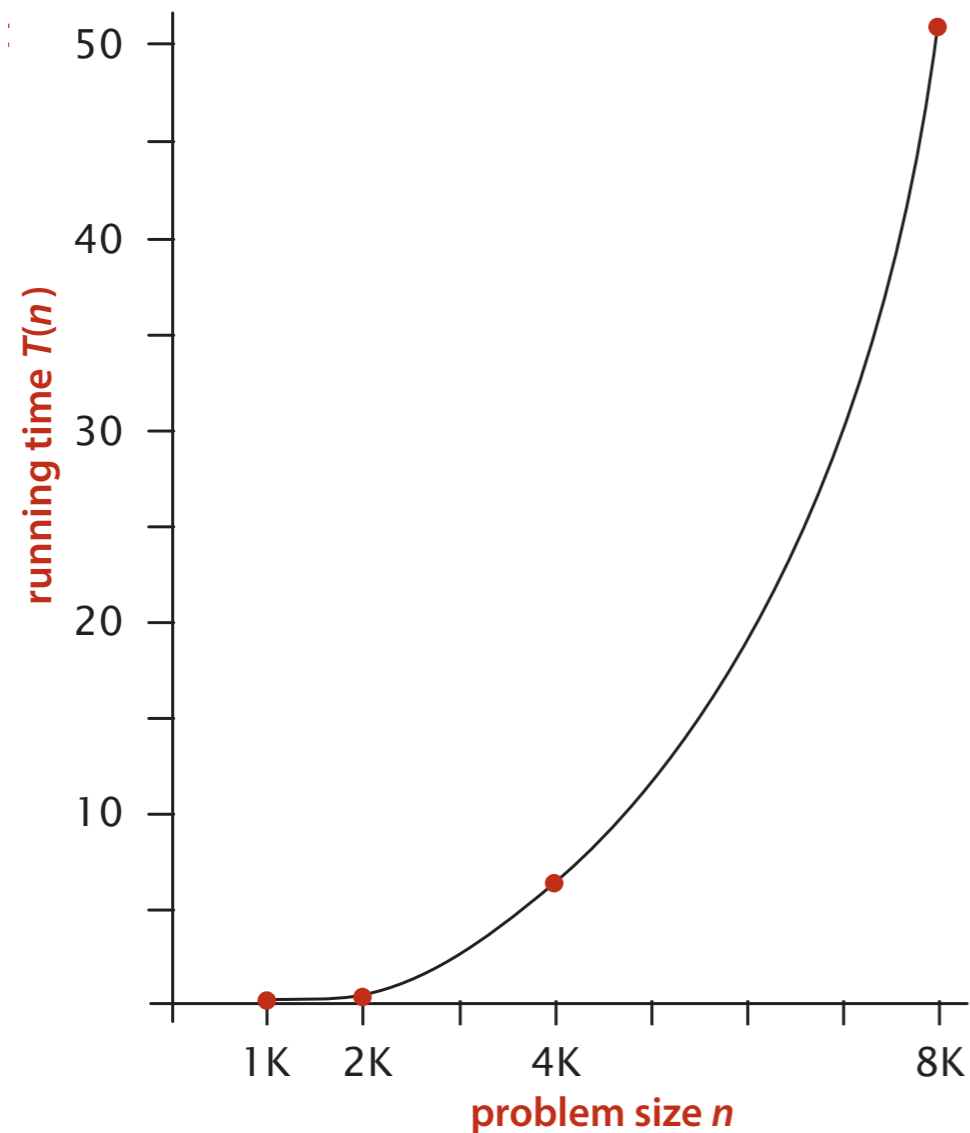
Run the program for various input sizes and measure running time.
(Note: different values compared to previous slide.)

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 $T(n)$ in seconds vs. input size n .



Hypothesis.

$$T(n) = a n^b$$

Q: How to validate hypothesis?

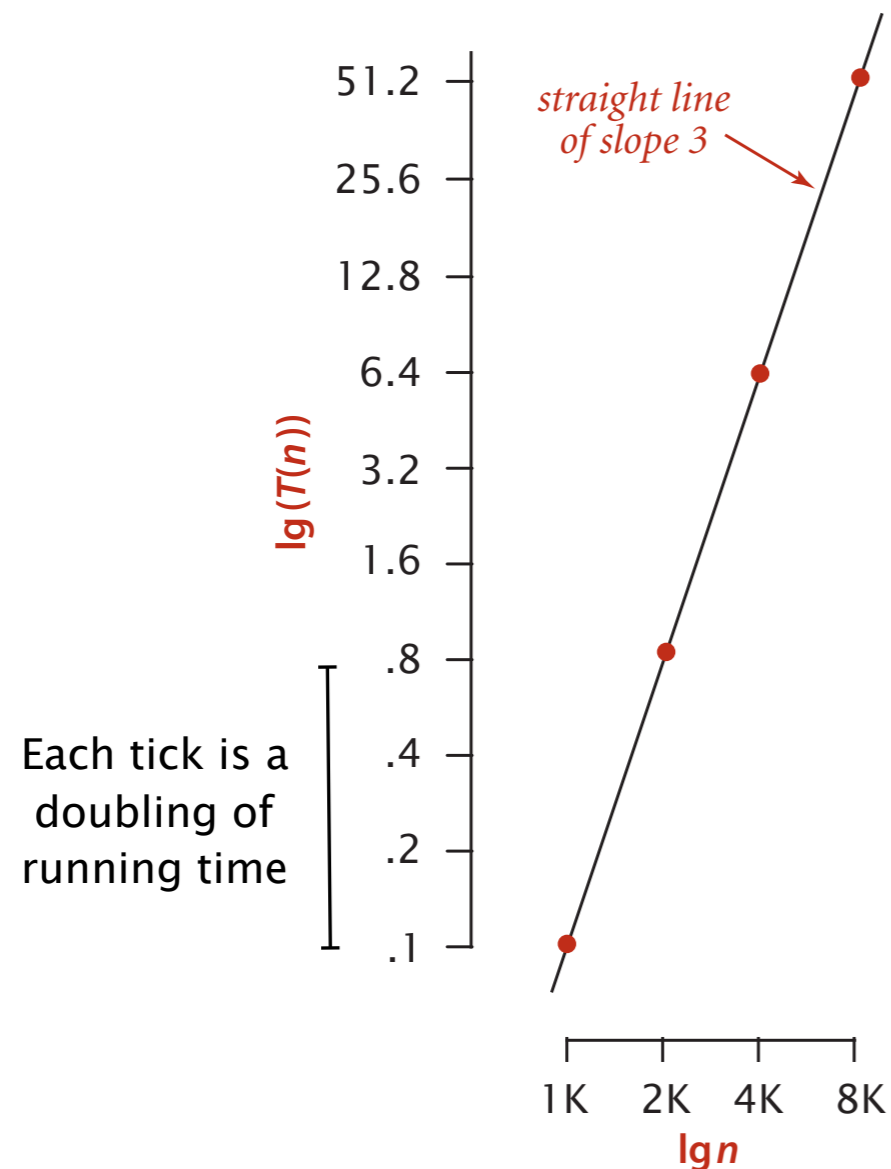
Q: How to estimate a and b from the data?

$$\log_2(T(n)) = b \log_2 n + \log_2 a$$

A: Plot $\log_2(T(n))$ vs. $\log_2 n$

Data analysis

Log-log plot. Plot running time $T(n)$ vs. input size n using **log-log scale**.



$$\log_2(T(n)) = 2.999 \log_2 n + (-33.2103)$$

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


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]



Prediction.

- 408.1 seconds for $n = 16,000$.

Observations.

n	time (seconds) †
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
8,000	51.1	8	3

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

← $\log_2(6.4 / 0.8) = 3.0$

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

Hypothesis. Running time is about an^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 sufficiently large value of n) and solve for a .

n	time (seconds) †
8,000	51.1
8,000	51
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



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 precise measurements.

Good news. Much easier and cheaper than other sciences.

An aside

Algorithmic experiments are virtually free by comparison with other sciences.



Chemistry
(1 experiment)



Mendelian genetics
Biology
(1 experiment)



Computer Science
(1 million experiments)



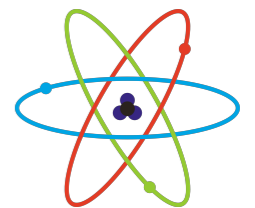
Physics
(1 experiment)

Scientific method applied to the analysis of algorithms

A framework for predicting performance and comparing algorithms.

Scientific method.

- **Observe** some feature of the natural world.
- **Hypothesize** a model that is consistent with the observations.
- **Predict** events using the hypothesis.
- **Verify** the predictions by making further observations.
- **Validate** by repeating until the hypothesis and observations agree.



Principles.

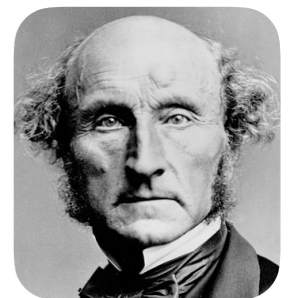
- Experiments must be **reproducible**.
- Hypotheses must be **falsifiable**.



Francis
Bacon



René
Descartes



John Stuart
Mills

Feature of the natural world. Computer itself.



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

1.4 ANALYSIS OF ALGORITHMS

- ▶ *introduction*
- ▶ *experimental analysis of running time*
- ▶ *mathematical models of running time*
- ▶ *order-of-growth classifications*
- ▶ *analysis of memory consumption*

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.
- Cost depends on machine, compiler.
- Frequency depends on algorithm, input data.


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	Sample cost in 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	10	n
increment	1/10	n to $2n$

Varies due to caching, locality, other factors; see COS 217





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. $n (n + 1)$
- D. $2 n^2$
- E. Help I'm confused

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++;
```

operation	frequency
variable declaration	$n + 2$
assignment statement	$n + 2$
less than compare	$\frac{1}{2} (n + 1) (n + 2)$
equal to compare	$\frac{1}{2} n (n - 1)$
array access	$n (n - 1)$
increment	$\frac{1}{2} n (n + 1)$ to n^2

Simplification 1: cost model

Cost model. Use some basic 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	frequency
variable declaration	$n + 2$
assignment statement	$n + 2$
less than compare	$\frac{1}{2} (n + 1) (n + 2)$
equal to compare	$\frac{1}{2} n (n - 1)$
array access	$n (n - 1)$
increment	$\frac{1}{2} n (n + 1)$ to n^2

← cost model = array accesses

Simplification 2: tilde notation

- Estimate running time (or memory) as a function of input size n .
- Ignore lower order terms.

Ex 1. $\frac{1}{6} n^3 + 20 n + 16 \sim \frac{1}{6} n^3$

Ex 2. $\frac{1}{6} n^3 + 100 n^{4/3} + 56 \sim \frac{1}{6} n^3$

Ex 3. $\frac{1}{6} n^3 - \frac{1}{2} n^2 + \frac{1}{3} n \sim \frac{1}{6} n^3$



discard lower-order terms

(e.g., $n = 1,000$: 166.67 million vs. 166.17 million)

Rationale.

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

Technical definition. $f(n) \sim g(n)$ means $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 1$

Simplification 2: tilde notation

- Estimate running time (or memory) as a function of input size n .
- Ignore lower order terms.

operation	frequency	tilde notation
variable declaration	$n + 2$	$\sim n$
assignment statement	$n + 2$	$\sim n$
less than compare	$\frac{1}{2} (n + 1) (n + 2)$	$\sim \frac{1}{2} n^2$
equal to compare	$\frac{1}{2} n (n - 1)$	$\sim \frac{1}{2} n^2$
array access	$n (n - 1)$	$\sim n^2$
increment	$\frac{1}{2} n (n + 1)$ to n^2	$\sim \frac{1}{2} n^2$ to $\sim n^2$

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”

Number of inner loop iterations = $\frac{1}{2} n(n-1) \sim \frac{1}{2} n^2$

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. $\sim 3 * \frac{1}{6} n^3 = \frac{1}{2} n^3$ array accesses.

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

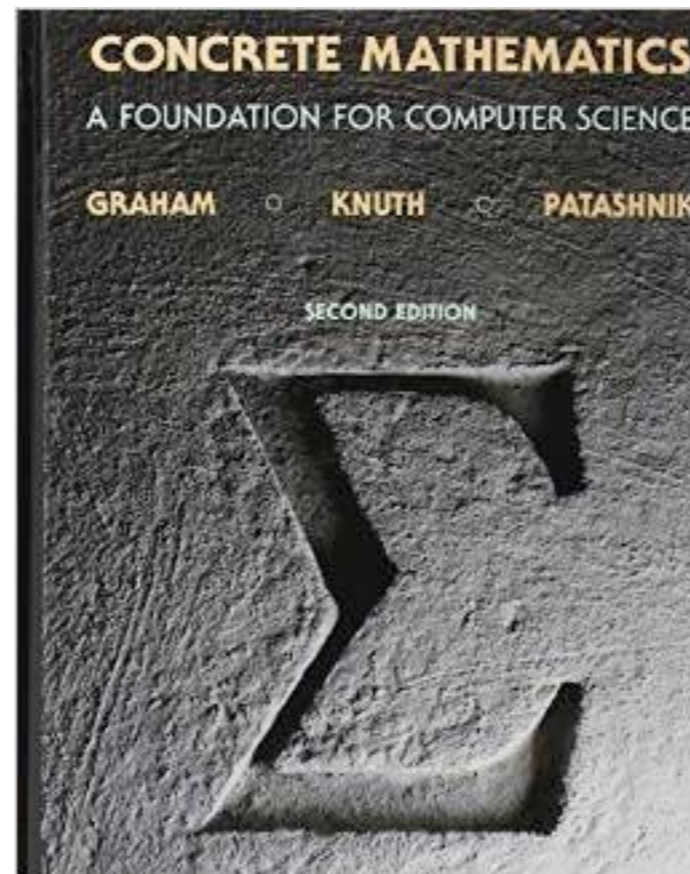
Next week's
precept

Bottom line. Use cost model and tilde notation to simplify counts.

Estimating a discrete sum

Q. How to estimate a discrete sum?

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



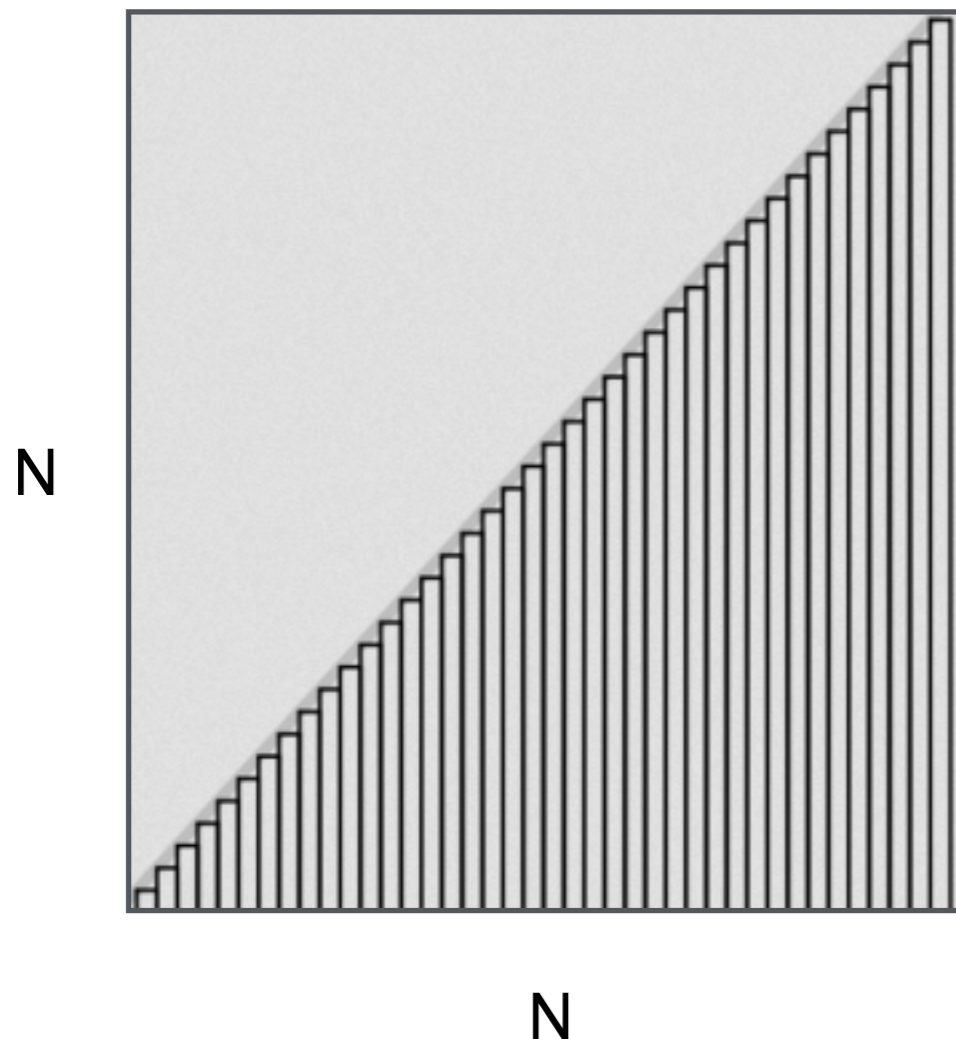
Estimating a discrete sum

Q. How to estimate a discrete sum?

A2. Replace the sum with an integral, and use calculus!

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

$$\sum_{i=1}^N i \sim \int_{x=1}^N x dx \sim \frac{1}{2} N^2$$



Visual proof:

Area occupied by the sum

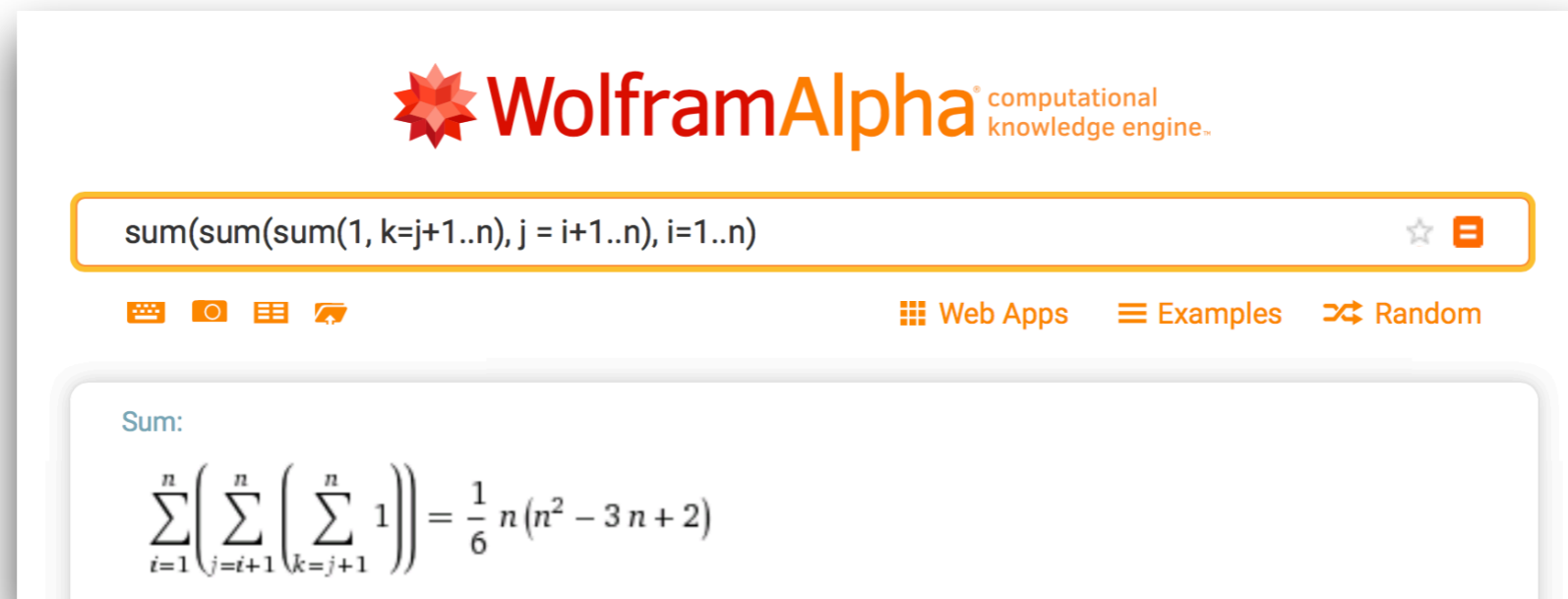
\approx

Half the area of the square

Estimating a discrete sum

Q. How to estimate a discrete sum?

A3. Use a tool like Maple or Wolfram Alpha.



The screenshot shows the Wolfram Alpha interface. At the top, the logo "WolframAlpha" is displayed with the tagline "computational knowledge engine." Below the logo is a search bar containing the input `sum(sum(sum(1, k=j+1..n), j = i+1..n), i=1..n)`. To the right of the search bar are icons for a star and a close button. Below the search bar are several navigation icons: a grid, a magnifying glass, a list, and a refresh icon. To the right of these icons are the links "Web Apps", "Examples", and "Random". Below the search bar, the word "Sum:" is displayed in blue. Underneath, the result is shown as a mathematical equation:
$$\sum_{i=1}^n \left(\sum_{j=i+1}^n \left(\sum_{k=j+1}^n 1 \right) \right) = \frac{1}{6} n (n^2 - 3n + 2)$$



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. $\sim 3/2 n^3$



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

1.4 ANALYSIS OF ALGORITHMS

- ▶ *introduction*
- ▶ *experimental analysis of running time*
- ▶ *mathematical models of running time*
- ▶ *order-of-growth classifications*
- ▶ *analysis of memory consumption*

Common order-of-growth classifications


Definition. If $f(n) \sim c g(n)$ for some constant $c > 0$, then the **order of growth** of $f(n)$ is $g(n)$.

- Ignores leading coefficient.
- Ignores lower-order terms.

Ex. The order of growth of the **running time** of this code is n^3 .

```
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++;
```

Typical usage. Mathematical analysis of running times.

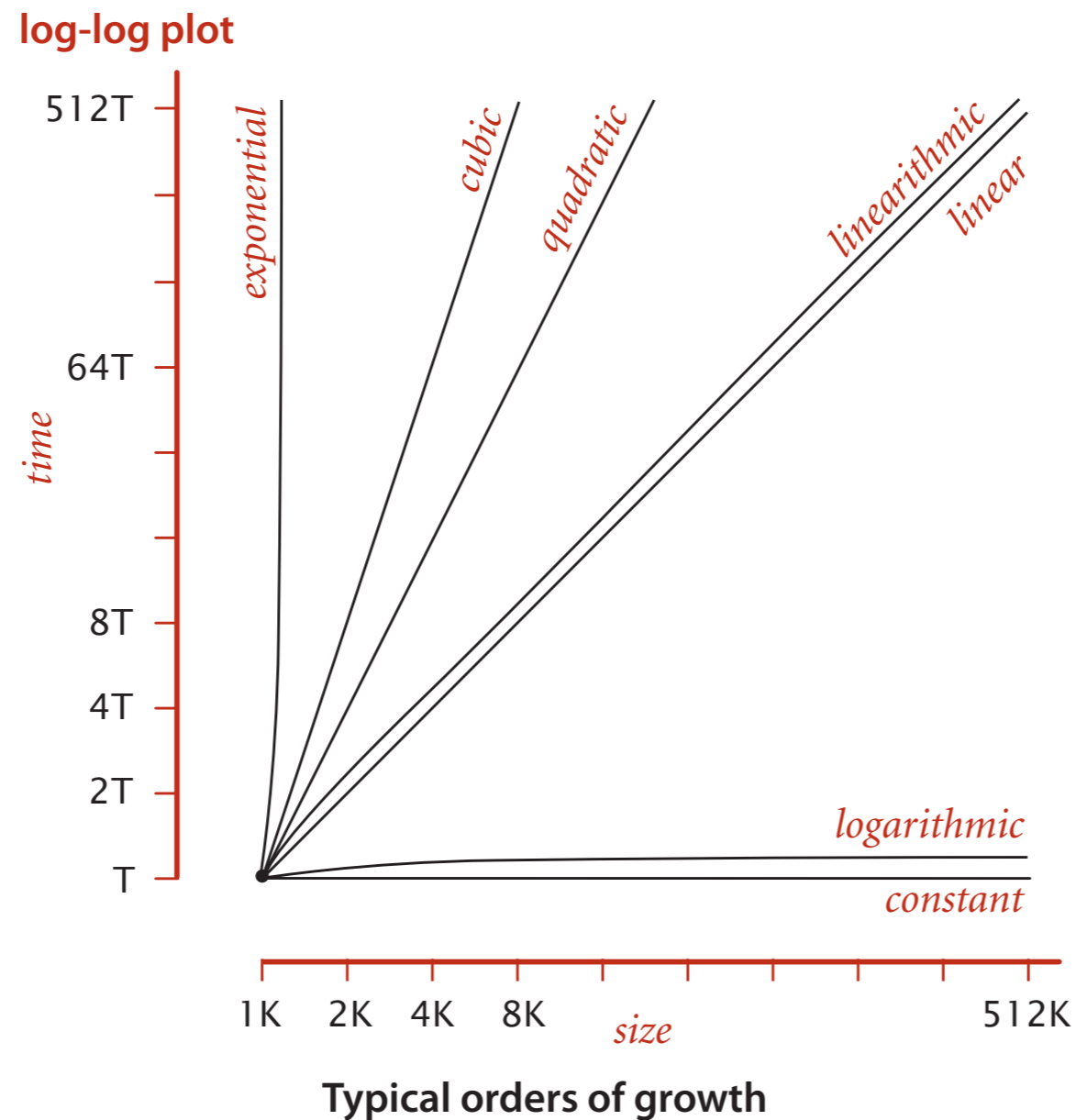
 where leading coefficient
depends on machine, compiler, JVM, ...

Common order-of-growth classifications

Good news. The set of functions

1, $\log n$, n , $n \log n$, n^2 , n^3 , and 2^n

suffices to describe the order of growth of most common algorithms.



Common order-of-growth classifications

order of growth	name	typical code framework	description	example	$T(2n) / T(n)$
1	constant	<pre>a = b + c;</pre>	statement	add two numbers	1
$\log n$	logarithmic	<pre>while (n > 1) { n = n/2; ... }</pre>	divide in half	binary search	~ 1
n	linear	<pre>for (int i = 0; i < n; i++) { ... }</pre>	single loop	find the maximum	2
$n \log n$	linearithmic	<i>see mergesort lecture</i>	divide and conquer	mergesort	~ 2
n^2	quadratic	<pre>for (int i = 0; i < n; i++) for (int j = 0; j < n; j++) { ... }</pre>	double loop	check all pairs	4
n^3	cubic	<pre>for (int i = 0; i < n; i++) for (int j = 0; j < n; j++) for (int k = 0; k < n; k++) { ... }</pre>	triple loop	check all triples	8
2^n	exponential	<i>see combinatorial search lecture</i>	exhaustive search	check all subsets	2^n

Example: 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: mathematical analysis

Proposition. Binary search uses at most $1 + \log_2 n$ key comparisons to search in a sorted array of length n .

Proof sketch.

- After 1 comparison, we are left with a sub-array of length at most $n/2$
- After $\log_2 n$ comparisons, we are left with a sub-array of length at most 1

(Not a complete proof; some details are missing.)

Aside: binary search is tricky to implement correctly

- 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.



<http://googleresearch.blogspot.com/2006/06/extra-extra-read-all-about-it-nearly.html>

Binary search: Java implementation

Invariant. If key appears in array $a[]$, then $a[lo] \leq key \leq a[hi]$.

Skipped
in class

```
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"



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

1.4 ANALYSIS OF ALGORITHMS

- *introduction*
- *experimental analysis of running time*
- *mathematical models of running time*
- *order-of-growth classifications*
- ***analysis of memory consumption***

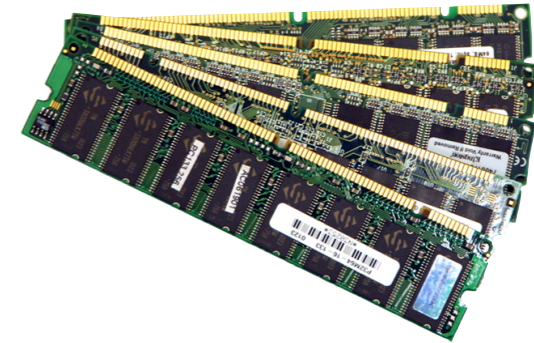
Basics

Bit. 0 or 1.

Byte. 8 bits.

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

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



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



In practice, machine dependent



Typical memory usage for primitive types and arrays

Primitive types.

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

primitive types

Array overhead. 24 bytes.

type	bytes
char[]	$2n + 24$
int[]	$4n + 24$
double[]	$8n + 24$

one-dimensional array (length n)

type	bytes
char[][]	$\sim 2 m n$
int[][]	$\sim 4 m n$
double[][]	$\sim 8 m n$

two-dimensional array (m-by-n)

Typical memory usage for objects in Java

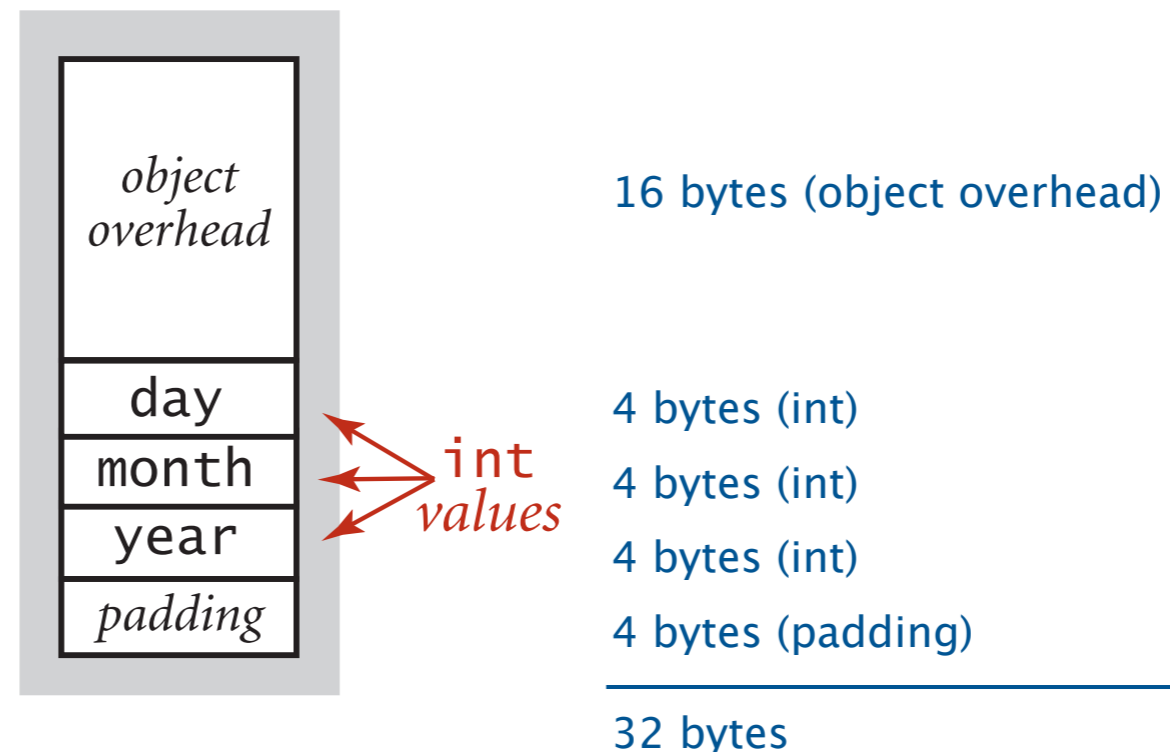
Object overhead. 16 bytes.

Reference. 8 bytes.

Padding. Each object uses 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;
    ...
}
```




Typical memory usage summary

Total memory usage for a data type value:

- 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.

+ 8 extra bytes per inner class object
(for reference to enclosing class)



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

“deep memory”





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;

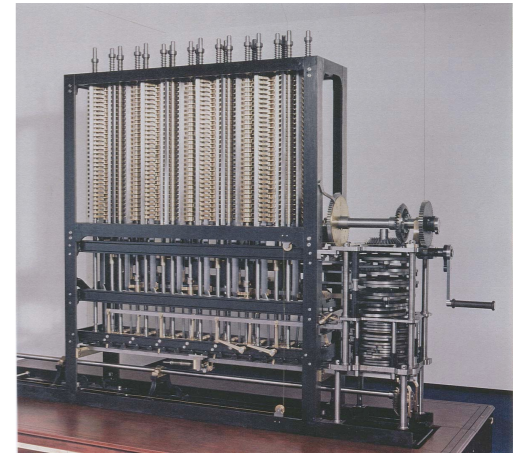
    public WeightedQuickUnionUF(int n)
    {
        parent = new int[n];
        size = new int[n];

        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**.

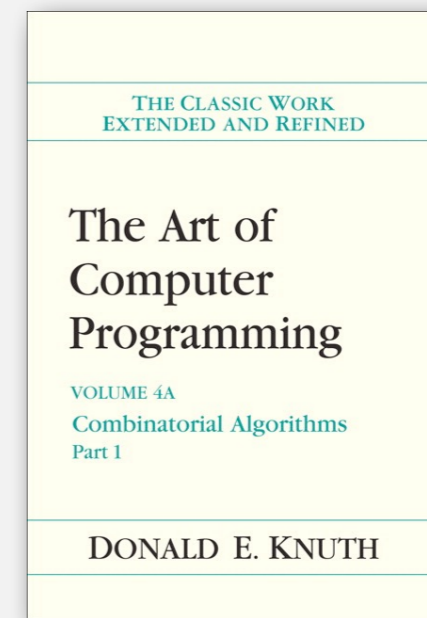
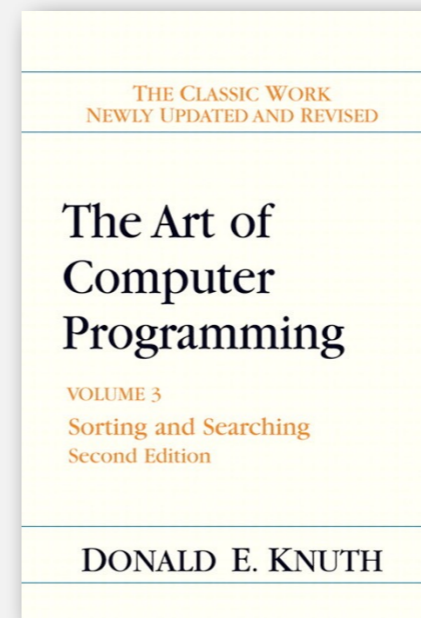
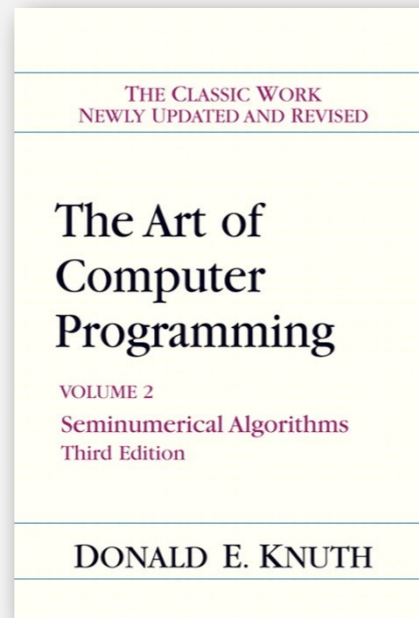
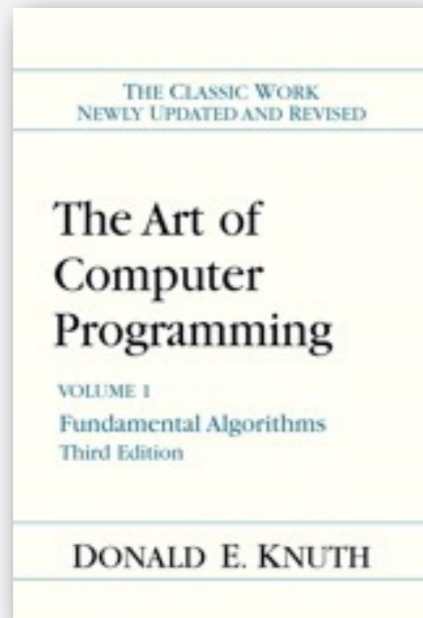


Mathematical analysis.

- Analyze algorithm to count frequency of operations.
- Use tilde notation to simplify analysis.
- Model enables us to **explain behavior**.

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

Historical note: analysis of algorithms was pioneered by Donald Knuth



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."

