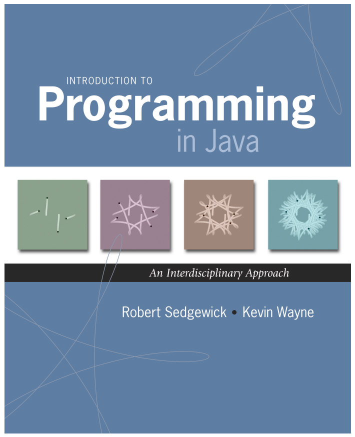
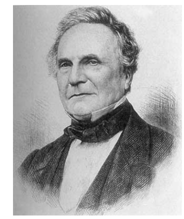
4.1 Performance

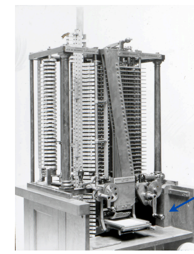


Running Time

"As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise - By what course of calculation can these results be arrived at by the machine in the shortest time?" – Charles Babbage



Charles Babbage (1864)



how many times do you have to turn the crank?

Analytic Engine

The Challenge

compile

debug on test cases

solve problems in practice

Will my program be able to solve a large practical problem?

Key insight (Knuth 1970s):

Use the scientific method to understand performance.

Scientific Method

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.

Reasons to Analyze Algorithms

Predict performance
- Will my program finish?
- When will my program finish?

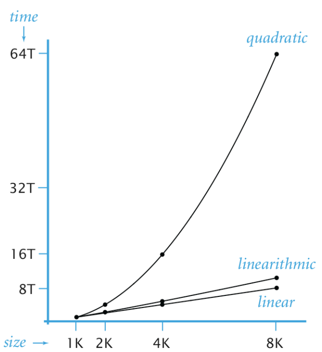
Compare algorithms
- Will this change make my program faster?
- How can I make my program faster?

Basis for inventing new ways to solve problems
- Enables new technology
- Enables new research

Algorithmic Successes

Discrete Fourier transform.
- Break down waveform of N samples into periodic components.
- Applications: DVD, JPEG, MRI, astrophysics, ….
- Brute force: N^2 steps.
- FFT algorithm: N log N steps, enables new technology.

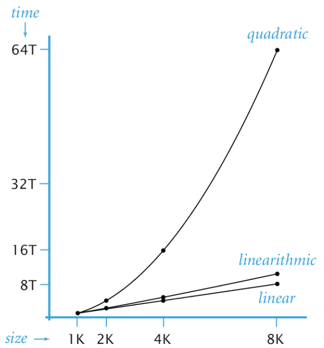
Freidrich Gauss
1805



Algorithmic Successes

N-body Simulation.
- Simulate gravitational interactions among N bodies.
- Brute force: N^2 steps.
- Barnes-Hut: N log N steps, enables new research.

Andrew Appel
PU '81



Example: Three-Sum Problem

Three-sum problem. Given N integers, find triples that sum to 0.
Context. Deeply related to problems in computational geometry.

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

% java ThreeSum < 8ints.txt
 4
 30 -30   0
 30 -20 -10
-30 -10  40
-10   0  10
```

Q. How would you write a program to solve the problem?

Three-Sum

```
public class ThreeSum
{
   // Return number of distinct triples (i, j, k)
   //     such that (a[i] + a[j] + a[k] == 0)
   public static int count(int[] a) {
      int N = a.length;
      int cnt = 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) cnt++;
      return cnt;
   }

   public static void main(String[] args) {
      int[] a = StdArrayIO.readInt1D();
      StdOut.println(count(a));
   }
}
```

all possible triplets
`i < j < k`

Empirical Analysis

Empirical Analysis

Empirical analysis. Run the program for various input sizes.

N	*time* †
512	0.03
1024	0.26
2048	2.16
4096	17.18
8192	136.76

† Running Linux on Sun-Fire-X4100 with 16GB RAM

Stopwatch

Q. How to time a program?
A. A stopwatch.

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


```
0
% java ThreeSum < 2Kints.txt
```


```
2
391930676 -763182495 371251819
-326747290 802431422 -475684132
```

Stopwatch

Q. How to time a program?
A. A `Stopwatch` object.

public class Stopwatch

	Stopwatch()	*create a new stopwatch and start it running*
double	elapsedTime()	*return the elapsed time since creation, in seconds*

```
public class Stopwatch
{
   private final long start;

   public Stopwatch()
   {
      start = System.currentTimeMillis();
   }

   public double elapsedTime()
   {
      return (System.currentTimeMillis() - start) / 1000.0;
   }
}
```


Stopwatch

Q. How to time a program?
A. A `Stopwatch` object.

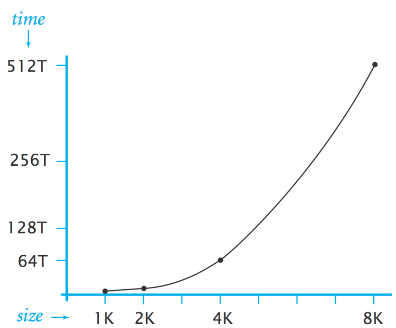
public class Stopwatch

	Stopwatch()	*create a new stopwatch and start it running*
double	elapsedTime()	*return the elapsed time since creation, in seconds*

```
public static void main(String[] args)
{
   int[] a = StdArrayIO.readInt1D();
   Stopwatch timer = new Stopwatch();
   StdOut.println(count(a));
   StdOut.println(timer.elapsedTime());
}
```


Empirical Analysis

Data analysis. Plot running time vs. input size N.



Q. How does running time grow as a function of input size N ?

Empirical Analysis

Data analysis. Plot running time vs. input size N on a log-log scale.

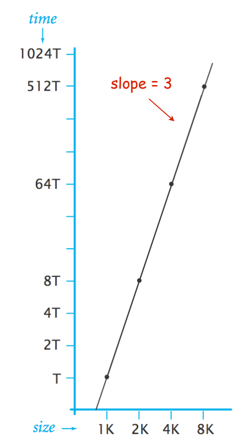
Initial hypothesis: Running time satisfies a power law

$$f(N) = a N^b$$

On log-log scale,

$$\lg (F(N)) = b \lg N + a$$

Estimate slope by fitting line through data points.



slope

Refined hypothesis. Running time grows as the cube of the input size: a N^3.

Doubling Hypothesis

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

Run program, doubling the size of the input.

N	time †	ratio
512	0.033	-
1024	0.26	7.88
2048	2.16	8.43
4096	17.18	7.96
8192	136.76	7.96

$$\frac{a\,(2N)^b}{a\,(N)^b} = 2^b$$

Seems to converge to a constant c_0?

Hypothesize that running time is about $a\,N^b$ with $b = \lg c_0$

Doubling Challenge 1

Let F(N) be the running time of program **Mystery** for input N.

```
public static Mystery
{
   ...
   int N = Integer.parseInt(args[0]);
   ...
}
```

Observation: F(2N)/F(N) is about 4.

What is the order of growth of the running time?

Doubling Challenge 1

Let F(N) be the running time of program **Mystery** for input N.

```
public static Mystery
{
   ...
   int N = Integer.parseInt(args[0]);
   ...
}
```

Observation: F(2N)/F(N) is about 4.

What is the order of growth of the running time?

A. Quadratic: $a\,N^2$

$$\frac{a\,(2N)^2}{a\,(N)^2} = 4$$

Doubling Challenge 2

Let F(N) be the running time of program **Mystery** for input N.

```
public static Mystery
{
   ...
   int N = Integer.parseInt(args[0]);
   ...
}
```

Observation: F(2N)/F(N) is about 2.

What is the order of growth of the running time?

Doubling Challenge 2

Let F(N) be the running time of program **`Mystery`** for input N.

```
public static Mystery
{
   ...
   int N = Integer.parseInt(args[0]);
   ...
}
```

Observation: F(2N)/F(N) is about 2.

What is the order of growth of the running time?

A. Linear: a N $\qquad \frac{a\,(2N)}{a\,(N)} = 2$

Could be a N lg N $\qquad \frac{a\,(2N)\,\lg\,(2N)}{a\,N\,\lg\,N} = 2 + \frac{2\,\lg\,2}{\lg\,N}$

Prediction and Validation

Hypothesis. Running time is about a N^3 seconds for input of size N.

Q. How to estimate a ?

A. Run the program!

N	time †
4096	17.18
4096	17.15
4096	17.17

† Running Linux on Sun-Fire-X4100 with 16GB RAM

$17.17 = a\ 4096^3$
$a = 17.17 / 4096^3$
$= 1.25 \times 10^{-10}$

Refined hypothesis. Running time is about $1.25 \times 10^{-10}\ N^3$ seconds.

Prediction. 1100 seconds for $N = 16{,}384$.

Observation.

N	time †
16384	1118.86

← validates hypothesis!

Mathematical Analysis

Donald Knuth
Turing award '74

Mathematical Analysis

Running time. Count up frequency of execution of each instruction and weight by its execution time.

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

operation	frequency
variable declaration	2
variable assignment	2
less than comparison	$N + 1$
equal to comparison	N
array access	N
increment	$\leq 2N$

between N (no zeros) and 2N (all zeros)

Mathematical Analysis

Running time. Count up frequency of execution of each instruction and weight by its execution 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	$N+2$
variable assignment	$N+2$
less than comparison	$1/2\ (N+1)\ (N+2)$
equal to comparison	$1/2\ N\ (N-1)$
array access	$N\ (N-1)$
increment	$\le N^2$

$0 + 1 + 2 + \ldots + (N-1) = 1/2\ N(N-1)$

becoming very tedious to count

Tilde Notation

Tilde notation.

- Estimate running time as a function of input size N.
- Ignore lower order terms.
  - when N is large, terms are negligible
  - when N is small, we don't care

Ex 1. $6N^3 + 17N^2 + 56 \sim 6N^3$

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

Ex 3. $6N^3 + 17N^2 \log N \sim 6N^3$

discard lower-order terms
(e.g., N = 1000: 6 trillion vs. 169 million)

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

Mathematical Analysis

Running time. Count up frequency of execution of each instruction and weight by its execution time.

```
{
   int N = a.length;
   int cnt = 0;                                  ← 1
   for (int i = 0; i < N; i++)
      for (int j = i+1; j < N; j++)              ← N
         for (int k = j+1; k < N; k++)           ← ~N²/2
            if (a[i] + a[j] + a[k] == 0)         ← ~N³/6
               cnt++;
   return cnt;
}
```

inner loop

depends on input data

Inner loop. Focus on instructions in "inner loop."

Constants in Power Law

Power law. Running time of a typical program is $\sim a N^b$.

Exponent b depends on: algorithm.

Constant a depends on:

- algorithm
- input data
- caching
- machine
- compiler
- garbage collection
- just-in-time compilation
- CPU use by other applications

system independent effects

system dependent effects

Our approach. Use doubling hypothesis (or mathematical analysis) to estimate exponent b, run experiments to estimate a.

Analysis: Empirical vs. Mathematical

Empirical analysis.

- Measure running times, plot, and fit curve.
- Easy to perform experiments.
- Model useful for predicting, but not for explaining.

Mathematical analysis.

- Analyze algorithm to estimate # ops as a function of input size.
- May require advanced mathematics.
- Model useful for predicting and explaining.

Critical difference. Mathematical analysis is independent of a particular machine or compiler; applies to machines not yet built.

Order of Growth Classifications

Observation. A small subset of mathematical functions suffice to describe running time of many fundamental algorithms.

```
while (N > 1) {
   N = N / 2;
   ...
}
```

$\lg N$

$\lg N = \log_2 N$

```
for (int i = 0; i < N; i++)
   ...
```

N

```
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      ...
```

N^2

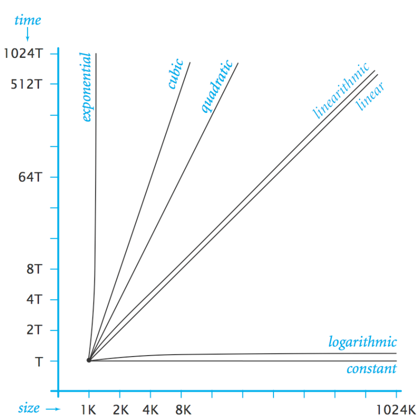
```
public static void g(int N) {
   if (N == 0) return;
   g(N/2);
   g(N/2);
   for (int i = 0; i < N; i++)
      ...
}
```

$N \lg N$

```
public static void f(int N) {
   if (N == 0) return;
   f(N-1);
   f(N-1);
   ...
}
```

2^N

Order of Growth Classifications



Orders of growth (log-log plot)

order of growth		factor for doubling hypothesis
description	function	
constant	1	1
logarithmic	$\log N$	1
linear	N	2
linearithmic	$N \log N$	2
quadratic	N^2	4
cubic	N^3	8
exponential	2^N	2^N

Order of Growth: Consequences

order of growth	predicted running time if problem size is increased by a factor of 100
linear	a few minutes
linearithmic	a few minutes
quadratic	several hours
cubic	a few weeks
exponential	forever

Effect of increasing problem size for a program that runs for a few seconds

order of growth	predicted factor of problem size increase if computer speed is increased by a factor of 10
linear	10
linearithmic	10
quadratic	3-4
cubic	2-3
exponential	1

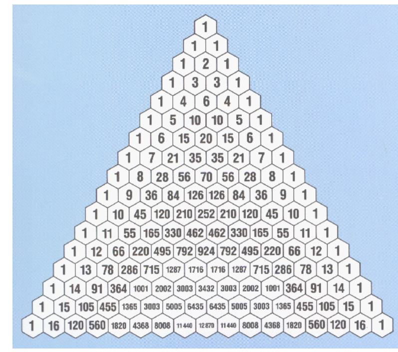
Effect of increasing computer speed on problem size that can be solved in a fixed amount of time

Dynamic Programming

Binomial Coefficients

Binomial coefficient. $\binom{n}{k}$ = number of ways to choose k of n elements.

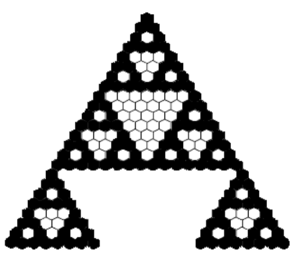
Pascal's identity. $\binom{n}{k} = \underbrace{\binom{n-1}{k-1}}_{\text{contains first element}} + \underbrace{\binom{n-1}{k}}_{\text{excludes first element}}$



Binomial Coefficients: Sierpinski Triangle

Binomial coefficient. $\binom{n}{k}$ = number of ways to choose k of n elements.

Sierpinski triangle. Color black the odd integers in Pascal's triangle.

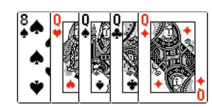


Binomial Coefficients: Poker Odds

Binomial coefficient. $\binom{n}{k}$ = number of ways to choose k of n elements.

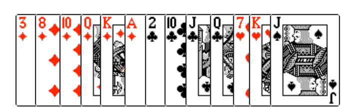
Probability of "quads" in Texas hold 'em:

$$\frac{\binom{13}{1} \times \binom{48}{3}}{\binom{52}{7}} = \frac{224{,}848}{133{,}784{,}560} \quad (\textit{about} \;\; 594:1)$$



Probability of 6-4-2-1 split in bridge:

$$\frac{\binom{4}{1} \times \binom{13}{6} \times \binom{3}{1} \times \binom{13}{4} \times \binom{2}{1} \times \binom{13}{2} \times \binom{1}{1} \times \binom{13}{1}}{\binom{52}{13}} = \frac{29{,}858{,}811{,}840}{635{,}013{,}559{,}600} \quad (\textit{about} \;\; 21:1)$$



Binomial Coefficients: First Attempt

```
public class SlowBinomial
{
   // Natural recursive implementation
   public static long binomial(long n, long k)
   {
      if (k == 0) return 1;
      if (n == 0) return 0;
      return binomial(n-1, k-1) + binomial(n-1, k);
   }

   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      int K = Integer.parseInt(args[1]);
      StdOut.println(binomial(N, K));
   }

}
```


Performance Challenge 3

Is this an efficient way to compute binomial coefficients?

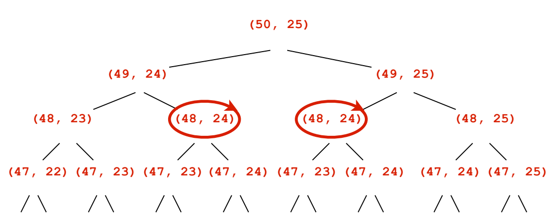
```
   public static long binomial(long n, long k)
   {
      if (k == 0) return 1;
      if (n == 0) return 0;
      return binomial(n-1, k-1) + binomial(n-1, k);
   }
```


Performance Challenge 3

Is this an efficient way to compute binomial coefficients?

```
   public static long binomial(long n, long k)
   {
      if (k == 0) return 1;
      if (n == 0) return 0;
      return binomial(n-1, k-1) + binomial(n-1, k);
   }
```

A. NO, NO, NO: same essential recomputation problem as naive Fibonacci.



recursion tree for naïve binomial function

Timing Experiments

Timing experiments: direct recursive solution.

(2n, n)	time †
(26, 13)	0.46
(28, 14)	1.27
(30, 15)	4.30
(32, 16)	15.69
(34, 17)	57.40
(36, 18)	230.42

increase n by 1, running time increases by about 4x

† Running Linux on Sun-Fire-X4100 with 16GB RAM

Q. Is running time linear, quadratic, cubic, exponential in n?

Performance Challenge 4

Let F(N) be the time to compute binomial(2N, N) using the naive algorithm.

```
public static long binomial(long n, long k)
{
   if (k == 0) return 1;
   if (n == 0) return 0;
   return binomial(n-1, k-1) + binomial(n-1, k);
}
```

Observation: F(N+1)/F(N) is about 4.

What is the order of growth of the running time?

Performance Challenge 4

Let F(N) be the time to compute binomial(2N, N) using the naive algorithm.

```
public static long binomial(long n, long k)
{
   if (k == 0) return 1;
   if (n == 0) return 0;
   return binomial(n-1, k-1) + binomial(n-1, k);
}
```

Observation: F(N+1)/F(N) is about 4.

What is the order of growth of the running time?

A. EXPONENTIAL: a 4^N

Will not finish unless N is small.

Dynamic Programming

Key idea. Save solutions to subproblems to avoid recomputation.

k

n	0	1	2	3	4
0	1	0	0	0	0
1	1	1	0	0	0
2	1	2	1	0	0
3	1	3	3	1	0
4	1	4	6	4	1
5	1	5	10	10	5
6	1	6	15	20	15

binomial(n, k)

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

20 = 10 + 10

Tradeoff. Trade (a little) memory for (a huge amount of) time.

Binomial Coefficients: Dynamic Programming

```
public class Binomial
{
   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      int K = Integer.parseInt(args[1]);
      long[][] bin = new long[N+1][K+1];

      // base cases
      for (int k = 1; k <= K; k++) bin[0][K] = 0;
      for (int n = 0; n <= N; n++) bin[N][0] = 1;

      // bottom-up dynamic programming
      for (int n = 1; n <= N; n++)
         for (int k = 1; k <= K; k++)
            bin[n][k] = bin[n-1][k-1] + bin[n-1][k];

      // print results
      StdOut.println(bin[N][K]);
   }
}
```

Timing Experiments

Timing experiments for binomial coefficients with dynamic programming.

(2n, n)	time †
(26, 13)	instant
(28, 14)	instant
(30, 15)	instant
(32, 16)	instant
(34, 17)	instant
(36, 18)	instant

† Running Linux on Sun-Fire-X4100 with 16GB RAM

Performance Challenge 5

Let F(N) be the time to compute binomial(2N, N) using dynamic programming.

```
for (int n = 1; n <= 2*N; n++)
   for (int k = 1; k <= N; k++)
      bin[n][k] = bin[n-1][k-1] + bin[n-1][k];
```

What is the order of growth of the running time?

Performance Challenge 5

Let F(N) be the time to compute binomial(2N, N) using dynamic programming.

```
for (int n = 1; n <= 2*N; n++)
   for (int k = 1; k <= N; k++)
      bin[n][k] = bin[n-1][k-1] + bin[n-1][k];
```

What is the order of growth of the running time?

A. Quadratic: a N^2

Effectively instantaneous for small N.

Key point: There is PROFOUND DIFFERENCE between 4^N and N^2

cannot solve a large problem (4^N) — can solve a large problem (N^2)

Stirling's Approximation

An alternative approach: $\binom{n}{k} = \frac{n!}{n!\,(n-k)!}$

Doesn't work: 52! overflows a long, even though final result doesn't.

Instead of computing exact values, use Stirling's approximation:

$$\ln n! \approx n \ln n - n + \frac{\ln(2\pi n)}{2} + \frac{1}{12n} - \frac{1}{360n^3} + \frac{1}{1260n^5}$$

Application. Probability of exact k heads in n flips with a biased coin.

$$\binom{n}{k} p^k (1-p)^{n-k}$$

Easy to compute approximate value with Stirling's formula

Memory

Typical Memory Requirements for Java Data Types

Bit. 0 or 1.
Byte. 8 bits.
Megabyte (MB). 2^{10} bytes ~ 1 million bytes.
Gigabyte (GB). 2^{20} bytes ~ 1 billion bytes.

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

type	*bytes*
`int[]`	$4N + 16$
`double[]`	$8N + 16$
`int[][]`	$4N^2 + 20N + 16$
`double[][]`	$8N^2 + 20N + 16$
`String`	$2N + 40$

typical computer '08 has about 1GB memory

Q. What's the biggest `double` array you can store on your computer?

Performance Challenge 6

How much memory does this program use (as a function of N)?

```
public class RandomWalk
{
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      int[][] count = new int[N][N];
      int x = N/2;
      int y = N/2;

      for (int i = 0; i < N; i++)  {
         // no new variable declared in loop
         ...
         count[x][y]++;
      }
   }
}
```


Performance Challenge 6

How much memory does this program use (as a function of N)?

```
public class RandomWalk
{
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      int[][] count = new int[N][N];
      int x = N/2;
      int y = N/2;

      for (int i = 0; i < N; i++)  {
         // no new variable declared in loop
         ...
         count[x][y]++;
      }
   }
}
```


A. ~ $4 N^2$ bytes.

Summary

Q. How can I evaluate the performance of my program?
A. Computational experiments, mathematical analysis, scientific method

Q. What if it's not fast enough? Not enough memory?
- Understand why.
- Buy a faster computer.
- Learn a better algorithm (COS 226, COS 423).
- Discover a new algorithm.

attribute	better machine	better algorithm
cost	$$$ or more.	$ or less.
applicability	makes "everything" run faster	does not apply to some problems
improvement	quantitative improvements	dramatic qualitative improvements possible