



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

## 1.4 ANALYSIS OF ALGORITHMS

---

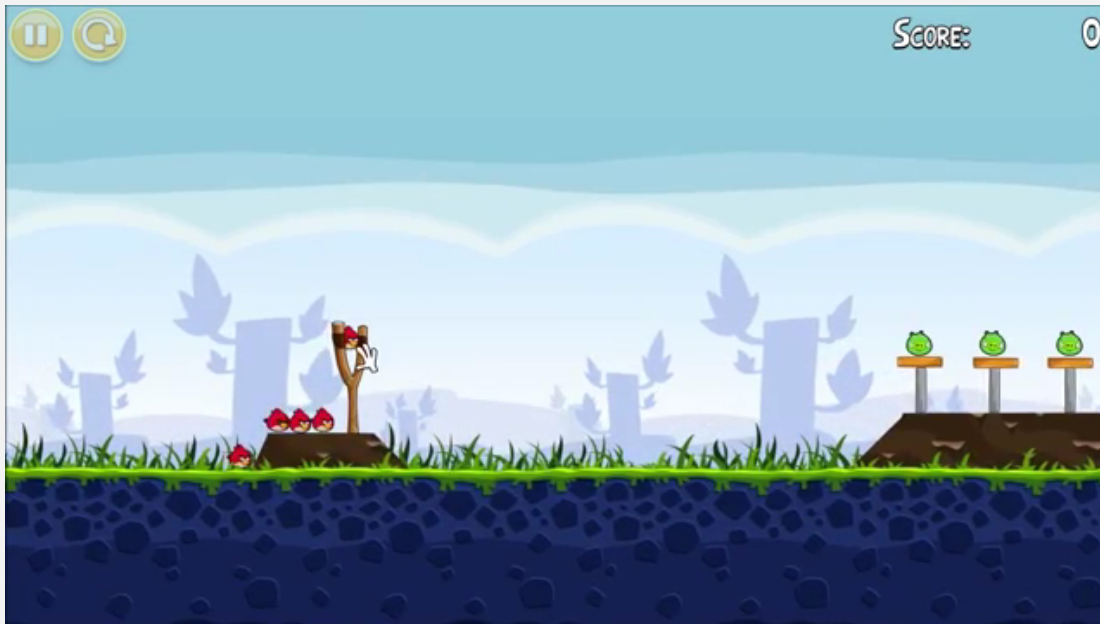
- ▶ *introduction*
- ▶ *observations*
- ▶ *mathematical models*
- ▶ *order-of-growth classifications*
- ▶ *theory of algorithms*
- ▶ *memory*

# 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.** Deeply 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. Manual.



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



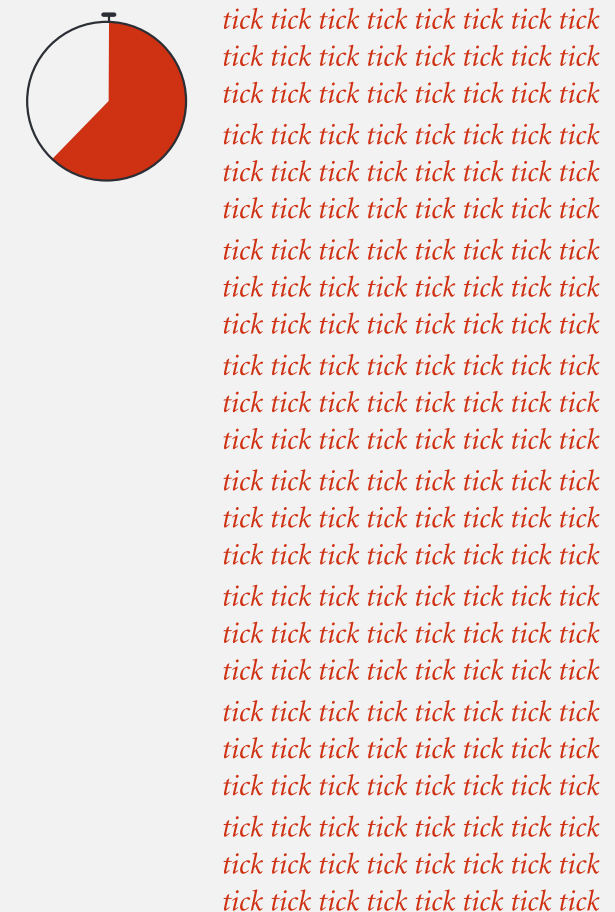
70

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



528

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



4039

# Measuring the running time

---

Q. How to time a program?

A. Automatic.

```
public class Stopwatch (part of stdlib.jar)
```

```
    Stopwatch() create a new stopwatch
```

```
    double elapsedTime() time since creation (in seconds)
```

```
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

---

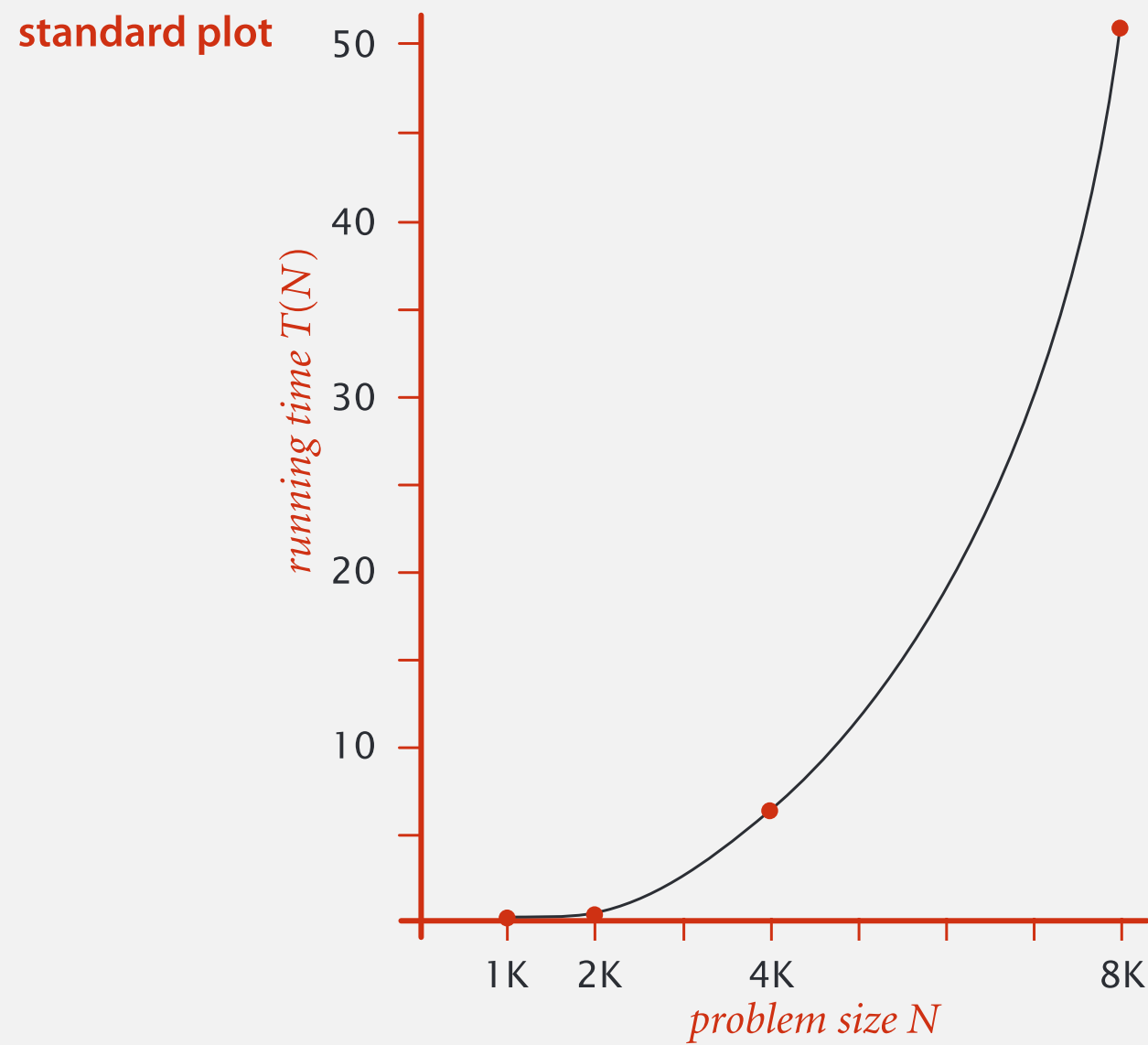
Run the program for various input sizes and measure running time.

| N      | time (seconds) † |
|--------|------------------|
| 250    | 0.0              |
| 500    | 0.0              |
| 1,000  | 0.1              |
| 2,000  | 0.8              |
| 4,000  | 6.4              |
| 8,000  | 51.1             |
| 16,000 | ?                |

# Data analysis

---

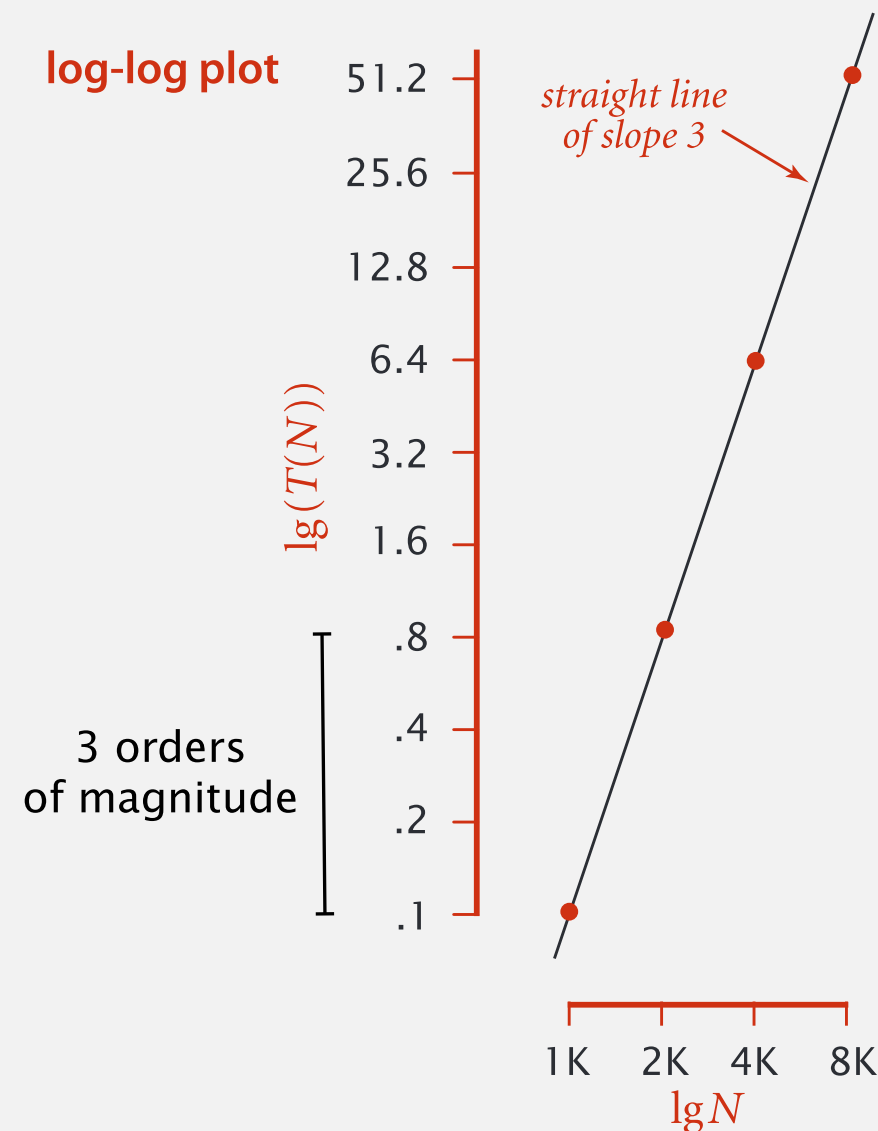
Standard plot. Plot running time  $T(N)$  vs. input size  $N$ .





# Data analysis

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



$$\lg(T(N)) = b \lg N + c$$

$$b = 2.999$$

$$c = -33.2103$$

$$T(N) = a N^b, \text{ where } a = 2^c$$

**Regression.** Fit straight line through data points:  $a N^b$ .

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

power law

slope

# 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   | 0.0              |       | –        |
| 500   | 0.0              | 4.8   | 2.3      |
| 1,000 | 0.1              | 6.9   | 2.8      |
| 2,000 | 0.8              | 7.7   | 2.9      |
| 4,000 | 6.4              | 8.0   | 3.0      |
| 8,000 | 51.1             | 8.0   | 3.0      |

$$\begin{aligned}\frac{T(2N)}{T(N)} &= \frac{a(2N)^b}{aN^b} \\ &= 2^b\end{aligned}$$

←  $\lg(6.4 / 0.8) = 3.0$

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

**Hypothesis.** Running time is about  $a N^b$  with  $b = \lg \text{ratio}$ .

**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             |

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

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

**Hypothesis.** Running time is about  $0.998 \times 10^{-10} \times N^3$  seconds.



almost identical hypothesis  
to one obtained via linear regression

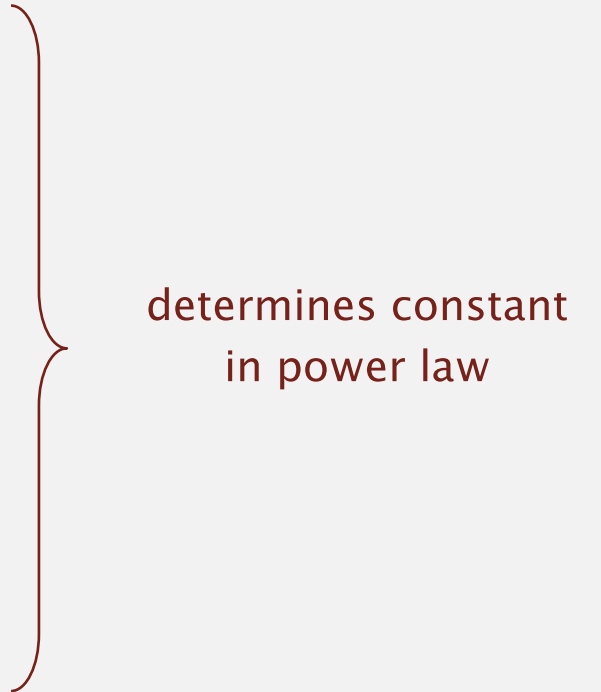
# Experimental algorithmics

---

## System independent effects.

- Algorithm.
  - Input data.
- 
- determines exponent  
in power law

## System dependent effects.

- Hardware: CPU, memory, cache, ...
  - Software: compiler, interpreter, garbage collector, ...
  - System: operating system, network, other apps, ...
- 
- determines constant  
in power law

**Bad news.** Difficult to get precise measurements.

**Good news.** Much easier and cheaper than other sciences.



e.g., can run huge number of experiments