Historical Quest for Speed

Multiplication: $a \times b$.
- Naïve: add $a$ to itself $b$ times. $N 2^N$ steps
- Grade school. $N^2$ steps
- Divide-and-conquer (Karatsuba, 1962). $N^{1.58}$ steps
- Ingenuity (Schönhage and Strassen, 1971).
  $N \log N \log \log N$ steps

Greatest common divisor: $\text{gcd}(a, b)$.
- Naïve: factor $a$ and $b$, then find $\text{gcd}(a, b)$. $2^N$ steps
- Euclid (20 BCE): $\text{gcd}(a, b) = \text{gcd}(b, a \mod b)$. $N$ steps

Linear Growth

- Grade school addition
  - Work is proportional to number of digits $N$
  - Linear growth: $kN$ for some constant $k$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>+</td>
<td>4</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

$N = 2$

- How many reads? How many writes? How many operations?

Sorting

- Given $n$ items, rearrange them so that they are in increasing order
- A key recurring problem
- Many different methods, how do we choose?
- Given a set of cards, describe how you would sort them:

  - Given a set of words, describe how you would sort them in alphabetical order?

Quadratic Growth

- Grade school multiplication
  - Work is proportional to square of number of digits $N$
  - Quadratic growth: $kN^2$ for some constant $k$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>

$N = 2$

- How many reads? How many writes? How many operations?
Better Machines vs. Better Algorithms

New machine.
- Costs $$$ or more.
- Makes “everything” finish sooner.
- Incremental quantitative improvements (Moore’s Law).
- May not help much with some problems.

New algorithm.
- Costs $ or less.
- Dramatic qualitative improvements possible! (million times faster)
- May make the difference, allowing specific problem to be solved.
- May not help much with some problems.

Why Does It Matter?

<table>
<thead>
<tr>
<th>Run time (nanoseconds)</th>
<th>$1.3 N^3$</th>
<th>$10 N^2$</th>
<th>$47 N \log_2 N$</th>
<th>$48 N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time to solve a problem of size</td>
<td>1000</td>
<td>1.3 seconds</td>
<td>10 msec</td>
<td>0.4 msec</td>
</tr>
<tr>
<td>10,000</td>
<td>22 minutes</td>
<td>1 second</td>
<td>6 msec</td>
<td>0.48 msec</td>
</tr>
<tr>
<td>100,000</td>
<td>15 days</td>
<td>1.7 minutes</td>
<td>78 msec</td>
<td>4.8 msec</td>
</tr>
<tr>
<td>million</td>
<td>41 years</td>
<td>2.8 hours</td>
<td>0.94 seconds</td>
<td>48 msec</td>
</tr>
<tr>
<td>10 million</td>
<td>41 millennia</td>
<td>1.7 weeks</td>
<td>11 seconds</td>
<td>0.48 seconds</td>
</tr>
</tbody>
</table>

Max size problem solved in one
- second: 920, 10,000, 1 million, 21 million
- minute: 3,600, 77,000, 49 million, 1.3 billion
- hour: 14,000, 600,000, 2.4 trillion, 76 trillion
- day: 41,000, 2.9 million, 50 trillion, 1.8 trillion

N multiplied by 10, time multiplied by
- 1,000, 100, 10+, 10

Impact of Better Algorithms

Example 1: N-body-simulation.
- Simulate gravitational interactions among N bodies.
  - physicists want $N = \#$ atoms in universe
- Brute force method: $N^2$ steps.

Example 2: Discrete Fourier Transform (DFT).
- Breaks down waveforms (sound) into periodic components.
  - foundation of signal processing
  - CD players, JPEG, analyzing astronomical data, etc.
- Grade school method: $N^2$ steps.
- Runge-Köning (1924), Cooley-Tukey (1965).
  - FFT algorithm: $N \log N$ steps, enables new technology.

Orders of Magnitude

<table>
<thead>
<tr>
<th>Seconds</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 second</td>
</tr>
<tr>
<td>10</td>
<td>10 seconds</td>
</tr>
<tr>
<td>$10^2$</td>
<td>1.7 minutes</td>
</tr>
<tr>
<td>$10^3$</td>
<td>17 minutes</td>
</tr>
<tr>
<td>$10^4$</td>
<td>2.8 hours</td>
</tr>
<tr>
<td>$10^5$</td>
<td>1.1 days</td>
</tr>
<tr>
<td>$10^6$</td>
<td>1.6 weeks</td>
</tr>
<tr>
<td>$10^7$</td>
<td>3.8 months</td>
</tr>
<tr>
<td>$10^8$</td>
<td>3.1 years</td>
</tr>
<tr>
<td>$10^9$</td>
<td>3.1 decades</td>
</tr>
<tr>
<td>$10^{10}$</td>
<td>3.1 centuries</td>
</tr>
<tr>
<td>...</td>
<td>forever</td>
</tr>
<tr>
<td>$10^{21}$</td>
<td>age of universe</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Meters Per Second</th>
<th>Imperial Units</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^{-10}$</td>
<td>1.2 in / decade</td>
<td>Continental drift</td>
</tr>
<tr>
<td>$10^{-9}$</td>
<td>1 ft / year</td>
<td>Hair growing</td>
</tr>
<tr>
<td>$10^{-8}$</td>
<td>3.4 in / day</td>
<td>Glacier</td>
</tr>
<tr>
<td>$10^{-7}$</td>
<td>1.2 ft / hour</td>
<td>Gastro-intestinal tract</td>
</tr>
<tr>
<td>$10^{-6}$</td>
<td>2 ft / minute</td>
<td>Ant</td>
</tr>
<tr>
<td>$10^{-5}$</td>
<td>220 mi / hour</td>
<td>Propeller airplane</td>
</tr>
<tr>
<td>$10^{-4}$</td>
<td>370 mi / min</td>
<td>Space shuttle</td>
</tr>
<tr>
<td>$10^{-3}$</td>
<td>620 mi / sec</td>
<td>Earth in galactic orbit</td>
</tr>
<tr>
<td>$10^{-2}$</td>
<td>62,000 mi / sec</td>
<td>1/3 speed of light</td>
</tr>
</tbody>
</table>

Powers of 2
- $2^{10}$ thousand
- $2^{20}$ million
- $2^{30}$ billion
Profiling Insertion Sort Analytically

How long does insertion sort take?
- Depends on number of elements N to sort.
- Depends on specific input.
- Depends on how long compare and exchange operation takes.

Best case.
- Elements in sorted order already.
  - $i^{th}$ iteration requires only 1 compare operation
  - total = $0 + 1 + 1 + . . . + 1 = N - 1$

```
A B C D E F G H I J
```

unsorted   active   sorted

Case Study: Sorting

Sorting problem:
- Given N items, rearrange them so that they are in increasing order.
- Among most fundamental problems.

Insertion sort
- Brute-force sorting solution.
- Move left-to-right through array.
- Exchange next element with larger elements to its left, one-by-one.

Profiling Insertion Sort Analytically

How long does insertion sort take?
- Depends on number of elements N to sort.
- Depends on specific input.
- Depends on how long compare and exchange operation takes.

Average case.
- Elements are randomly ordered.
  - $i^{th}$ iteration requires $i / 2$ comparison on average
  - total = $0 + 1/2 + 2/2 + . . . + (N-1)/2 = N (N-1) / 4$
  - check with profile: 249,750 vs. 256,313

```
B E F R T U O R C E
```

unsorted   active   sorted

Profiling Insertion Sort Analytically

How long does insertion sort take?
- Depends on number of elements N to sort.
- Depends on specific input.
- Depends on how long compare and exchange operation takes.

Worst case.
- Elements in reverse sorted order.
  - $i^{th}$ iteration requires $i - 1$ compare and exchange operations
  - total = $0 + 1 + 2 + . . . + N-1 = N (N-1) / 2$

```
E F G H I J D C B A
```

unsorted   active   sorted
Profiling Insertion Sort Analytically

How long does insertion sort take?
- Depends on number of elements $N$ to sort.
- Depends on specific input.
- Depends on how long compare and exchange operation takes.

Worst case: $N (N - 1) / 2$.

Best case: $N - 1$.

Average case: $N (N - 1) / 4$.

Easier alternative.

(i) Analyze asymptotic growth.
(ii) For medium $N$, run and measure time.
For large $N$, use (i) and (ii) to predict time.

Asymptotic growth rates.
- Ignore lower order terms and leading coefficients.
- Ex. $6N^3 + 17N^2 + 56$ is proportional to $N^3$.

Insertion sort is quadratic. On arizona: 1 second for $N = 10,000$.
- How long for $N = 100,000$? 100 seconds (100 times as long).
- $N = 1$ million? 2.78 hours (another factor of 100).
- $N = 1$ billion? 317 years (another factor of $10^6$).
**Profiling Mergesort Analytically**

How long does mergesort take?
- Bottleneck = merging (and copying).
  - merging two files of size $N/2$ requires $N$ comparisons
- $T(N) =$ comparisons to mergesort $N$ elements.

$$T(N) = \begin{cases} 0 & \text{if } N = 1 \\ \frac{N}{2} T\left(\frac{N}{2}\right) + N & \text{otherwise} \end{cases}$$

**Quicksort**

Quicksort.
- Partition array so that:
  - some partitioning element $a[m]$ is in its final position
  - no larger element to the left of $m$
  - no smaller element to the right of $m$
- Sort each "half" recursively.

Quicksort.
- Partition array so that:
  - some partitioning element $a[m]$ is in its final position
  - no larger element to the left of $m$
  - no smaller element to the right of $m$
- Sort each "half" recursively.
Profiling Mergesort Analytically

How long does mergesort take?
- Bottleneck = merging (and copying).
  - merging two files of size $N/2$ requires $N$ comparisons
  - $N \log_2 N$ comparisons to sort ANY array of $N$ elements.
  - even already sorted array!

How much space?
- Can’t do "in-place" like insertion sort.
- Need auxiliary array of size $N$.

Profiling Quicksort Analytically

Partition on median element.
- Proportional to $N \log_2 N$ in best and worst case.

Partition on rightmost element.
- Proportional to $N^2$ in worst case.
  - Already sorted file: takes $N^2/2 + N/2$ comparisons.

Partition on random element.
- Roughly $2 N \log_2 N$ steps.
- Choose random partition element.

Check profile.
- $2 N \log_2 N$: 13815 vs. 12372 (5708 + 6664).
- Running time for $N = 100,000$ about 1.2 seconds.
  - How long for $N = 1$ million?
    - slightly more than 10 times (about 12 seconds)

Sorting Analysis Summary

Running time estimates:
- Home pc executes $10^9$ comparisons/second.
- Supercomputer executes $10^{12}$ comparisons/second.

<table>
<thead>
<tr>
<th>computer</th>
<th>thousand</th>
<th>million</th>
<th>billion</th>
</tr>
</thead>
<tbody>
<tr>
<td>home</td>
<td>instant</td>
<td>2.8 hours</td>
<td>317 years</td>
</tr>
<tr>
<td>super</td>
<td>instant</td>
<td>1 second</td>
<td>1.6 weeks</td>
</tr>
</tbody>
</table>

Lessons: good algorithms are more powerful than supercomputers.

Design, Analysis, and Implementation of Algorithms

Algorithm.
- "Step-by-step recipe" used to solve a problem.
- Generally independent of programming language or machine on which it is to be executed.

Design.
- Find a method to solve the problem.

Analysis.
- Evaluate its effectiveness and predict theoretical performance.

Implementation.
- Write actual code and test your theory.