4.1 - 4.2 Analysis of Algorithms


Scientific method.
- Observe some feature of the universe.
- Hypothesize a model that is consistent with observation.
- Predict events using the hypothesis.
- Verify the predictions by making further observations.
- Validate the theory by repeating the previous steps until the hypothesis agrees with the observations.

Universe = computer itself.

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.

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.

Sorting.
- Rearrange N items in ascending order.
- Fundamental information processing abstraction.

Case Study: Sorting

Sorting problem. Given N items, rearrange them in ascending order.

Applications. Statistics, databases, data compression, computational biology, computer graphics, scientific computing, ...

Andrew Appel PU '81
Freidrich Gauss 1805
Jon von Neumann IAS 1945
Hanley Hauser
Haskell Hong
Hsu Hayes
Hayes Hong
Hanley Hsu
Hornet Hsu
Insertion Sort

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

Observe and tabulate running time for various values of $N$.
- Data source: $N$ random numbers between 0 and 1.
- Machine: Apple G5 1.8GHz with 1.5GB memory running OS X.
- Timing: Skagen wristwatch.

<table>
<thead>
<tr>
<th>$N$</th>
<th>Comparisons</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,000</td>
<td>6.2 million</td>
<td>0.13 seconds</td>
</tr>
<tr>
<td>10,000</td>
<td>25 million</td>
<td>0.43 seconds</td>
</tr>
<tr>
<td>20,000</td>
<td>99 million</td>
<td>1.5 seconds</td>
</tr>
<tr>
<td>40,000</td>
<td>400 million</td>
<td>5.6 seconds</td>
</tr>
<tr>
<td>80,000</td>
<td>1600 million</td>
<td>23 seconds</td>
</tr>
</tbody>
</table>

Helper Functions

Sorting helper functions.
- Is real number $x$ strictly less than $y$?

```java
public static boolean less(double x, double y) {
    return (x < y);
}
```

- Swap real numbers stored in $a[i]$ and $a[j]$.

```java
public static void exch(double[] a, int i, int j) {
    double swap = a[i];
    a[i] = a[j];
    a[j] = swap;
}
```

Insertion Sort: Experimental Hypothesis

Data analysis. Plot # comparisons vs. input size on log-log scale.

Regression. Fit line through data points $\sim a N^p$.

Hypothesis. # comparisons grows quadratically with input size $N^2/4$. 
**Insertion Sort: Theoretical Hypothesis**

**Experimental hypothesis.**  # comparisons $\sim N^2/4$.

**Prediction.**  400 million comparisons for $N = 40,000$.

<table>
<thead>
<tr>
<th>N</th>
<th>Comparisons</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>40,000</td>
<td>401.3 million</td>
<td>5.595 sec</td>
</tr>
<tr>
<td>40,000</td>
<td>399.7 million</td>
<td>5.573 sec</td>
</tr>
<tr>
<td>40,000</td>
<td>401.6 million</td>
<td>5.648 sec</td>
</tr>
<tr>
<td>40,000</td>
<td>400.0 million</td>
<td>5.632 sec</td>
</tr>
</tbody>
</table>

**Observations.**  401.3 million - 400.0 million $\sim 1/2$ million agreements.

**Prediction.**  10 billion comparisons for $N = 200,000$.

<table>
<thead>
<tr>
<th>N</th>
<th>Comparisons</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>200,000</td>
<td>9.997 billion</td>
<td>145 seconds</td>
</tr>
</tbody>
</table>

**Observation.**  Agrees.

**Numerical Examples.**
- $0+1+2+\ldots+N-2+N-1 = N(N-1)/2$
- $E F G H I J A B C D$

**Difference.**  Theoretical model can apply to machines not yet built.

**Insertion Sort: Experimental Hypothesis**

**Number of comparisons depends on input family.**
- **Ascending:** $N$.
- **Random:** $N^2/4$.
- **Descending:** $N^2/2$.

**Insertion Sort: Validation**

**Worst case.** (descending)
- Iteration $i$ requires $i$ comparisons.
- Total $= 0 + 1 + 2 + \ldots + N-2 + N-1 = N(N-1)/2$.

**Average case.** (random)
- Iteration $i$ requires $i/2$ comparisons on average.
- Total $= 0 + 1/2 + 2/2 + \ldots + (N-1)/2 = N(N-1)/4$. 

**Insertion Sort: Prediction and Verification**

**Experimental hypothesis.**  # comparisons $\sim N^2/4$.

**Prediction.**  400 million comparisons for $N = 40,000$.

**Observations.**

<table>
<thead>
<tr>
<th>N</th>
<th>Comparisons</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>40,000</td>
<td>401.3 million</td>
<td>5.595 sec</td>
</tr>
<tr>
<td>40,000</td>
<td>399.7 million</td>
<td>5.573 sec</td>
</tr>
<tr>
<td>40,000</td>
<td>401.6 million</td>
<td>5.648 sec</td>
</tr>
<tr>
<td>40,000</td>
<td>400.0 million</td>
<td>5.632 sec</td>
</tr>
</tbody>
</table>

**Prediction.**  401.3 million comparisons for $N = 40,000$.

**Observations.**  Agrees.
Insertion Sort: Theoretical Hypothesis

Partition array so that:
- some partitioning element \(a[i]\) is in its final position
- no larger element to the left of \(i\)
- no smaller element to the right of \(i\)

Quick sort.

- Partition array so that:
  - some partitioning element \(a[i]\) is in its final position
  - no larger element to the left of \(i\)
  - no smaller element to the right of \(i\)

- Sort each "half" recursively.

Partition array so that:
- some partitioning element \(a[i]\) is in its final position
- no larger element to the left of \(i\)
- no smaller element to the right of \(i\)

Sort each "half" recursively.
Quicksort: Java Implementation

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.

```java
public static void quicksort(double[] a, int left, int right) {
  if (right <= left) return;
  int i = partition(a, left, right);
  quicksort(a, left, i-1);
  quicksort(a, i+1, right);
}
```

Quicksort: Observation

Observe and tabulate running time for various values of \( N \).
- Data source: first \( N \) words of Charles Dicken’s life work.
- Machine: Apple G5 1.8GHz with 1.5GB memory running OS X.

<table>
<thead>
<tr>
<th>( N )</th>
<th>Comparisons</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>200,000</td>
<td>4.5 million</td>
<td>0.10 sec</td>
</tr>
<tr>
<td>400,000</td>
<td>9.5 million</td>
<td>0.23 sec</td>
</tr>
<tr>
<td>1 million</td>
<td>26 million</td>
<td>0.47 sec</td>
</tr>
<tr>
<td>2 million</td>
<td>55 million</td>
<td>0.96 sec</td>
</tr>
<tr>
<td>4 million</td>
<td>120 million</td>
<td>2.0 sec</td>
</tr>
<tr>
<td>8 million</td>
<td>240 million</td>
<td>4.2 sec</td>
</tr>
</tbody>
</table>

Remark. Takes 1.8 seconds to generate input of size 8 million!

Quicksort: Implementing Partition

Q. How to partition in-place efficiently?

```java
public static int partition(double[] a, int left, int right) {
  int i = left - 1;
  int j = right;
  while(true) {
    while (less(a[++i], a[right]))
      if (i == right) break;
    while (less(a[right], a[--j]))
      if (j == left) break;
    if (i >= j) break;
    exch(a, i, j);
    exch(a, i, right);
    return i;
  }
}
```

QuickSort: Preliminary Hypothesis

Experimental hypothesis. Number of comparisons \( \approx 30N \).
Quicksort: Prediction and Verification

**Experimental hypothesis.** Number of comparisons $\approx 30N$.

**Prediction.** 120 million comparisons for $N = 4$ million.

<table>
<thead>
<tr>
<th>N</th>
<th>Comparisons</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 million</td>
<td>112.9 million</td>
<td>2.04 sec</td>
</tr>
<tr>
<td>4 million</td>
<td>116.7 million</td>
<td>2.07 sec</td>
</tr>
<tr>
<td>4 million</td>
<td>116.8 million</td>
<td>2.02 sec</td>
</tr>
</tbody>
</table>

**Observations.** Agrees.

**Prediction.** 600 million comparisons for $N = 20$ million.

<table>
<thead>
<tr>
<th>N</th>
<th>Comparisons</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 million</td>
<td>638 million</td>
<td>11.1 sec</td>
</tr>
<tr>
<td>100 million</td>
<td>3.6 billion</td>
<td>60.6 sec</td>
</tr>
</tbody>
</table>

**Observations.** Not quite.

Quicksort: Hypothesis

**Quickso**rt. To sort $N$ elements, need to do $N$ comparisons, and then quicksort two files of (roughly) size $N/2$.

**Lesson.** Great algorithms can be more powerful than supercomputers.

<table>
<thead>
<tr>
<th>Computer</th>
<th>Comparisons Per Second</th>
<th>Insertion</th>
<th>Quicksort</th>
</tr>
</thead>
<tbody>
<tr>
<td>laptop</td>
<td>$10^7$</td>
<td>3 centuries</td>
<td>3 hours</td>
</tr>
<tr>
<td>super</td>
<td>$10^{12}$</td>
<td>2 weeks</td>
<td>instant</td>
</tr>
</tbody>
</table>

$N = 1$ billion

Quicksort: Theoretical Hypothesis

**Average case.** [random]
- Number of comparisons $\approx 2N \ln N$ (stddev $\approx 0.65N$).
- Number of exchanges $\approx 1/3 N \ln N$.

**Worst case.** Number of comparisons $\approx 1/2 N^2$.

**Validation.**
- Partition on random element to eliminate worst case.
- Theory now agrees with observations.
Summary

How can I evaluate the performance of my algorithm?
- Computational experiments.
- Theoretical analysis.

What if it’s not fast enough?
- Understand why.
- Buy a faster computer.
- Find a better algorithm in a textbook.
- Discover a new algorithm.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Better Machine</th>
<th>Better Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>$$$ or more</td>
<td>$ or less</td>
</tr>
<tr>
<td>Applicability</td>
<td>Makes &quot;everything&quot; run faster.</td>
<td>Does not apply to some problems.</td>
</tr>
<tr>
<td>Improvement</td>
<td>Quantitative improvements.</td>
<td>Dramatic qualitative improvements possible.</td>
</tr>
</tbody>
</table>

Scientific Method

Scientific method applies to estimate running time.
- Experimental analysis: not difficult to perform experiments.
- Theoretical analysis: may require advanced mathematics.
- Small subset of mathematical functions suffice to describe running time of many fundamental algorithms.

```
for (int i = 0; i < N; i++)
...
2^N
N^3
```

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

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

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

Order of Growth

Asymptotic running time.
- Estimate time as a function of input size N.
- Ignore lower order terms and leading coefficients.
  - when N is large, terms are negligible
  - when N is small, we don’t care
- Ex: 6N^3 + 17N^2 + 56 is asymptotically proportional to N^3.

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Description</th>
<th>When N doubles, running time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Constant algorithm is independent of input size.</td>
<td>does not change</td>
</tr>
<tr>
<td>log N</td>
<td>Logarithmic algorithm gets slightly slower as N grows.</td>
<td>increases by a constant</td>
</tr>
<tr>
<td>N</td>
<td>Linear algorithm is optimal for processing N inputs.</td>
<td>doubles</td>
</tr>
<tr>
<td>N log N</td>
<td>Linearithmic algorithm scales to huge N.</td>
<td>slightly more than doubles</td>
</tr>
<tr>
<td>N^2</td>
<td>Quadratic algorithm is impractical for large N.</td>
<td>quadruples</td>
</tr>
<tr>
<td>2^N</td>
<td>Exponential algorithm is not usually practical.</td>
<td>squares</td>
</tr>
</tbody>
</table>