Analysis of Algorithms

Overview

**Analysis of algorithms**: framework for comparing algorithms and predicting performance.

**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.**

---

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) 
Analytic Engine (schematic)

---

Case Study: Sorting

**Sorting problem:**
- Given N items, rearrange them in ascending order.
- Applications: statistics, databases, data compression, computational biology, computer graphics, scientific computing, ...

<table>
<thead>
<tr>
<th>name</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hauser</td>
<td>Hanley</td>
</tr>
<tr>
<td>Hang</td>
<td>Haskell</td>
</tr>
<tr>
<td>Hsu</td>
<td>Hauser</td>
</tr>
<tr>
<td>Hayes</td>
<td>Hayes</td>
</tr>
<tr>
<td>Haskell</td>
<td>Hong</td>
</tr>
<tr>
<td>Hanley</td>
<td>Hornet</td>
</tr>
<tr>
<td>Hornet</td>
<td>Hsu</td>
</tr>
</tbody>
</table>
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.

```java
public static void insertionSort(double[] a) {
    int N = a.length;
    for (int i = 0; i < N; i++) {
        for (int j = i; j > 0; j--) {
            if (less(a[j], a[j-1]))
                exch(a, j, j-1);
            else break;
        }
    }
}
```

Insertion Sort: Observation

Observe and tabulate running time for various values of $N$.
- Data source: $N$ random numbers between 0 and 1.

<table>
<thead>
<tr>
<th>$N$</th>
<th>Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,000</td>
<td>6.2 million</td>
</tr>
<tr>
<td>10,000</td>
<td>25 million</td>
</tr>
<tr>
<td>20,000</td>
<td>99 million</td>
</tr>
<tr>
<td>40,000</td>
<td>400 million</td>
</tr>
<tr>
<td>80,000</td>
<td>16 million</td>
</tr>
</tbody>
</table>

Insertion Sort: Experimental Hypothesis

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

Regression. Fit line through data points $\sim aN^p$.
Hypothesis. # comparisons grows quadratically with input size $= N^2/4$.

Observations.

<table>
<thead>
<tr>
<th>$N$</th>
<th>Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>40,000</td>
<td>401.3 million</td>
</tr>
<tr>
<td>40,000</td>
<td>399.7 million</td>
</tr>
<tr>
<td>40,000</td>
<td>401.6 million</td>
</tr>
<tr>
<td>40,000</td>
<td>400.0 million</td>
</tr>
</tbody>
</table>

Agrees.

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

Observation.

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

Agrees.
Insertion Sort: Theoretical Hypothesis

**Experimental hypothesis.**
- Measure running times, plot, and fit curve.
- Model useful for predicting, but not for explaining.

**Theoretical hypothesis.**
- Analyze algorithm to estimate # comparisons as a function of:
  - number of elements N to sort
  - average or worst case input
- Model useful for predicting and explaining.
- Model is independent of a particular machine or compiler.

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

**Insertion Sort: Observation**

**Analysis**

<table>
<thead>
<tr>
<th>Analysis</th>
<th>Input</th>
<th>Comparisons</th>
<th>Stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst</td>
<td>Descending</td>
<td>(N^2/2)</td>
<td>-</td>
</tr>
<tr>
<td>Average</td>
<td>Random</td>
<td>(N^2/4)</td>
<td>(1/6 N^{1/2})</td>
</tr>
<tr>
<td>Best</td>
<td>Ascending</td>
<td>N</td>
<td>-</td>
</tr>
</tbody>
</table>

**Validation.** Theory agrees with observations.

**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: \(0 + 1/2 + 2/2 + \ldots + (N-1)/2 = N(N-1)/4\).

**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.

<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>16 million</td>
<td>23 seconds</td>
</tr>
<tr>
<td>200,000</td>
<td>10 billion</td>
<td>145 seconds</td>
</tr>
</tbody>
</table>
Insertion Sort: Experimental Hypothesis

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

Regression. Fit line through data points $\sim$ a $N^2$.

Hypothesis. Running time grows quadratically with input size.

Measuring Running Time

Factors that affect running time.
- Machine.
- Compiler.
- Algorithm.
- Input data.

More factors.
- Caching.
- Garbage collection.
- Just-in-time compilation.
- CPU used by other processes.

Bottom line. Often hard to get precise measurements.

Timing in Java

Wall clock. Measure time between beginning and end of computation.
- Automatic: Stopwatch.java library.

```java
Stopwatch.tic();
...
double elapsed = Stopwatch.toc();
```

```java
public class Stopwatch {
    private static long start;
    public static void tic() {
        start = System.currentTimeMillis();
    }
    public static double toc() {
        long stop = System.currentTimeMillis();
        return (stop - start) / 1000.0;
    }
}
```

Summary

Analysis of algorithms: framework for comparing algorithms and predicting performance.

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.

Remaining question. How to formulate a hypothesis?
How To Formulate a Hypothesis

Types of Hypotheses

Worst case running time. Obtain bound on largest possible running time of algorithm on input of a given size N.

- Generally captures efficiency in practice.
- Draconian view, but hard to find effective alternative.

Average case running time. Obtain bound on running time of algorithm on random input as a function of input size N.

- Hard to accurately model real instances by random distributions.
- Algorithm tuned for a certain distribution may perform poorly on other inputs.

Amortized running time. Obtain bound on running time of sequence of N operations as a function of the number of operations.

Estimating the Running Time

Total running time: sum of cost \times frequency for all of the basic ops.
- Cost depends on machine, compiler.
- Frequency depends on algorithm, input.

Cost for sorting.
- \( A = \# \) exchanges.
- \( B = \# \) comparisons.
- Cost on a typical machine = \( 11A + 4B \).

Frequency of sorting ops.
- \( N = \# \) elements to sort.
- Selection sort: \( A = N-1, B = N(N-1)/2 \).

Donald Knuth
1974 Turing Award

Asymptotic Running Time

An easier alternative.

(i) Analyze asymptotic growth as a function of input size N.
(ii) For medium N, run and measure time.
(iii) For large N, use (i) and (ii) to predict time.

Asymptotic growth rates.
- Estimate as a function of input size N.
  - \( N, N \log N, N^2, N^3, 2^N, N! \)
- Ignore lower order terms and leading coefficients.
  - Ex. \( 6N^3 + 17N^2 + 56 \) is asymptotically proportional to \( N^3 \).
Big Theta, Oh, and Omega notation.

- $\Theta(N^2)$ means \{ $N^2$, $17N^2$, $N^2 + 17N^{1.5}$, $3N$, \ldots \} 
  - ignore lower order terms and leading coefficients
- $O(N^2)$ means \{ $N^2$, $17N^2$, $N^2 + 17N^{1.5}$, $3N$, $N^{1.5}$, $100N$, \ldots \} 
  - $\Theta(N^2)$ and smaller
  - use for upper bounds
- $\Omega(N^2)$ means \{ $N^2$, $17N^2$, $N^2 + 17N^{1.5}$, $3N$, $N^3$, $100N^5$, \ldots \} 
  - $\Theta(N^2)$ and larger
  - use for lower bounds

Never say: insertion sort makes at least $O(N^2)$ comparisons.

---

### Orders of Magnitude

<table>
<thead>
<tr>
<th>Seconds</th>
<th>Equivalent</th>
<th>Meters Per Second</th>
<th>Imperial Units</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 second</td>
<td>$10^{10}$</td>
<td>1.2 in / decade</td>
<td>Continental drift</td>
</tr>
<tr>
<td>10</td>
<td>10 seconds</td>
<td>$10^8$</td>
<td>1 ft / year</td>
<td>Hair growing</td>
</tr>
<tr>
<td>$10^2$</td>
<td>1.7 minutes</td>
<td>$10^6$</td>
<td>3.4 in / day</td>
<td>Glacier</td>
</tr>
<tr>
<td>$10^3$</td>
<td>17 minutes</td>
<td>$10^4$</td>
<td>1.2 ft / hour</td>
<td>Gastro-intestinal tract</td>
</tr>
<tr>
<td>$10^4$</td>
<td>2.8 hours</td>
<td>$10^2$</td>
<td>2 ft / minute</td>
<td>Ant</td>
</tr>
<tr>
<td>$10^5$</td>
<td>11 days</td>
<td>$10^1$</td>
<td>2.2 mi / hour</td>
<td>Human walk</td>
</tr>
<tr>
<td>$10^6$</td>
<td>1.6 weeks</td>
<td>$10^0$</td>
<td>220 mi / hour</td>
<td>Propeller airplane</td>
</tr>
<tr>
<td>$10^7$</td>
<td>3.8 months</td>
<td></td>
<td>370 mi / min</td>
<td>Space shuttle</td>
</tr>
<tr>
<td>$10^8$</td>
<td>3.1 years</td>
<td></td>
<td>620 mi / sec</td>
<td>Earth in galactic orbit</td>
</tr>
<tr>
<td>$10^9$</td>
<td>3.1 decades</td>
<td></td>
<td>62,000 mi / sec</td>
<td>1/3 speed of light</td>
</tr>
<tr>
<td>$10^{10}$</td>
<td>3.1 centuries</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>forever</td>
<td>Powers of 2</td>
<td>$2^{10}$ thousand</td>
<td></td>
</tr>
<tr>
<td>$10^{17}$</td>
<td>age of universe</td>
<td></td>
<td>$2^{20}$ million</td>
<td></td>
</tr>
</tbody>
</table>

---

### Why It Matters

<table>
<thead>
<tr>
<th>Run time in nanoseconds</th>
<th>$1.3 N^3$</th>
<th>$10 N^2$</th>
<th>$47 N \log N$</th>
<th>$48 N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>1.3 seconds</td>
<td>10 msec</td>
<td>0.4 msec</td>
<td>0.048 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>

<table>
<thead>
<tr>
<th>Time to solve a problem of size</th>
<th>second</th>
<th>920</th>
<th>10,000</th>
<th>1 million</th>
<th>21 million</th>
</tr>
</thead>
<tbody>
<tr>
<td>minute</td>
<td>3,600</td>
<td>77,000</td>
<td>49 million</td>
<td>1.3 billion</td>
<td></td>
</tr>
<tr>
<td>hour</td>
<td>14,000</td>
<td>600,000</td>
<td>2.4 trillion</td>
<td>76 trillion</td>
<td></td>
</tr>
<tr>
<td>day</td>
<td>41,000</td>
<td>2.9 million</td>
<td>50 trillion</td>
<td>1,800 trillion</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Max size problem solved in one</th>
<th>second</th>
<th>1,000</th>
<th>100</th>
<th>10+</th>
<th>10</th>
</tr>
</thead>
</table>

Reference: More Programming Pearls by Jon Bentley

---

### Constant Time

**Linear time.** Running time is $O(1)$.

**Elementary operations.**

- Function call.
- Boolean operation.
- Arithmetic operation.
- Assignment statement.
- Access array element by index.
Logarithmic Time

**Logarithmic time.** Running time is $O(\log N)$.

**Searching in a sorted list.** Given a sorted array of items, find index of query item.

**O(\log N) solution.** Binary search.

```java
public static int binarySearch(String[] a, String key) {
    int left = 0;
    int right = a.length - 1;
    while (left <= right) {
        int mid = (left + right) / 2;
        int cmp = key.compareTo(a[mid]);
        if (cmp < 0) right = mid - 1;
        else if (cmp > 0) left = mid + 1;
        else return mid;
    }
    return -1;
}
```

Linear Time

**Linear time.** Running time is $O(N)$.

**Find the maximum.** Find the maximum value of $N$ items in an array.

```java
double max = Integer.NEGATIVE_INFINITY;
for (int i = 0; i < N; i++) {
    if (a[i] > max) {
        max = a[i];
    }
}
```

Logarithmic Time

**Logarithmic time.** Running time is $O(\log N)$.

**Searching in a sorted list.** Given a sorted array of items, find index of query item.

**O(\log N) solution.** Binary search.

**Remark.** $O(\log N)$ comparisons required. [stay tuned]

Quadratic Time

**Quadratic time.** Running time is $O(N^2)$.

**Closest pair of points.** Given a list of $N$ points in the plane, find the pair that is closest.

**O($N^2$) solution.** Enumerate all pairs of points.

```java
double min = Math.POSITIVE_INFINITY;
for (int i = 0; i < N; i++) {
    for (int j = i+1; j < N; j++) {
        double dx = (x[i] - x[j]);
        double dy = (y[i] - y[j]);
        if (dx*dx + dy*dy < min) {
            min = dx*dx + dy*dy;
        }
    }
}
```

**Remark.** $O(N^2)$ seems inevitable, but this is just an illusion.

Linear Time

**Linear time.** Running time is $O(N \log N)$.

**Sorting.** Given an array of $N$ elements, rearrange in ascending order.

**O(N \log N) solution.** Mergesort. [stay tuned]

**Remark.** $\Omega(\log N)$ comparisons required. [stay tuned]
**Exponential Time**

Exponential time. Running time is $O(a^N)$ for some constant $a > 1$.

**Fibonacci sequence**: 1 1 2 3 5 8 13 21 34 55 ...

$O(\phi^N)$ solution. Spectacularly inefficient! $\phi = \frac{1}{2}(1 + \sqrt{5}) = 1.618034...$

Efficient solution. $F(N) = \lfloor \phi^N \rfloor$ nearest integer function

```
public static int F(int N) {
    if (n == 0 || n == 1)
        return n;
    else
        return F(n-1) + F(n-2);
}
```

**Summary of Common Hypotheses**

<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 if you need to process N inputs.</td>
<td>doubles</td>
</tr>
<tr>
<td>N log N</td>
<td>Linearithmic algorithm scales to huge problems.</td>
<td>slightly more than doubles</td>
</tr>
<tr>
<td>N²</td>
<td>Quadratic algorithm practical for use only on relatively small problems.</td>
<td>quadruples</td>
</tr>
<tr>
<td>$2^N$</td>
<td>Exponential algorithm is not usually practical.</td>
<td>squares!</td>
</tr>
</tbody>
</table>