Little Experiments on Algorithms

Jon Bentley
Avaya Labs Research
Outline

Introduction
  A Classic Tiny Experiment
  Themes
A Collection of Little Experiments
Conclusions
Jumbo Engineering: The Challenger

A Big Project

A Big Problem

A Big Clue

O-rings
Their job: Expand so that no flames escape
Could they have failed?
A Tiny Experiment

Facts

The O-rings worked successfully in all previous launches
January 28, 1986, was a cold Florida day: 29°F
All earlier launches were on warmer days (≥53°F)

Hypothesis

O-ring expansion at 29°F
is substantially slower
than at higher temps

Testing the Hypothesis

The O-ring material, a C-clamp, a glass of ice water, and one Nobel laureate
Themes of this Workshop

1. Extrapolation to Asymptopia
2. Fair Comparisons
3. Drawing Robust Conclusions
4. Data Overload
   No problem is so big that it can’t be run away from.
5. Components of Running Time
6. Comparing Convergence Histories
7. Optimizing Parameter Settings
   A little bit.
8. Sampling Streams
9. Run Time Uncertainty
0. Experiment Design
   My own axe to grind

2. Fair Comparison (to other algs)

“Our Pascal implementation of the cell nearest neighbor algorithm required 11 lines to place the points in cells and 34 lines to search. The optimal number of points per cell was 3; densities ranging from 1 to 9, however, decreased the running time by only 10 percent.

1. (Casual) Extrapolation to Asymptopia

The average running time for nearest neighbor searching was the constant 2765 microseconds per search, on a PDP-KL10. This compares with $52n$ microseconds required by the linear search; the break-even point is at $n = 53$. To find all nearest neighbors in a 1000-point planar set, the linear-expected-time cell method required less than 2.8 seconds while the quadratic algorithm required 52 seconds.

3. Conclusions Robust across non-uniform inputs?
Cells, Cont.

3. Robust Conclusions
“We ran the program on two data sets representing the population centroids of political areas. The first set represented all census tracts in San Diego County and was fairly uniform over about half of a square; the second set represented all precincts in the (roughly square) State of New Mexico and was very clustered in the few large cities in the state.”

<table>
<thead>
<tr>
<th></th>
<th>San Diego</th>
<th>New Mexico</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of points</td>
<td>318</td>
<td>1122</td>
</tr>
<tr>
<td>Quadratic algorithm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Predicted secs</td>
<td>5.25</td>
<td>65.5</td>
</tr>
<tr>
<td>Observed secs</td>
<td>5.35</td>
<td>66.5</td>
</tr>
<tr>
<td>Cell algorithm</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Predicted secs</td>
<td>0.88</td>
<td>3.11</td>
</tr>
<tr>
<td>Observed secs</td>
<td>1.40</td>
<td>7.56</td>
</tr>
<tr>
<td>Optimum cell density</td>
<td>1.7</td>
<td>3.0</td>
</tr>
</tbody>
</table>

7. Parameter Settings (no longer smooth)
A Simpler Problem Today

Which sort algorithm is faster: Quicksort or Heap sort?

Both are known to be \( \Theta(N \lg N) \) for shuffled inputs

Assume both have run times of \( c N \lg N \)

Run each at one value to determine each’s constant

Assume \( c_1N \lg N + c_2N \)

Run each at two values to determine both constants

Alternative: Plot a graph

What kind?

<table>
<thead>
<tr>
<th>Run time</th>
<th>Run time/N</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N (log scale)</td>
</tr>
</tbody>
</table>
CPU Times for Sorting

What happens at n=1?

Some possible lines
Sorting on Other Machines

Sorting – K6 400

Sorting – R10000 250

nanosecs/n

n

0

100

1000

10000

1E+06

1E+07

0

100

1000

10000

1E+06

1E+07

qsort

hsort
Workshop Themes

1. Extrapolation to Asymptopia
   Not there yet –
   disk is right around the corner
2. Fair Comparisons
3. Drawing Robust Conclusions
5. Components of Running Time
9. Run Time Uncertainty
   Why the hiccup near 10,000,000?
0. Experiment Design
   Is there a better way?
A Cost Model for Memory

Goal: A little experiment to estimate memory costs

Remove sorting to get to the essence of caching

The critical loop (\(n\) is array size, \(d\) is delta)

```c
for (i = 0; i < count; i++) {
    sum += x[j];
    j += d;
    if (j >= n)
        j -= n;
}
```

The complete MemEx program is ~30 lines of C
Results of the Model

MemEx -- PII 400

- d = 1
- d = 11
Other Machines

An insight that can explain *and* predict.
Simon’s Parable of the Ant

“We watch an ant make his laborious way across a wind- and wave molded beach. He moves ahead, angles to the right to ease his climb up a steep dunelet, detours around a pebble, …. I sketch his path on a piece of paper….

“I show the unlabeled sketch to a friend. Whose path is it? An expert skier, perhaps, slaloming down a steep and somewhat rocky slope. Or a sloop, beating upwind in a channel dotted with islands or shoals….

“An ant, viewed as a behaving system, is quite simple. The apparent complexity of its behavior over time is largely a reflection of the complexity of the environment in which it finds itself.”

MemEx is a simple program in a complex environment
Statistical Questions

How can I \{explain, formalize, reason about\} experiments across memory domains?

How can I reason about large experiments built on this infrastructure?

Tukey, *EDA*, 6C: “We can make many good uses of a close fit, whether or not it is ‘a basic law’.”

How can I talk about the components of a computation?

How can I design experiments to get the most insight bang for the least computational buck?

This tiny experiment did well; how should I design an experiment on cell-based NN searching?

How can I design algorithms for this environment?

How ought I deal with hiccups?
Statements about Sorting?

In memory systems with equal costs for random and sequential access, this Heap sort is about 30%-50% slower than this Quicksort.

In memory systems with random access a factor of $K$ more expensive than sequential access, the dominant term of this Heap sort is about a factor of $1.3K$ greater than the dominant term of this Quicksort.
A Tale of Two Sorts

Heap sort is $O(n \log n)$ in the worst case
   Pretty fast on the average
Quicksort is expected $O(n \log n)$
   Faster yet on the average (30% or more)
   Frequently used to implement the C `qsort`
   $\Theta(n^2)$ in the worst case
A Great Bug Report

We [Wilks & Becker] found that `qsort` is unbearably slow on “organ-pipe” inputs like “0123443210”:

```c
main(int argc, char **argv)
{
    int n=atoi(argv[1]), i, x[100000];
    for (i = 0; i < n; i++)
        x[i] = i;
    for ( ; i < 2*n ; i++)
        x[i] = 2*n-i-1;
    qsort(x, 2*n, sizeof(int), intcmp);
}
```

(Continued …)
Here are the timings on a Pentium:

```
$ time a.out 2000
real 5.85s
$ time a.out 4000
real 21.65s
$ time a.out 8000
real 85.11s
$
```

This is clearly quadratic behavior – each time we double the input size, the run time goes up by a factor of four.

A simple experiment to reveal that a sort that should be \(O(n \log n)\) is in fact quadratic

Distilled from a huge program (hundreds of thousands of lines of code)
Observation

The run time of the little program is quadratic

Explanations?

A flaw in qsort itself

Expensive underlying structures

Memory management: caching?

Memory allocation: malloc?
A Production Malloc

A malloc driver

```c
void main(int argc, char *argv[])
{
    int n = atoi(argv[1]), m = atoi(argv[2]);
    while (n-- > 0)
        malloc(m);
}
```

Some runs for 16-byte nodes

```
$ time a.out 50000 16
1.8u
$ time a.out 100000 16
8.5u
$ time a.out 200000 16
38.3u
$
```

MclIIroy’s explanation
How Firm a Foundation?

Bumps so far

Memory hierarchy
  Caching, paging
Memory management
  Storage allocation

Additional traps lurking

  Software: Compiler optimizations
  Hardware: Deep pipelines
  Multiprocessing, Networks, Parallel computation, …

Simon’s Parable of the Ant suggests that simple experiments may provide a powerful way to explore complex environments
Return to Qsort

Becker and Wilks observed quadratic CPU time

Many potential sources

Critical operations

Comparisons and swaps

Add to the code

```c
int c = 0;

int intcmp (int *i, int *j)
{
    c++; return *i - *j;
}

printf("comps: %d\n", c);
```
A Hypothesis About Qsort

A sequence of runs

$ a.out 1000  
comps: 1000000$

$ a.out 2000  
comps: 4000000$

$ a.out 4000  
comps: 16000000$

Next Steps

Why \( n^2 \) time to sort \( 2n \) elements?
A better \texttt{qsort}
String Reversal in Awk

How do algorithms on English words perform right-to-left, rather than left-to-right?

1.) Rewrite a suite of programs
2.) Reverse the words in the English input files

A “Production” Program

```awk
function rev1(s, len, i, t) {
    len = length(s)
    t = ""
    for (i = 1; i <= len; i++)
        t = substr(s, i, 1) t
    return t
}

{   print rev1($1)
}
```

A “Theoretical” Question

How much time to reverse a string?
Is there a better way?
Faster String Reversal?

A Divide-and-Conquer Algorithm

```javascript
function rev2(s, len, m) {
    len = length(s)
    if (len <= 1)
        return s
    m = int(len/2)
    return rev2(substr(s, m+1)) \n        rev2(substr(s, 1, m))
}
```

Questions

Correct?
How fast compared to \texttt{rev1}?
Summary of Reversal

Data

<table>
<thead>
<tr>
<th>Alg</th>
<th>N</th>
<th>Secs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>64,000</td>
<td>2.53</td>
</tr>
<tr>
<td>1</td>
<td>128,000</td>
<td>11.38</td>
</tr>
<tr>
<td>2</td>
<td>256,000</td>
<td>3.51</td>
</tr>
<tr>
<td>2</td>
<td>512,000</td>
<td>6.99</td>
</tr>
</tbody>
</table>

Hypotheses

Alg 1 is quadratic
Alg 2 is $O(N \log N)$

Comparing Algs 1 and 2  A tiny “horse race”
Alg 2 is faster (for values of $N$ in this neighborhood)
Alg 1 takes 7 lines of code; Alg 2 takes 8 lines
awk 'function rev1(s, len, i, t) {
    len = length(s)
    t = ""
    for (i = 1; i <= len; i++)
        t = substr(s, i, 1) t
    return t
}

function rev2(s, len, m) {
    len = length(s)
    if (len <= 1)
        return s
    m = int(len/2)
    return rev2(substr(s, m+1)) \ 
       rev2(substr(s, 1, m))
}

{ alg = $1
  n = $2
  s = "a"
  while (length(s) < n)
      s = s s
  if (alg == 1)
      s = rev1(s)
  else
      s = rev2(s)
  print alg, length(s)
} ' $*

time reverse <<End
1 64000
End
time reverse <<End
1 128000
End
time reverse <<End
2 256000
End
time reverse <<End
2 512000
End

1. Extrapolation
2. Comparisons
3. Robust Conclusions
5. Components
   (string costs)
Compressing American Names

General Idea: Store unique large items in a master table, and then represent them elsewhere by small indices into that table

Application to Last Names

Idea

Represent Smith by 1, Johnson by 2, etc.

Performance

Are names nonuniform?

How often do frequent names appear?

How to phrase a precise question?
## Data on Last Names

From [www.census.gov/genealogy/names](http://www.census.gov/genealogy/names)

<table>
<thead>
<tr>
<th>Last Name</th>
<th>Frequency</th>
<th>Open Electone</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMITH</td>
<td>1.006</td>
<td>1.006</td>
<td>1</td>
</tr>
<tr>
<td>JOHNSON</td>
<td>0.810</td>
<td>1.816</td>
<td>2</td>
</tr>
<tr>
<td>WILLIAMS</td>
<td>0.699</td>
<td>2.515</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>HOLLAND</td>
<td>0.042</td>
<td>27.786</td>
<td>256</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>BOBECK</td>
<td>0.000</td>
<td>88.093</td>
<td>65536</td>
</tr>
</tbody>
</table>

- Top name accounts for 1% of the 1990 US population
- Top 256 names account for 28% of the population
- Top 65,536 names account for 88% of the population

1 byte encodes 28%; 2 bytes encode 88%

This data is not dispositive, but may be typical
### Data on Male First Names

From [www.census.gov/genealogy/names](http://www.census.gov/genealogy/names)

<table>
<thead>
<tr>
<th>Name</th>
<th>Frequency</th>
<th>Total</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAMES</td>
<td>3.318</td>
<td>3.318</td>
<td>1</td>
</tr>
<tr>
<td>JOHN</td>
<td>3.271</td>
<td>6.589</td>
<td>2</td>
</tr>
<tr>
<td>ROBERT</td>
<td>3.143</td>
<td>9.732</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>GEORGE</td>
<td>0.927</td>
<td>28.939</td>
<td>16</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>DWAYNE</td>
<td>0.059</td>
<td>76.480</td>
<td>256</td>
</tr>
</tbody>
</table>

- Top name accounts for 3.3% of the population
- Top 16 names account for 29% of the population
- Top 256 names account for 76% of the population

1 nybble encodes 29%; 1 byte encodes 76%
Where Does The Time Go?

A Built-In Profiler

Detailed counts on the time spent in each function give the complete distribution

Dunlavey’s Call-Stack Sampling

Run the program under a debugger, halt it with a “pause” button, and examine the call stack. Make a record of the call stacks observed.

Any statement that appears on more than one call stack might be a time hog.

Invoking a statement less frequently (or eliminating it) reduces execution time by the fraction of time it resided on the call stack.

[Details in SIGPLAN Notices and Wikipedia]
Ancient History

*Programming Pearls*, “The Back of the Envelope”

“Bob Martin read from a proposal for the system that his team was building for the Olympic games, and went through a similar sequence of calculations.

He estimated one key parameter as we spoke by measuring the time required to send himself a one-character piece of mail.

His calculations showed that, under generous assumptions, the proposed system could work only if there were at least a hundred and twenty seconds in each minute.”
A Huge System

How long to send e-mail to 100,000 recipients?

Time to send to $N$ recipients:

<table>
<thead>
<tr>
<th>$N$</th>
<th>Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>11.4</td>
</tr>
<tr>
<td>2000</td>
<td>39.6</td>
</tr>
<tr>
<td>4000</td>
<td>167.1</td>
</tr>
</tbody>
</table>

Tukey, *EDA*, Section 6B

“Three points can take us a long way. If they are well chosen, they can do very well for us.”
"If I ever believed in the myth of the ‘exact sciences’ or ‘hard sciences’, my belief was wholly dissipated by encounters with such topics as air quality, eutrophication of lake waters, global warming, dietary standards, effects of low-level radiation, meteorology (for example, cloud seeding), and cold fusion. All of these topics contain uncertainties about the facts and their implications at least as serious as those we are accustomed to in the social sciences.

The true line is not between ‘hard’ natural science and ‘soft’ social sciences, but between

precise science limited to highly abstract and simple phenomena in the laboratory and 1968: MIX 1009

inexact science and technology dealing with complex phenomena in the real world.” 2008: microprocessors

H. A. Simon, *Models of My Life*, p. 304
The Meaning of “Little”

Two Definitions

A few slides to describe
Less than an hour to conduct

Counting Time

Conducting the final experiment

From scratch: Compressing names, string reversal, MemEx, Dunlavey’s sampling
From working code: Cell NN searching, sorting
After days of debugging: Qsort, malloc
After organizational hurdles: E-mail system

A new experiment in an existing framework
Next two stories: bin packing, k-d trees
History: Analysis of K-d Trees

K-d trees

Nearest Neighbor search

13 August 1974

The power of graphical displays
Any graph at all
A graph of the right variable

Trim the data
An Old Graph

Nearest Neighbor search in K-d trees

13 Aug 1974

Q: Why did it take me so long to draw this graph?
A: A skim of my three Prob/Stat books showed one scatter plot (of 8 points!)

Why the bumps?
Later Graphs

Two separable issues

Work going down the tree: node visits

Work at the leaves: dist calcs in the buckets

The Whole Truth

A Peek at Asymptopia

1. Extrapolation
2. Components of a count
3. Robust Conclusions

Observations: tradeoffs and cyclicity (binary trees)

Plea for statistical help: How to design this experiment?
Graphs of Bin Packing

Graph 1: Empty space, \( n = 128,000 \)

Observation: spread is about 1

A single packing of bins (l-to-r)

Graph 2: Empty except last

Insight: ignore the final bin!
## Zoo of Little Experiments

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Year(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NN searching with cells</td>
<td>1980</td>
</tr>
<tr>
<td>Sort times under caching</td>
<td>1999</td>
</tr>
<tr>
<td>Memory cost model</td>
<td>2000</td>
</tr>
<tr>
<td>A broken qsort</td>
<td>1991</td>
</tr>
<tr>
<td>An expensive malloc</td>
<td>1995</td>
</tr>
<tr>
<td>String reversal in Awk</td>
<td>2004</td>
</tr>
<tr>
<td>Compressing USA names</td>
<td>2006</td>
</tr>
<tr>
<td>Dunlavey’s sampling</td>
<td>2007</td>
</tr>
<tr>
<td>An e-mail system</td>
<td>2007</td>
</tr>
<tr>
<td>NN searching in k-d trees</td>
<td>1974, 1991</td>
</tr>
<tr>
<td>Bin packing</td>
<td>1983, 2005</td>
</tr>
</tbody>
</table>

A third of a century of inducing intraocular trauma

Sampler of colleagues: Friedman, Weide, Yao, Leighton, Johnson, McGeoch, Kernighan, McIlroy, …
Context: Tiny MSE

Problem from colleagues

Hashing \(~20,000\) 100-character strings into 32-bit ints

Too many collisions?

How many should we expect?

Birthday problem

Math

With 23 people, about a 50% chance of shared birthdays

When tossing balls into \(N\) urns, probability of 50% of shared urn after about \(N^{\frac{1}{2}}\) tosses

Observe about 10 collisions after 100 hashes

Solution

Science

A tiny horse race: testbed to count collisions for several hash functions

A couple hours later, the best was “good enough”
# Kinds of Experiments

## Parameter Estimation
- NN searching with cells
- Sort times under caching
- Memory cost model
- String reversal in Awk
- Compressing USA names
- Bin packing
- E-mail system

## Functional Form
- Sort times under caching
- Memory cost model
- Costly qsort, malloc
- String reversal in Awk
- E-mail system
- NN searching in k-d trees
- Bin packing

## Hypothesis Testing
- Sort times under caching
- Memory cost model
- Costly qsort, malloc

## Horse Races
- NN searching with cells
- Sort times under caching
- String reversal in Awk
Lessons

Reduce a huge problem to its tiny essence
  Theory: NN searching, bin packing
  Practice: Broken qsort and malloc; e-mail system
A fundamental iteration
  Measure run time ⇔ count key operations
Identify critical environmental issues
  Caching, malloc, qsort, etc.
Draw graphs
  Plot the right variables
  Look at enough detail to expose the real shape
Think small
Why Small Experiments on Algorithms?

Small is Cheap
  Useful for making engineering decisions
  Dunlavey’s sampling

Small is Insightful
  Results are usually straightforward to interpret
  Decompose a big system into critical components
  Memory cost model

Small is Beautiful
  Document serious flaws in real systems
    Qsort, malloc, e-mail
  New insights for bin packing and $k$-d trees
Plea for Statistical Help

1. Statistical common sense
   Few-point analyses: revise the lost art of *EDA*
   Drawing simple graphs
   Most of these graphs are cost as a function of size

More substantial issues

2. How do I reason about layered systems (such as caching)?
   What canonical tests must I perform?
   What precise statements do I then make?

3. How do I design experiments to get the most insight bang for the least computational buck?
   Neil Sloane: A *sequence* of experiments – given goals and results of experiments 1 … *N*, how to design *N*+1?
Special Bonus Material!

Probably not time in the workshop

A Normal Form for describing experiments

Feedback

  Algorithms people: How can we describe little experiments?

  Statisticians: How do you address similar problems?
A Small Medical Experiment

Lesson 2: Format


Hypothesis. It is proposed that the HAINES (High Arm IN Endangered Spine) modified recovery position reduces movement of the neck. In this modification, one of the patient's arms is raised above the head to support the head and neck.

Methods. Neck movements in two healthy volunteers were measured by video-image analysis and radiographic studies.

Results. For both subjects, the total degree of lateral flexion of the cervical spine in the HAINES position was less than half of that measured during use of the lateral recovery position.

Conclusion. An unconscious person with a suspected neck injury should be positioned in the HAINES modified recovery position. There is less neck movement than when the lateral recovery position is used, and, therefore, HAINES use carries less risk of spinal-cord damage.

Lesson 1: Size – 2002: 38 subjects, more analysis
Sort Algorithms in the Presence of Caching

Background

Domain: Sorting algorithms
Motivation: Determine the effect of caching on the run time of sort algorithms
Goal: A simple model to predict performance under caching

Algorithm

Design techniques: Implement existing Quicksort and heapsort algorithms
Code: ~50 lines of C

Experimental Apparatus

Overview: Single program implementing driver and algs
Generated Input Data: Arrays of uniformly distributed integers
Output: Run times of sorts

Analysis

Technique: Graph time per element as a function of size (log scale)
Results: Without caching, plots would be linear. With caching, linear within L1, L2 and RAM
Conclusions: Simple model explains caching times
A Slow Malloc

**Background**
Domain: C library malloc function
Motivation: Production library malloc was flawed
Goal: Document that the library malloc is unreasonably slow

**Algorithm**
Name: Unknown storage allocation algorithm
Design techniques: Unknown
Code: ~5 lines of driver + library malloc

**Experimental Apparatus**
Overview: Simple driver to malloc $n$ blocks each of size $m$
Generated Input Data: None
Output: Run times via time(1)

**Analysis**
Technique: Few-point analysis
Results: Run time is quadratic for a simple class of inputs
Conclusions: A system library function should be rewritten
References: “With malloc aforethought”
A Slow Qsort

Background
Domain: C library qsort function
Motivation: Production library sort was flawed
Goal: Document that the library sort is unreasonably slow

Algorithm
Name: Quicksort implementation of C qsort interface
Design techniques: Divide-and-conquer
Code: ~10 lines of driver + ~100 lines of broken sort

Experimental Apparatus
Overview: Simple driver to generate one input and call qsort
Generated Input Data: “Organ pipe” input arrays of the form 012345543210
Output: Run times via time(1)

Analysis
Technique: Few-point analysis
Results: Run time is quadratic for a plausible class of inputs
Conclusions: A system library function should be rewritten
References: “The trouble with Qsort”, “Engineering a sort function”
String Reversal in Awk

Background
Domain: Efficient string operations in a high-level language
Motivation: Fundamental problem; useful algorithm design techniques; cost models
Goal: Confirm that the cost of a straightforward string reversal algorithm is quadratic, and design a more efficient algorithm

Algorithm
Design techniques: Iteration, divide-and-conquer
Code: ~30 lines of Awk

Experimental Apparatus
Overview: Single program to implement and exercise two algorithms
Generated Input Data: Strings of the form $a^N$
Output: Run times via time(1)

Analysis
Technique: Few-point analysis
Results: Simple algorithm is apparently $\Theta(N^2)$, while divide-and-conquer algorithm is $\Theta(N \lg N)$
Conclusions: Divide-and-conquer can be simple and useful
Representing Common Names

Background

**Domain:** Data compression

**Motivation:** Represent a directory in an 8MB telephone

**Goal:** Represent a set of names in as little space as possible

Algorithm

**Design techniques:** Replacing a string by an index into a table of strings

**Code:** None; measurement only

Experimental Apparatus

**Overview:** Examine publicly available data on distribution of USA first and last names

Analysis

**Technique:** Few-point analysis

**Results:** For American last names, one byte represents 28% and two bytes represents 88% of the population. For American male first names, one nybble represents 29%, and one byte represents 76%.

**Conclusions:** Simple techniques could provide effective compression