VISUALIZING
THE ANALYSIS of ALGORITHMS

ROBERT SEDGEWICK
Princeton University

Can graphics be
"more precise and revealing"
than mathematical formulae?
Ground rules

Graphical images can
* enhance understanding of technical concepts
* help identify or clarify technical goals
* reveal otherwise unnoticable characteristics

Advances in technology enable
creation of new types of images

Analysis of Algorithms researcher’s toolkit
[GFs, asymptotics, special functions....]
TeX
Maple
C/C++
PostScript
Java

THEMES
Apply basic programming skills
Learn basic graphic design principles
Exploit mass-market technologies

“What tools would Euler be using?”
Graphic design for data visualization

GOAL: Communicate complex ideas with clarity, precision, and efficiency

Ref: E. W. Tufte
The Visual Display of Quantitative Information (1983)
Envisioning Information (1990)
Visual Explanations (1997)
Oriented towards statistical data sets but basic principles are generally applicable

TUFTE: “Graphical displays should
* show the data
* induce the viewer to think about substance
* avoid distorting what the data say
* present many numbers in a small place
* encourage comparison of different data pieces
* reveal the data at several levels of detail, broad overview to fine structure
* serve a clear purpose:
  description, exploration, tabulation, decoration
* be closely integrated with statistical and verbal description of data”

Tufte’s books elaborate on these ideas with extensive illustrative examples
Low-tech example

Basic divide-and-conquer recurrences
study of mergesort
properties of bitstrings
arithmetic algorithms
divide-and-conquer algorithms

Ex: C program to print values of recurrence

```c
#include <math.h>
#include <stdio.h>

void main(int argc, char* argv[])

{
    int i, N, c[32], d[32];
    c[0] = 0; c[1] = 0;
    for (N = 2; N < 32; N++)
    {
        c[N] = 2*c[N/2] + N;
        d[N] = N*(log((float) N)/log(2.0))-c[N];
        printf("%2d %3d %3d ", N, c[N], d[N]);
        for (i = 0; i < d[N]; i++) printf(" ");
        printf("* ");
    }
}
```

"Graphical" version: loop to print spaces and *
"Higher tech": use Postscript (stay tuned)
Aside: ancient low-tech example

Best case of quicksort
(with cutoff to insertion for small files)
Which partitioning values
minimize the total number of comparisons?

```c
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char* argv[])
{
    int i, j, N = atoi(argv[1]), M = atoi(argv[2]);
    int c[1000], t[1000];
    for (i = 0; i < M; i++) c[i] = 0;
    for (i = M; i < N; i++)
    {
        for (j = 0; j < i; j++)
            t[j] = c[j] + c[i-j-1] + i;
        for (j = 0, c[i] = 1000000; j < i; j++)
            if (t[j] < c[i]) c[i] = t[j];
        for (j = 0; j < 40-i; j++) printf(" ");
        for (j = 0; j < i; j++)
            if (t[j] == c[i]) printf("* ");
            else printf(" ");
        printf(" ");
    }
}

Ref: Sedgewick, "Quicksort," 1972
### Example: a familiar table of numbers

#### Binomial coefficients

<table>
<thead>
<tr>
<th></th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 1</td>
</tr>
<tr>
<td>1</td>
<td>1 2 1</td>
</tr>
<tr>
<td>1</td>
<td>1 3 3 1</td>
</tr>
<tr>
<td>1</td>
<td>1 4 6 4 1</td>
</tr>
<tr>
<td>1</td>
<td>1 5 10 10 5 1</td>
</tr>
<tr>
<td>1</td>
<td>1 6 15 20 15 6 1</td>
</tr>
<tr>
<td>1</td>
<td>1 7 21 35 35 21 7 1</td>
</tr>
<tr>
<td>1</td>
<td>1 8 28 56 70 56 28 8 1</td>
</tr>
<tr>
<td>1</td>
<td>1 9 36 84 126 126 84 36 9 1</td>
</tr>
</tbody>
</table>

#### Binomial distribution

<table>
<thead>
<tr>
<th></th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/2</td>
<td>1/2</td>
</tr>
<tr>
<td>1/4</td>
<td>2/4 1/4</td>
</tr>
<tr>
<td>1/8</td>
<td>3/8 3/8 1/8</td>
</tr>
<tr>
<td>1/16</td>
<td>4/16 6/16 4/16 1/16</td>
</tr>
<tr>
<td>1/32</td>
<td>5/32 10/32 10/32 5/32 1/32</td>
</tr>
</tbody>
</table>
PostScript

Available in all modern computing environments
[ basic language in printing industry]
Has all basic components for our [modest] needs

Postfix language, uses abstract stack machine
Ex: convert 9753 from hex to decimal (Horner alg)
  9 16 mul 7 add 16 mul 5 add 16 mul 3 add

Coordinate system: rotate, translate, scale, ...
Turtle commands: moveto, lineto, rmoveto, rlineto,
Graphics commands: stroke, fill, ...
Arithmetic: add, sub, mul, div, ...
Stack commands: exch, dup, currentpoint, ...
Control: if, ifelse, while, for, ...
Define: /xx { ... } def

Everyone’s first program: draw a box
%
36 36 translate 0 0 moveto
0 72 rlineto 72 0 rlineto
0 -72 rlineto -72 0 rlineto
stroke
showpage
%%
/inch { 72 mul } def
/Xsize 6.5 inch def /Ysize 4 inch def
/myscale
    { Ysize mul 2 N exp div exch
      N div Xsize mul exch } def
2 1 100
{ /N exch def
  newpath
  /Y 1 def
  0 Y myscale moveto
  1 1 N
  { /k exch def
    /Y Y N k sub 1 add mul k div def
    k Y myscale lineto
  } for
  stroke
} for
showpage
Other familiar distributions

Graphic design ideas for binomial plot
  scale Y axis to \([0, 1]\)
  scale X axis to \([0, 1]\)
  superimpose all plots

Apply to numerous distributions
  Stirling numbers
  Catalan
  Eulerian

size-cost in analysis of algorithms
  Quicksort
  tries
  AVL trees

... [cf. Flajolet-Sedgewick]

Trivial changes in programs
  lead to striking differences in images

Ex: spread curves by a few points

See ASCII slides on talk web page
  for Postscript code
Binomial distribution with spread curves
Cost of computation

How much time is required to produce a plot?

"Analysis of Algorithms" problem
[If we can't do this, who can!!]

Binomial plot: time proportional to $N^2$?
[not really: depends on resolution]

Modern personal computer
over 250 million operations per second

Million bits
three-inch square image at 300 dpi

BOTTOM LINE
now feasible to do 100s or 1000s of ops per bit

Note: EVERY "image processing" program does
significant computation for each bit!

Aside: bitmap computational geometry algs
now practical for huge problems

Caveat: printer may not be as fast as PC
Example: complex functions

Graphic design idea:
use color scale for absolute values

Postscript implementation
simple library of complex functions
compute each bit (!!!)

Basic loop to plot each point
0 1 nY
{ /Y exch def
  0 1 nX
  { /X exch def
    X dx mul Y dy mul color pt
  } for
} for

Graphics functions (scaling omitted)
/pt { sz 0 360 arc fill } def
/color
{
  X Y scale /y exch def /x exch def f ABS
dup MAX gt
  { pop 0.0 }
  { MAX div 1 exch sub}
ifelse
dup dup sethsbcolor
} def
Complex functions (continued)

PostScript functions implement "complex" type

/Z { x y } def

/SUB
    { /d exch def /c exch def
      /b exch def /a exch def
      a c sub b d sub } def

/DIV
    { /d exch def /c exch def
      /b exch def /a exch def
      /dd c dup mul d dup mul add def
      a c mul b d mul add dd hackdiv
      b c mul a d mul sub dd hackdiv
    } def

/ABS { dup mul exch dup mul add sqrt } def

/f { 1 0 1 0 Z SUB DIV } def

Now can "plot" arbitrary complex functions

Examples from the analysis of algorithms
   rational polynomial GFs
   quicksort
   tries
   Catalan
   ...
Perspective

Three examples
  - printing asterisks
  - plotting curves
  - set colors in bitmap

Characteristics
  - easy to implement
  - follow Tufte’s principles
  - expose essential algorithm-analysis concept

Why not use Maple, Mathematica plot packages?
  - plus:
    - extensive built-in library
    - 3D rendering, etc., etc
  - minus:
    - design inflexibility
    - graphics computation cost

Scratching the surface of design possibilities
  - black-and-white
  - grayscale
  - color
  - animation

Basic results broadly applicable