Creating “Algorithms”

Robert Sedgewick
Princeton University
## Brief history of books

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>1982</th>
<th>Pascal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Second Edition</td>
<td>1988</td>
<td>Pascal</td>
</tr>
<tr>
<td></td>
<td>1990</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>1992</td>
<td>C++</td>
</tr>
<tr>
<td></td>
<td>1993</td>
<td>Modula-3</td>
</tr>
<tr>
<td>Third Edition 1-4</td>
<td>1997</td>
<td>C</td>
</tr>
<tr>
<td>basic/ADTs/sort/search</td>
<td>1998</td>
<td>C++</td>
</tr>
<tr>
<td>~700 pages</td>
<td>2002</td>
<td>Java</td>
</tr>
<tr>
<td>Third Edition 5</td>
<td>2001</td>
<td>C</td>
</tr>
<tr>
<td>graph algorithms</td>
<td>2001</td>
<td>C++</td>
</tr>
<tr>
<td>~500 pages</td>
<td>2003</td>
<td>Java</td>
</tr>
</tbody>
</table>

Translations: Japanese, French, German, Spanish, Italian, Polish, Russian
20 years, 11 books, 17+ translations, 400,000+ copies in print
Ground rules for book authors

1. You are on your own
2. Deadlines exist
3. Content over form
4. Focus on the task at hand
5. Tell the truth about what you know
6. Revise, revise, revise
Goals:

- Algorithms for the masses
- Use real code, not pseudocode
- Exploit computerized typesetting technology

Problems:

- Real code hard to find for many algorithms
- Laser printers unavailable outside research labs
- Low resolution
- Software to create figures?

Approach:

- emacs + TeX for text
- Pen-and-ink for figures
1977 historical context

[Images: Star Wars movie poster, two men shaking hands, a sign reading 'Palo Alto Research Center', and a computer from the 1970s.]
The code for this method is straightforward. In the following implementation, `insert` adds a new item to `a[N]`, then calls `upheap(N)` to fix the heap condition violation at `N`:

```pascal
procedure upheap(k: integer);
  var v: integer;
  begin
    v := a[k]; a[0] := maxint;
    while a[k div 2] < v do
      begin
        a[k] := a[k div 2]; k := k div 2;
        a[k] := v;
      end;
  end;

procedure insert(v: integer);
  begin
    N := N + 1; a[N] := v;
    upheap(N);
  end;
```

As with insertion sort, it is not necessary to do a full exchange within the loop, because `v` is always involved in the exchanges. A sentinel key must be put in `a[0]` to stop the loop for the case that `v` is greater than all the keys in the heap.

The `replace` operation involves replacing the key at the root with a new key, then moving down the heap from top to bottom to restore the heap condition. For example, if the `X` in the heap above is to be replaced with `C`, the first step is to store `C` at the root. This violates the heap condition, but the violation can be fixed by exchanging `C` with `T`, the larger of the two sons of the root. This creates a violation at the next level, which can be fixed.
Goal:
All the book’s code should be real code.

Problems:
Pascal compiler expects code in .p file
TeX formatter expects code in .tex file
Not all the code goes into the book
Code has to be formatted
Continually need to fix bugs and test fixes

Solution:
Add comments in .p files to id and name code fragments
Add “include” lines to source that refer to names
loom: shell script to build .tex file
Example program
One of the simplest algorithms for this task works as follows: first do this, then do that. This code does this and that:

```
program example(input,output);
var a: array[1..100] of integer;
N,i: integer;
{include code}
procedure solve;
var i,j,t: integer;
begin
... 
end;
{end include code}
beg
... 
solve;
... 
end.
```

This algorithm is sometimes useful. Its running time is proportional to ...

Typescript (.tex file)

```
Example program
One of the simplest algorithms for this task works as follows: first do this, then do that. This code does this and that:

```
procedure solve;
var i,j,t: integer;
begin
... 
end;
```

This algorithm is sometimes useful. Its running time is proportional to ...

Text (.loom file)

```
\Sh{Example program}
One of the simplest algorithms for this task works as follows: first do this, then do that. This code does this and that:
\prog{
\%include example.p code
}
\noindent
This algorithm is sometimes useful. Its running time is proportional to ...
```

Program (.p file)
Second edition 1986-87

Goals:
Make content more widely accessible
Eliminate pen-and-ink
Add visual representations of data structures

Problem:
Figures are numerous and intricate

Opportunities:
LaserWriter + PostScript
Algorithm animation research

Approach:
Add introductory material; move math algs to end
dsdraw: package for drawing data structures
fig: use loom to include program output in figs
dsdraw

PostScript code to draw data structures
basic graphics
automatic layout of snapshots

Ex: points in the plane

```
/points
  % Points in the plane
  % Stack: array containing the points
  % (label,x,y] for each node).
  % (Example: [[(C) 1 3] [(B) 2 5] [(D) 3 5]
  % [(A) 3 1]])
  % Optional fourth argument can change nodestyle
  % Put a dummy point [N M] to fool (size) (?)
{/option exch def
  option (size) eq
  {dup
    /xmax 0 def /ymax 0 def
    {aload length 4 eq {pop} if
      dup ymax gt {ymax exch def}{pop} ifelse
      dup xmax gt {xmax exch def}{pop} ifelse
      pop} forall
    xmax ymax} if
  option (plot) eq
  {{aload length 3 eq {nodestyle} if drawnode}
    forall} if
  } def
```
dsdraw: basic data structure drawings

permutation
array of ints
2D array
points
completetree
tree
polygon
graph

[[[(X)]
  [(T) A][(P)]
  [(G)][(S) A][(O)][(N)]
  [(A)][(E)][(R) A][(A)][(I)][(M)][(M)]]]
(completetree)

[[[[A) 1 7][(B) 2 5][(C) 4 5][(D) 2 3][(E) 4 3]
  [(F) 1 1][(G) 6 5][(H) 8 6][(I) 10 6][(J) 8 3]
  [(K) 10 3][(L) 8 1][(M) 10 1]]
  [] 1 7[] 1 2[] 1 3[] 12 13[] 10 13]
  [] 10 12[] 10 11[] 5 4[] 6 4[] 8 9
  [] 6 5[] 1 6[] 7 5]][]]
(graph)

[[[...]]
(polygon)
Goal:
   Use programs to produce figures

Problem:
   figures are PostScript programs

Opportunities:
   loom

Solution:
   instrument Pascal code to produce .ps code
   use loom to include program output in .ps files
   (filter out instrumentation)
   include refs to .ps files in .tex files
One of the simplest algorithms for this task works as follows: first do this, then do that. This code does this and that:

```
{include code}
procedure solve;
  var i,j,t: integer;
  begin
    ...{IE}
    for i:= l to r do
      {IE }
        write(a[i]:4);
      ...{end include code}
  end;
{end include code}
```

This algorithm is sometimes useful. This figure shows how it works:

```
{fig{... psfile: fig1.ps ...}}
```

Note: can use loom here, too!
dsdraw: automatic layout of snapshots

```
[ [ (A)][ (S)][ (O)][ (R)][ (T)]
 [ (I)][ (N)][ (G)][ (E)][ (X)]
 [ (A)][ (M)][ (P)][ (E) B]
 [ (A)][ (S)][ (O)][ (R)][ (T)]
 [ (P) A][ (N)][ (G)][ (E)][ (X)]
 [ (A)][ (M) B][ (I) A][ (L)][ (E)]]
 [ [ (A)][ (S)][ (O)][ (R)][ (T)]
 [ (P)][ (N)][ (G)][ (E)][ (T) A]
 [ (A) B][ (M)][ (I)][ (L)][ (E)]]
 [ [(A)][ (S)][ (O)][ (R) A][ (X)]
 [ (P)][ (N)][ (G) B][ (E) B][ (T)]
 [ (A)][ (M)][ (I)][ (L)][ (E)]]
 [ [(A)][ (S)][ (P) A][ (R)][ (X)]
 [ (O) A][ (N) B][ (G)][ (E)][ (T)]
 [ (A)][ (M)][ (I)][ (L)][ (E)]]
 [ [(A)][ (X) A][ (P)][ (R) B][ (T) A]
 [ (O)][ (N)][ (G)][ (E)][ (S) A]
 [ (A) B][ (M)][ (I)][ (L)][ (E)]]
 [ [(X) A][ (T) A][ (P) B][ (R) B]
 [ (S) A][ (O)][ (N)][ (G)][ (E)]
 [ (A) A][ (A) B][ (M)][ (I)][ (L)]
 [ (E)]]
]
(completetree)
```
Beyond manual drafting
Second edition features

- Algorithms for the masses
- Uses real code, not pseudocode
- Fully exploits technology

Original goals realized, PLUS

Innovative, detailed visualizations

Done?
Other languages (1990-1993)

**Mandate:**
Spread the word in other programming languages

**Challenges:**
Which languages? (Answer: C, C++, and Modula-3)
Who translates?
Early versions of new languages are unstable

**Solution:**
Copy-and-edit to implement programs in new language
Use conditionals in typescript for language-dependent text

**Problems:**
(figs were produced by Pascal programs)
difficult to take advantage of language features
typescript is a mess; layout is painful
Third edition 1993-

Goals:
- Full coverage, not summary
- Take visualizations to next level
- Analyses with empirical verification

Challenges:
- Typescript filled with conditionals
- Program code filled with instrumentation
- Figs made with Pascal code
- Many algorithms not well-understood

Approach:
- START OVER, one language at a time
- Status: 9 books, 6 done

Algorithms

- Parts 1-4
- Part 5
- Parts 6-8

Languages

- C
- C+
- Java
Starting over (third edition)

Layout:
Structured text, figures, exercises, programs, tables
Multiple story flows (figs with captions in margins)

Figures:
Direct PostScript implementations
Visualize “large” examples
Explanatory captions

Programs:
Full implementations to support empirical studies
Emphasize ADTs in all languages
Use consultants to champion language features

Exercises:
All questions addressed

Tables:
Summarize full empirical studies
PostScript as algorithm visualization tool

/insert
{
/X rand 1000 idiv N mod def
/N N 1 add def
/sum 0 def
/a [
0 1 a length 1 sub
{
  a exch get /nd exch def
  X sum ge X sum nd add lt and
  {
    nd 1 add M 1 add ge
    {
      M 1 add 2 div dup
      /S S 1 add def }
    { nd 1 add } ifelse
    } { nd } ifelse
  } /sum sum nd add def
} for
] def
} def

/doit
{
/a [ M ] def showline
Nmax { insert showline } repeat
} def
Third edition features programs C, C++, Java figures dsdrawn direct tables empirical summaries exercises (1000s) properties (theorems) layout design links**

** not enough (stay tuned)
Creating “Algorithms”

text sections

**GRAPH PROPERTIES AND TYPES**

Many computational applications naturally involve not just a set of items, but also a set of connections between pairs of those items. The relationships implied by these connections lead immediately to a host of natural questions: Is there a way to get from one item to another by following the connections? How many other items can be reached from a given item? What is the best way to get from this item to this other item?

To model such situations, we use abstract objects called graphs. In this chapter, we examine basic properties of graphs in detail, setting the stage for us to study a variety of algorithms that are useful for answering questions of the type just posed. These algorithms make effective use of many of the computational tools that we considered in Parts 1--4. They also serve as the basis for attacking problems in important applications whose solution we could not even contemplate without good algorithmic technology.

exercises

Write a representation-independent graph-initialization ADT function that, given an array of edges, returns a graph.

programs

```
program euclid(input,output);
var x,y: integer;
function gcd(u,v:integer): integer;
beging
if v=0 then gcd:=u
else gcd:=gcd(v, u mod v)
end;
begin
while not eof do
begin
readln(x,y);
if x<0 then x:=-x;
if y<0 then y:=-y;
writeln(x,y,gcd(x,y));
end;
end.
```
Bookmaker (the lonely author)

juggler image from Northern Lights Software
## Facts and figures

<table>
<thead>
<tr>
<th></th>
<th>pages</th>
<th>programs</th>
<th>figures</th>
<th>tables</th>
<th>exercises</th>
<th>files</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithms</strong></td>
<td>550</td>
<td>140</td>
<td>150</td>
<td>0</td>
<td>400</td>
<td>600</td>
</tr>
<tr>
<td><strong>Second edition</strong></td>
<td>650</td>
<td>200</td>
<td>350</td>
<td>0</td>
<td>400</td>
<td>6,000</td>
</tr>
<tr>
<td>(typical)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Third edition</strong></td>
<td>1200</td>
<td>250</td>
<td>500</td>
<td>75</td>
<td>2,000</td>
<td>25,000</td>
</tr>
<tr>
<td>1–5 (typical)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Third edition</strong></td>
<td>2000</td>
<td>400</td>
<td>800</td>
<td>120</td>
<td>3,500</td>
<td>40,000</td>
</tr>
<tr>
<td>1–8 (est.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
digression: PostScript as math visualization tool

```
/doit
{ /M exch def /Nmax exch def
 /A [0 M 1 sub M div 1 M div 0] def
 3 1 Nmax
  { /N exch def
    [ 0
      1 1 N
      { /k exch def
        A k 1 sub get M div A k get M 1 sub mul M div add
      } for
      0 ] /A exch def
    A drawcurve
  } for
} def
```
Fourth edition 2003-??

Goals:
- Do answers to exercises
- Stabilize content
- Create interactive and dynamic eBook supplements

Problems:
- Tens of thousands of files
- Thousands of exercises
- Different typescripts for C, C++, Java
- Deep hacks throughout figs (need new dsdraw)
- Ancient typesetting engine

Approach:
- Back to single typescript??
- Layout language??
- Scripting language??
Needs for fourth edition

1. Structured-document authoring and editing tool
   - simple system- and machine-independent editor
   - manage nonlinear organization of fragments
   - TeX-like plugin for equations
   - application-independent primary source format
   - cross-reference/indexing across all types of fragments

2. Programming tools
   - Source language with flexible ADT and IO mechanisms
   - Postscript

3. Flexible document-creation engine
   - semiautomatic layout
   - programming language
   - smart filters with link/embed/unlink/unembed
Q: Where is the “Algs” e-/dynamic-/interactive- book?


1985 choice: content over form

Triumph of content leads to (reasonable) demand for:
   Answers to exercises
   Online lecture notes
   Customizable versions
   Dynamic figures
   Interactive testing/drill
   ...

Inventing the Future
Q: Where is the “Algs” e-/dynamic-/interactive- book?

A: (2002): Where are the tools that an individual author could use to make one?