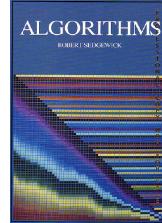
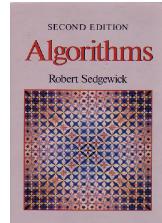
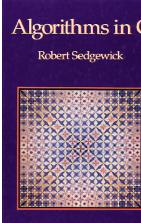
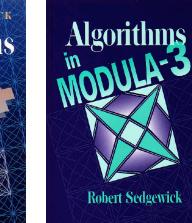
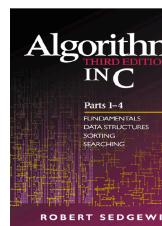
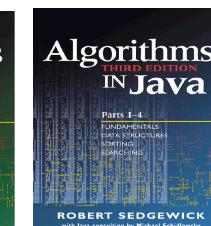
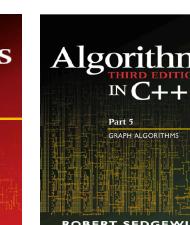
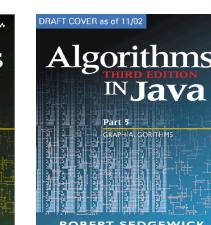


# **Creating “Algorithms”**

**Robert Sedgewick  
Princeton University**

# Brief history of books

<b>Algorithms</b> ~550 pages	1982	Pascal	
<b>Second Edition</b> ~650 pages	1988	Pascal	
	1990	C	
	1992	C++	
	1993	Modula-3	
<b>Third Edition 1-4</b> basic/ADTs/sort/search ~700 pages	1997	C	
	1998	C++	
	2002	Java	
<b>Third Edition 5</b> graph algorithms ~500 pages	2001	C	
	2001	C++	
	2003	Java	

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

# First edition 1977-1982

## Goals:

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

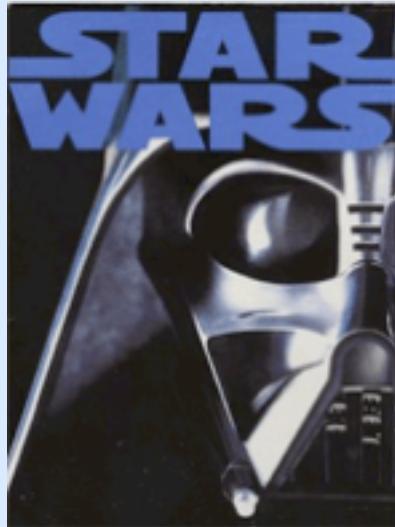
## Problems:

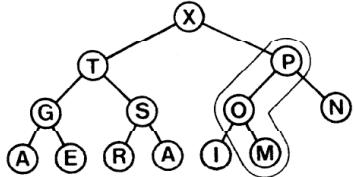
- Real code hard to find for many algorithms
- Laser printers unavailable outside research labs
- low resolution
- software to create figures?

## Approach:

- emacs + TeX for text
- pen-and-ink for figures

## 1977 historical context





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`:

---

```

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

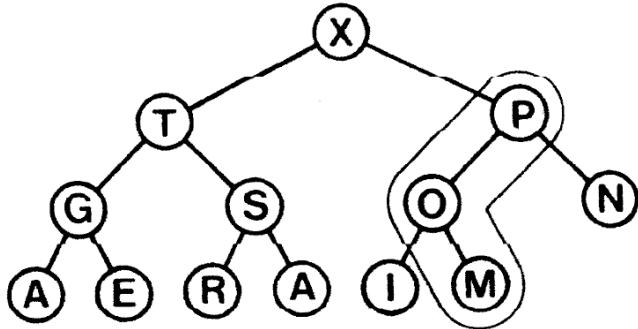
---

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

## First edition features:

**phototypeset final copy  
real Pascal code  
pen-and-ink drawings**



not enough G's in Paris

# loom

## 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

# loom example (1st edition)

## Text (.loom file)

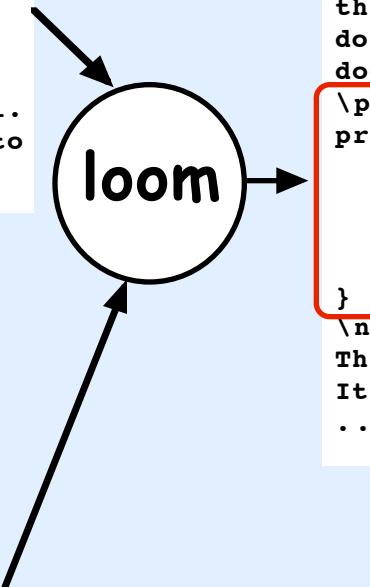
```
\Sh{Example program}
One of the simplest algorithms for
this task works as follows: first
do this, then do that. This code
does this and that:
\prog{
%include example.p code
}
\noindent
This algorithm is sometimes useful.
Its running time is proportional to
...
```

## Program (.p file)

```
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}
begin
...
solve;
...
end.
```

## TypeScript (.tex 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{
procedure solve;
var i,j,t: integer;
begin
...
end;
}
\noindent
This algorithm is sometimes useful.
Its running time is proportional to
...
```



## 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

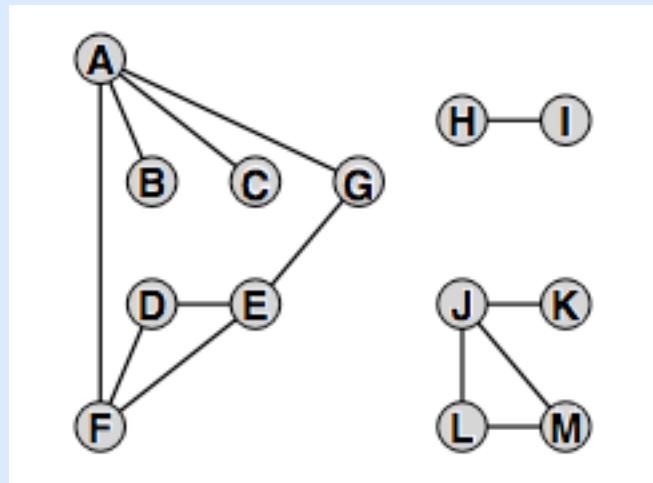


```
/points
    % Points in the plane
    % Stack: array containing the points
    % ([label,x,y] for each node).
    % (Example: [[(C) 1 3] [(B) 2 5] [(D) 3 5]
    % [(A) 3 1]])
    % Optional fourth argument can change nodestyle
    % Put a dummy point [N M] to fool (size) (?)
{/option exch def
    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
```

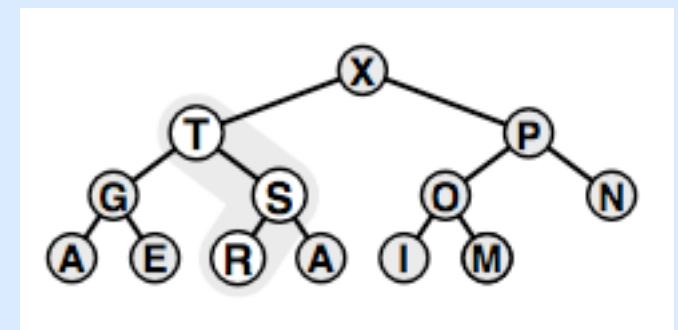
# dsdraw: basic data structure drawings

permutation  
array of ints  
2D array  
points

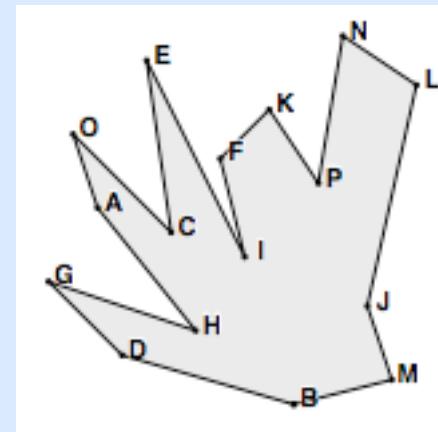
completetree  
tree  
polygon  
graph



```
[[[[[(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)
```



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



```
[ [...]
  (polygon)
```

# fig

**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

# fig example (2nd edition)

## 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 | grep -v IE
}
\noindent
This algorithm is sometimes useful.
This figure shows how it works:
\fig{... psfile: fig1.ps ...}
...
```

## TypeScript (.tex 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{
procedure solve;
var i,j,t: integer;
begin
...
end;
}
\noindent
This algorithm is sometimes useful.
This figure shows how it works:
\fig{... psfile: fig1.ps ...}
...
```

## Program (.p file)

```
...
{include code}
procedure solve;
  var i,j,t: integer;
  begin
  ...
{IE} for i:= 1 to r do
{IE }      write(a[i]:4);
  ...
  end;
{end include code}
...
```

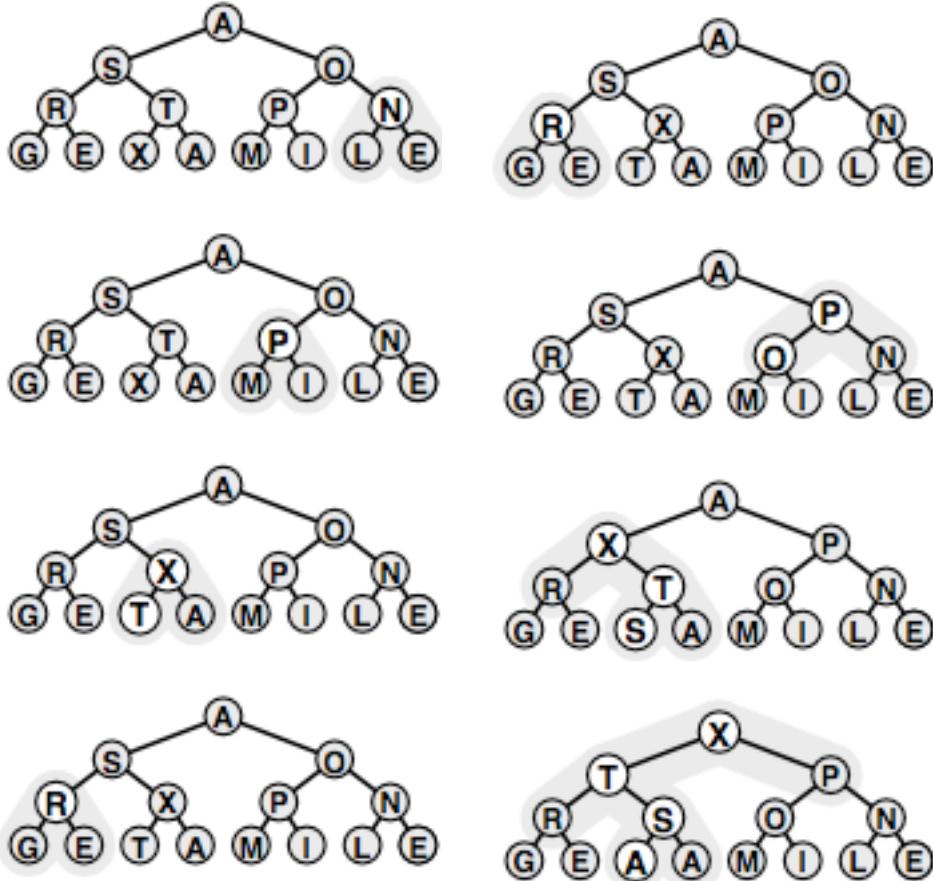
loom

## Figure (.ps file)

```
[
[127 125 126 124 115 117 122 123 ... ]
]
(permuation) showdata
```

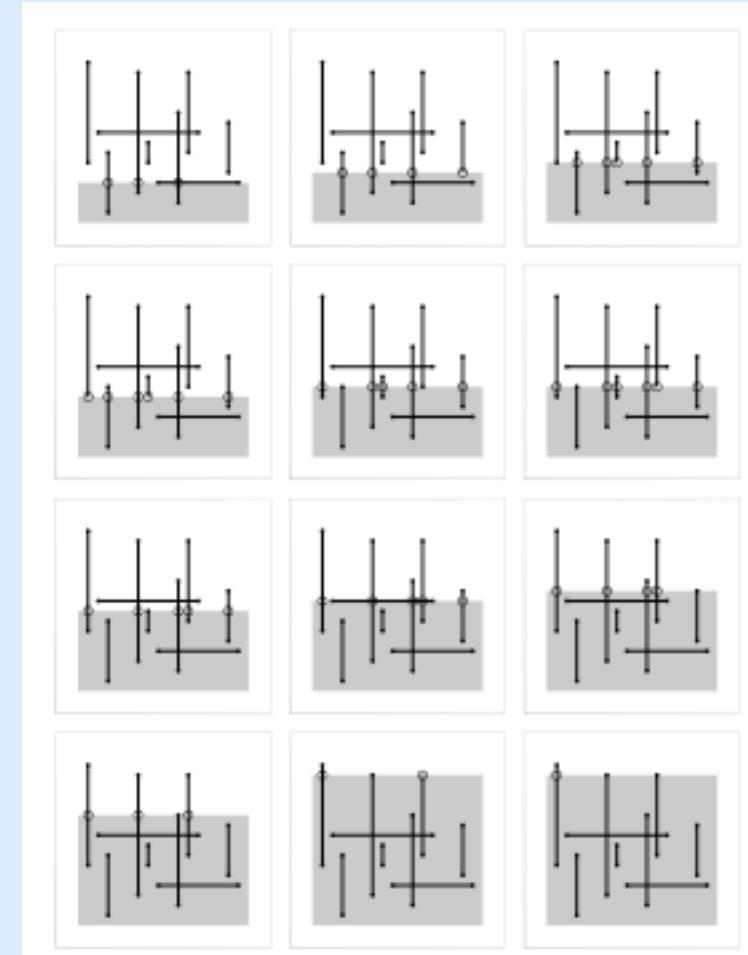
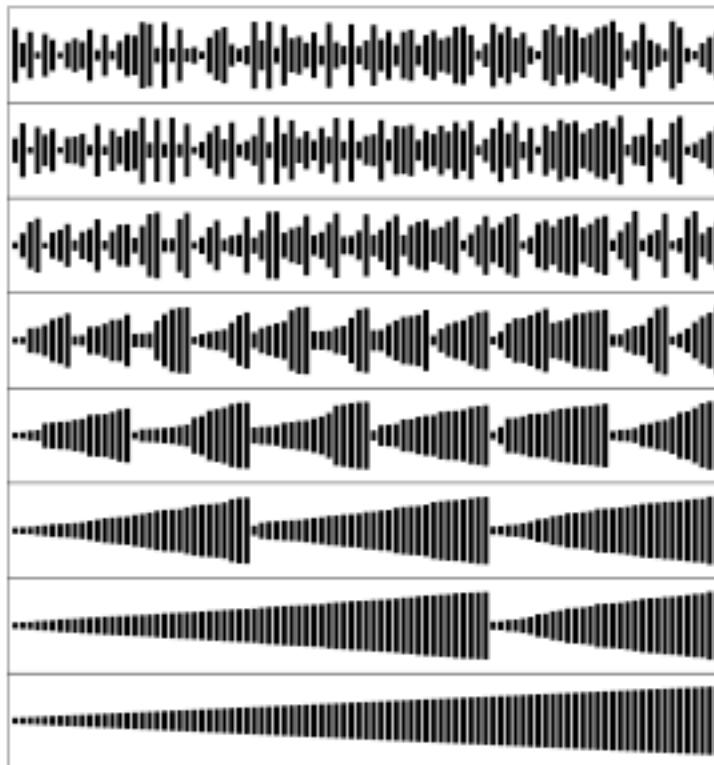
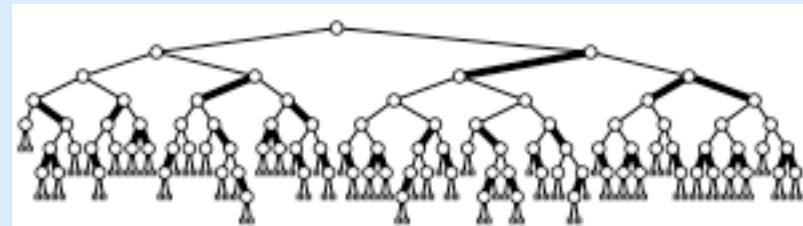
Note: can use loom here, too!

# dsdraw: automatic layout of snapshots

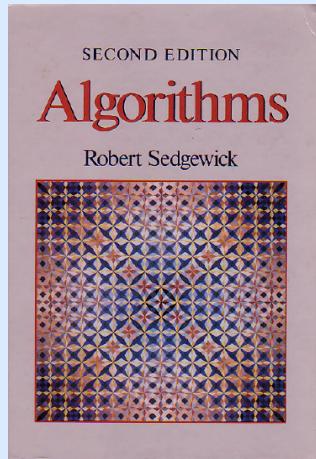


```
[  
[[ (A) ][(S)][(O)][(R)][(T)]  
[(I)][(N) A][(G)][(E)][(X)]  
[(A)][(M)][(P)][(L) B][(E) B]]  
[[ (A) ][(S)][(O)][(R)][(T)]  
[(P) A][(N)][(G)][(E)][(X)]  
[(A)][(M) B][(I) A][(L)][(E)]]  
[[ (A) ][(S)][(O)][(R)][(X) A]  
[(P)][(N)][(G)][(E)][(T) A]  
[(A) B][(M)][(I)][(L)][(E)]]  
[[ (A) ][(S)][(O)][(R)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

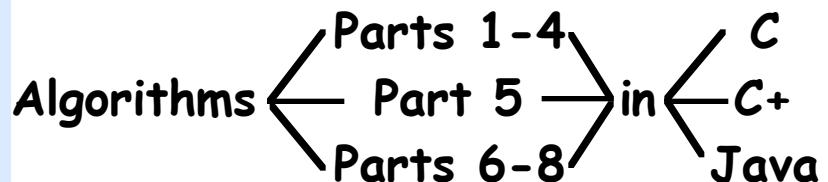
### Challenges:

- 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



# 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

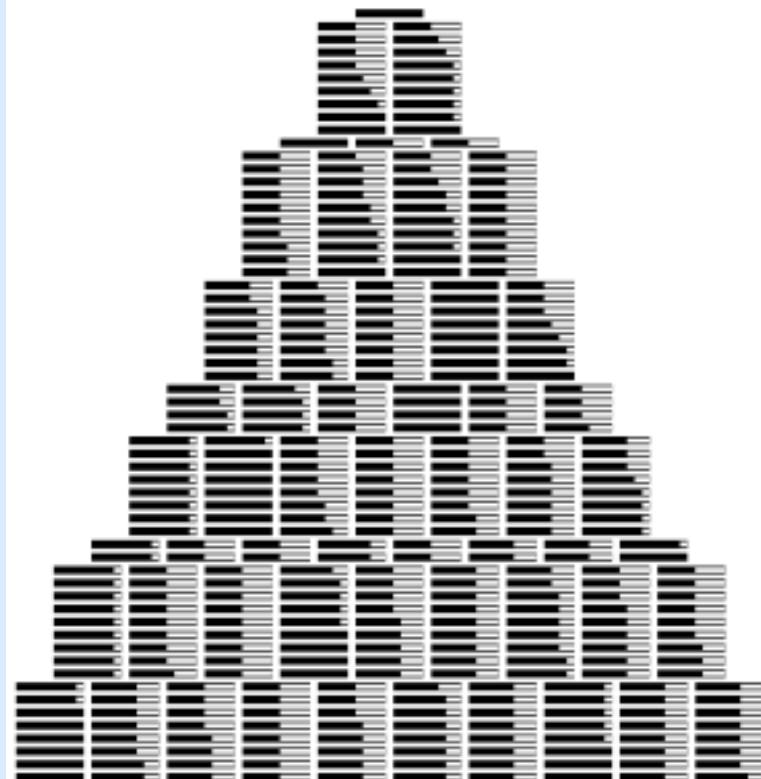
## Exercises:

- 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
```

be used for practical applications when space is not at premium and a guaranteed worst-case running time is desirable. Both algorithms are of interest as prototypes of the general *divide-and-conquer* and *combine-and-conquer* algorithm design paradigms.

#### Exercises

- 8.14 Show the merges that bottom-up mergesort (Program 8.5) does for the keys EASYQUESTION.
- 8.15 Implement a bottom-up mergesort that starts by sorting blocks of  $M$  elements with insertion sort. Determine empirically the value of  $M$  for which your program runs fastest to sort random files of  $N$  elements, for  $N = 10^3$ ,  $10^4$ ,  $10^5$ , and  $10^6$ .
- 8.16 Draw trees that summarize the merges that Program 8.5 performs, for  $N = 16, 24, 31, 32, 33$ , and 39.
- 8.17 Write a recursive mergesort that performs the same merges that bottom-up mergesort does.
- 8.18 Write a bottom-up mergesort that performs the same merges that top-down mergesort does. (This exercise is much more difficult than is Exercise 8.17.)
- 8.19 Suppose that the file size is a power of 2. Remove the recursion from top-down mergesort to get a nonrecursive mergesort that performs the same sequence of merges.
- 8.20 Prove that the number of passes taken by top-down mergesort is also the number of bits in the binary representation of  $N$  (see Property 8.6).

#### 8.6 Performance Characteristics of Mergesort

Table 8.1 shows the relative effectiveness of the various improvements that we have examined. As is often the case, these studies indicate that we can cut the running time by half or more when we focus on improving the inner loop of the algorithm.

In addition to netting the improvements discussed in Section 8.2, a good Java VM implementation might avoid unnecessary array accesses to reduce the inner loop of mergesort to a comparison (with conditional branch), two index increments ( $i$  and either  $i + 1$  or  $j$ ), and a test with conditional branch for loop completion. The total number of instructions in such an inner loop is slightly higher than that for quicksort, but the instructions are executed only  $N \lg N$  times, where quicksort's are executed 39 percent more often (or 29 percent with the

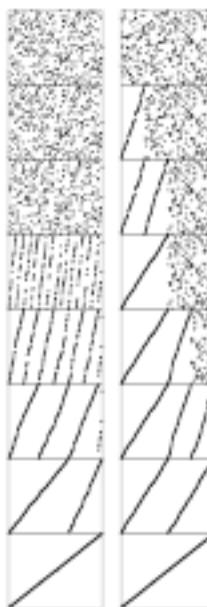


Figure 8.7  
Bottom-up versus top-down mergesort

Bottom-up mergesort (left) consists of a series of passes through the file that merge together sorted subfiles, until just one remains. Every element in the file, except possibly a few at the end, is involved in each pass. By contrast, top-down mergesort (right) sorts the first half of the file before proceeding to the second half (recursively), so the pattern of its progress is decidedly different.

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.

...

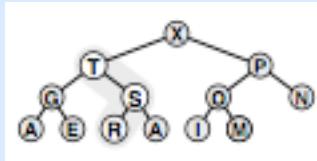
## exercises

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

## programs

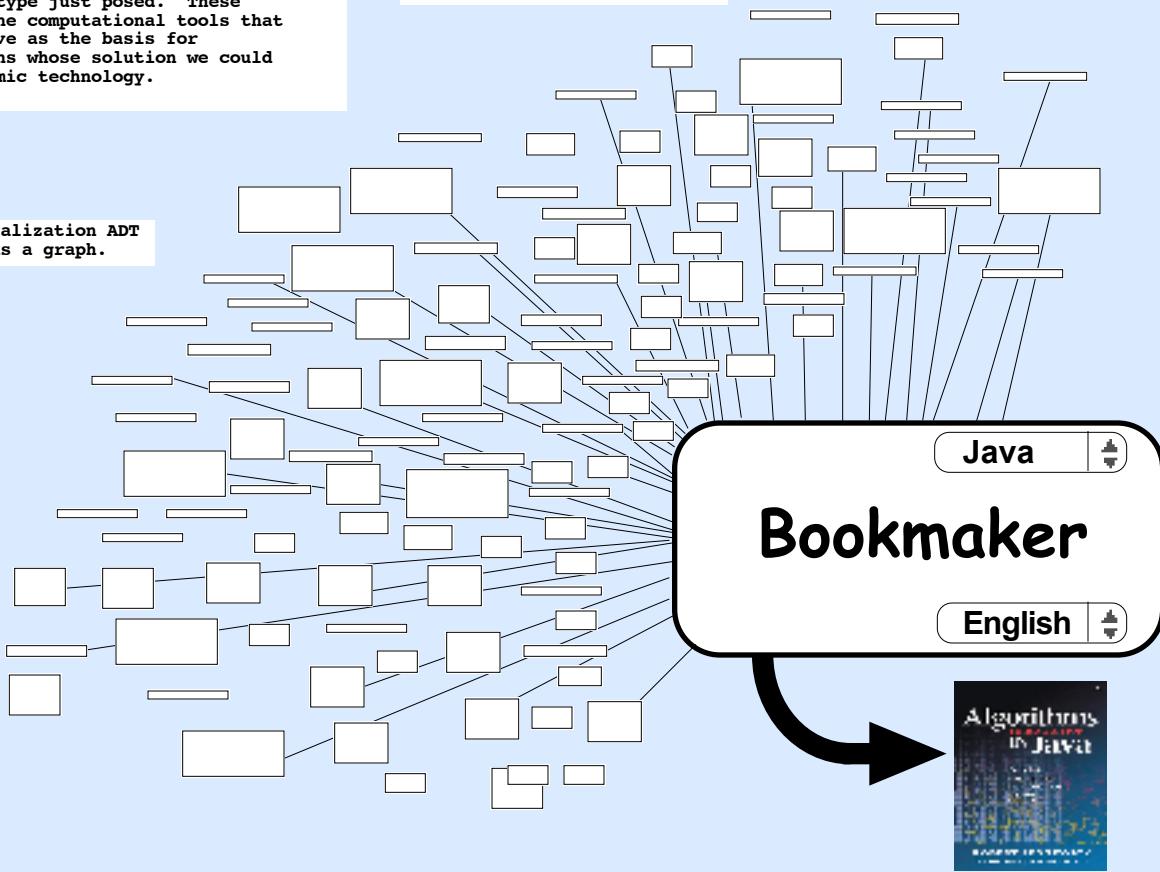
```
program euclid(input,output);
var x,y: integer;
function gcd(u,v:integer): integer;
begin
  if v=0 then gcd:=u
  else gcd:=gcd(v, u mod v)
end;
begin
while not eof do
begin
  readln(x,y);
  if x<0 then x:=-x;
  if y<0 then y:=-y;
  writeln(x,y,gcd(x,y));
end;
end.
```

## figures



## tables

N	G	T	T'	O	B	B'	S
121000	23	27	16	18	30	20	19
200000	26	43	34	27	42	28	28
500000	46	91	79	52	92	77	58
1000000	36	138	104	111	204	175	117
2000000	201	421	352	294	455	398	256
4000000	446	934	764	529	932	873	529
8000000	927	1916	1629	1134	2136	1871	1119



# Bookmaker (the lonely author)

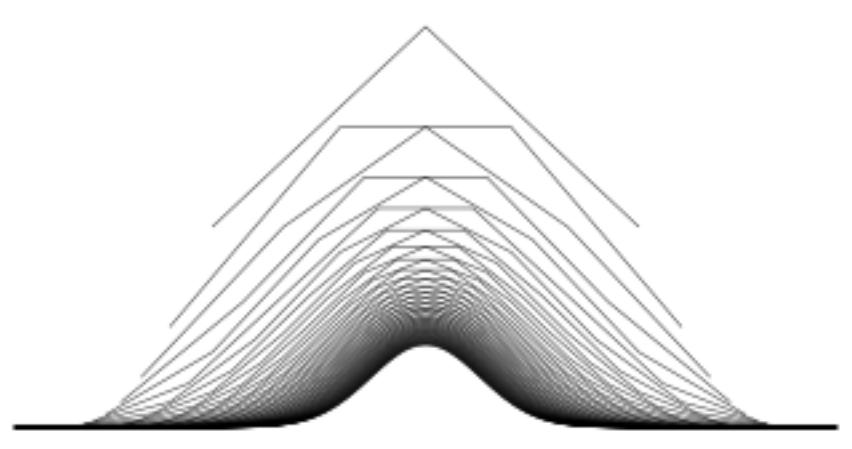


juggler image from Northern Lights Software

## Facts and figures

	pages	programs	figures	tables	exercises	files
Algorithms	550	140	150	0	400	600
Second edition (typical)	650	200	350	0	400	6,000
Third edition 1-5 (typical)	1200	250	500	75	2,000	25,000
Third edition 1-8 (est.)	2000	400	800	120	3,500	40,000

## 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

# Inventing the Future

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

**A:** (1984): Done. Balsa (with M. Brown).

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??