Computational Thinking:
A Necessary Subject in Education

Walter Gander

ETH Zurich and HKBU

May 12, 2011

50th Anniversary
香港浸會大學理學院
HKBU Faculty of Science
Department of Mathematics

Computer Development by Decades

- 1950–1960: Hardware Development
- 1960–1970: First Higher Programming Languages, Numerical Computing
- 1970–1980: Mainframes, Data Processing
- 1980–1990: Microprocessors, Personal Computer
- 1990–2000: Network and Communications, WWW
- 2000–2010: Ubiquitous Computing

Computers Determine Our Life

Communication: e-mail, cell-phone, sms, social networks: facebook, twitter, LinkedIn . . .

Writing: text-processing, spreadsheets, presentation tools

Reading: Google eBooks, e-Reader: Kindle, iPad, Sony Reader, Digital Book Index provides links to more than 165,000 full-text digital books

Music: iTune, e-music, MP3

Radio and Television: digital, Internet

Photography: software has replaced chemically processed films

Search for Information: libraries, archives available on-line, Wikipedia

many more examples . . .

However!

Teaching K-12 Computer Science in the Digital Age Fails! [a]

- Computer science and the technologies it enables now lie at the heart of our economy, our daily lives, and scientific enterprise.
- The digital age has transformed the world and workforce, but education has fallen woefully behind in preparing students with the fundamental CS knowledge and skills they need for future success.
- To be a well-educated citizen as we move toward an ever-more computing-intensive world and to be prepared for the jobs of the 21st Century, students must have a deeper understanding of the fundamentals of computer science.

---

[a] ACM and CSTA released the startling findings of their computer science education standards report, Running on Empty: The Failure to Teach K-12 Computer Science in the Digital Age, at the National Press Club in Washington, DC.

Same Failure in Other Countries

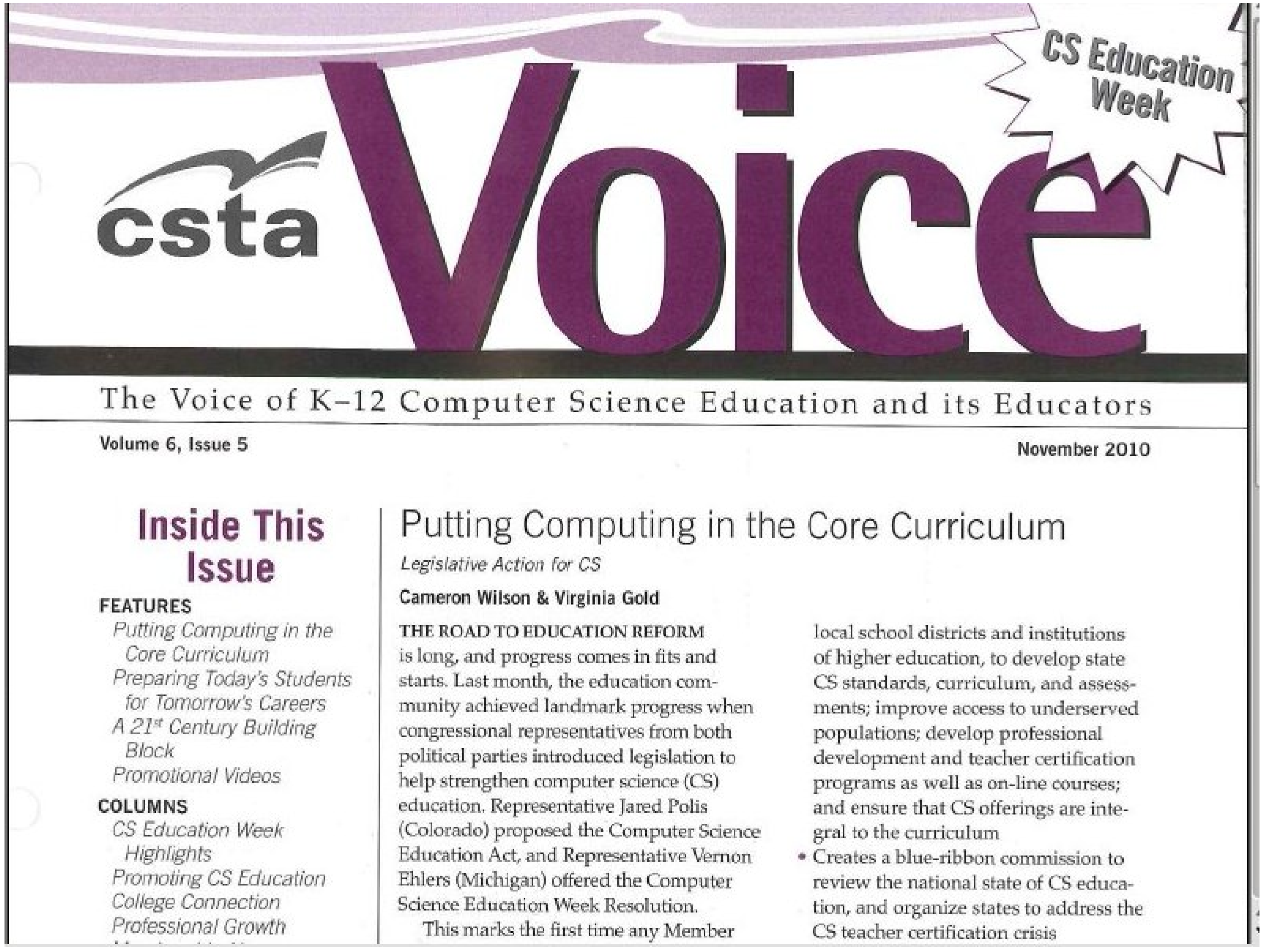
- Switzerland: only since two years CS is introduced as an elective. ICTswitzerland[a] and SVIA[b] push for a mandatory subject equivalent to mathematics, physics or chemistry.
- Germany: GI [c] and BITKOM [d] observe
  - CS as an Interdisciplinary technology is of special importance because it advances innovation in many other disciplines
  - We demand therefore in the curriculum of the schools CS for all students as an independent subject.
  - In high school the subjects biology, chemistry, computer science and physics have to be offered equivalently.

---

[a] umbrella organization of the computer science and telecommunication sector
[b] Computer Science Teacher Association
[c] Gesellschaft für Informatik
[d] Bundesverband Informationswirtschaft, Telekommunikation und neue Medien

The CSTA Voice is a bi-monthly publication for members of the Computer Science Teachers Association

http://csta.acm.org/Communications/sub/CSTAVoice.html



csta Voice

CS Education Week

The Voice of K–12 Computer Science Education and its Educators

Volume 6, Issue 5

November 2010

Inside This Issue

FEATURES
- *Putting Computing in the Core Curriculum*
- *Preparing Today's Students for Tomorrow's Careers*
- *A 21st Century Building Block*
- *Promotional Videos*

COLUMNS
- *CS Education Week Highlights*
- *Promoting CS Education*
- *College Connection*
- *Professional Growth*

Putting Computing in the Core Curriculum

Legislative Action for CS

Cameron Wilson & Virginia Gold

THE ROAD TO EDUCATION REFORM is long, and progress comes in fits and starts. Last month, the education community achieved landmark progress when congressional representatives from both political parties introduced legislation to help strengthen computer science (CS) education. Representative Jared Polis (Colorado) proposed the Computer Science Education Act, and Representative Vernon Ehlers (Michigan) offered the Computer Science Education Week Resolution.

This marks the first time any Member

local school districts and institutions of higher education, to develop state CS standards, curriculum, and assessments; improve access to underserved populations; develop professional development and teacher certification programs as well as on-line courses; and ensure that CS offerings are integral to the curriculum

- Creates a blue-ribbon commission to review the national state of CS education, and organize states to address the CS teacher certification crisis

A surgeon of 1900 would not recognize anything in todays operating room. A mathematics teacher of 1900 in todays classroom would just continue teaching the same way. *TED*
NICHOLAS NEGROPONTE, *MIT Media Lab*

.

Change Curriculum?

- Very hard to realize, many excuses. Most frequent is: "We cannot add more new material"
- Good way is to redefine the necessary knowledge and skills needed in the 21 century
- Done in France by Ministry of Education:
  "Le socle commun des connaissances et des compétences",
  Tout ce qu'il est indispensable de maîtriser à la fin de la scolarité obligatoire.[a] (Décret du 11 juillet 2006)
- Such a decree is not possible everywhere, especially not in Switzerland

[a] The common knowledge and skills, the essentials to master at the end of compulsory education.

Progress in USA: CS Education Act

www.acm.org/press-room/news-releases/2010/cs-ed-act

Landmark progress July 30, 2010: congressional representatives from both political parties introduced legislation to strengthen CS education

- Defines CS education and its concepts to clarify the confusion of terms around K–12 CS education
- Establishes planing grants for 5 years implementation to develop CS standards curriculum, teachers certification programs and on-line courses
- Create blue-ribbon commission to review state of CS education and to address CS teacher certification crisis
- Establishes K–12 teacher preparation programs at institutions of higher education

CS: Fundamentals versus Application

- We all need to be able to use a computer ICT↔CS:David Braben
- However, we need to know more to understand todays world
- Computational Thinking

  Definition by Jan Cuny, Larry Snyder, and Jeannette M. Wing, Carnegie Mellon University, USA:

  > *Computational Thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent.*

  – CMU `www.cs.cmu.edu/~CompThink/`

- Also supported by Google

  `www.google.com/edu/computational-thinking/index.html`

If you don't understand the fundamentals then . . .

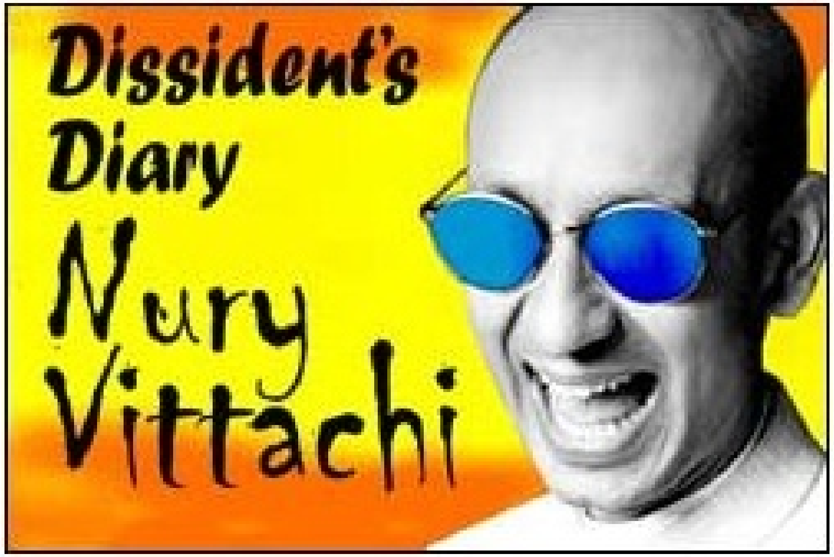
It may not be Dewali yet but I have to go pray to my laptop

Friday, January 28, 2011

Heard some shocking news about a businessman I know.

"Jahan decided not to sing to his account books this time," his wife said. "He chanted a series of verses to his laptop instead."

It's progress, I suppose. But it's also a reminder of the difference between East and West. Business people in Europe and North America never sing to their computers. (In Texas, they shoot the poor things!).



Computer

- originally: calculating machine
- today: information-processing machine for digital data:
  texts, pictures, music, speech . . .
- Properties
  - can store vast amounts of data
  - can compute extremely fast
  - can communicate with other computers
  - can be programmed for special tasks

The possibility of being programmed makes the computer a universal machine

Solving Problems with Computer

- analyze a task or problem, model and formulate it mathematically
- search for a way to solve it, find or design an algorithm
- program
- run the program: let the computer work, maybe correct, modify the program

Why is programming important for general education?

- creative and
- constructive activity work of engineers!
- teaches precise working and
- computational thinking

How do I get the computer to solve a problem?

- The algorithm has to be programmed.

  The single steps to be executed (like a recipe) have to be described in a language which the computer understands.

- The exists many programming languages e.g. FORTRAN, Algol, BASIC, Java, C, C++, C$^{\#}$, Ada, A$^{\#}$, Pascal, Matlab, Scilab, Python, Oberon, Eiffel, Maple, Mathematica . . .

- For each such language there exist compilers which translate the program in executable machine code for the specific computer.

- I will use in my examples Matlab. However, the language is not relevant for my goal to hopefully arouse your enthusiasm for programming!

- Problem: decide if a given number n is prime.
- Analysis: a number is prime if it has no divisors except 1 and itself
- Solution: Check if n cannot be factored by a smaller number
- Program:

```
function prim=primetest(n)
%  n is prime if prim=1
prim=1;                       % we are optimistic
k=2;                          % smallest possible divisor
while k<=n-1 & prim           % for all smaller numbers
   prim=rem(n,k) ~= 0;        % test if remainder nonzero
   k=k+1;
end
```

- Run the program:

```
>> [primetest(13), primetest(10)]
ans =
     1     0
```

All Primes up to a Number m

- Problem: Compute primes and store them as a list
- Analysis and Solution: test each number from 2 to m using `primetest` and store it if it is prime
- Program: `primes1`

```
function [p,hm]=primes1(m);
p=[];                % empty list
for n=2:m
  if primetest(n)  % if prime
     p=[p,n];        % then store in list
  end
end
hm=length(p);       % length of list
```

- Run the program:

```
>> [p,hm] = primes1(30)
p =
     2     3     5     7    11    13    17    19    23    29
hm =
    10
```

Creative Improvement!

- is it necessary to divide by all smaller numbers?
- No, it is sufficient to divide up to $\sqrt{n}$!

$$
\begin{array}{rccc}
 & & & \text{remainder} \\
29 = & 2 \times 14 & + & 1 \\
 & 3 \times 9 & + & 2 \\
 & 4 \times 7 & + & 1 \\
 & 5 \times 5 & + & 4 \\
 & 6 \times 4 & + & 5 \\
 & 7 \times 4 & + & 1 \\
 & 8 \times 3 & + & 5 \\
 & 9 \times 3 & + & 2 \\
 & 10 \times 2 & + & 9 \\
 & 11 \times 2 & + & 7 \\
 & 12 \times 2 & + & 5 \\
 & \cdots & & \\
 & 28 \times 1 & = & 1
\end{array}
$$

```
function prim=primetest2(n)
%  n is prime if prim=1
prim=1;                      % we are optimistic
k=2;                         % smallest possible divisor
while k<=sqrt(n) & prim      % for all numbers up to sqrt(n)
   prim=rem(n,k) ~= 0;       % test if remainder nonzero
   k=k+1;
end
```

since $\sqrt{29} = 5.3852$ test only up to 5

Even smarter

- not test for all smaller numbers up to $\sqrt{n}$ but only for already found smaller primes!
- We want to compute and store all primes up to m

```
function [p,hm]=primes2(m);
p=[];                      % empty list
for n=2:m                  % for each n test if prime
  prim=1;                  % optimistic
  wu=sqrt(n);              % upper bound
  for k=p                  % for all primes in list
    if k>wu, break,end     % which are < sqrt(n)
    prim=rem(n,k)~=0;      % test remainder
    if ~prim, break, end   % exit loop if not prime
  end
  if prim, p=[p,n]; end    % store if prime
end
hm=length(p);              % length of list
```

Different Approach, Other Algorithm Sieve of Eratosthenes

Analysis: a number prime if it is not a multiple of another number

Solution: cross out in the list of the numbers $2, 3, \ldots, m$ all multiples of $2, 3, \ldots$: the numbers left are the primes

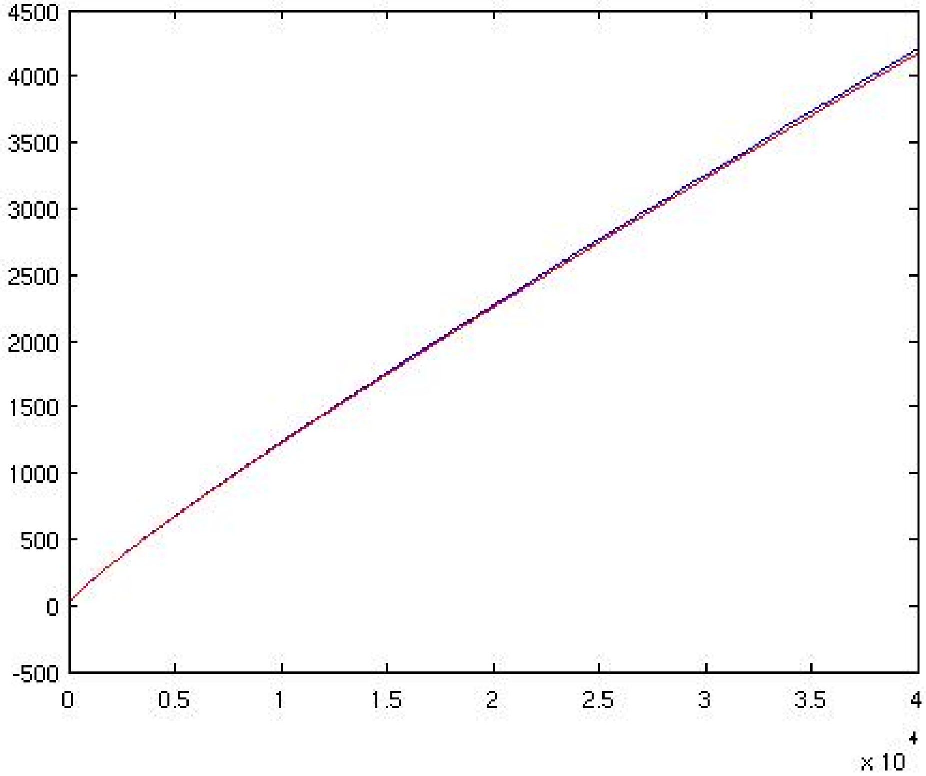
Program: t1.m

```
function [p,hm]=sieve(m);
s=1:m; s(1)=0;          % 1 is not prime
p=[]; k=2;              % 2 is first prime
while k<m
   p=[p,k];             % store prime
   j=k+k;               % first multiple
   while j<=m
      s(j)=0; j=j+k;    % cross out multiples
   end
   j=k+1;               % find next prime
   while (s(j)==0)&(j<m)
      j= j+1;
   end
   k=j;                 % k is prime
end
hm=length(p);
```

Prime Number Theorem (the second question)

- Gauss (1800), Hadamard, Vallée Poussin (1896) conjectured and proved for the prime-counting function $\pi(x) \sim \dfrac{x}{\ln x - 1}$
- Verification by computer:

```
% Prime Number Theorem
clear, clf
n=50000
[p,hm] = primes2(n);
%[p,hm] = sieve(n);
y=[];
x = [2:n];
for k = 2:n
   y = [y sum(p<k)];
end
plot(x,y)
hold
plot(x,x./(log(x)-1),'r')
```



Shipwrecked Sailors (First question, Quiz in American. Newspaper 1926)

- 5 sailors strand on an island, collect coconuts and want to divide them next day. Go to sleep.
- First sailor wakes up, divides the nuts, one is left for the monkey, hides his part, shuffles the leftover together, goes back to sleep.
- The same repeats with the other sailors.
- Next morning, no one makes a remark, they divide the pile again, and again one nut is left for the monkey.
- How many nuts did they collect?

Solution:

- 1926 solve diophantine equation.
- Today: brute force! Program the dividing process for nuts $n = 1, 2, 3, \ldots$ until a number is found which fulfills the conditions.

Program: Shipwrecked Sailors

```
function [n,parts]=nuts;
n=0;                                % initialize number of nuts
good=0;                             % boolean variable
while ~good
  n=n+1;                            % try with next n
  leftover=n;
  good=1;                           % optimistic
  i=0;
  while (i<5) & good                % try to divide for all sailors
    good=rem(leftover,5)==1;        % good if one nut remains
    if good,
      i=i+1;                        % count sailor
      parts(i)=fix(leftover/5);     % saylor i takes his part
      leftover =parts(i)*4;         % shuffles the leftover together
    end
  end
  good=good & (rem(leftover,5)==1);% next morning:one nut left for monkey
  parts=(leftover-1)/5+parts;       % add morning share to each sailor
end
```

Results

- ```
  >> [n,parts]=nuts
  n = 15621
  parts = 4147   3522   3022   2622   2302
  ```
- for the variant that no nut is leftover for the monkey in the morning we change

  ```
  good=good & (rem(leftover,5)==1); % next morning:one nut left for monkey
  parts=(leftover-1)/5+parts;       % add morning share to each sailor
  ```

  to

  ```
  good=good & (rem(leftover,5)==0); % next morning: no nut for monkey
  parts=leftover/5+parts;           % add morning share to each sailor
  ```

  and get

  ```
  >> [n,parts]=nuts
  n = 3121
  parts = 828   703   603   523   459
  ```
  ```

Sorting Algorithms

- Problem: The numbers 19 11 8 3 12 14 are to be sorted in numerical order.
- Solution: we look for the minimum value and swap it with the value in the first position (selection sort)

19	11	8	3	12	14	swap 19 and 3
3	|11	8	19	12	14	swap 8 and 11
3	8	|11	19	12	14	no swap
3	8	11	|19	12	14	swap 19 and 12
3	8	11	12	|19	14	swap 19 and 14
3	8	11	12	14	19	sorted!

Program Selection Sort

```
function a=minsort(a)
n=length(a);
for i=1:n-1                        % we need n-1 steps
  k=i;                             % assume a(k)is min
  for j=i+1:n
    if a(j)<a(k), k=j; end         % look for smaller element
  end
  if k~=i                          % swap if i~=k
    h=a(k); a(k)=a(i); a(i)=h;
    bar(a); pause(0.1)
  end
end
```

Bubble Sort

Sweep through numbers, compare pairs and swap adjacent numbers. Repeat sweeps until no swap occur anymore.

19	11	8	3	12	14	1. sweep
11	19	8	3	12	14	
11	8	19	3	12	14	
11	8	3	19	12	14	
11	8	3	12	19	14	
11	8	3	12	14	19	
11	8	3	12	14	19	2. sweep
8	11	3	12	14	19	
8	3	11	12	14	19	
8	3	11	12	14	19	3. sweep
3	8	11	12	14	19	sorted!

Program Bubble Sort

```
function a=bubble(a)
n=length(a);
done=0;                                  % boolean variable
while ~done
  done=1;                                % optimistic
  for k=1:n-1
    if a(k)>a(k+1),                      % if pair not ordered
      h=a(k); a(k)=a(k+1); a(k+1)=h;     % swap
      done=0;                            % needs another sweep
      bar(a); pause(0.01)
    end
  end
end
```

Quicksort

Ingenious, more complex (recursive) but very fast!

- choose a number in the middle of the sequence

  19 11 8 3 12 14

- look left for a number ≥ 8 and right for a number ≤ 8.
  Swap both numbers

  3 11 8 19 12 14

- repeat the process

  $\{3\}$ 8 $\{11$ 19 12 $14\}$

- we obtained two sets with all numbers ≤ 8 and ≥ 8
- apply the same procedure to the two sets (recursion)

Program Quicksort

sortieren(60)

```
function quick(left,right)
global a;
mid=fix((left+right)/2);          % choose middle element
i=left; j=right; x=a(mid);        % sort a(i) ... a(j)
while i<=j
  while a(i)<x, i=i+1; end        % search left  a(i)>=x
  while x<a(j), j=j-1; end        % search right a(j)<=x
   if i<=j                        % swap if found
     u=a(i); a(i)=a(j); a(j)=u;
     i=i+1; j=j-1;                % advance indices
     bar(a); pause(0.01)
   end
end                               % sort the two sets
if left<j, quick(left,j) ; end    % recursively
if i<right,quick(i,right); end
```

Programming with Recursion

Problem: prime factors of a number

Algorithm: search factor j of n, then search factor of $\frac{n}{j}$

Program: t2.m

```
function factors= factorize(n)
factors =[]; wu=sqrt(n);
j=1; remainder=1;
while j<=wu & remainder~=0              % search factor
  j=j+1; remainder=rem(n,j);
end
if remainder~=0, factors=[factors n];  % n is a prime, store
else factors=[factors j];              % j is a factor of n, store
     factors=[factors factorize(n/j)]; % recursion with n/j
end
```

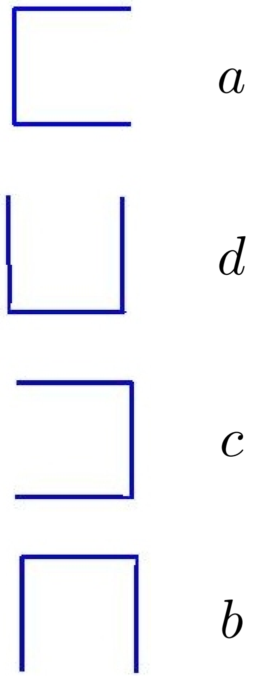
Example: `>> f = factorize(36284)`

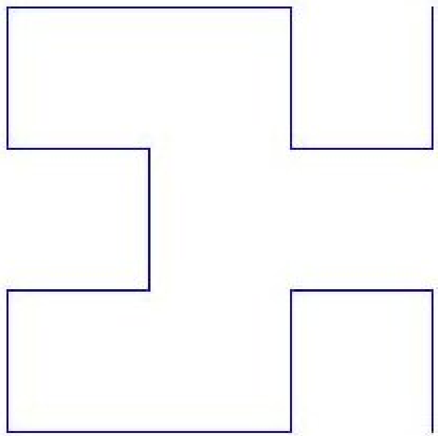
```
f =    2     2    47   193
```

Simple recursion when only one branch $=$ iteration

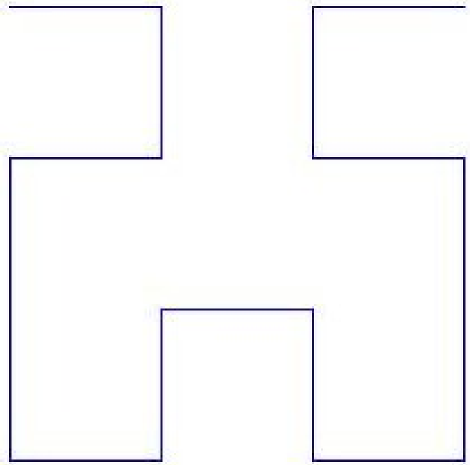
Genuine Recursion: Hilbert Curve

(N. Wirth: Algorithms + Data Structures = Programs)





. $a_2 : d \leftarrow a \downarrow a \rightarrow b$

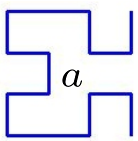


. $d_2 : a \downarrow d \leftarrow d \uparrow c$

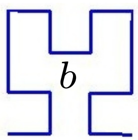
Programming the Four Cases

```
function a(i);
global x y h;
if i>0,
    d(i-1); plot([x-h,x],[y,y]); x=x-h;
    a(i-1); plot([x,x],[y-h,y]); y=y-h;
    a(i-1); plot([x,x+h],[y,y]); x=x+h;
    b(i-1);
end
function b(i);
global x y h;
if i>0,
    c(i-1); plot([x,x],[y,y+h]); y=y+h;
    b(i-1); plot([x,x+h],[y,y]); x=x+h;
    b(i-1); plot([x,x],[y-h,y]); y=y-h;
    a(i-1);
end
```

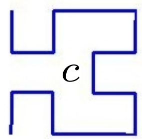
```
function c(i);
global x y h;
if i>0,
    b(i-1); plot([x,x+h],[y,y]); x=x+h;
    c(i-1); plot([x,x],[y,y+h]); y=y+h;
    c(i-1); plot([x-h,x],[y,y]); x=x-h;
    d(i-1);
end
function d(i);
global x y h;
if i>0,
    a(i-1); plot([x,x],[y-h,y]); y=y-h;
    d(i-1); plot([x-h,x],[y,y]); x=x-h;
    d(i-1); plot([x,x],[y,y+h]); y=y+h;
    c(i-1);
end
```



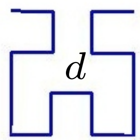
$a_2 : d \leftarrow a \downarrow a \rightarrow b$



$b_2 : c \uparrow b \rightarrow b \downarrow a$



$c_2 : b \rightarrow c \uparrow c \leftarrow d$



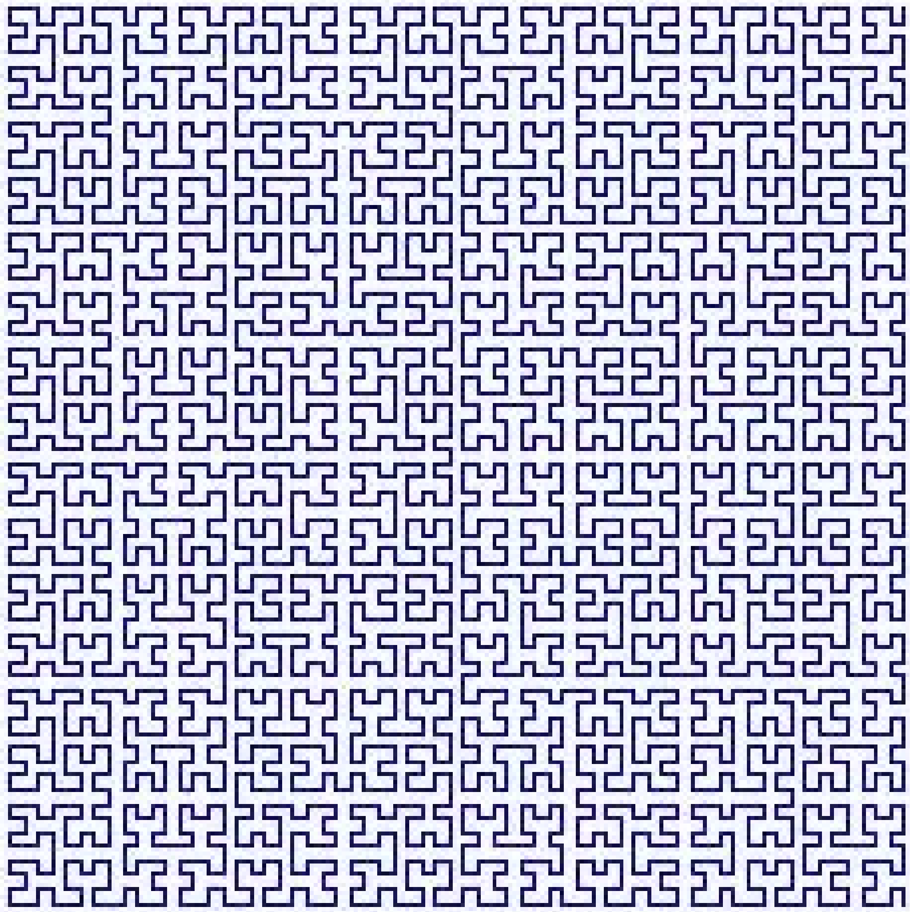
$d_2 : a \downarrow d \leftarrow d \uparrow c$

Hilbert Curve

$a(6)$

hilbert2.m

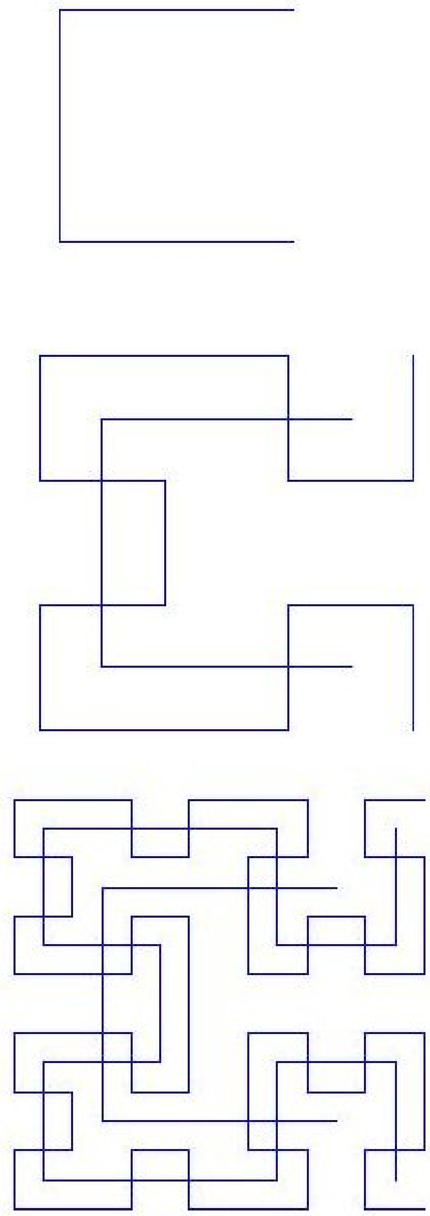
```
% HILBERT CURVES
global x y h;
h0=1024;
for n=1:6
  clf
  axis([-600,800,-600,800])
  axis square, hold
  x=600; y=600 ;
  h=h0/2^n; n
  a(n)
  pause(2)
end
```



Superposed Hilbert Curves

hilbert.m

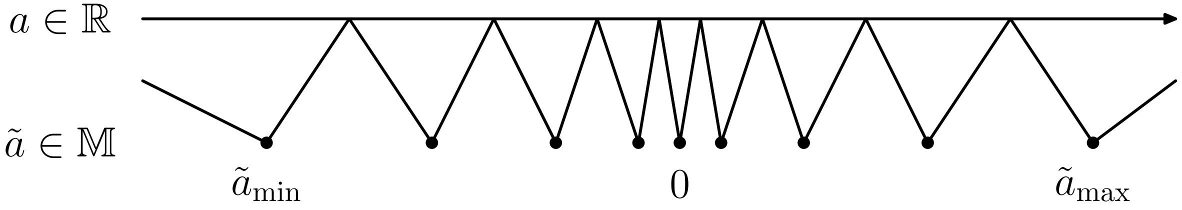
```
% Superposed Hilbert Curves
clear, clf
global x y h ;
h0=512;
n=5
axis([-600,800,-600,800])
axis square, hold
x0=h0/4; y0=h0/4; h=h0;
for i=1:n
  x0=x0 + h/2; y0=y0+h/2;
  x=x0; y=y0;
  a(i)
  h=h/2;
end
```



Number Representation in a Computer

real numbers $\leftrightarrow$ machine numbers

- mathematics: real numbers $\mathbb{R}$ = continuum
  every interval $(a,b) \in \mathbb{R}$ with $a < b$ contains ∞ set of numbers
- computer: finite machine, can only
  - store a finite set of numbers
  - perform a finite number of operations
- computer: the machine numbers $\mathbb{M}$ (finite set)
- mapping $\mathbb{R} \to \mathbb{M}$: a whole interval $\in \mathbb{R} \to \tilde{a} \in \mathbb{M}$:



IEEE Floating Point Standard (since 1985)

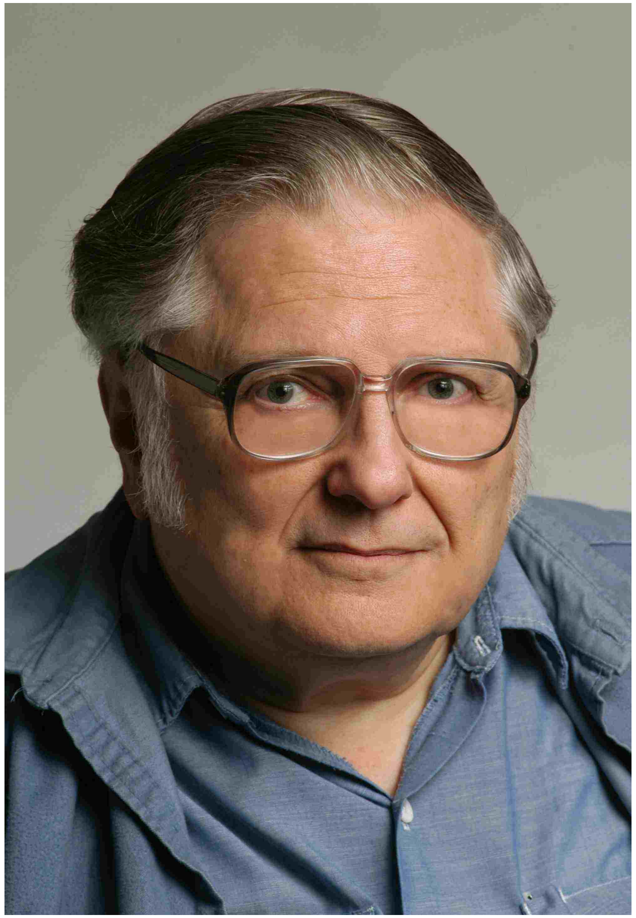
- a real number is represented as floating point number using 64 bits

$$\begin{array}{llll} S & \overbrace{EEEEEEEEEEE}^{e} & & \overbrace{FFFFF\cdots FFFFF}^{m} \\ 0 & 1 \qquad\qquad\qquad 11 & & 12 \qquad\qquad\qquad 63 \end{array}$$

- normal case: $0 < e < 2047, \quad (2^{11} = 2048)$

$$\tilde{a} = (-1)^S \times 2^{e-1023} \times 1.m$$

- realmin $= 2.2251 \cdot 10^{-308} = 2^{-1022}$
  realmax $= 1.7977 \cdot 10^{308}$
- underflow, overflow



William Kahan
(Father IEEE F.P.S.)

computer calculate on principle inaccurately!

Matlab program			results		
a	$=$	10	a	$=$	10
b	$=$	$a/7$	b	$=$	1.428571428571429
c	$=$	$sqrt(sqrt(sqrt(sqrt(b))))$	c	$=$	1.022542511383932
d	$=$	$exp(16 * log(c))$	d	$=$	1.428571428571427
e	$=$	$d * 7$	e	$=$	9.999999999999991
$a - e$			ans	$=$	$8.881784197001252e{-}15$

Rounding errors: machine precision=spacing of number in $(1, 2)$ is $\varepsilon = 2.22 \cdot 10{-}16$. For a basic operation $\otimes \in \{+, -, \times, /\}$ we have:

$$a \,\tilde{\otimes}\, b = (a \otimes b)(1 + \eta), \quad |\eta| < \varepsilon$$

study/controlling of rounding errors $\Rightarrow$ numerical analysis

Correct Results in Spite of Rounding Errors

- Example: computing the square root

$$x = \sqrt{a} \iff x^2 = a, \; x > 0$$

  using only the basic operations $\{+, -, \times, /\}$

- Method: Guess and correct. We want to find x such that

$$\frac{a}{x} = x$$

- Start with some initial value x_1, compute $\dfrac{a}{x_1}$

$$\text{if} \quad \frac{a}{x_1} \neq x_1 \quad \text{take the mean} \quad x_2 = \frac{1}{2}\left(x_1 + \frac{a}{x_1}\right)$$

- Iterate and obtain sequence $\{x_k\}$ converging to $\sqrt{a}$

$\sqrt{20} = ?$

approximation	divide		take mean
4	$\dfrac{20}{4} = 5$	larger	$4.5 = \dfrac{4+5}{2}$
4.5	$\dfrac{20}{4.5} = 4.4444$	smaller	$4.4722 = \dfrac{4.5+4.4444}{2}$
4.4722	$\dfrac{20}{4.4722} = 4.4721$	smaller	$\ldots$

- sequence $x_k \to \sqrt{a}$ as $k \to \infty$

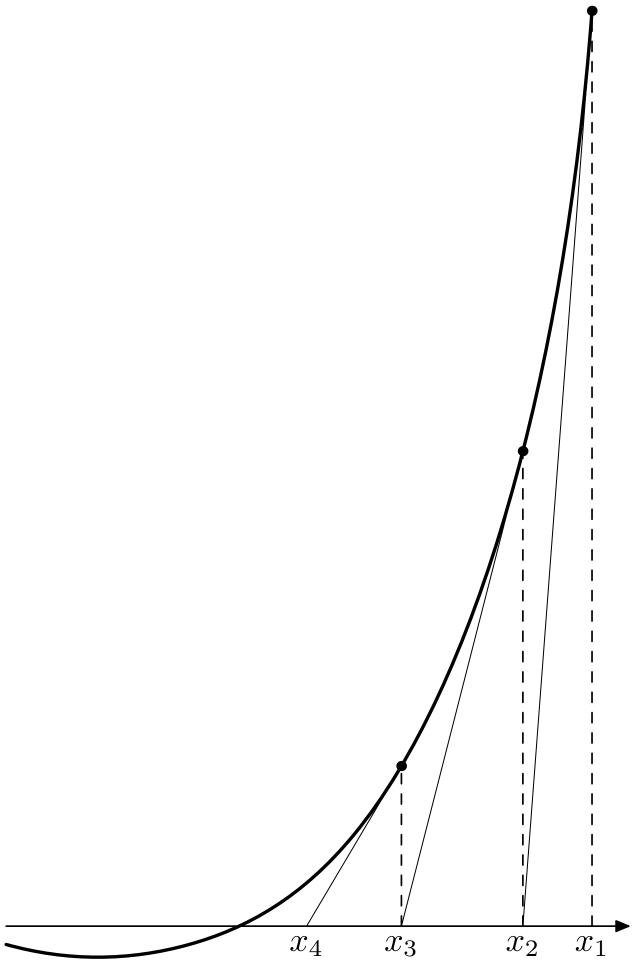
$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{a}{x_k}\right) \quad \text{Heron}$$

- initial value? termination criterion?

Alternative Derivation of Heron's Iteration Solve $f(x) = x^2 - a = 0$ with Newton's method

$$\begin{aligned} x_{k+1} &= x_k - \frac{f(x_k)}{f'(x_k)} \\ f(x) &= x^2 - a, \quad f'(x) = 2x \\ \Rightarrow x_{k+1} &= x_k - \frac{x_k^2 - a}{2x_k} \\ &= \frac{1}{2}\left(x_k + \frac{a}{x_k}\right) \end{aligned}$$

if $x_1 > \sqrt{a}$ then monotonous convergence: $\sqrt{a} < \cdots < x_2 < x_1$



Program mysqrt with Smart Termination

Stop iteration when monotonicity is violated!

```
function xnew=mysqrt(a);
% computes w=sqrt(a) using Heron's algorithm
xold=(1+a)/2;              % start > sqrt(a)
xnew=(xold+a/xold)/2;      % first iterate
while xnew<xold            % if monotone
  xold=xnew;               % iterate
  xnew=(xold+a/xold)/2;
end
>> a= 12345.654321;
>> RelErr=(sqrt(a)-mysqrt(a))/sqrt(a)
RelErr = 1.2790e-16
```

Relative error is smaller than machine precision $\varepsilon = 2.22 \cdot 10^{-16}$

Summary: Why is Programming FUN?

- Creative activity (inventing algorithms is fascinating)
- Constructive – one designs and constructs a machine (in software) which then can be run.
- Programming trains accuracy and discipline – good programs are elegant and aesthetical.
- When programming, the student is active not a passive consumer.
- Debugging programs is often an interesting detective work.
- Programming has a playful component: teach a machine to do something.
- Multidisciplinary: when programming, one gets to know applications in various disciplines.

Fred Brooks: Mythical Man Month (1974)

Why is programming fun? What delights may its practioner expect as his reward?

First is the sheer joy of making things. As the child delights in his mud pie, so the adult enjoys building things, especially things of his own design. I think this delight must be an image of God's delight in making things, a delight shown in the distinctness and newness of each leaf and each snowflake.

Second is the pleasure of making things that are useful to other people. Deep within, we want others to use our work and to find it helpful. In this respect the programming system is not essentially different from the child's first clay pencil holder "for Daddy's office."

Third is the fascination of fashioning complex puzzle-like objects of interlocking moving parts and watching them work in subtle cycles, playing out the consequences of principles built in from the beginning. The programmed computer has all the fascination of the pinball machine or the jukebox mechanism, carried to the ultimate.

Fourth is the joy of always learning, which springs from the nonrepeating nature of the task. In one way or another the problem is ever new, and its solver learns something: sometimes practical, sometimes theoretical, and sometimes both.

Finally, there is the delight of working in such a tractable medium. The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures.