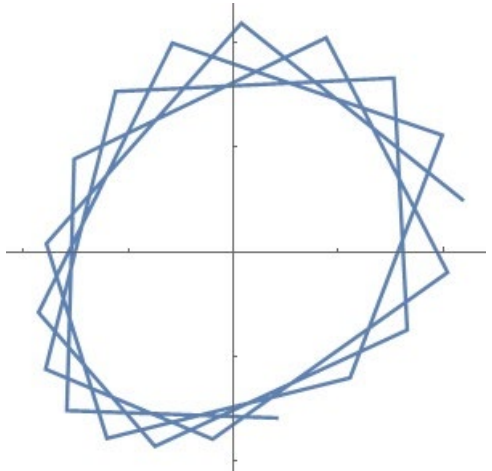
What is Complexity?

Notes for CAS, 28-4-2020

Loe Feijs



What is complexity and how to measure complexity?

- Computational complexity

  - ✓ practice: example in Processing
  - ✓ theory: big-O notation
  - ✓ theory: NP-complete problems

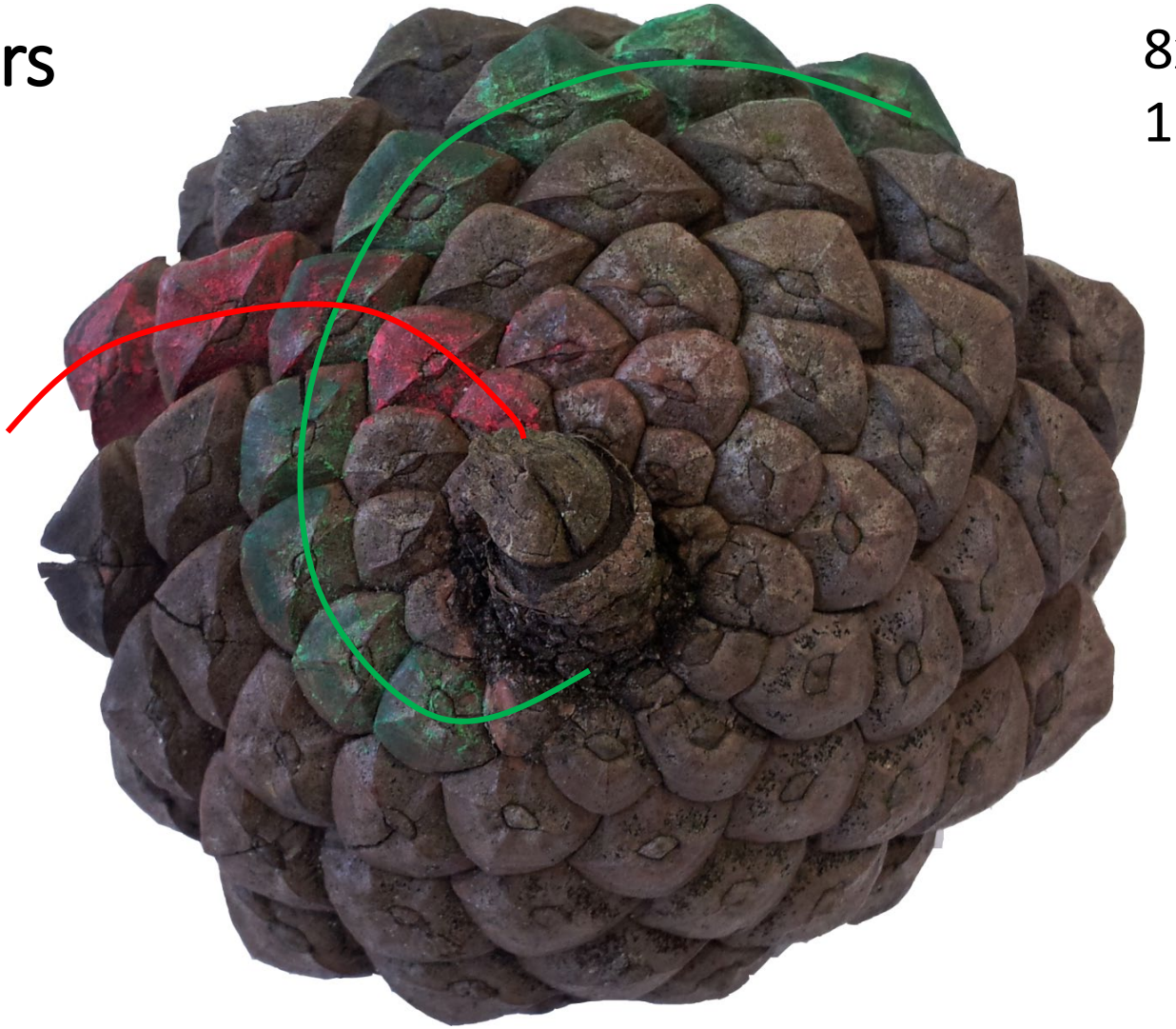
- Information theory complexity

  - ✓ Shannon’s aproach:
  - ✓ Kolmogorov’s approach:

Example:
computing Fibonacci numbers

1
1
2
3
5
8
13
21
34

Each number is the sum of the previous two numbers



8x (green)
13x (red)

naive | Processing 2.2.1

File Edit Sketch Tools Help

naive

```
//fibonacci numbers using recursion
//Loe Feijs and TU/e for CAS 2020
long fibonacci(long n){
     if (n <= 1)
        return 1;
        else return fibonacci(n - 1) + fibonacci(n - 2);
}

int n;
long t;
void setup(){
     n = 0;
}
void draw(){
     n++;
     t = System.currentTimeMillis();
     long fibo = fibonacci(n);
     long time = System.currentTimeMillis() - t;
     println("input, output, time: " + n + ", " + fibo + ", " + time);
}
```

Done Saving.

22

naive | Processing 2.2.1

File Edit Sketch Tools Help

Stop

Java

naive

```
//fibonacci numbers using recursion
//Loe Feijs and TU/e for CAS 2020
long fibonacci(long n){
     if (n <= 1)
        return 1;
```

```
input, output, time: 1, 1, 0
input, output, time: 2, 2, 0
input, output, time: 3, 3, 0
input, output, time: 4, 5, 0
input, output, time: 5, 8, 0
input, output, time: 6, 13, 0
input, output, time: 7, 21, 0
input, output, time: 8, 34, 0
input, output, time: 9, 55, 0
input, output, time: 10, 89, 0
input, output, time: 11, 144, 0
input, output, time: 12, 233, 0
input, output, time: 13, 377, 0
input, output, time: 14, 610, 0
input, output, time: 15, 987, 0
input, output, time: 16, 1597, 0
input, output, time: 17, 2584, 0
input, output, time: 18, 4181, 0
input, output, time: 19, 6765, 0
input, output, time: 20, 10946, 0
input, output, time: 21, 17711, 0
input, output, time: 22, 28657, 0
input, output, time: 23, 46368, 0
```

5

naive | Processing 2.2.1

File Edit Sketch Tools Help

Stop

Java

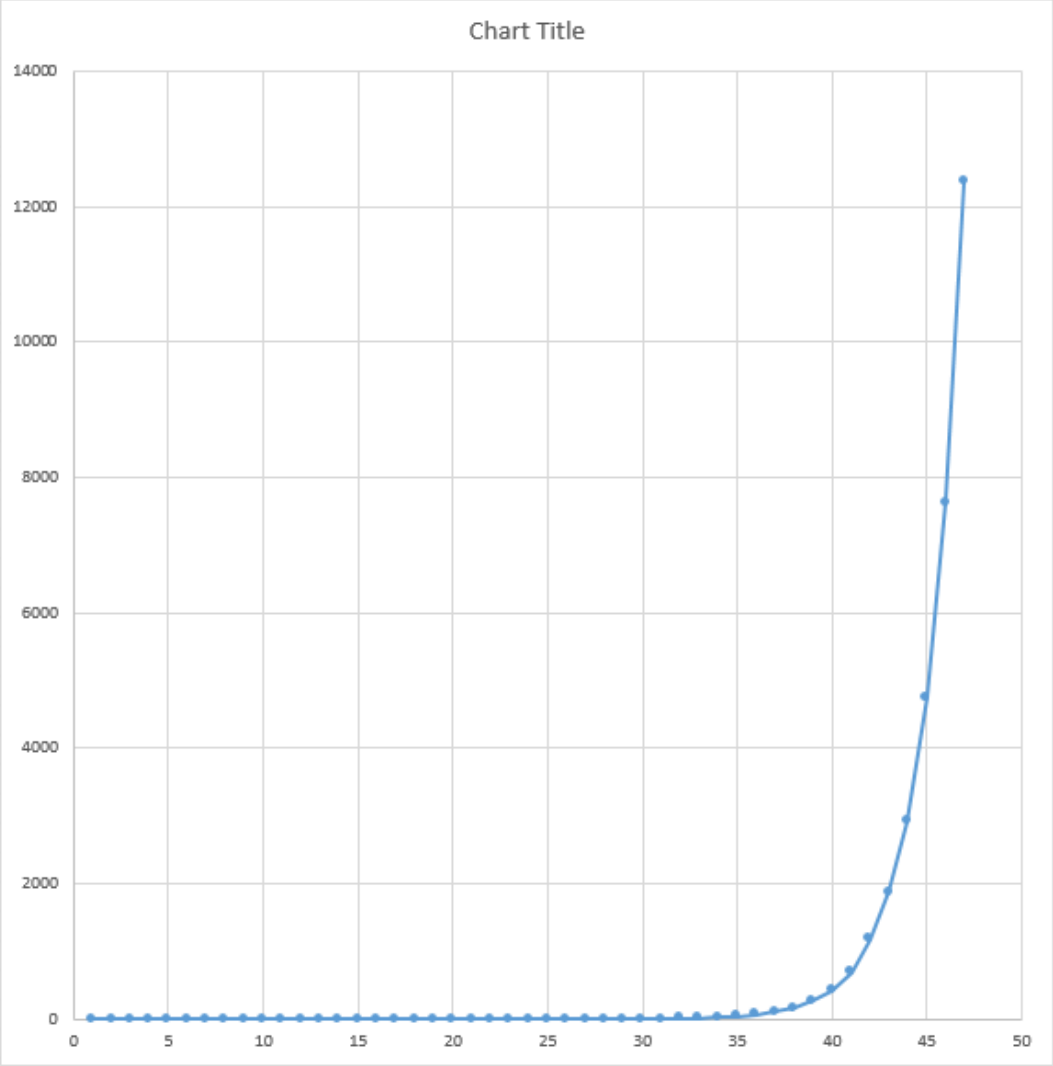
naive

```
//fibonacci numbers using recursion
//Loe Feijs and TU/e for CAS 2020
long fibonacci(long n){
     if (n <= 1)
        return 1;
```

```
input, output, time: 25, 121393, 0
input, output, time: 26, 196418, 0
input, output, time: 27, 317811, 1
input, output, time: 28, 514229, 1
input, output, time: 29, 832040, 2
input, output, time: 30, 1346269, 3
input, output, time: 31, 2178309, 5
input, output, time: 32, 3524578, 9
input, output, time: 33, 5702887, 15
input, output, time: 34, 9227465, 24
input, output, time: 35, 14930352, 41
input, output, time: 36, 24157817, 60
input, output, time: 37, 39088169, 98
input, output, time: 38, 63245986, 166
input, output, time: 39, 102334155, 262
input, output, time: 40, 165580141, 436
input, output, time: 41, 267914296, 682
input, output, time: 42, 433494437, 1164
input, output, time: 43, 701408733, 1882
input, output, time: 44, 1134903170, 3043
input, output, time: 45, 1836311903, 4778
input, output, time: 46, 2971215073, 7936
input, output, time: 47, 4807526976, 12856
```

5

	A	B	C	D	E
1	input	output	time: 1	1	0
2	input	output	time: 2	2	0
3	input	output	time: 3	3	0
4	input	output	time: 4	5	0
5	input	output	time: 5	8	0
6	input	output	time: 6	13	0
7	input	output	time: 7	21	0
8	input	output	time: 8	34	0
9	input	output	time: 9	55	0
10	input	output	time: 10	89	0
11	input	output	time: 11	144	0
12	input	output	time: 12	233	0
13	input	output	time: 13	377	0
14	input	output	time: 14	610	0
15	input	output	time: 15	987	0
16	input	output	time: 16	1597	0
17	input	output	time: 17	2584	0
18	input	output	time: 18	4181	0
19	input	output	time: 19	6765	0
20	input	output	time: 20	10946	0
21	input	output	time: 21	17711	0
22	input	output	time: 22	28657	0
23	input	output	time: 23	46368	0
24	input	output	time: 24	75025	0
25	input	output	time: 25	121393	0
26	input	output	time: 26	196418	0
27	input	output	time: 27	317811	1
28	input	output	time: 28	514229	1
29	input	output	time: 29	832040	2
30	input	output	time: 30	1346269	4
31	input	output	time: 31	2178309	6
32	input	output	time: 32	3524578	9
33	input	output	time: 33	5702887	15
34	input	output	time: 34	9227465	27
35	input	output	time: 35	14930352	41
36	input	output	time: 36	24157817	65
37	input	output	time: 37	39088169	107
38	input	output	time: 38	63245986	164
39	input	output	time: 39	102334155	266
40	input	output	time: 40	165580141	425
41	input	output	time: 41	267914296	688
42	input	output	time: 42	433494437	1185
43	input	output	time: 43	701408733	1867
44	input	output	time: 44	1134903170	2924
45	input	output	time: 45	1836311903	4745
46	input	output	time: 46	2971215073	7623
47	input	output	time: 47	4807526976	12365



smart | Processing 2.2.1

File Edit Sketch Tools Help

smart

```
//fibonacci numbers using memoization
//Loe Feijs and TU/e for CAS 2020
long fibonacci(int n){
     if (memo[n] > 0)
        return memo[n];
        else if (n <= 1)
                return 1;
                else return memo[n] = fibonacci(n - 1) + fibonacci(n - 2);
}

int n;
long t;
long[] memo;
void setup(){
     n = 0;
     memo = new long[100];
     for (int i = 0; i < 100; i++)
          memo[i] = 0;
}
void draw(){
     if (n < 47){
         n++;
         t = System.currentTimeMillis();
         long fibo = fibonacci(n);
         long time = System.currentTimeMillis() - t;
         println("input, output, time: " + n + ", " + fibo + ", " + time);
     }
}
```

smart | Processing 2.2.1

File Edit Sketch Tools Help

Stop

Java

smart

```
//fibonacci numbers using memoization
//Loe Feijs and TU/e for CAS 2020
long fibonacci(int n){
     if (memo[n] > 0)
        return memo[n];
```

```
input, output, time: 1, 1, 0
input, output, time: 2, 2, 0
input, output, time: 3, 3, 0
input, output, time: 4, 5, 0
input, output, time: 5, 8, 0
input, output, time: 6, 13, 0
input, output, time: 7, 21, 0
input, output, time: 8, 34, 0
input, output, time: 9, 55, 0
input, output, time: 10, 89, 0
input, output, time: 11, 144, 0
input, output, time: 12, 233, 0
input, output, time: 13, 377, 0
input, output, time: 14, 610, 0
input, output, time: 15, 987, 0
input, output, time: 16, 1597, 0
input, output, time: 17, 2584, 0
input, output, time: 18, 4181, 0
input, output, time: 19, 6765, 0
input, output, time: 20, 10946, 0
input, output, time: 21, 17711, 0
input, output, time: 22, 28657, 0
input, output, time: 23, 46368, 0
```

21

smart | Processing 2.2.1

File Edit Sketch Tools Help

smart

```
//fibonacci numbers using memoization
//Loe Feijs and TU/e for CAS 2020
long fibonacci(int n){
     if (memo[n] > 0)
        return memo[n];
```

Done Saving.

```
input, output, time: 23, 46368, 0
input, output, time: 24, 75025, 0
input, output, time: 25, 121393, 0
input, output, time: 26, 196418, 0
input, output, time: 27, 317811, 0
input, output, time: 28, 514229, 0
input, output, time: 29, 832040, 0
input, output, time: 30, 1346269, 0
input, output, time: 31, 2178309, 0
input, output, time: 32, 3524578, 0
input, output, time: 33, 5702887, 0
input, output, time: 34, 9227465, 0
input, output, time: 35, 14930352, 0
input, output, time: 36, 24157817, 0
input, output, time: 37, 39088169, 0
input, output, time: 38, 63245986, 0
input, output, time: 39, 102334155, 0
input, output, time: 40, 165580141, 0
input, output, time: 41, 267914296, 0
input, output, time: 42, 433494437, 0
input, output, time: 43, 701408733, 0
input, output, time: 44, 1134903170, 0
input, output, time: 45, 1836311903, 0
input, output, time: 46, 2971215073, 0
input, output, time: 47, 4807526976, 0
```

30

Theory: Big-O notation

- Two programs:
  - ✓ naive.pde (compute the nth Fibonacci number)
  - ✓ smart.pde (idem)

- Big-O notation characterizes a function according to its growth rate
  - ✓ write $T_1(n)$ for the execution time of the first program, given n

- $T_1(n) = O(2^n)$
  - ✓ meaning that for sufficiently large n
    $T_1(n)$ does not grow faster than 2^n
  - ✓ formally:
    there exist constants C and N such that
    $T_1(n) \leq C \times 2^n$ for all $n > N$

Recommended reading:
http://web.mit.edu/16.070/www/lecture/big_o.pdf

Fact: $T_1(n) = O(2^n)$

Stronger statement: $T_1(n) = O(1.63^n)$

What about the second program?

Fact: $T_2(n) = O(n)$

which is a significant improvement

Typical complexities:

$O(1)$	constant
$O(n)$	linear
$O(n^2)$	quadratic
$O(n^c)$	polynomial
$O(c^n)$	exponential

Recommended reading:
http://web.mit.edu/16.070/www/lecture/big_o.pdf

0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
1
1
2
4
6
9
15
27
41
65
107
164
266
425
688
1185
1867
2924
4745
7623
12365

12365 / 7623 = 1.6221

NP-complete problems

- A problem is a well-defined program specification
  - ✓ compute the nth Fibonacci number
  - ✓ sort any given list of *n* numbers
  - ✓ find a shortest path from a to b in any network of *n* nodes (distances given)
  - ✓ find a shortest return path along all *n* nodes in a network (distances given)

https://commons.wikimedia.org/wiki/File:TSP_Deutschland_3.png

The last problem is called the "Travelling Salesman Problem"

Theory:

it belongs to the class of *NP-complete* problems

for which any solution program's execution time is $O(c^n)$

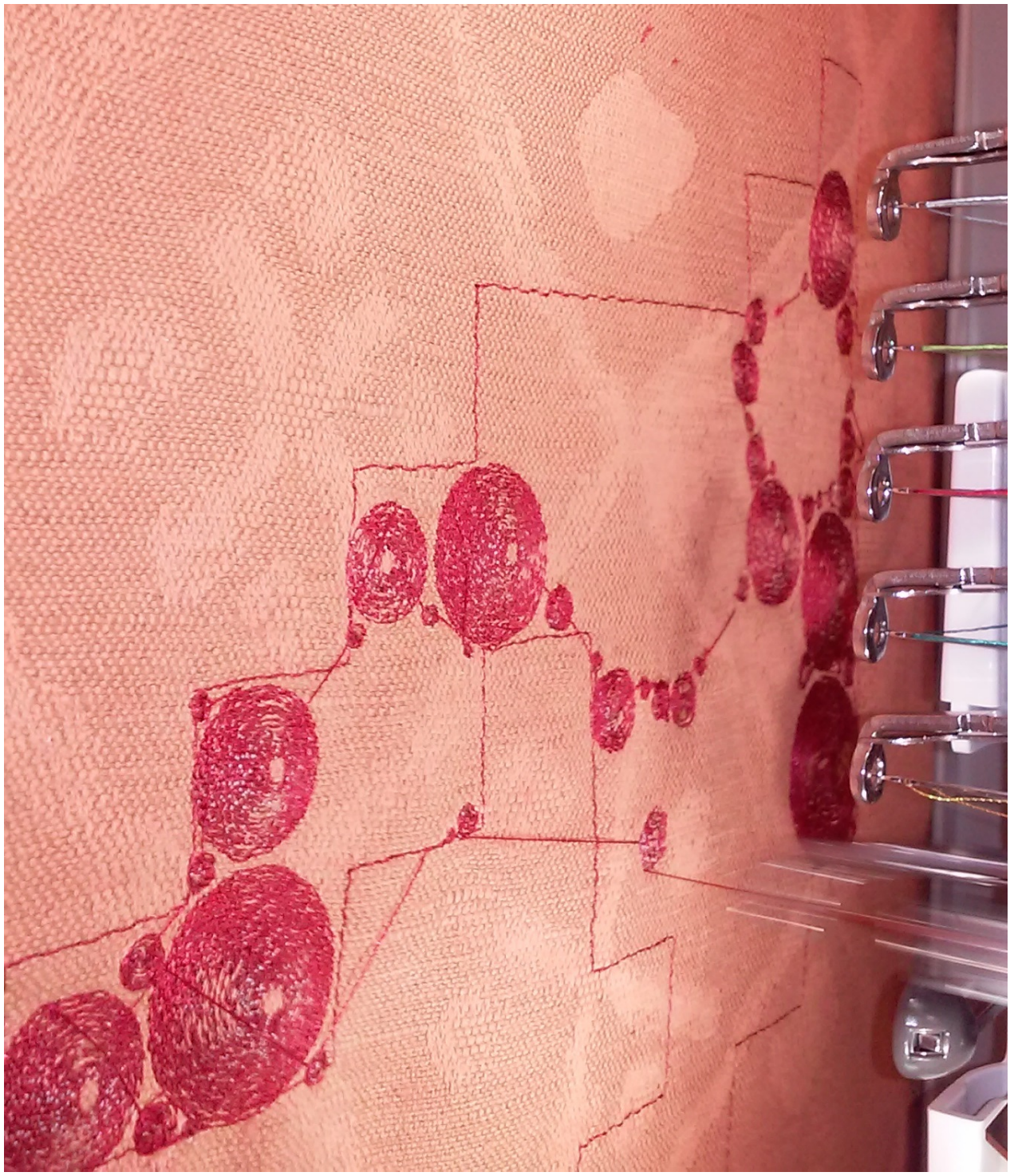
whether polynomial solution programs exist is an open problem

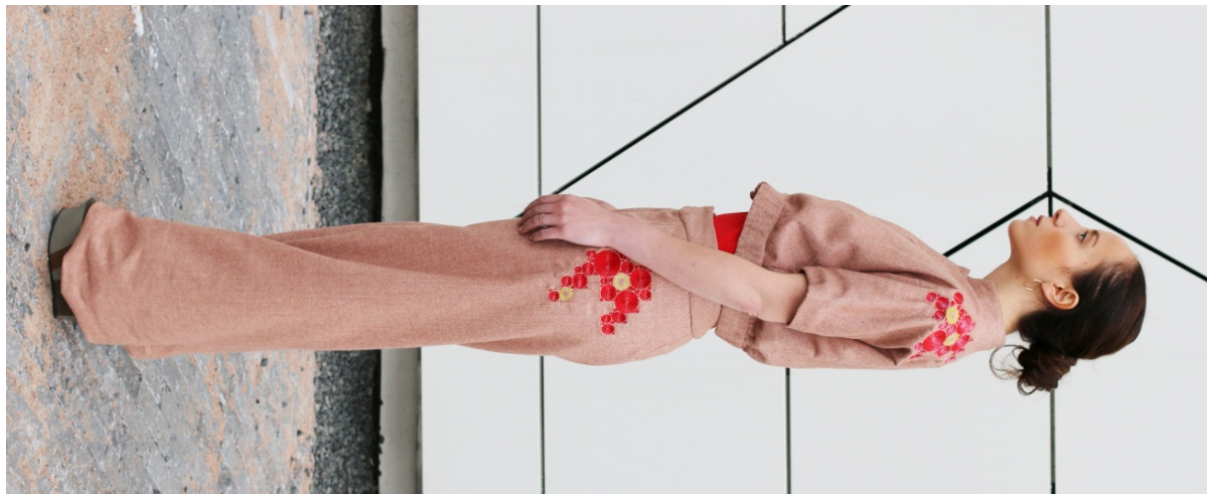
solving this "P=NP" problem deserves a $1,000,000 Millenium-prize

Practice:

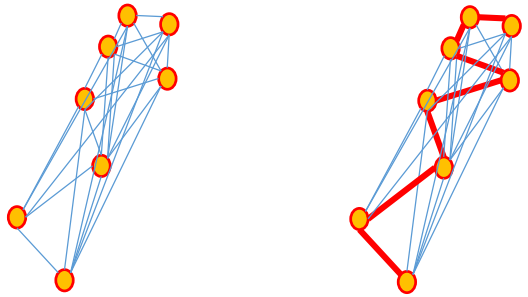
find good-enough approximations, not shortest

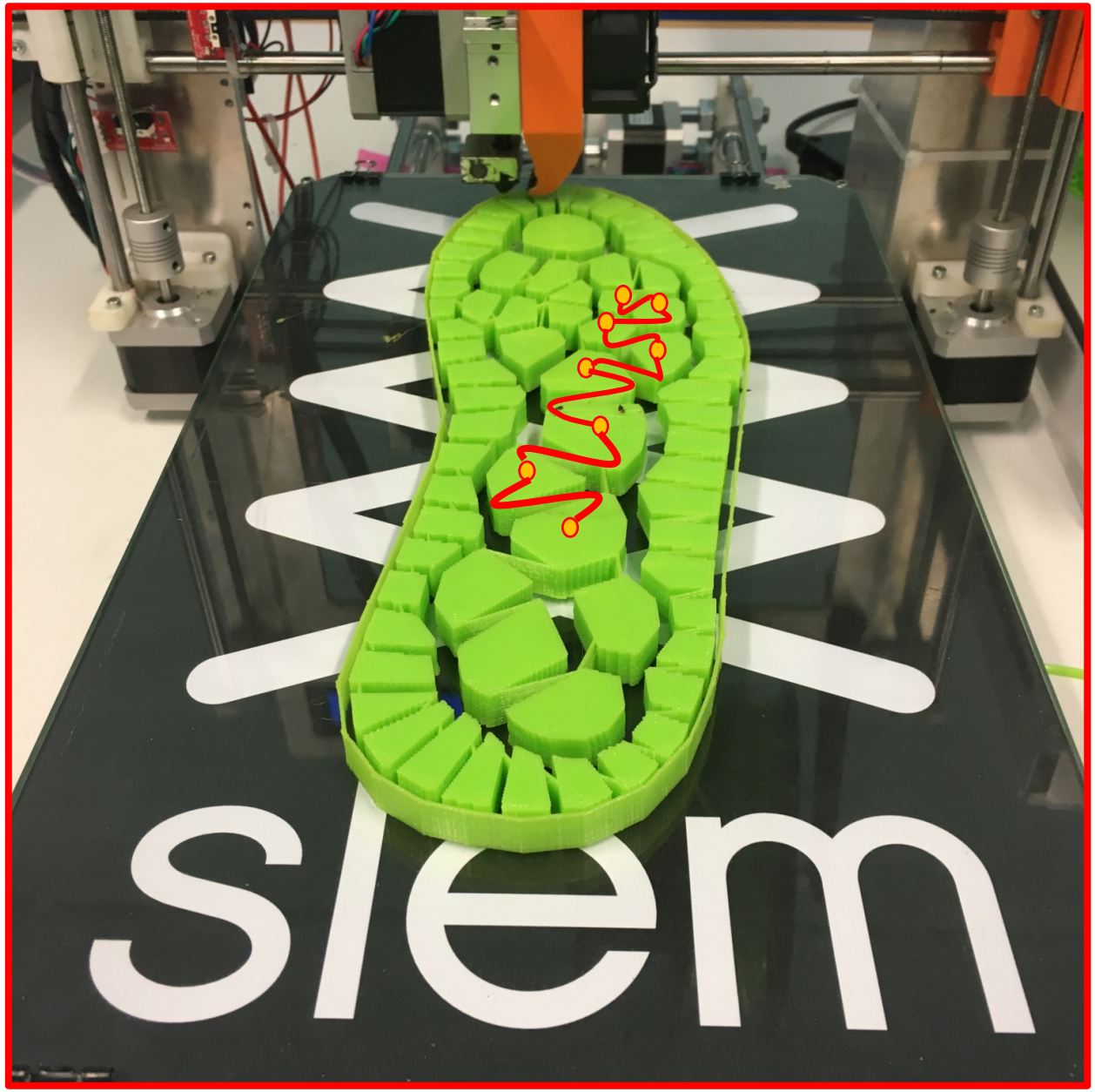
these problems pop-up in design, e.g. our WS lab





Pied de Pulse, Concept & Design: Loe Feijs & Marina Toeters, Technology: Anna Pagano & Geert van den Boomen, Embroidery: Anna Pagano & Lucia van den Hoven, Styling: Giorgia Presti, Photos & Video: Flora Macleod, Model: Anna Pagano, More info: by-wire.net/pied-de-pulse/





Solemaker.io was created by Troy Nachtigall - Head Designer, Loe Feijs, Stephan Wensveen, Oscar Tomico, Admar Shoonen, Bart Pruijmboom, Henry Lin, Erwin Hoogerwoord, Fiore Basile, Max Pirsky, Bart van Overbeeke, Sigrídur Helga Hauksdóttir

Computational complexity: an overview

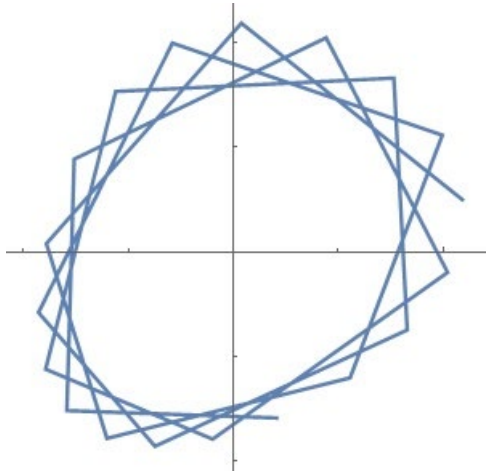
- Each program has a run-time complexity
  - ✓ naive fibonacci: exponential
  - ✓ smart Fibonacci: linear

- Each problem belongs to a complexity class, for example:
  - ✓ problems having polynomial solutions (e.g. Fibonacci)
  - ✓ problems having exponential solutions, no polynomial (if P≠NP)
  - ✓ problems having no solutions at all (e.g. the halting problem)

Information entropy as a measure for complexity

Notes for CAS

Feijs, 2019-2020



How to measure complexity?

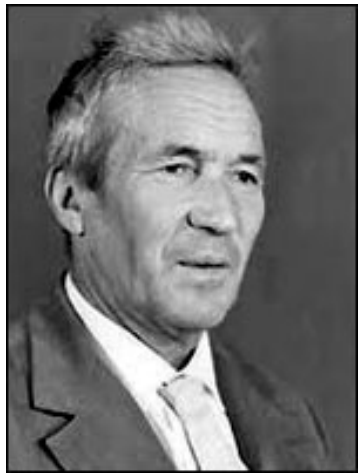
- Shannon's aproach:
  assume a source which produces random messages,
  how many bits do we need on average to code a message?

  A Mathematical Theory of Communication", Bell System Technical Journal, vol. 27, pp. 379-423 & 623-656, 1948

Claude Shannon in 1948
Source: www.i-programmer.info

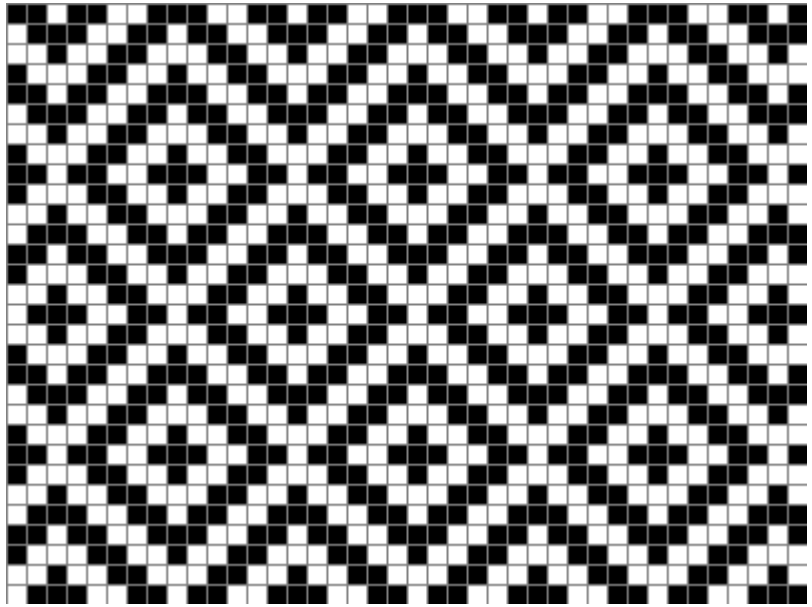
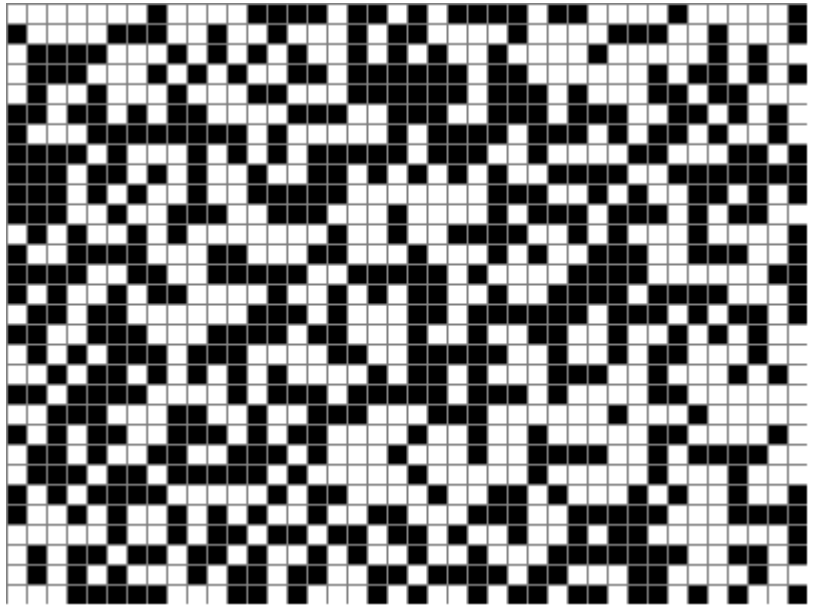
- Kolmogorov's approach:
  assume a fixed message
  (a tekst, or an image, for example),
  how long is the shortest program
  that will reproduce the message?

  Kolmogorov, A. (1968). Logical basis for information theory and probability theory. *IEEE Transactions on Information Theory*, *14*(5), 662-664.



Andrei Kolmogorov
Source: http://www.mi.ras.ru

Examples:

- “0100110101101001100100111001010010010110 01”
- “0000000000000100000000000000100000000000001”
- int tm(int i, int N){int im = i%(2*N); return im<N? im : (2*N)-im;}
- “a[i][j] = (tm(i,N)+1000+1 - tm(j-1,N+1))%4 <2? true : false;”
- 
- 

Shannon's aproach:

Assume source X
with alphabet {A,B,C,D}
and probabilities P(A)=0.5, P(B) = 0.25, P(C)= 0.125, P(D) = 0.125

Information per letter
$H(A) = -\,{}^{2}\log 0.5 = -(-1) = 1$ bit
$H(B) = -\,{}^{2}\log 0.25 = -(-2) = 2$ bit
$H(C) = -\,{}^{2}\log 0.125 = -(-3) = 3$ bit
$H(D) = -\,{}^{2}\log 0.125 = -(-3) = 3$ bit

Complexity of this source
$H(X) = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3 = 1\frac{3}{4} = 1.75$ bit per letter

Shannon's aproach:

Claim: we can code these letters in 1.75 bit (on average)

"Huffman coding"

A → 0

B → 10

C → 110

D → 111

Decoding: BABADACA was 10010011101100

DDDDDDDD was 111111111111111111111111

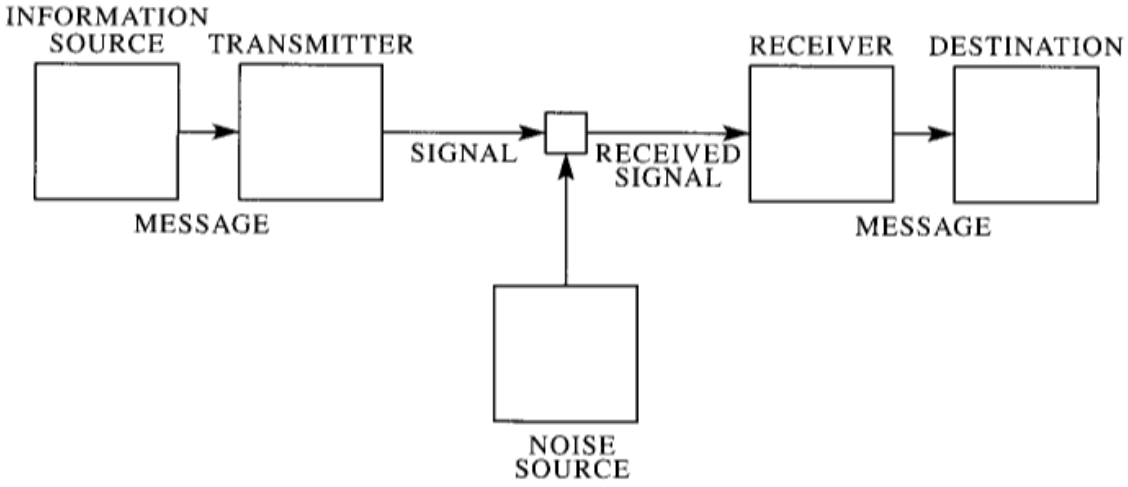


Fig. 1—Schematic diagram of a general communication system.

Source: A Mathematical Theory of Communication", Bell System Technical Journal, vol. 27, pp. 379-423 & 623-656, 1948

$$H(X) = \sum_i -p_i \, {}^2\log p_i$$

Special case:
two-letter alphabet

$P(A) = p_1 = p$

$P(B) = p_2 = (1 - p)$

$$H(X) = -p \, {}^2\log p - (1 - p) \, {}^2\log (1 - p)$$

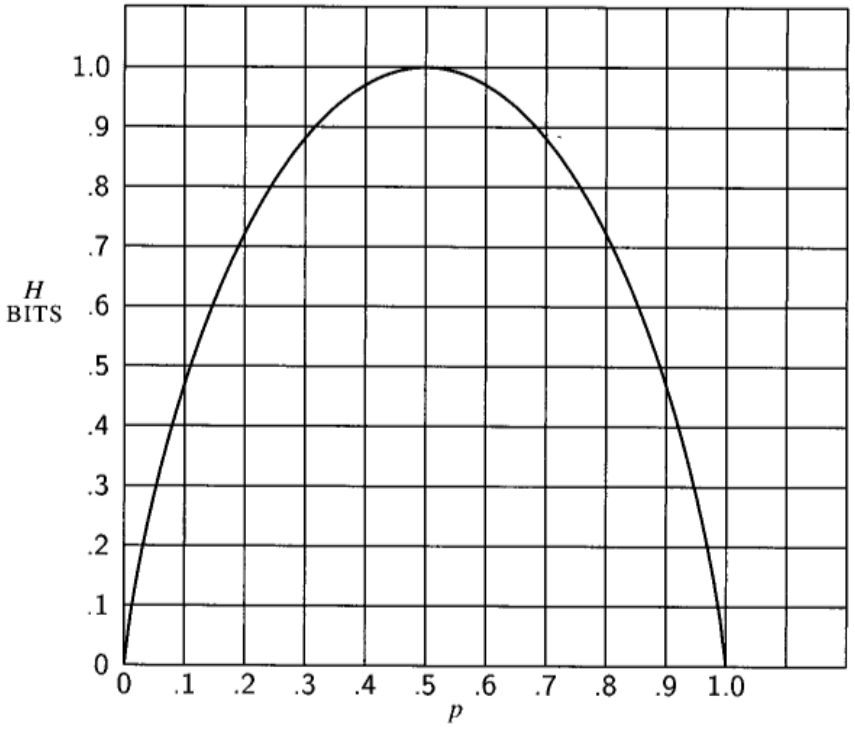


Fig. 7—Entropy in the case of two possibilities with probabilities p and $(1 - p)$.

Source: A Mathematical Theory of Communication", Bell System Technical Journal, vol. 27, pp. 379-423 & 623-656, 1948

III. The Series of Approximations to English

To give a visual idea of how this series of processes approaches a language, typical sequences in the approximations to English have been constructed and are given below. In all cases we have assumed a 27-symbol "alphabet," the 26 letters and a space.

1. Zero-order approximation (symbols independent and equiprobable).

   XFOML RXKHRJFFJUJ ZLPWCFWKCYJ FFJEYVKCQSGHYD QPAAMKBZAACIBZLHJQD.

2. First-order approximation (symbols independent but with frequencies of English text).

   OCRO HLI RGWR NMIELWIS EU LL NBNESEBYA TH EEI ALHENHTTPA OOBTTVA NAH BRL.

3. Second-order approximation (digram structure as in English).

   ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY ACHIN D ILONASIVE TUCOOWE AT TEASONARE FUSO TIZIN ANDY TOBE SEACE CTISBE.

4. Third-order approximation (trigram structure as in English).

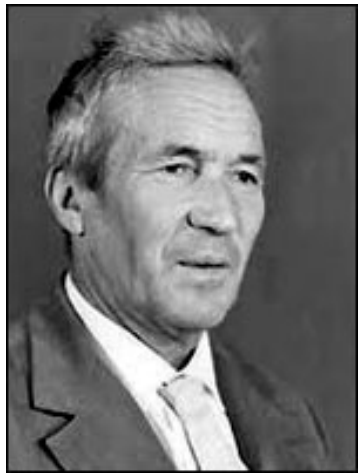
   IN NO IST LAT WHEY CRATICT FROURE BIRS GROCID PONDENOME OF DEMONSTURES OF THE REPTAGIN IS REGOACTIONA OF CRE.

Source: A Mathematical Theory of Communication", Bell System Technical Journal, vol. 27, pp. 379-423 & 623-656, 1948

Later, (1951, Shannon): English between .5 and 1.3 bit per char

How to measure complexity, 2nd approach?

- Kolmogorov's approach:
  assume a fixed message
  (a tekst, or an image, for example),
  how long is the shortest program
  that will reproduce the message?



Andrei Kolmogorov
Source: http://www.mi.ras.ru

Kolmogorov, A. (1968). Logical basis for information theory and probability theory. *IEEE Transactions on Information Theory*, *14*(5), 662-664.

Logical Basis for Information Theory and Probability Theory

ANDREI N. KOLMOGOROV

Abstract—**A new logical basis for information theory as well as probability theory is proposed, based on computing complexity.**

Section I

WE SHALL be concerned with the main basic concepts of information theory, beginning with the traditional concept of the conditional entropy of x when the value of y is known, $H(x \mid y)$, which can be interpreted as the quantity of information required for computing ("programming") the value x when the value y is already known. By using ϕ to denote a particular given known value, we get the unconditional entropy

$$H(x \mid \phi) = H(x).$$

Information given by y concerning the value of x can, as is well known, be expressed:

$$I(x \mid y) = H(x) - H(x \mid y).$$

It is evident that

$$I(x \mid x) = H(x).$$

The ordinary definition of entropy uses probability concepts, and thus does not pertain to individual values, but to random values, i.e., to probability distributions within a given group of values. In order to stress this difference, we will denote random values by Greek letters. I believe that the need for attaching definite meaning to the expressions $H(x \mid y)$ and $I(x \mid y)$, in the case of individual values x and y that are not viewed as a result of random tests with a definite law of distribution, was realized long ago by many who dealt with information theory.

As far as I know, the first paper published on the idea of revising information theory so as to satisfy the above conditions was the article by Solomonov [1]. I came to similar conclusions, before becoming aware of Solomonov's work, in 1963–1964, and published my first article on the subject [2] in early 1965. A young Swedish mathematician, Martin-Löf, who worked in Moscow during 1964–1965, began developing this concept. His lectures [3] which he gave in Erlangen in 1966 represent a better introduction to the subject of my paper.

The meaning of the new definition is very simple. Entropy $H(x \mid y)$ is the minimal length of the recorded sequence of zeros and ones of a "program" P that permits construction of the value of x, the value of y being known,

$$H(x \mid y) = \min_{A(P,y)=x} l(P). \tag{2}$$

This concept is supported by the general theory of "computable" (partially recursive) functions, i.e., by the theory of algorithms in general. We will return again to the interpretation of the notation $A(P, y) = x$.

$H(x) = \min_{P \to x} \text{length}(P)$

length(P_1) = 100

P_1 = `print(“aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa”);`

$P_1 \to$

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

length(P_2) = 32

P_2 = `for(int i=0;i<90;i++)print("a");`

$P_2 \to$

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

An analogous situation exists in the principles of information theory. Essentially, it is applicable to large quantities of information, when the initial information (contained in the method on which the theory is based) is infinitesimal. Our basic formula (1) implies a "universal programming method" A, which exists because there are programming methods A possessing the quality

$$H_A(x) \leq H_{A'}(x) + C_{A'}$$

They allow the programming of anything with a program length that exceeds the length of any other programming method by not greater than a constant and is dependent only on this second programming method and not on values of x

From Kolmogorov, 1968, simplified, no y, LF

Kolmogorov, A. (1968). Logical basis for information theory and probability theory. *IEEE Transactions on Information Theory*, *14*(5), 662-664.

Example (Mandelbrot):

- simple program
- fascinating complexity

mandelbrot_simple | Processing 2.2.1

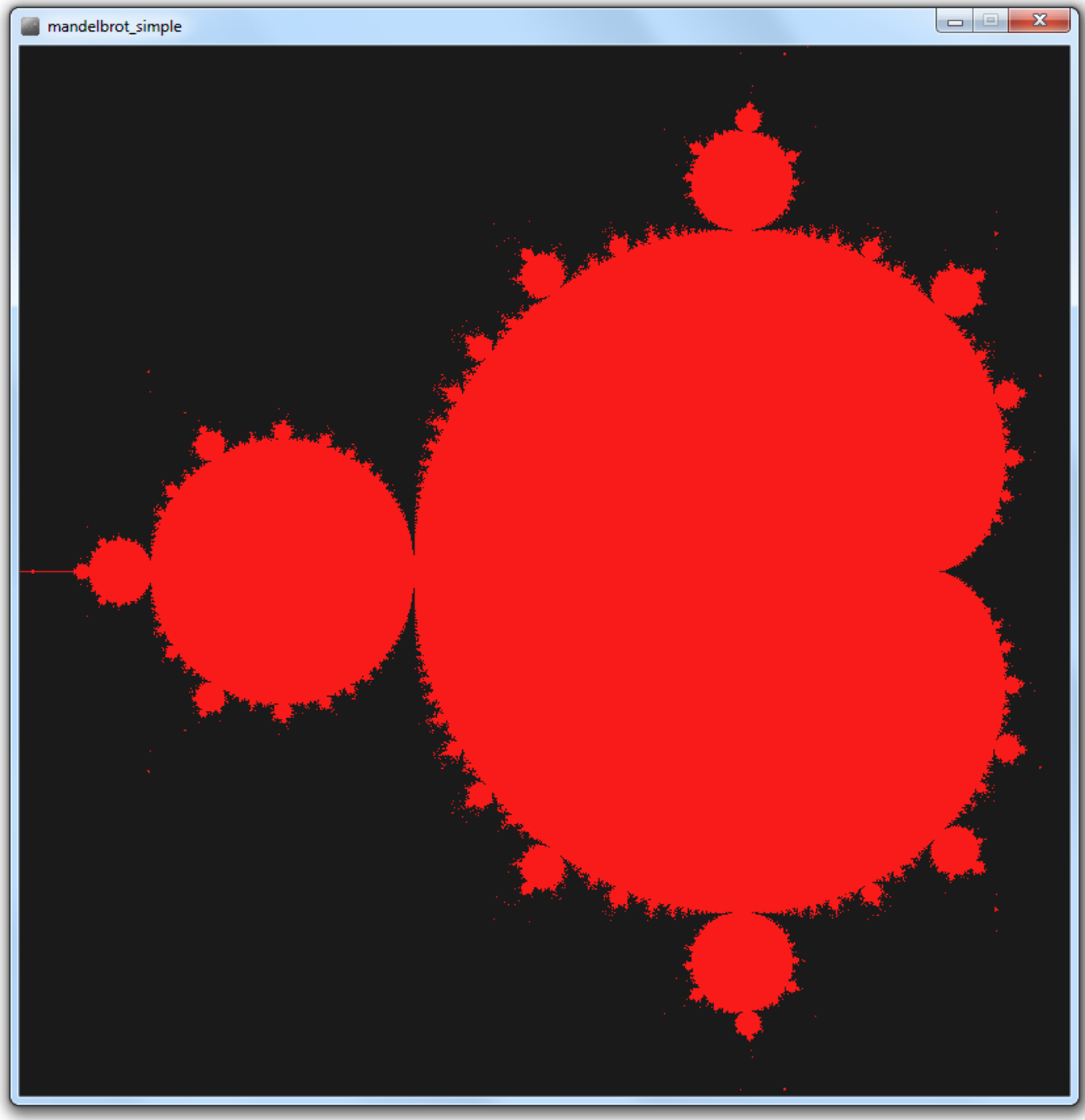
File Edit Sketch Tools Help

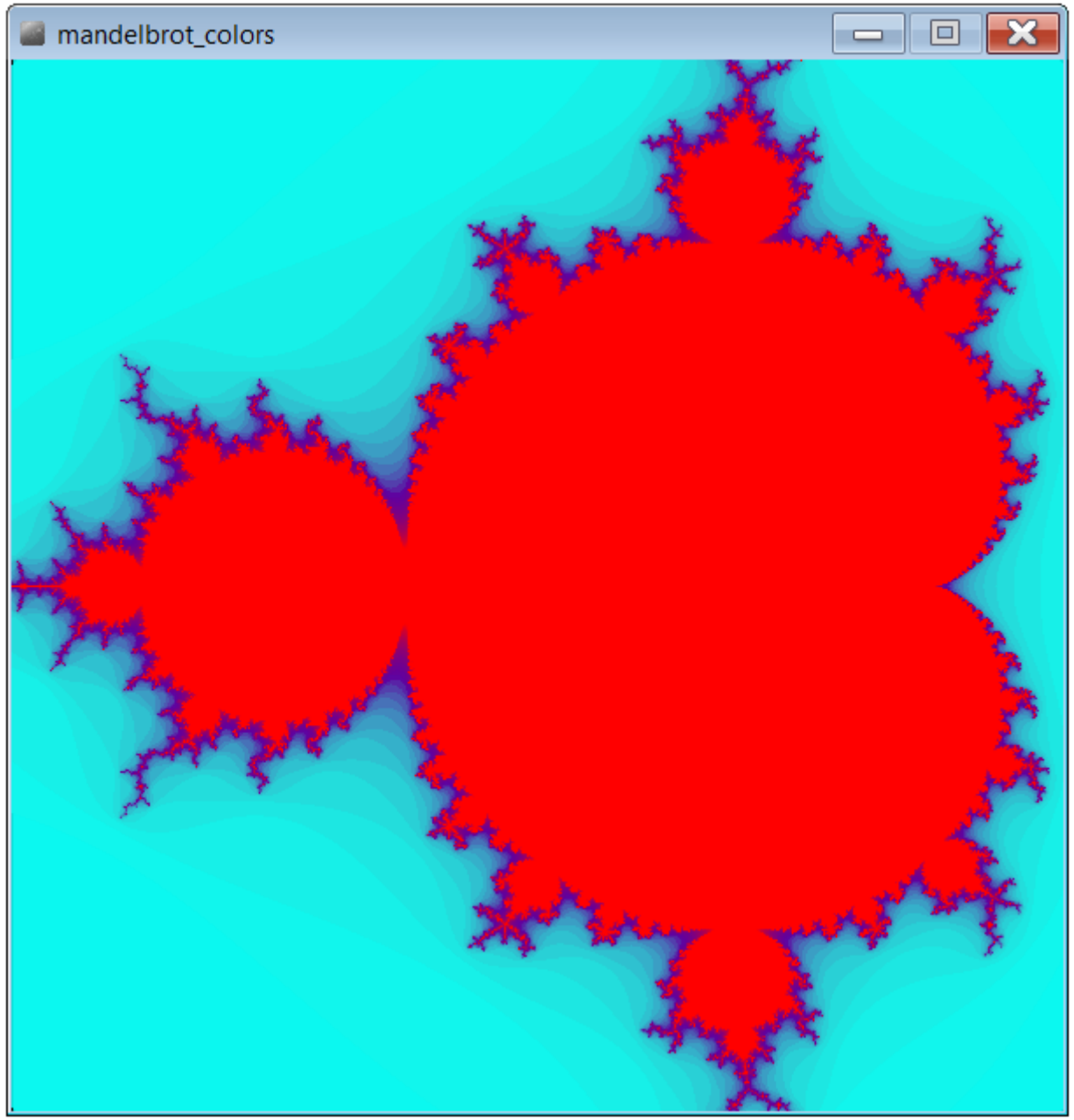
Java

mandelbrot_simple | complex

```
Complex  F(Complex c, int n){
         if (n == 0)
             return new Complex(0,0);
         else { Complex z = F(c,n - 1);
                return z.mul(z).add(c);
         }
}

void setup(){
     size(400,400);
     float scale = width / 2;
     for (float x = 0; x < width; x++)
          for (float y = 0; y < height; y++){
               float zx = 2*x / width  - 1.5;
               float zy = 2*y / height - 1.0;
               Complex zn = F(new Complex(zx,zy),100);
               stroke(zn.modulus() < 2? 255 : 0,0,0);
               point(x,y);
     }
}
```




mandelbrot_colors