Information Science
8: Algorithms and complexity 1
Fibonacci numbers

Naonori Kakimura

垣村尚徳

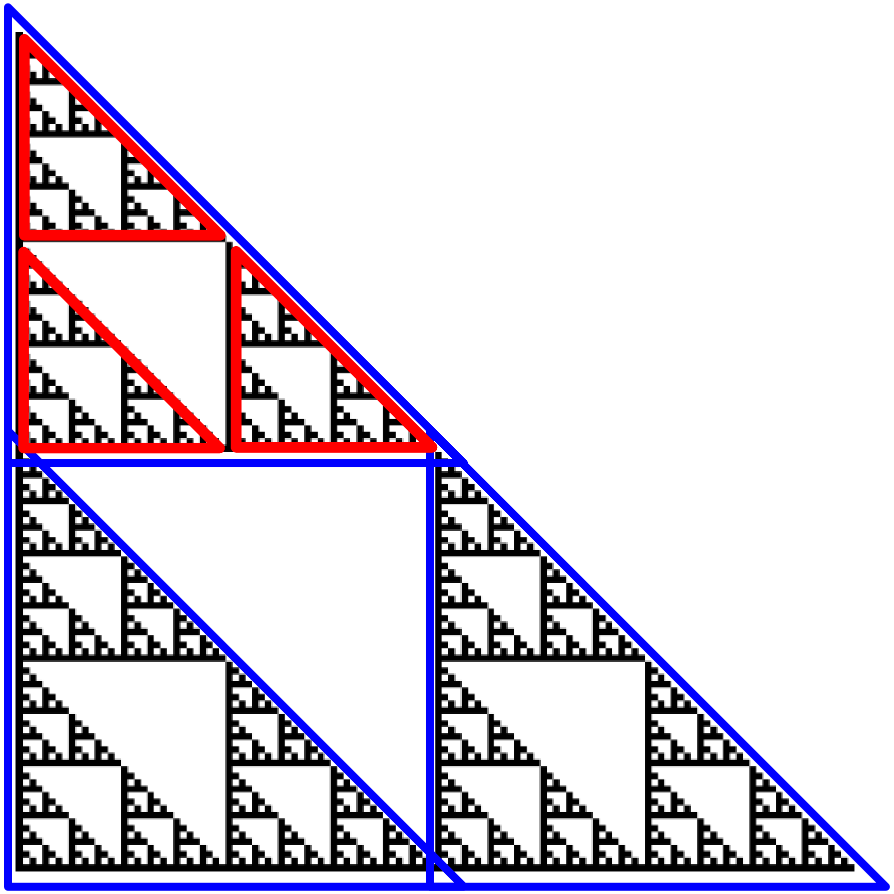
kakimura@global.c.u-tokyo.ac.jp

PEAK

Programs in English at Komaba

Recursive Definition for Sierpinski Triangle

- Three same triangles
  - In each small triangle there are same three triangles

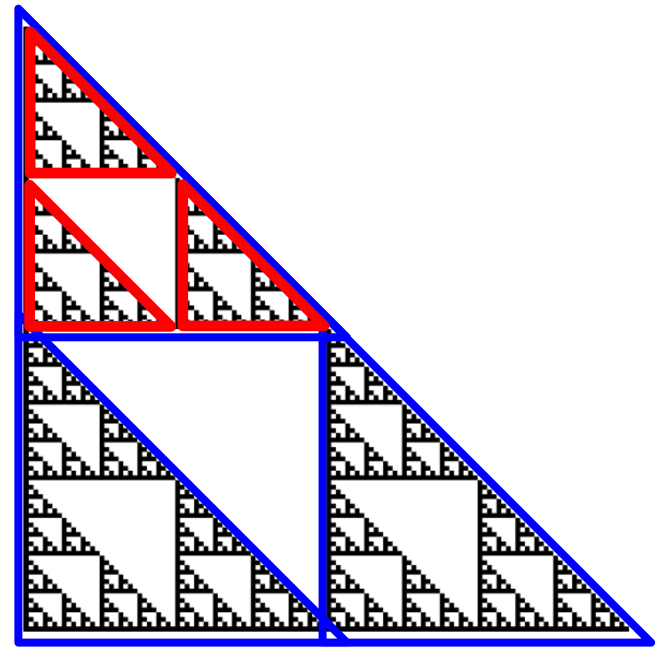


Recursion for Sierpinski Triangle

- Function: sierpinski_rec(n)
  - Make an image of size 2^n

```
def sierpinski_rec(n)
   a = make2d(2**n, 2**n)
   subsierpinski(a, n, 0, 0)
   a
end
```

Make a triangle of size n whose top is (0,0)



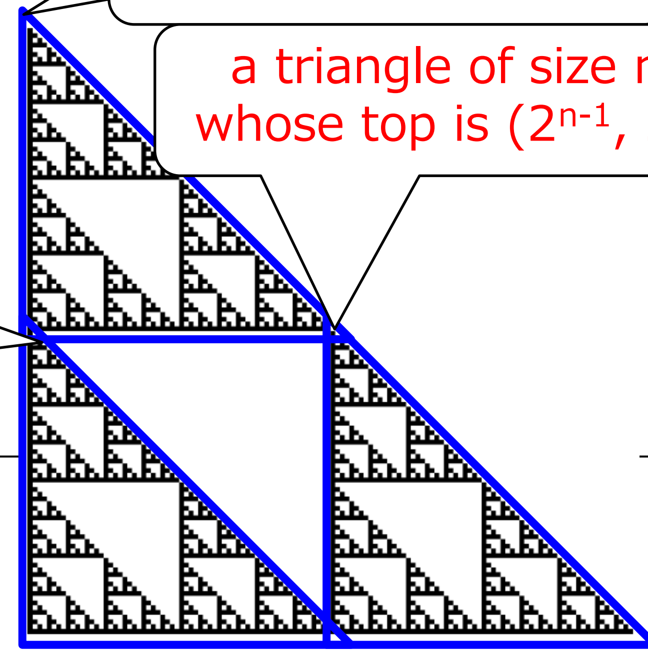
Size of triangle

- Function: subsierpinski(a, n, x, y)
  - Put a “triangle” of size n from (x,y) in a

(x,y): Coordinate of the “top” of triangle

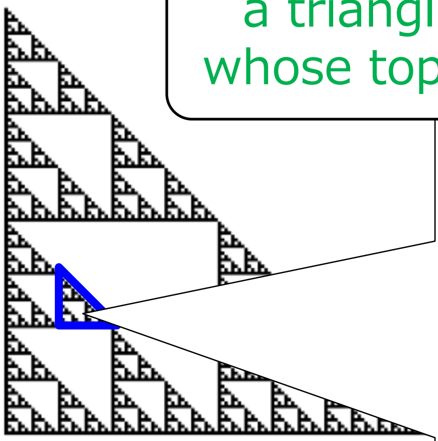
At the Beginning

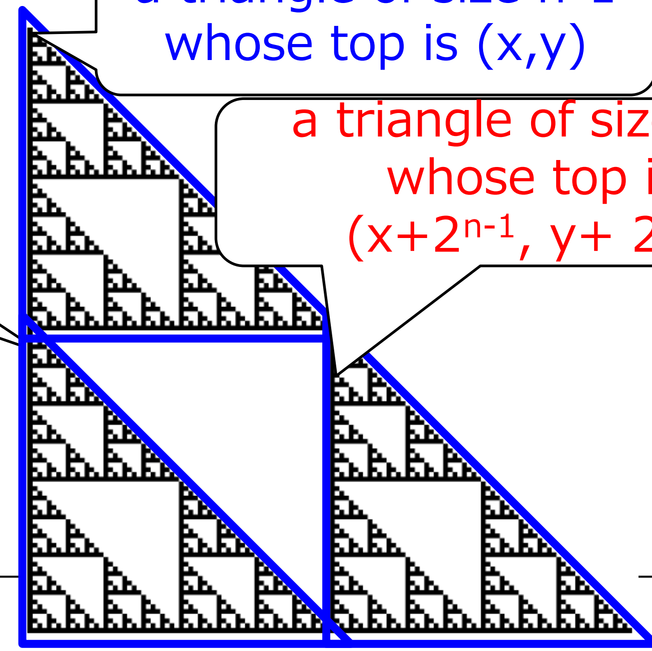
- If n=0, it is white
  - to be flipped
- OW, draw 3 triangles




```
def subsierpinski(a, n, x, y)
   if n == 0
     a[y][x] = 1
   else
         # when (x, y)=(0, 0)
         subsierpinski(a, n-1, 0, 0)
         subsierpinski(a, n-1, 0, 2**(n-1))
         subsierpinski(a, n-1, 2**(n-1), 2**(n-1))
    end
end
```

For (x, y)

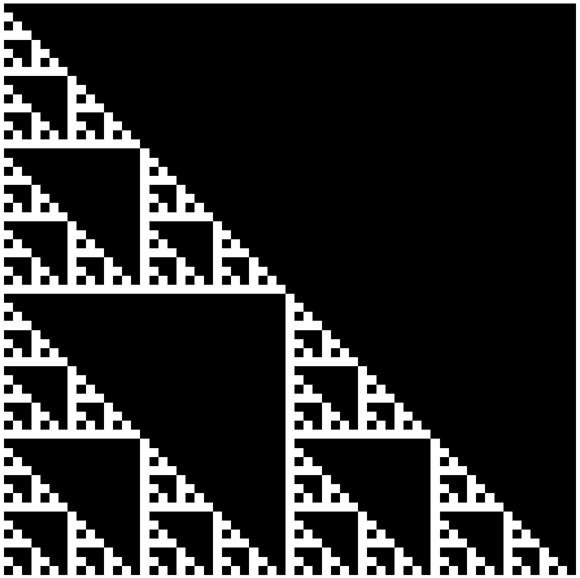


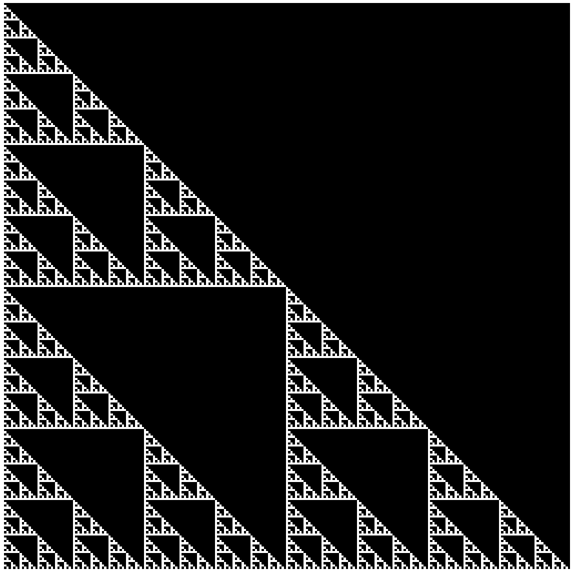



```
def subsierpinski(a, n, x, y)
   if n == 0
     a[y][x] = 1
   else
     subgasket(a, n-1, x, y)
     subgasket(a, n-1, x, y + 2**(n-1))
     subgasket(a, n-1, x + 2**(n-1), y + 2**(n-1))
   end
end
```

Recursion for Sierpinski Triangle

- When n = 6 and 8





- Need to be inversed (exchange 1 and 0)

Algorithms and complexity

What we study is

- Algorithms
  - Method to solve a problem in finite time
    - Cf. Applications
      - Car navigation system
      - scheduling of machines in factories
  - There are many ways to solve the same problem
    - Ex. two programs for n choose k last week
  - Computation time depends on algorithms

- How to estimate the computational time
  - Measure the execution time in practice
  - Estimate theoretically
    - common technique to all problems

Through Examples

- Today
  - Fibonacci number
- Next week
  - Further examples
- The week after the next
  - Sorting algorithm

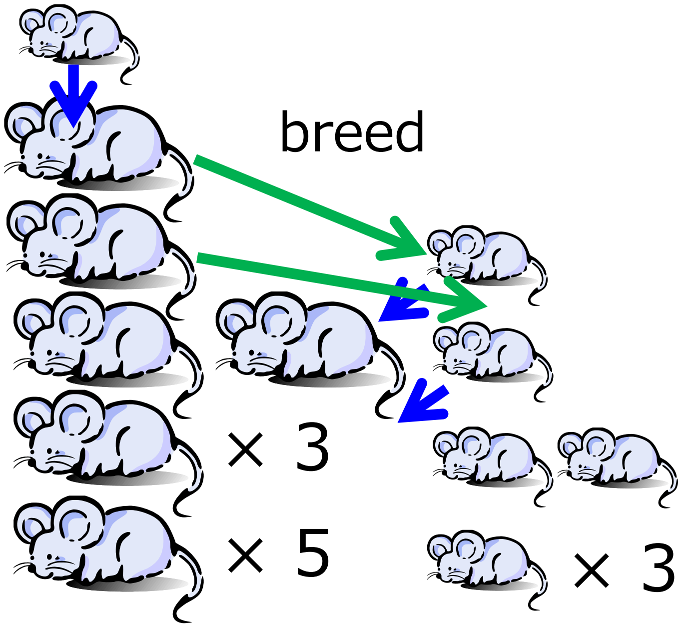
Today's Contents

- Fibonacci numbers
  - Introduction
  - Definition-based algorithm
  - Enumeration-based algorithm
  - Speed competition
- How to estimate computational time
  - Concept
  - Fibonacci case
  - Order notation
- Exercises

Fibonacci Number: introduction

- Each mouse(at least one-day old) breeds one baby everyday
  - Do not care about gender
  - They live a long time

0day	1 baby mouse
1day	1 adult mouse
2day	2 mice (1 adult & 1 baby)
3day	3 mice
4day	5 mice
5day	8 mice
6day	13 mice
7day	21 mice
…	



Sequence of #mice is called the Fibonacci numbers

Fibonacci Numbers

- 1, 1, 2, 3, 5, 8, 13, 21, 34, $\cdots$
  - Definition: Letting $\boldsymbol{f_k}$ be the kth number,

mice on
the (k-1)th day

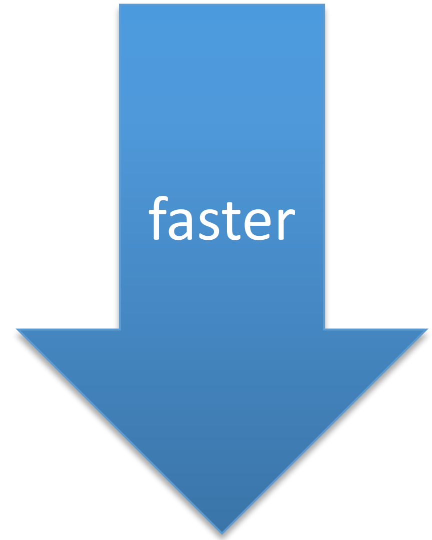
mice bred on the *k*th day
= # adult mice on the (k-1)th day
= # mice on the (k-2)th day

$$f_k = \begin{cases} f_{k-1} + f_{k-2} & (k \geq 2) \\ 1 & (k = 0, 1) \end{cases}$$

- Problem: find the kth number of the sequence
  - How many mice are there on the kth day?

Algorithms for Fibonacci Numbers

- Three algorithms
  - Compute from the recursive definition
    - in “fib.rb” on ITC-LMS
  - Enumeration using loop
    - in “fib.rb” on ITC-LMS
  - Using matrix computation(see Exercises)



- Aim: Compare computational times

Today’s Contents

- Fibonacci numbers
  - Introduction
  - Definition-based algorithm
  - Enumeration-based algorithm
  - Speed competition
- How to estimate computational time
  - Concept
  - Fibonacci case
  - Order notation
- Exercises

Computing based on the definition

- Definition

$$f_k = \begin{cases} f_{k-1} + f_{k-2} & (k \geq 2) \\ 1 & (k = 0, 1) \end{cases}$$

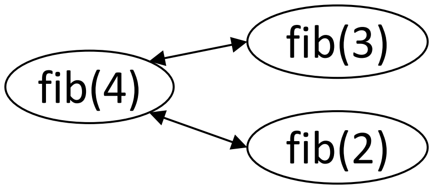
- $\boldsymbol{f_k}$ = **fibr(k)** : function
  - the postfix "r" stands for Recursion

```
def fibr(k)
  if k==0 || k==1
    1
  else
    fibr(k-1)+fibr(k-2)
  end
end
```

Computing based on the (

```
def fibr(k)
  if k==0 || k==1
    1
  else
    fibr(k-1)+fibr(k-2)
  end
end
```

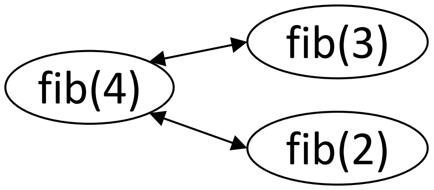
- fibr(4)
  - fibr(4)=fibr(3)+fibr(2)



Computing based on the

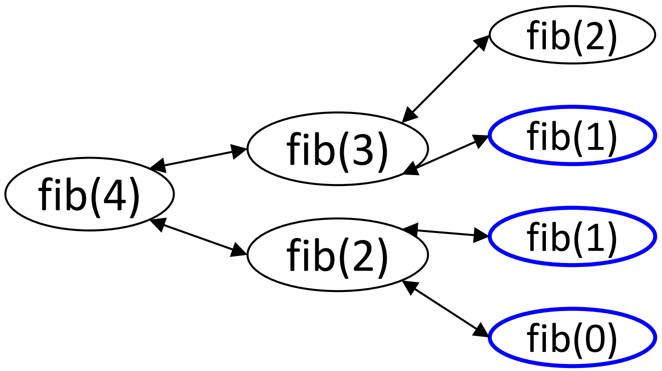
```
def fibr(k)
  if k==0 || k==1
    1
  else
    fibr(k-1)+fibr(k-2)
  end
end
```

- fibr(4)
  - fibr(4)=fibr(3)+fibr(2)
    - fibr(3)=fibr(2)+fibr(1)
    - fibr(2)=fibr(1)+fibr(0)



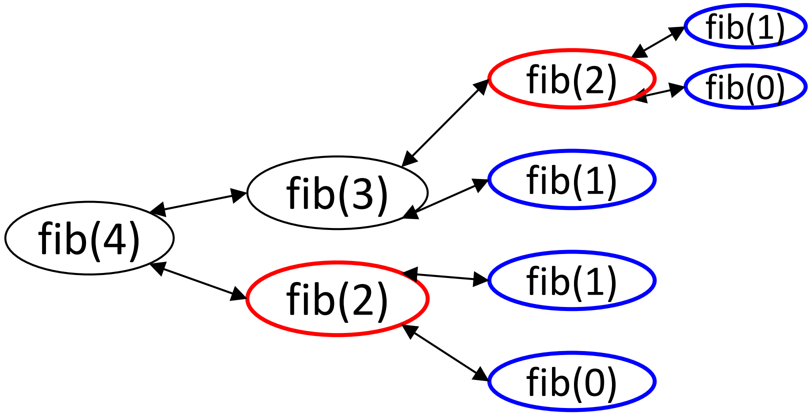
Computing based on the definition

- fibr(4)
  - fibr(4)=fibr(3)+fibr(2)
    - fibr(3)=fibr(2)+fibr(1)
      - fibr(2)=fibr(1)+fibr(0)
      - fibr(1)=1
    - fibr(2)=fibr(1)+fibr(0)
      - fibr(1)=1
      - fibr(0)=1



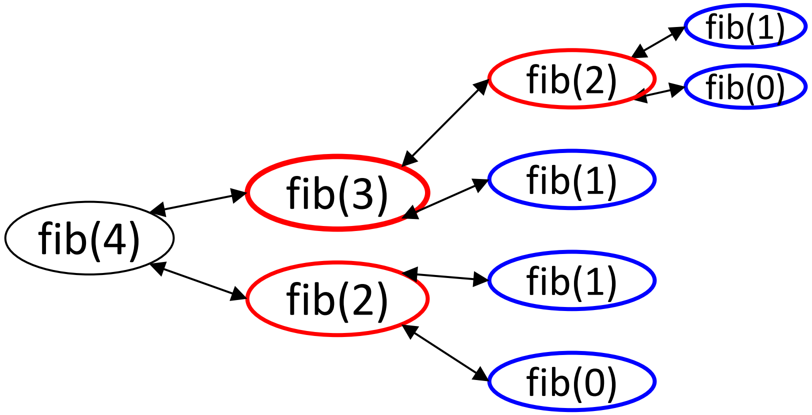
Computing based on the definition

- fibr(4)
  - fibr(4)=fibr(3)+fibr(2)
    - fibr(3)=fibr(2)+fibr(1)
      - fibr(2)=fibr(1)+fibr(0) **= 1+1 = 2**
        - fibr(1)=1
        - fibr(0)=1
      - fibr(1)=1
    - fibr(2)=fibr(1)+fibr(0) **= 1+1 = 2**
      - fibr(1)=1
      - fibr(0)=1



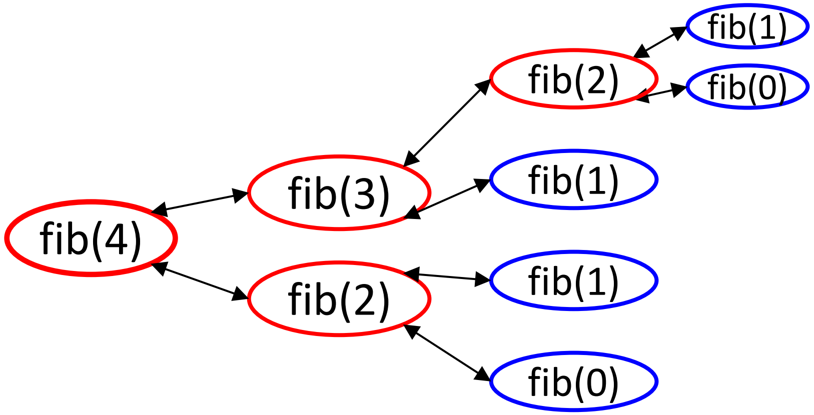
Computing based on the definition

- fibr(4)
  - fibr(4)=fibr(3)+fibr(2)
    - fibr(3)=fibr(2)+fibr(1) **= 2+1 = 3**
      - fibr(2)=fibr(1)+fibr(0) **= 1+1 = 2**
        - fibr(1)=1
        - fibr(0)=1
      - fibr(1)=1
    - fibr(2)=fibr(1)+fibr(0) **= 1+1 = 2**
      - fibr(1)=1
      - fibr(0)=1



Computing based on the definition

- fibr(4)
  - fibr(4)=fibr(3)+fibr(2) **= 3+2 = 5**
    - fibr(3)=fibr(2)+fibr(1) **= 2+1 = 3**
      - fibr(2)=fibr(1)+fibr(0) **= 1+1 = 2**
        - fibr(1)=1
        - fibr(0)=1
      - fibr(1)=1
    - fibr(2)=fibr(1)+fibr(0) **= 1+1 = 2**
      - fibr(1)=1
      - fibr(0)=1



Necessary to call fibr 9 times

Exercises during session: Fibonacci Numbers

- Try fibr(10)
  - #mice on the 10th day

- Find the minimum k satisfying fib(k)>1000000
  - feel it takes much time to compute
  - We can stop the process by pressing Ctrl C

- More efficient ways to compute it?

Today's Contents

- Fibonacci numbers
  - Introduction
  - Definition-based algorithm
  - Enumeration-based algorithm
  - Speed competition
- How to estimate computational time
  - Concept
  - Fibonacci case
  - Order notation
- Exercises

You can see that there're a lot of overlapping

#overlapping explodes if k is large

- fibr(4)
  - fibr(4)=fibr(3)+fibr(2) **= 3+2 = 5**
    - fibr(3)=fibr(2)+fibr(1) **= 2+1 = 3**
      - fibr(2)=fibr(1)+fibr(0) **= 1+1 = 2**
        - fibr(1)=1
        - fibr(0)=1
      - fibr(1)=1
    - fibr(2)=fibr(1)+fibr(0) **= 1+1 = 2**
      - fibr(1)=1
      - fibr(0)=1

Computing by using iteration

We can compute $\boldsymbol{f_k}$ from $\boldsymbol{f_{k-1}}$ and $\boldsymbol{f_{k-2}}$

$\rightarrow$ Enough to remember the last two numbers

Keeping the last one and the one before last, and add them

Repeat from i=1 to k

k	0	1	2	3	4	5	6	7	8	9	10	11	12
fib(k): f	1	1	2	3	5	8	13	21	34	55	89	144	233
Last : p1	–	1	1	2	3	5	8	13	21	34	55	89	144
Before last p2	–	–	1	1	2	3	5	8	13	21	34	55	89

$\boldsymbol{f_2, f_1}$ $\boldsymbol{f_3, f_2}$

Computing by using iteration


```
def fibl(k)
  f=1
  p1=1
  for i in 2..k
    p2 = p1      #fib(i-2)
    p1 = f       #fib(i-1)
    f = p1 + p2 #fib(i)
  end
  f              #fib(k)
end
```

For each i, update p2, p1

Orderings in the for-loop is important
(see Quizzes)

Ordering in the for-loop is important

```
def fibl(k)
  f=1
  p1=1
  for i in 2..k
    p2 = p1      #fib(i-2)
    p1 = f       #fib(i-1)
    f = p1 + p2 #fib(i)
  end
  f            #fib(k)
end
```

p2 becomes (previous) p1
p1 becomes (previous) f
f becomes (new)p1+(new)p2

	k	0	1	2	3	4	5	6	7	8	9	10	11	12
	fib(k): f	1	1	2	3	5	8	13	21	34	55	89	144	233
Last	p1	–	1	1	2	3	5	8	13	21	34	55	89	144
Before last	: p2	–	–	1	1	2	3	5	8	13	21	34	55	89

Today's Contents

- Fibonacci numbers
  - Introduction
  - Definition-based algorithm
  - Enumeration-based algorithm
  - Speed competition
- How to estimate computational time
  - Concept
  - Fibonacci case
  - Order notation
- Exercises

Speed Competition of two programs

- Download bench.rb and fib.rb from ITC-LMS

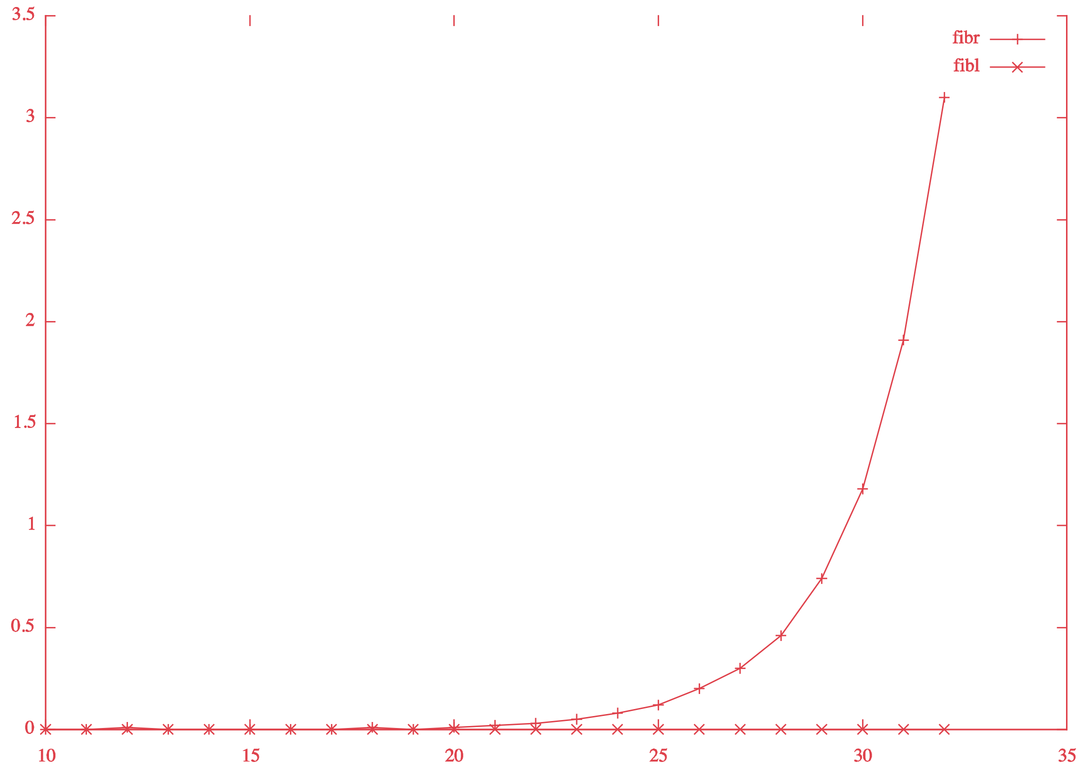
```
irb(main):004:0> load("./bench.rb")
irb(main):005:0> load ("./fib.rb")
irb(main):006:0> run("fibr", 10)
irb(main):007:0> run ("fibl", 10)

irb(main):009:0> for k in 10..32
irb(main):010:1>   run("fibr", k)
irb(main):011:1>   run("fibl", k)
irb(main):012:1> end
```

Parameters for fibr

Graphs of fibr(k) and fibl(k) for 10≦k≦32

sec.



Parameter k

↑ fibl(k) is almost zero

※ Depends on your computational environments

Remarks of bench.rb: Change Display Format

reset(): delete data

command("set logscale y"): change the y-axis to log scale

command("set logscale x"): change the x-axis to log scale

command("unset logscale"): change to the normal scale

command("set xrange [a:b]"):
set the range of x-axis to from a to b

command("set yrange [a:b]"): same as yrange

command("set autoscale"): set the range automatically

Commands in "()" is commands of the software GNUPLOT

See a manual of GNU plot for other commands

Examples: Logarithmic Scale for fibr(k)

- Observe the curve of fibr in more details

```
irb(main):026:0> command("set logscale y")
irb(main):027:0> for k in 10..32
irb(main):028:1>   run("fibr", k)
irb(main):029:1> end
```

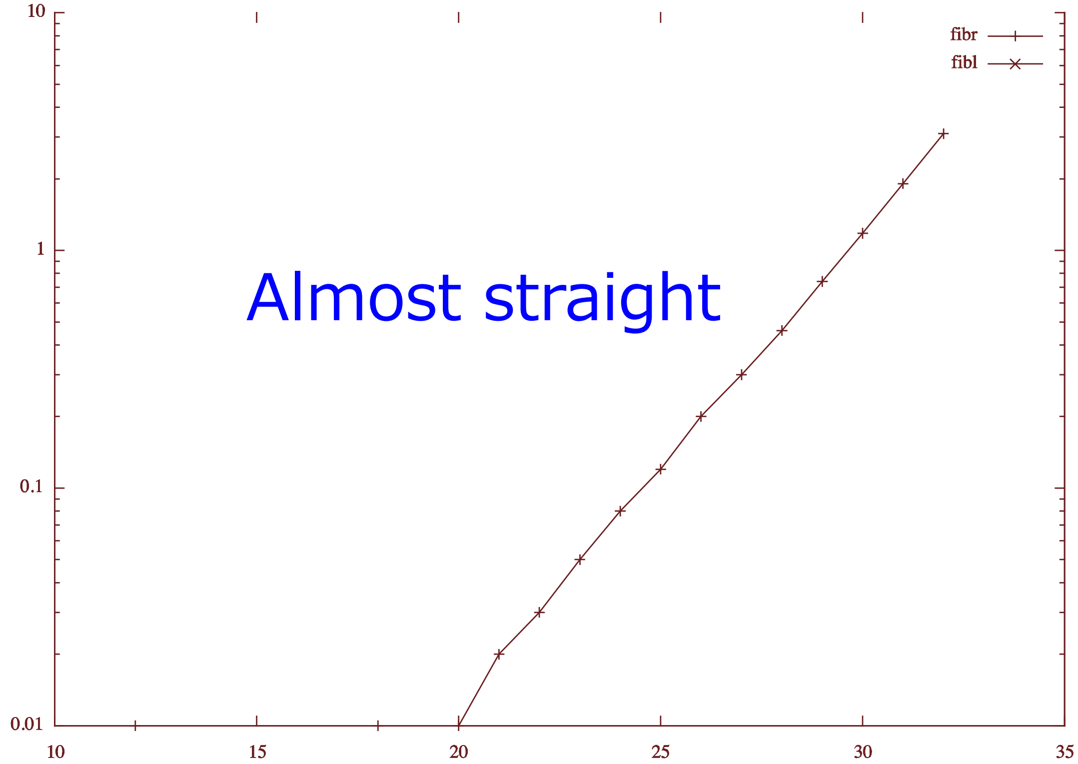
Examples: Logarithmic Scale for fibr(k)

- Almost straight line
  → proportional to an exponential function

log of seconds

$$\text{time} = 10^{ak+b}$$

$$= C10^{ak}$$



k

Time is proportional to an exponential function:
← log(time)=$ak+b$ (a: slope)

Observation So Far

- fibr(k)'s running time >> fibl(k)'s running time
  - when k → large

- fibr(k)'s running time ∝ an exponential function of k

  proportional

- fibl(k)'s running time : almost 0

→ consider a much larger k to investigate fibl

Let's use fibl6(k) in "fib.rb"

We want to avoid handling large numbers

Computing only 6 Digits in Fibl

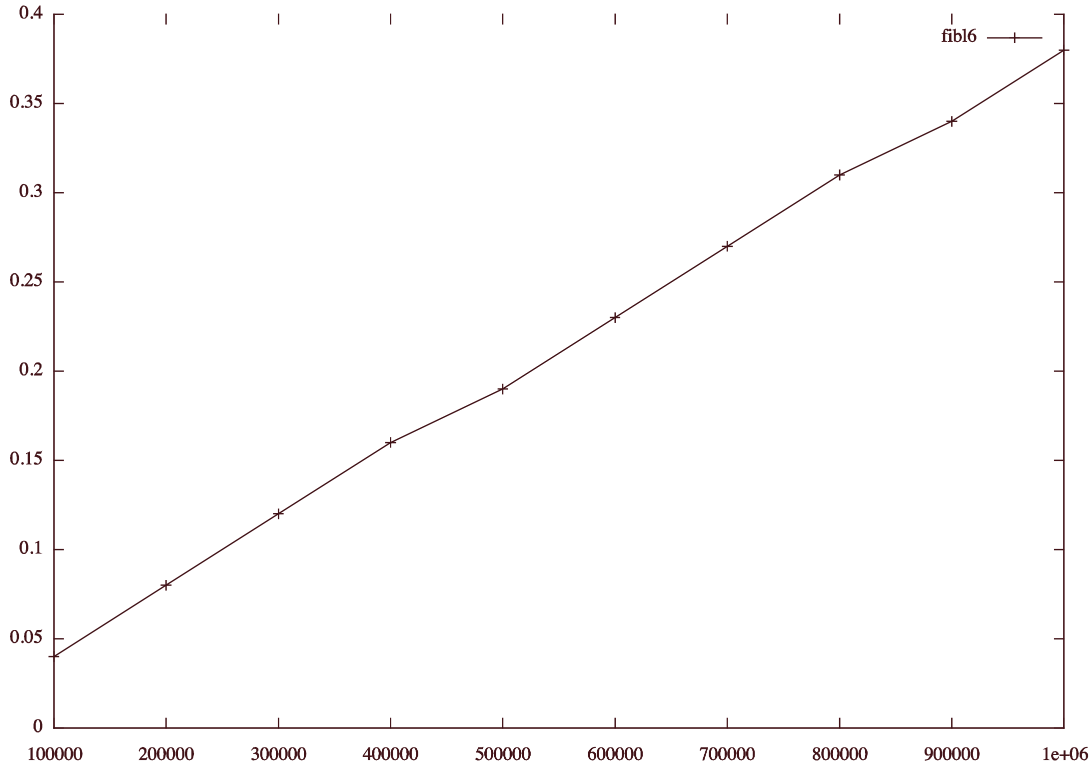

```
def fibl6(k)
  f=1
  p1=1
  for i in 2..k
    p2 = p1
    p1 = f
    f = (p1 + p2) % 1000000
  end
  f
end
```

store only the first 6 digits

```
irb(main):030:0> reset()
irb(main):031:0> command("unset logscale")
irb(main):032:0> for m in 1..10
irb(main):033:1>   run("fibl6", 100000*m)
irb(main):034:1> end
```

Graph for fibl6(k)

- 100thousands ≦ k ≦ 1million



Computation time for fibl6

- almost straight in a normal graph
  - → proportional to a linear function of k: $\mathrm{time} = ck+d$

Summary So Far

- fibr(k)'s running time >> fibl(k)'s running time
  - when k $\rightarrow$ large

- fibr(k)'s running time $\propto$ an exponential function of k

  proportional

- fibl(k)'s running time : almost 0
  - When k is large $\propto$ a linear function of k

Cf) To use bench.rb at home

- You need to install gnuplot
  - Gnuplot: software for making a graph

Today's Contents

- Fibonacci numbers
  - Introduction
  - Definition-based algorithm
  - Enumeration-based algorithm
  - Speed competition
- How to estimate computational time
  - Concept
  - Fibonacci case
  - Order notation
- Exercises

Expect the speed of algorithms

- Useful to estimate the time BEFORE execution

  BEFORE making a program

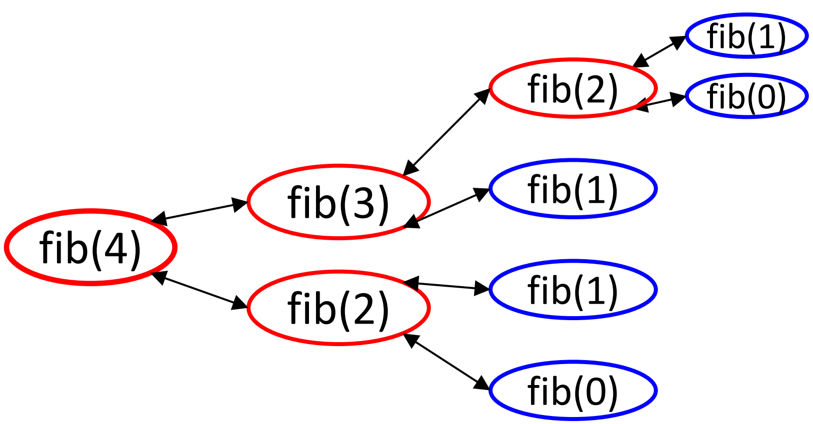
- Time ≈ # arithmetic operations & comparisons
  - Rough approximation of practical computational time
  - Independent of computer environments
  - Ignore difference of operations
    - Same speed: comparison vs addition

$T_r(k)$: Computational Time for fibr

- 3 computations
  - Comparison btw k and 1(or 0)
  - Expand fibr(k) to fibr(k-1) and fibr(k-2)
  - Add fibr(k-1) and fibr(k-2)

$$T_r(k) = T_r(k-1) + T_r(k-2) + 3$$

$$= \left[\# \quad \text{and} \leftrightarrow \text{in the chart} \right]$$




```
def fibr(k)
  if k==0 || k==1
    1
  else
    fibr(k-1)+fibr(k-2)
  end
end
```

$T_r(k)$: Computational Time for fibr

$$\phi = \frac{1+\sqrt{5}}{2}$$

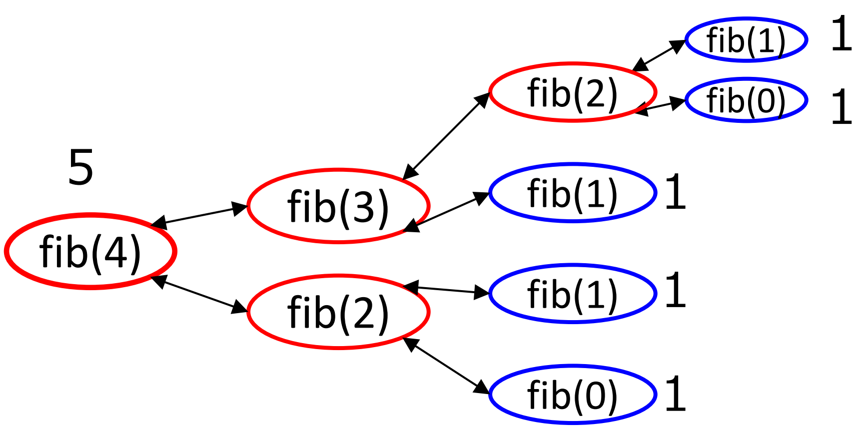
- 3 computations
  - Comparison btw k and 1(or 0)
  - Expand fibr(k) to fibr(k-1) and fibr(k-2)
  - Add fibr(k-1) and fibr(k-2)

$$T_r(k) = T_r(k-1) + T_r(k-2) + 3$$

= [# and $\leftrightarrow$ in the chart] $\approx$ 4f(k) $\propto \phi^k$

k-th Fib number

Exercise 6.1.3



= f(k)

having value 1

= f(k)-1

merge two by one

$\leftrightarrow$ = 2

Fibonacci number by Repetition

- Let $T_l(k)$ be # operations to compute fibl(k)
  - Each iteration takes 3 computations

$$T_l(k) \cong 3(k-1) \propto k$$

```
def fibl(k)
  f=1
  p1=1
  for i in 2..k
    p2 = p1        #fib(i-2)
    p1 = f         #fib(i-1)
    f = p1 + p2 #fib(i)
  end
  f              #fib(k)
end
```

Decide if i is at most k

Assign

Assign

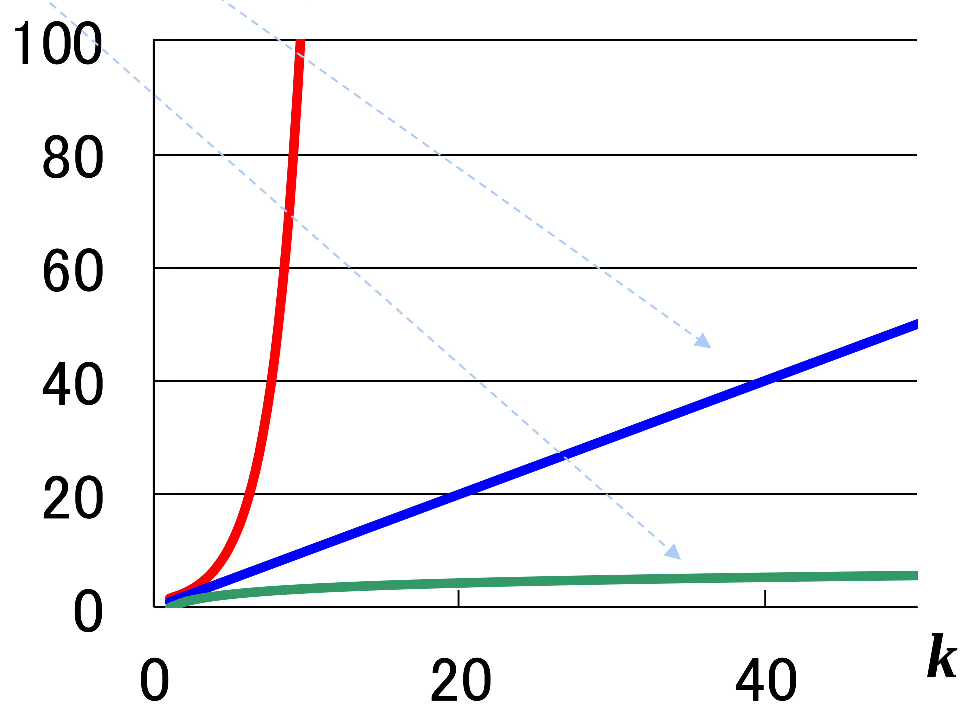
Add

Complexity for Fibonacci Numbers

- Finding the ***k*** th Fibonacci number
  - Definition-based: proportional to Φ^k
  - Enumeration: proportional to $\boldsymbol{k}$
  - Matrix-computation: proportional to $\mathbf{log}\ \boldsymbol{k}$

  See Exercise 6.1.6-7

$$\phi = \frac{1+\sqrt{5}}{2}$$



Today’s Contents

- Fibonacci numbers
  - Introduction
  - Definition-based algorithm
  - Enumeration-based algorithm
  - Speed competition

- How to estimate computational time
  - Concept
  - Fibonacci case
  - Order notation (next week?)

- Exercises

Order Notation

- Use “order” notation instead of detailed eqn
  - Ex: $\mathbf{O}(\boldsymbol{n})$, $\mathbf{O}(\boldsymbol{n}^2)$, $\mathbf{O}(\mathbf{log}\ \boldsymbol{n})$
    - $\mathbf{O}(\cdot)$: ”time is proportional to ・”
  - $\mathbf{O}(\boldsymbol{n})$: $\boldsymbol{n}$ increases 100 times → time 100 times
  - $\mathbf{O}(\boldsymbol{n}^2)$: $\boldsymbol{n}$ increases 100 times → time 10K times
  - $\mathbf{O}(\Phi^{\boldsymbol{n}})$: $\boldsymbol{n}$ increases 100 times → time $\Phi^{99\boldsymbol{n}}$ times
  - $\mathbf{O}(\mathbf{log}\ \boldsymbol{n})$: $\boldsymbol{n}$ increases $\boldsymbol{n}$ times → time 2 times

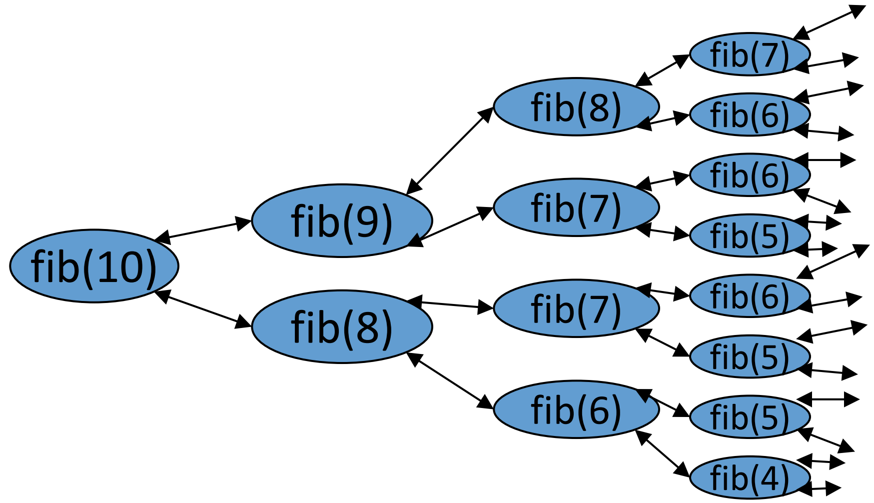
Order Notation

- Use "order" notation instead of detailed eqn
  - Ex: $\mathbf{O}(n)$, $\mathbf{O}(n^2)$, $\mathbf{O}(\log n)$
    - $\mathbf{O}(\cdot)$: "time is proportional to ・"
  - $\mathbf{O}(n)$: n increases 100 times → time 100 times
  - $\mathbf{O}(n^2)$: n increases 100 times → time 10K times
  - $\mathbf{O}(\Phi^n)$: n increases 100 times → time Φ^{99n} times
  - $\mathbf{O}(\log n)$: n increases n times → time 2 times

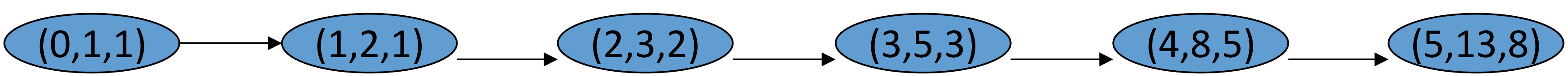
- Point: We do not care small details
  - Ignore coefficients, rounding-up/down
    - $\mathbf{O}(n)$「proportional to n」when n, $2n$, $100000n$ times
  - Leave the most dominant term only
    - $n+8 = \mathbf{O}(n)$, $n+\log n = \mathbf{O}(n)$

Difference of Algorithms: Fibonacci

- Definition-based
  - $\mathbf{O(\Phi^{\mathit{k}})}$



- Enumeration-based
  - $\mathbf{O(\mathit{k})}$



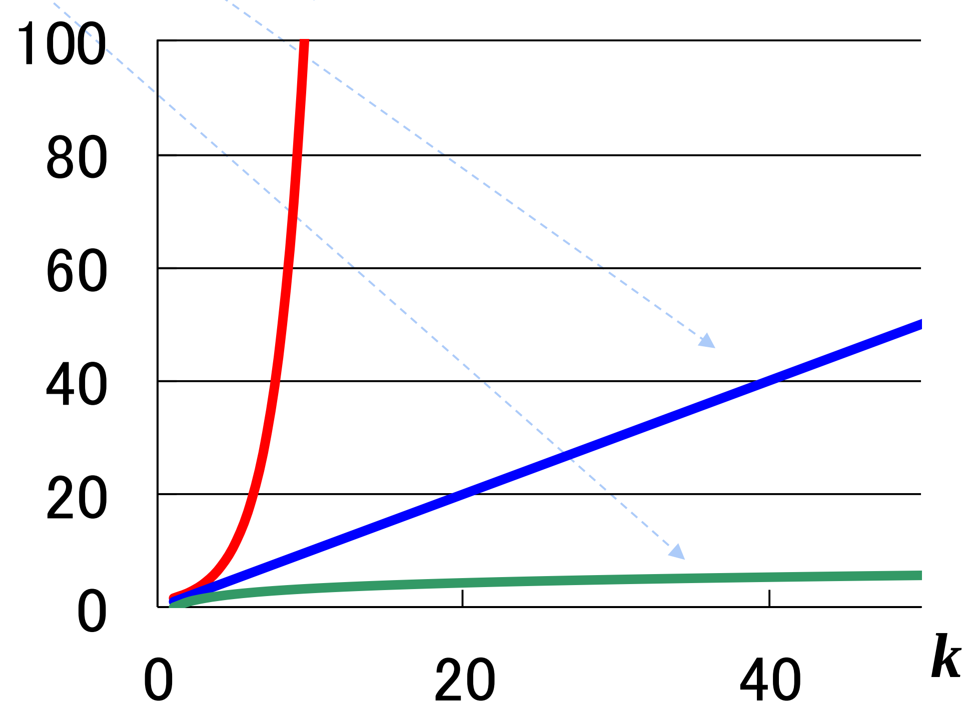
Complexity for Fibonacci Numbers

- Finding the ***k*** th Fibonacci number

$$\phi = \frac{1+\sqrt{5}}{2}$$

  - Definition-based — $\mathbf{O(\Phi^k)}$
  - Enumeration — $\mathbf{O(k)}$
  - Matrix-computation — $\mathbf{O(\log k)}$

  See Exercise 6.1.6-7



Today’s Contents

- Fibonacci numbers
  - Introduction
  - Definition-based algorithm
  - Enumeration-based algorithm
  - Speed competition

- How to estimate computational time
  - Concept
  - Fibonacci case
  - Order notation

- Exercises

Exercises for Today

- Solve the following 4 exercises
  - Exercise4. math rather than info science

Exercise1: What is the 100th Fibonacci number?

1. Even number
2. Odd number

Fibonacci numbers: 1,1,2,3,5,8,13,21,34,…

Explain why (without computing)

Exercise 2: Ordering of Assignment

- Suppose that we swap the two lines of fibl as follows. Explain what sequence will be obtained by completing the table
  - Text file or by hand-writing

```
def fibl_error(k)
  f=1
  p1=1
  for i in 2..k
    p1 = f        #fib(i-1)
    p2 = p1       #fib(i-2)
    f = p1 + p2 #fib(i)
  end
  f             #fib(k)
end
```

k	0	1	2	3	4	5	6	7	8	9	10	11	12
fib(k): f													
Last p1													
Before last : p2													

Exercise 3: Computational Time for fibr

- Explain why fibr(k) takes about 4f(k) operations
  - Hint: we can use *either of* the following strategies.
    - Show $T_r(k) = 4f(k) - 3$ by induction using

$$T_r(k) = T_r(k-1) + T_r(k-2) + 3$$

  - or
    - Estimate the number of nodes and edges in the generated tree as in Slide 44

Exercise 4: Review of Eigenvalues

- Explain also how you obtain solutions

6. Let f_k be the kth Fibonacci number. Then it holds that

$$\begin{pmatrix} f_{k+1} \\ f_k \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_k \\ f_{k-1} \end{pmatrix}.$$

(a) Explain why the above equation holds.

(b) Let A be the matrix in the form of

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}.$$

Determine the eigenvalues λ_1 and λ_2 of A.

(c) By the definition of eigenvalues, there exists a nonsingular(invertible) matrix U such that

$$A = U \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} U^{-1}$$

Determine the matrix U and U^{-1}.

(d) By (a), we have

$$\begin{pmatrix} f_{k+1} \\ f_k \end{pmatrix} = A^k \begin{pmatrix} f_1 \\ f_0 \end{pmatrix} = A^k \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Using the equation in (c), express the kth Fibonacci number f_k with λ_1 and λ_2.

Exercise 5: (optional)

6. Define the function matpower(a, n) that computes the nthe power of a matrix a. We may suppose that a is a 2×2 matrix and it is given as a 2-dimensional array.

   (a) Define the function matmul(a,b) that computes the product of two matrices a and b.

   Hint: Since two matrices have size 2 by 2, it holds that

$$\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix} = \begin{pmatrix} a_{00}b_{00} + a_{01}b_{10} & a_{00}b_{01} + a_{01}b_{11} \\ a_{10}b_{00} + a_{11}b_{10} & a_{10}b_{01} + a_{11}b_{11} \end{pmatrix}.$$

   (b) Define the function matsquare(a) that computes the square of a matrix a.

Exercise 5: (optional)

(d) We can observe that the power of matrices can be computed more efficiently. For example, for a matrix A, $A^{16} = (((A^2)^2)^2)^2$. Hence A^{16} can be obtained by taking a square four times. This reduces the number of matrix multiplications from 16 to 4. More precisely, we have the following.

$$A^n = \begin{cases} I & \text{if } n = 0, \\ (A^{n/2})^2 & \text{if } n \text{ is an even number with } n \geq 2, \\ A \times A^{n-1} & \text{if } n \text{ is an odd number,} \end{cases}$$

where I is the identity matrix (unit matrix). Using this relationship, we can make a recursive program matpower(a,n) that computes the nth power of a matrix a.

(e) Estimate the computational time of matpower(a,n) using the order notation.

Related to (e)

8. Define the function fibm6(k) that computes the first 6 digits of kth Fibonacci number using matpower(a,n). Using bench.rb, confirm that the computational time is proportional to $\log k$, which means $O(\log k)$.

Submission Deadline

- **Dec. 7 (Wed) 23:59**
  - By ITC-LMS
    - **a text file (.txt)**
    - **a PDF file (.pdf)** made by Word or scanning etc.
  - It is OK to submit a hand-written one if you want to do homework manually
    - Recommend to scan and submit it
    - Or hand in to me during the class

  - You can use program notation to express math, e.g.,
    - sqrt(2) for $\sqrt{2}$
    - 2-dim array to represent a matrix

Next Session (Dec. 5)

- No class on Nov. 28 due to Komaba Festival

- Today
  - Fibonacci number
- Next session
  - Further examples
- The week after the next
  - Sorting algorithm

Cf） Importance of Algorithms

- 『フカシギのかぞえかた』 “Time with class! Let’s count!”
  - http://www.youtube.com/watch?v=Q4gTV4r0zRs
- Explaining importance of algorithms