

ソフトウェア工学

2009年度 5セメスタ開講

履修にあたって

教科書:「アルゴリズムとデータ構造」
平田富夫著 森北出版

参考書: 「データ構造とアルゴリズム」
エイホ他著、倍風館

講義種類: 5セメスター開講、専門科目、
時間割: 金曜3時限(12:50-14:20)
講義室: K325

担当:
草苅 良至
GI511(内線 2095)、kusakari@akita-pu.ac.jp

講義予定

1回 4/10(金)	2回 4/17(金) レポート 提出 (S0)	3回 4/24(金)	4回 5/1(金) レポート 提出 (S1)	5回 5/8(金)	6回 5/15(金)	7回 5/22(金) レポート 提出 (S2)	8回 5/29(金)
6/5(金) 開学記 念日 (休講)	9回 6/12(金)	10回 6/19(金) レポート 提出 (S3)	11回 6/26(金)	12回 7/3(金)	13回 7/10(金) レポート 提出 (S4)	14回 7/17(金)	試験 7/31(金) 持込不 可

評価

- 授業内演習(出席)15%
- レポート25%
- 試験60%

本講義の目的

- よいソフトウェアを作成するための基礎を身に着ける。
- 良いソフトウェアであることの客観的な評価法を身に着ける。

本講義のレポート

- 主にC言語によるプログラミングが伴う。
- レポート作成の際には、プログラミング演習室を用いることができる。
 - ◆ ただし、木曜日と、金曜日の午後は、3セメスターのプログラミング演習があるので、他の時間帯に利用すること。

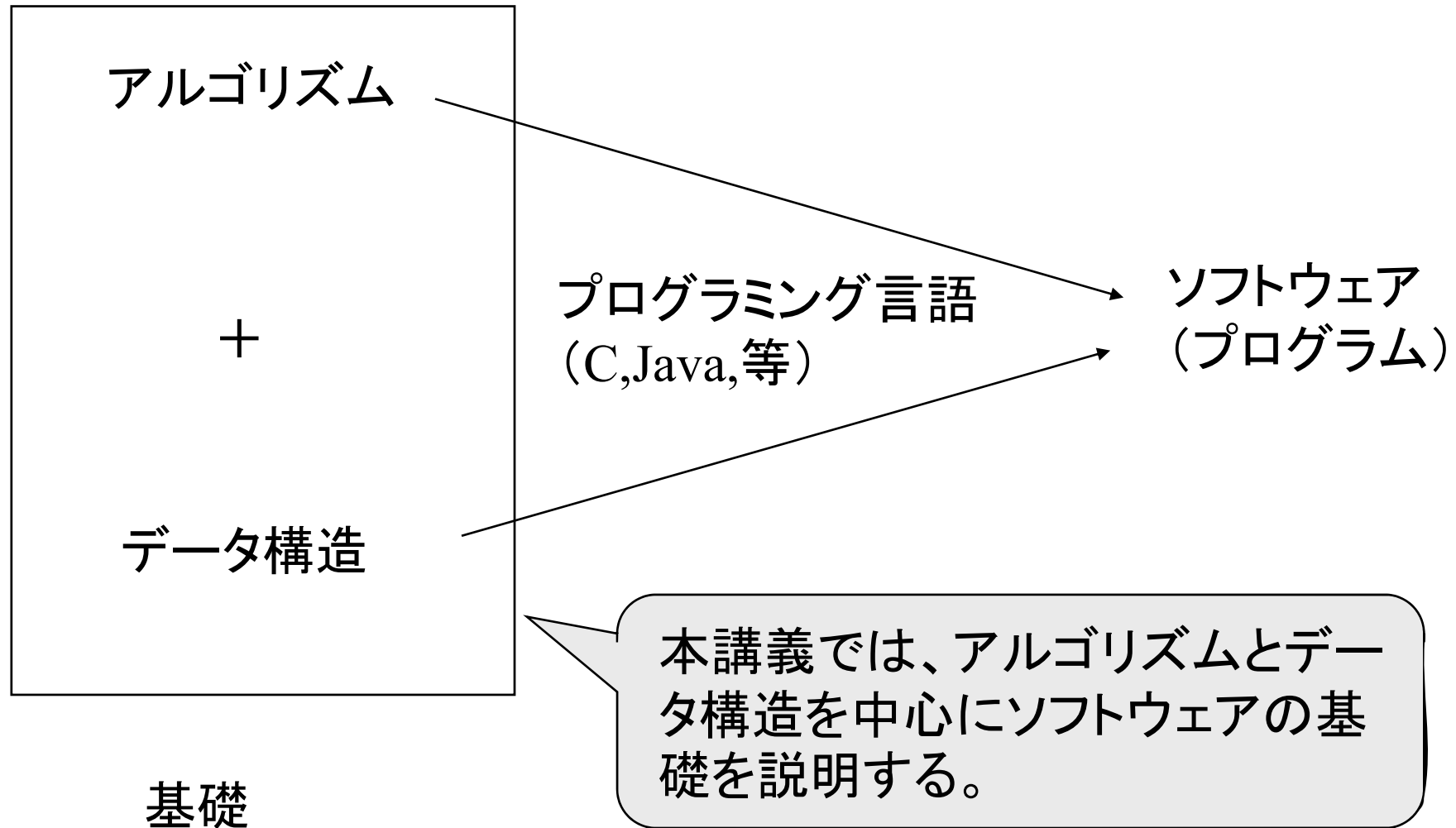
1. アルゴリズム入門

よいソフトウェアとは

本講義では、
主にこの部分
に注目する。

- 正しく動作するソフトウェア
- 同じ処理を速く実行できるソフトウェア（同じハードウェアで動作させた場合。）
- 同じ処理を少ないメモリで実行できるソフトウェア
- 再利用が可能なソフトウェア
- 誤動作のないソフトウェア
- 使いやすいソフトウェア
- 等々

ソフトウェア作成の基礎



本講義での主な注目点

- 正しく動作するプログラムの作成とアルゴリズムの正当性
- 高速に動作するプログラムの作成とアルゴリズムの評価
 - なお、アルゴリズムとは、計算機の基本操作の有限個の組み合わせである。すわわち、機械的な手順で、有限であるもの。厳密には、チューリング機械やRAM(Random Access Machine)を用いて定義されるが、本講義では省略する。

アルゴリズムの解析

- 正当性

- 数学的証明 帰納法や背理法
- 実験的解析 実装とテスト

講義で解説

レポートで検証

- 速度の解析

- 数学的解析 O 記法による時間量解析
- 実験的解析 実装と時間計測

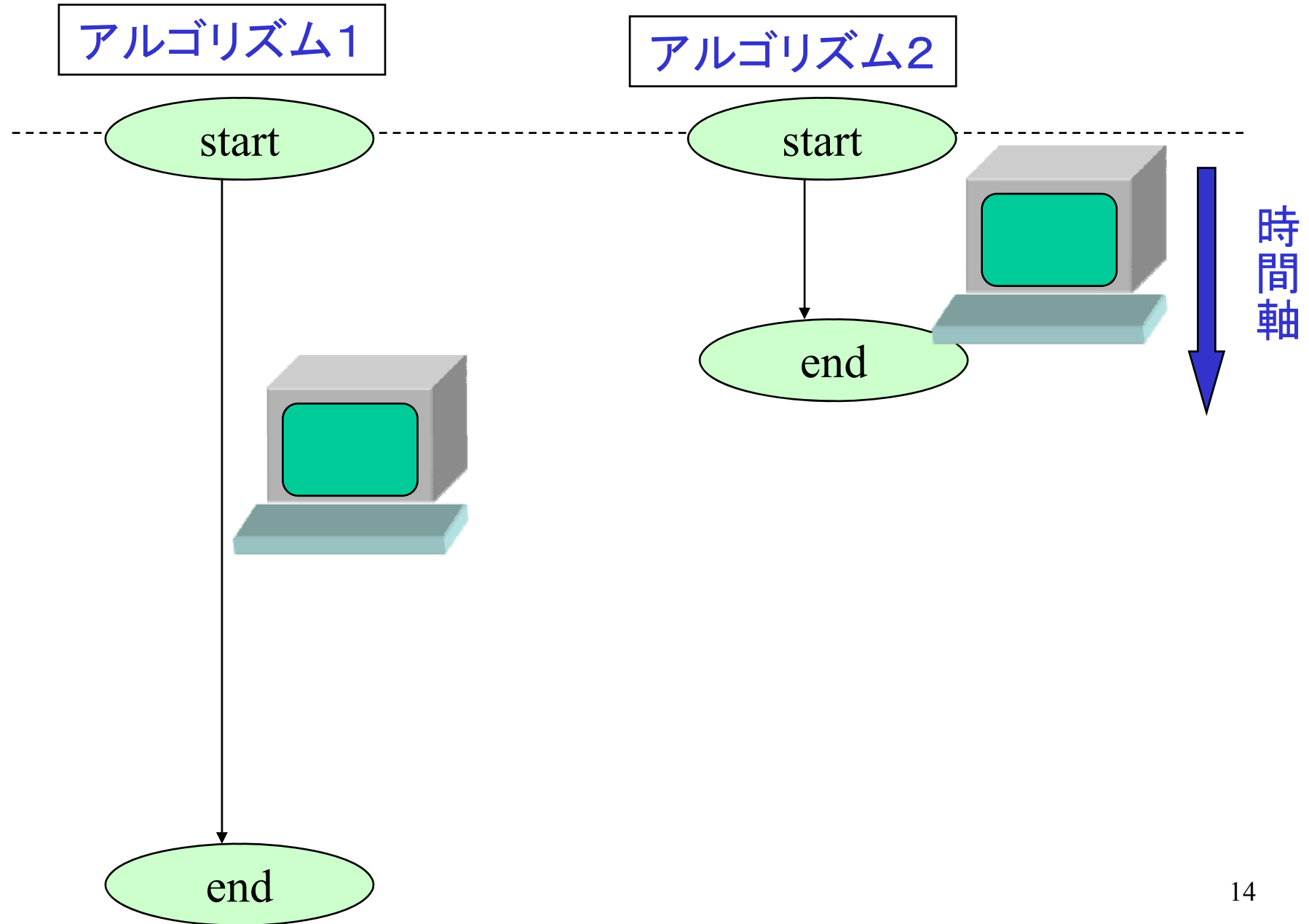
レポートで検証

アルゴリズムの計算量 (complexity)

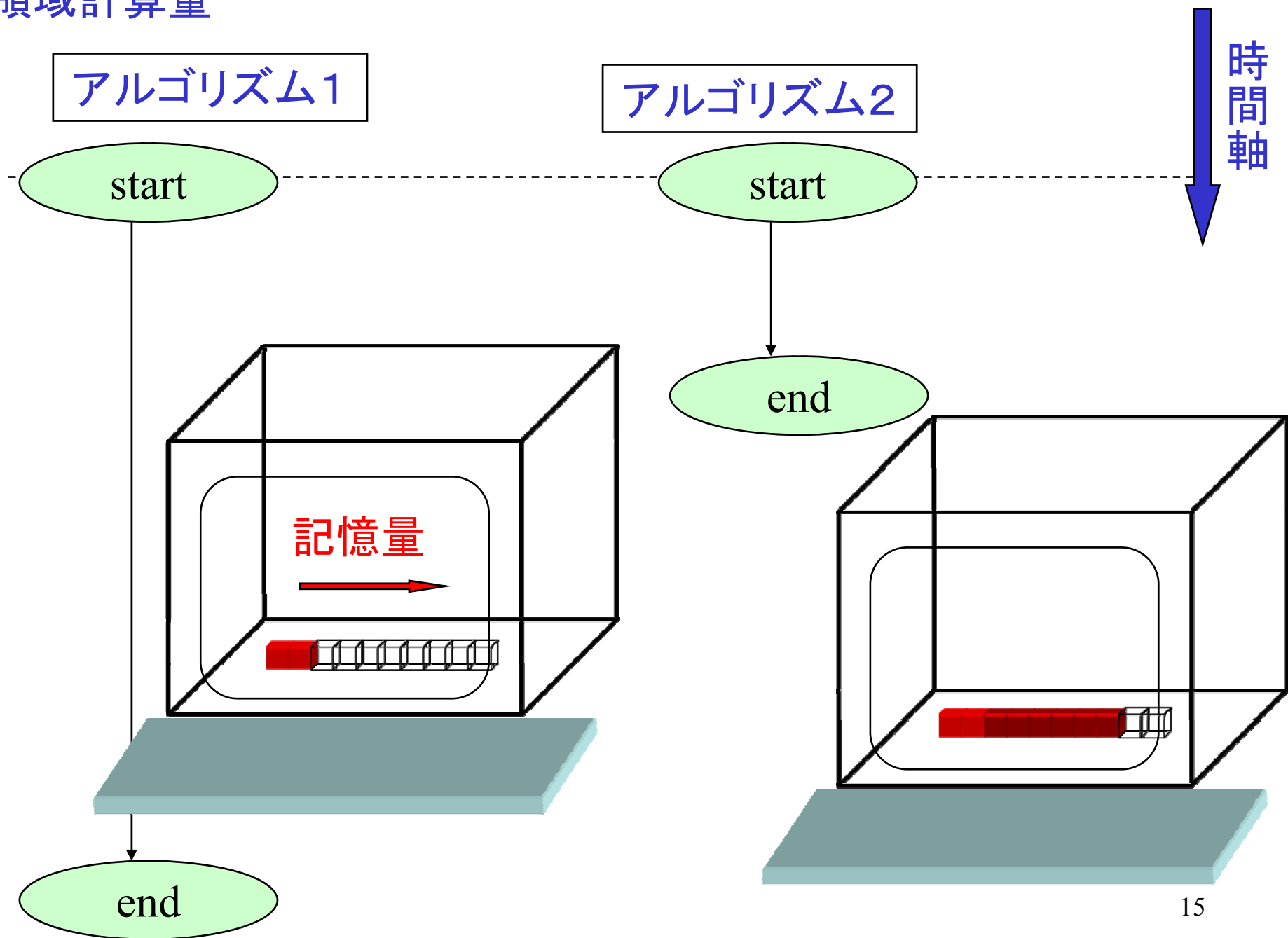
アルゴリズムの計算量1

- 時間計算量 (time complexity)
 - 総ステップ数 (基本演算の総数、アルゴリズムでは ∞ にはならない。)
 - 同じハードウェアでも速く実行できるプログラム作成のための指標。
- 領域計算量 (space complexity)
 - アルゴリズム実行時に、開始から終了までの間に使用するメモリやディスクなどの利用量
 - 記憶量ともいう。

時間計算量



領域計算量



アルゴリズムの計算量2

- 最大時間計算量

(worst case time complexity)

- 同じ入力サイズの問題に対して、最も遅く動作する場合を想定したときの時間計算量。
- 最悪計算量ともいう。

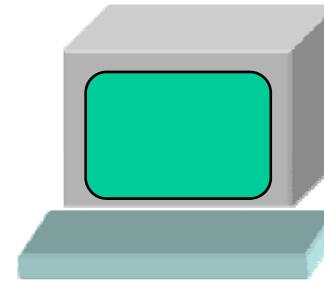
- 平均時間計算量

(average case time complexity)

- 同じ入力サイズの問題に対して、入力の分布を考えて、時間計算量を平均したもの。

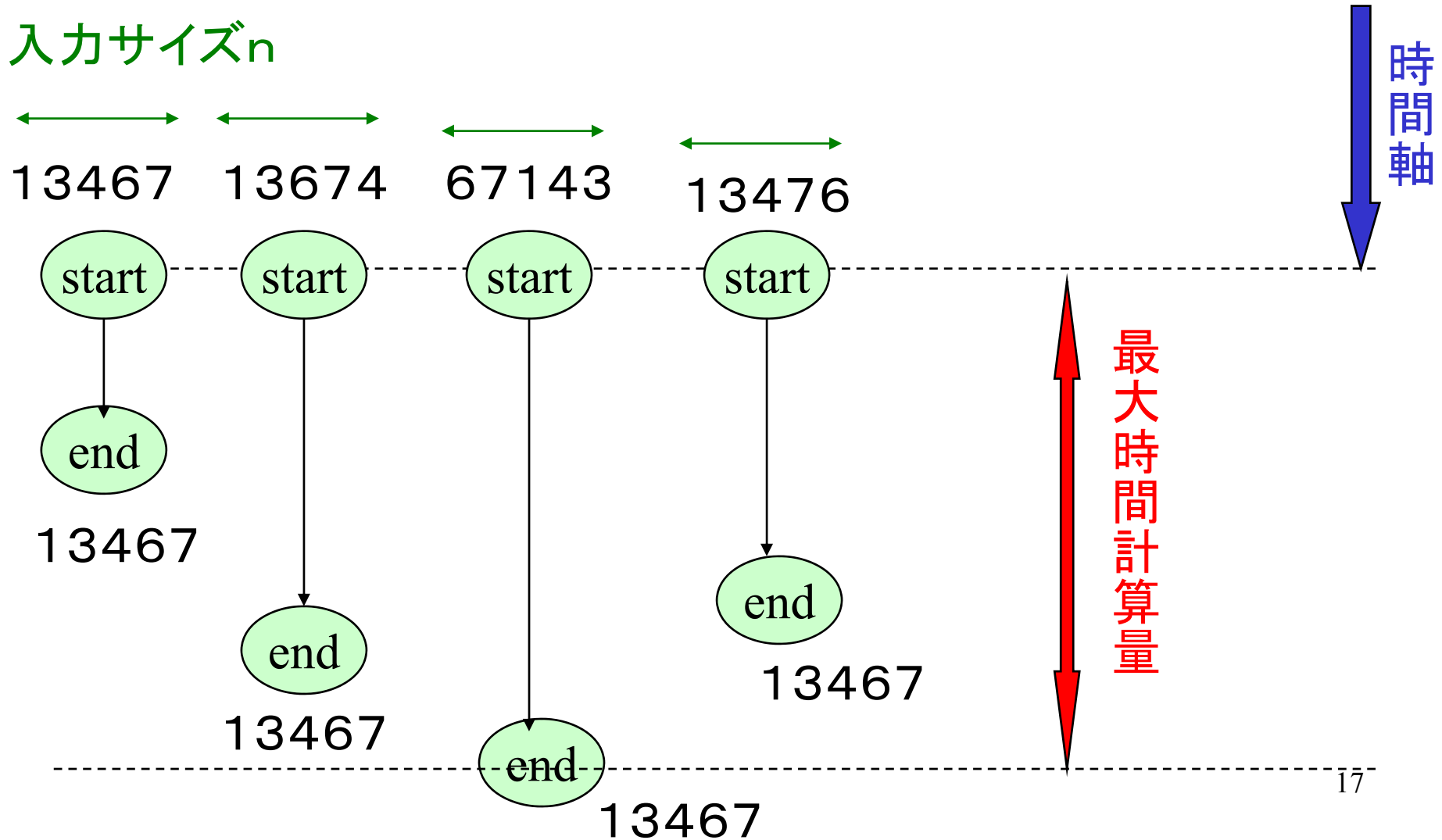
最大時間計算量

アルゴリズム1



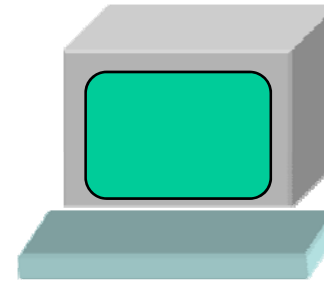
ソートアルゴリズム

入力サイズ n



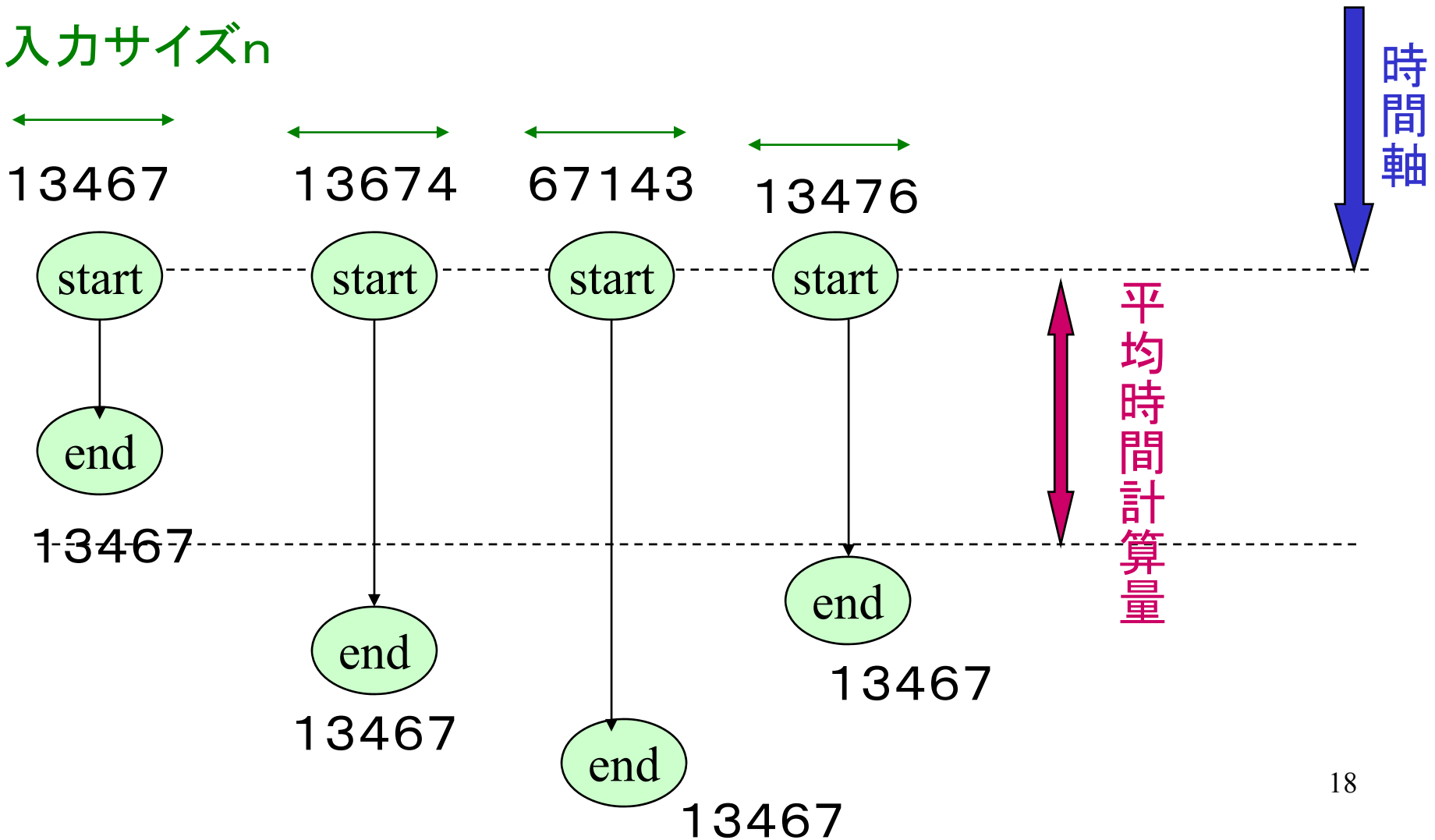
平均時間計算量

アルゴリズム1



ソートアルゴリズム

入力サイズ n



アルゴリズムの解析例

簡単なアルゴリズム例 (最大値を求める。)

アルゴリズムmax

```
1. big=A[0];  
2. for(i=1;i<n;i++){  
3.     if(A[i]>big){  
4.         big=A[i];  
5.     }  
6. }
```

1回の代入

n回の比較

n-1回の比較

最悪n-1回の代入

最大時間計算量 $T(n) = 3n - 1$ のアルゴリズム

アルゴリズム_{max}の正当性

次の命題を帰納法によって証明する。

命題1

forループが i 回実行されたとき、
 big には $A[0] \sim A[i]$ の最大値が保持されている。

証明

基礎

$i=0$

このときは、 big には $A[0]$ が保持されており、明らかに命題は成り立つ。

帰納

$i=k$ の時、命題1が成り立つと仮定する。(帰納法の仮定)
このとき、 $i=k+1$ を考える。

帰納法の仮定より、

$$\text{big} = \max \{A[0], A[1], \dots, A[k]\}$$

このとき、2つの場合に分けて考える。

場合1 $A[k+1] > \text{big}$ のとき。

このときは、アルゴリズム max の3. のif文の条件分岐が真なので、 $\text{big} = A[k+1]$ に更新される。

よって、 $k+1$ 回目の繰り返し終了時には、

$$\text{big} = \max \{A[0], A[1], \dots, A[k+1]\}$$

場合2 $A[k+1] \leq \text{big}$ のとき。

$$\max \{A[0], A[1], \dots, A[k+1]\}$$

$$= \max \{ \max \{A[0], A[1], \dots, A[k]\}, A[k+1] \}$$

$$= \max \{ \text{big}, A[k+1] \}$$

$$= \text{big}$$

どちらの場合も命題が成り立つ。

アルゴリズム max の停止性

次の命題を証明する。

命題2

forループの反復部分は、丁度 $n-1$ 回実行される。

証明

ループカウンタ i は1からはじまる。

また、ループカウンタ i が繰り返し事に1増加する。

ループカウンタが $i=n$ になったときには、ループの反復部分は実行されない。したがって、丁度 $n-1$ 回反復部分は実行される。

QED

命題1と命題2より、アルゴリズム max は正しいことがわかる。

漸近的解析

(Asymptotic Analysis)

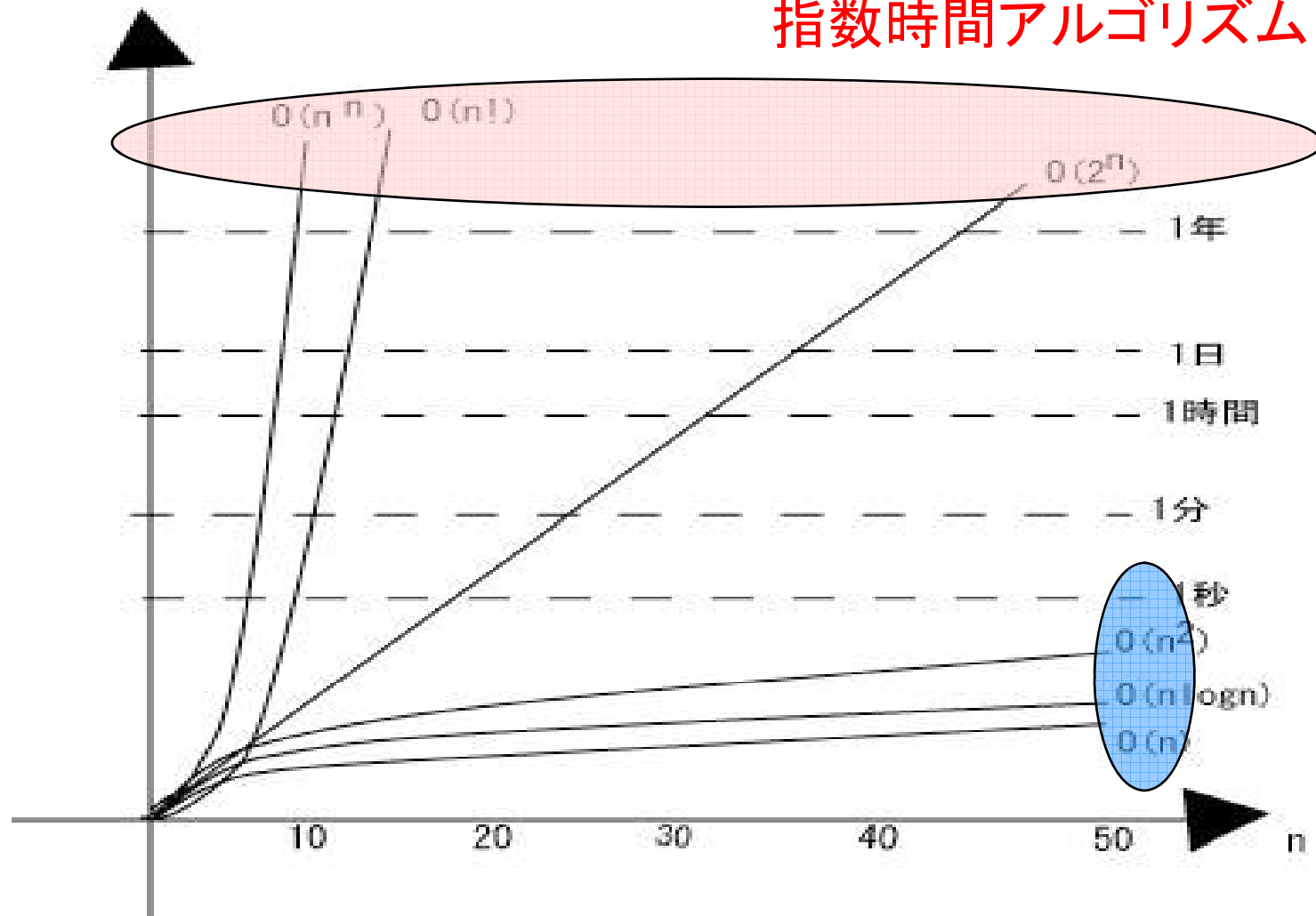
入力 サイズ	アルゴリズム	A n	B $n \log n$	C n^2	D n^3	E 2^n	F $n!$
$n = 10$		1.0×10^{-7}	3.3×10^{-7}	1.0×10^{-6}	1.0×10^{-5}	1.0×10^{-5}	0.04
15		1.5×10^{-7}	0.6×10^{-6}	2.3×10^{-6}	0.3×10^{-4}	3.3×10^{-4}	$3.6(h)$
20		2.0×10^{-7}	0.8×10^{-6}	0.4×10^{-5}	0.8×10^{-4}	1.1×10^{-2}	$772(y)$
30		3.0×10^{-7}	1.5×10^{-6}	0.9×10^{-5}	0.3×10^{-3}	11	
50		5.0×10^{-7}	2.8×10^{-6}	1.6×10^{-5}	1.3×10^{-3}	$4 \times 10^{14}(y)$	
100		1.0×10^{-6}	6.6×10^{-6}	1.0×10^{-4}	0.01		
500		5.0×10^{-6}	4.5×10^{-5}	2.5×10^{-3}	1.3		
1000		1.0×10^{-5}	0.1×10^{-3}	1.0×10^{-2}	10		
10000		1.0×10^{-4}	1.3×10^{-3}	1.0	$2.8(h)$		
10^5		1.0×10^{-3}	1.7×10^{-2}	100	$116(d)$		

100MIPSの計算機(1命令あたり 10^{-8} 秒)

単位: 秒(sec)

関数の漸近的ふるまい (関数の増加率による分類)

指数時間アルゴリズム



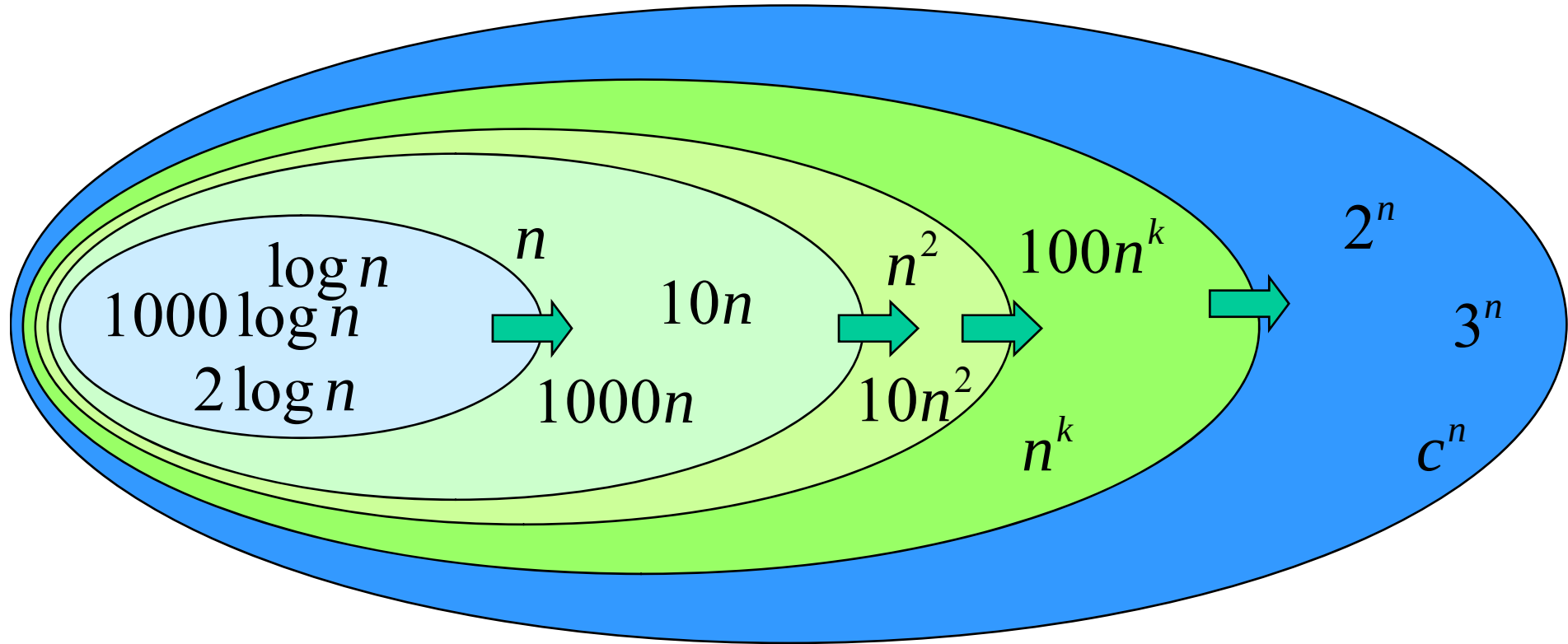
多項式時間
アルゴリズム

関数の分類1 (計算量の漸近的評価1):

オーダー記法

重要!

関数の増加傾向により、関数を大まかに分類したい。



$$O(\log n) \rightarrow O(n) \rightarrow O(n^2) \rightarrow O(n^k) \rightarrow O(c^n)$$

対数(時間)

多項式(時間)

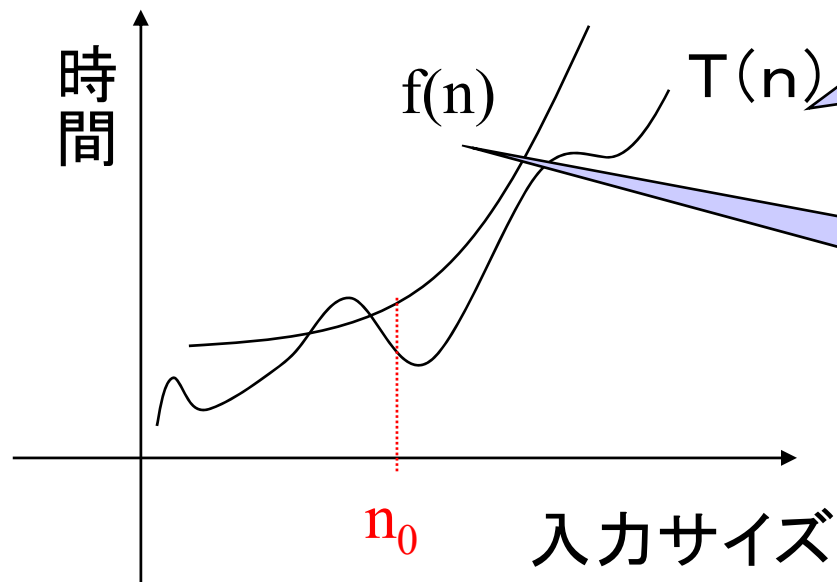
指数(時間)

定義: オーダー記法

ある関数 $f(n)$ に対して、計算量 $T(n)$ が $O(f(n))$ であるとは、
適当な2つの正定数 n_0 と c が存在して、 n_0 以上のすべての
 n に対して

$T(n) \leq cf(n)$
が成り立つことである。

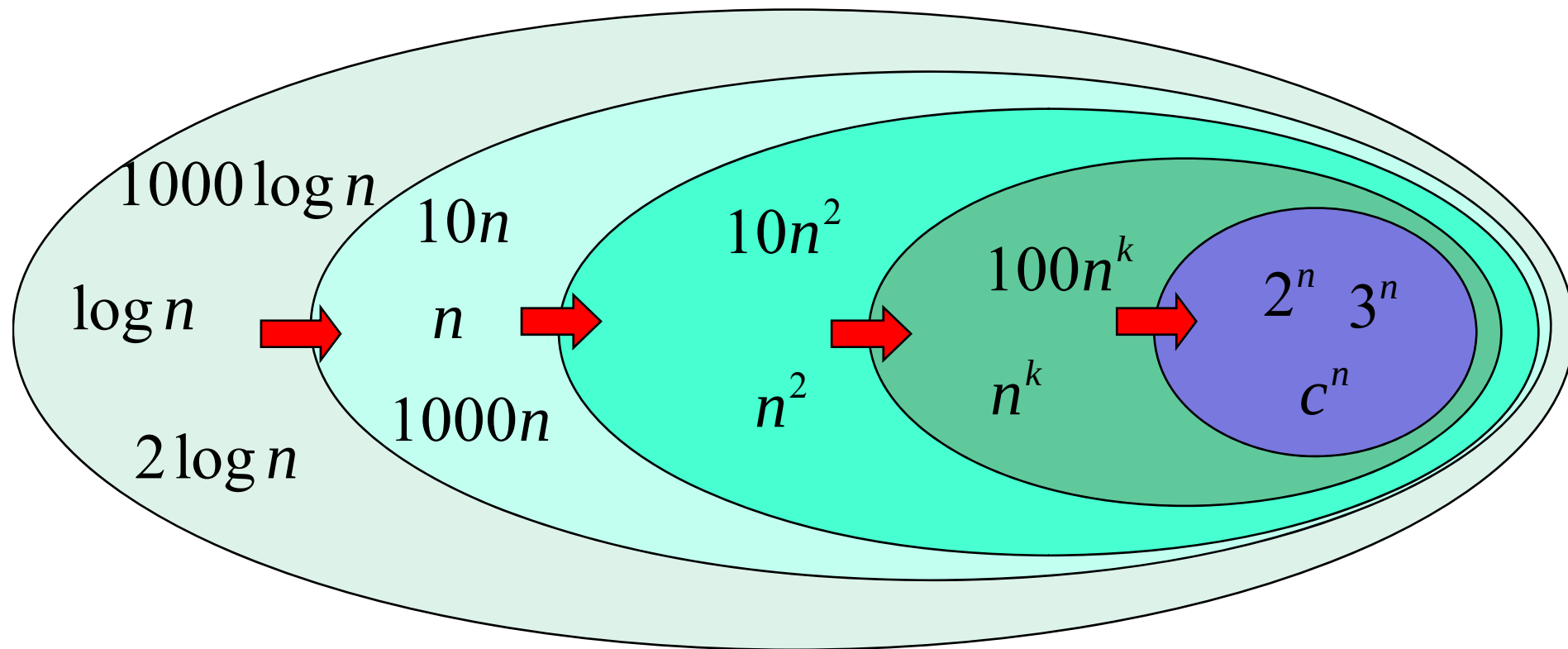
O-記法は“以下”を表す記法。計算時間の上界を見積もる。



実際の時間計算量は、
一般に複雑になることが多い。

O-記法を用いれば、
簡単な関数で時間計算量を見積もれる。

関数の分類2(計算量の漸近的評価2): オメガ記法



$$\Omega(\log n) \rightarrow \Omega(n) \rightarrow \Omega(n^2) \rightarrow \Omega(n^k) \rightarrow \Omega(c^n)$$

←
対数(時間)

←
多項式(時間)

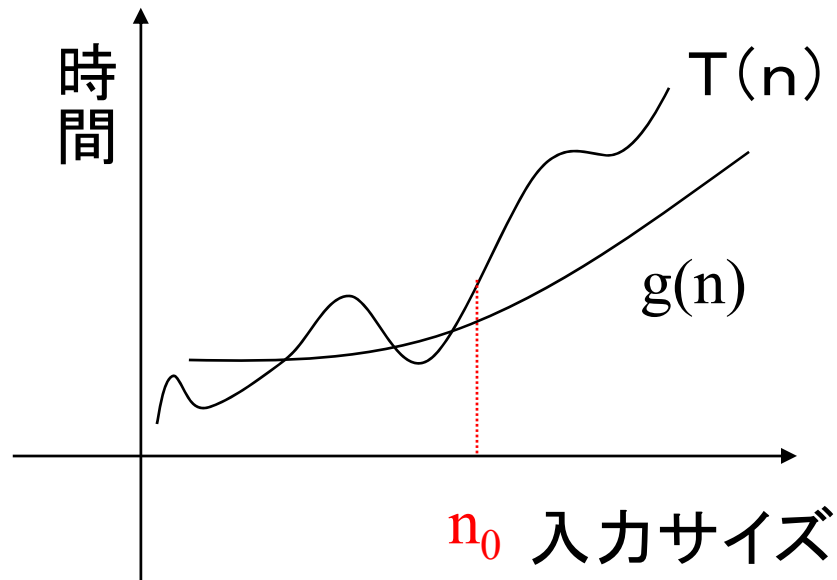
指数(時間)

定義: オメガ記法

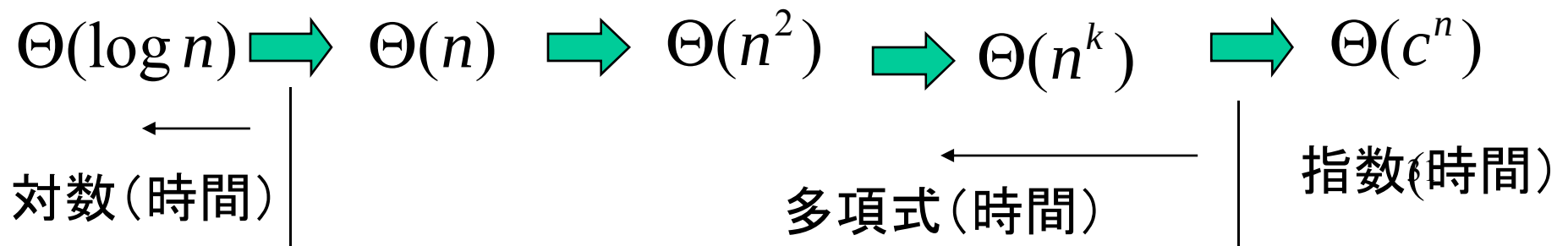
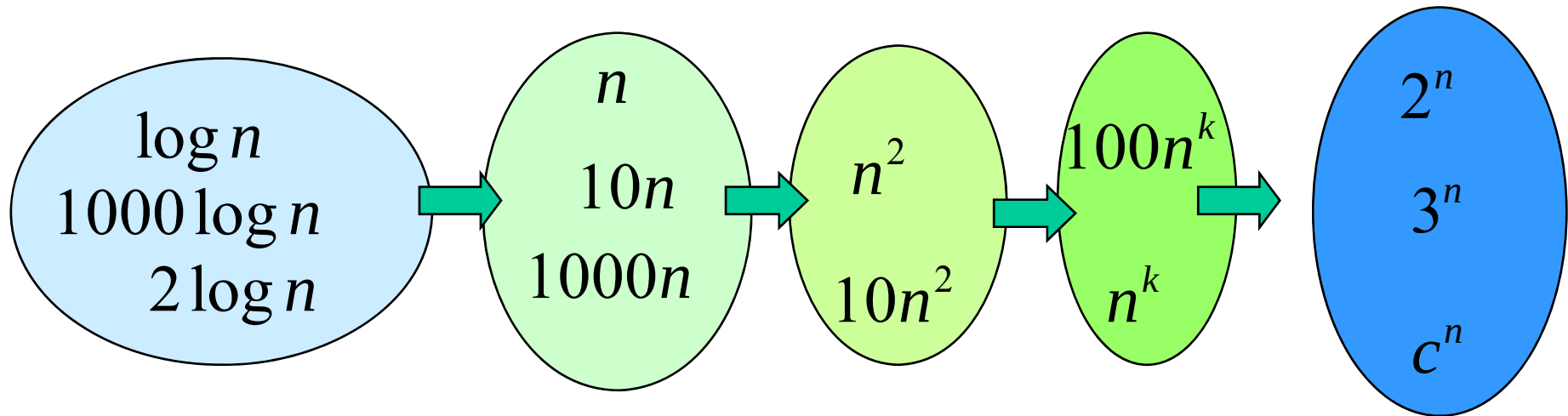
ある関数 $g(n)$ に対して、計算量 $T(n)$ が $\Omega(g(n))$ であるとは、
適当な2つの正定数 n_0 と c が存在して、 n_0 以上のすべての
 n に対して

$T(n) \geq cg(n)$
が成り立つことである。

Ω 記法は“以上”を表す記法。計算時間の下界を見積もる。



関数の分類3 (計算量の漸近的評価3): シータ記法

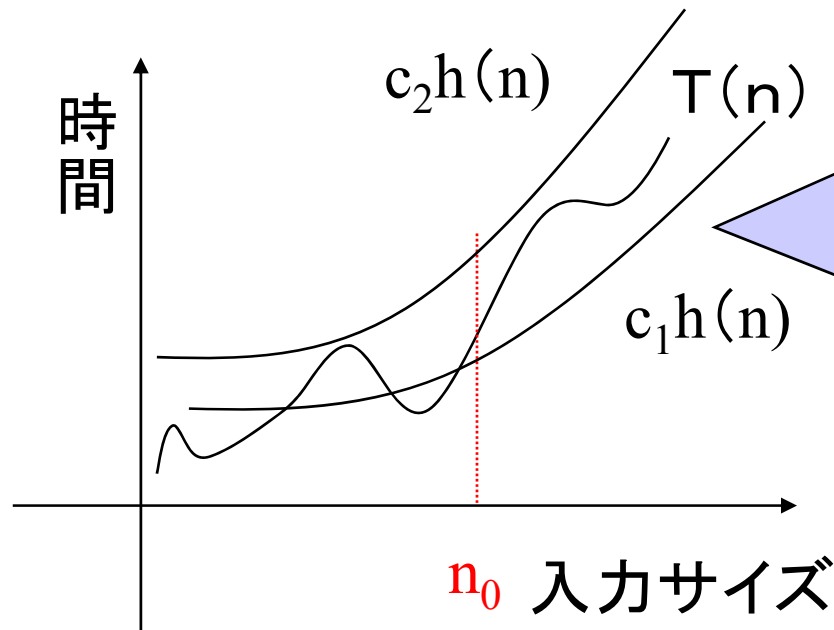


定義:シータ記法

ある関数 $h(n)$ に対して、計算量 $T(n)$ が $\Theta(h(n))$ であるとは、
適当な3つの正定数 n_0 、 c_1 、 c_2 が存在して、 n_0 以上のす
べての n に対して

$c_1 h(n) \leq T(n) \leq c_2 h(n)$
が成り立つことである。

Θ記法は“ほぼ等し
い”を表す記法。



Θ記法は漸近的な時間計算量を
定数倍の差の範囲で見積もれる。
Θ記法で表されるとき、その時間
計算量はタイト(tight)といわれる。

○記法の例

(1) $f(n) = 2n$

$n_0 = 1, c = 2, g(n) = n$ とすれば、

$n \geq 1$ に対して、 $f(n) \leq 2g(n)$

よって、 $2n = O(n)$

(2) $f(n) = 5n^2 + 100n$

$n_0 = 100, c = 6, g(n) = n^2$ とすれば、

$n \geq 100$ に対して、 $f(n) \leq 6g(n)$

よって、 $5n^2 + 100n = O(n^2)$

注意2: 通常○記法では、最も簡単な関数で表す。

○記法の例

$$(3) f(n) = 100000000000$$

$n_0 = 1, c = 100000000000, g(n) = 1$ とすれば、

$$n \geq 1 \quad \text{に対して、} \quad f(n) \leq cg(n)$$

よって、 $100000000000 = O(1)$

$$(4) f(n) = 2^n + n^{100}$$

$n_0 = 1000, c = 2, g(n) = 2^n$ とする。

$$2^{1000} = (2^{10})^{100} = (1024)^{100} > (1000)^{100} \quad \text{なので、}$$

$$n \geq 1000 \quad \text{に対して、} \quad f(n) \leq cg(n)$$

よって、 $2^n + n^{100} = O(2^n)$

○記法の練習問題

次の数列の一般項(関数)を○記法で表せ。

$$(1) f(n) = 10n - 500$$

$$(2) f(n) = 10n\sqrt{n} + 100000n^2$$

$$(3) f(n) = 1000 \log_2 n + 10 \log_{10} n$$

$$(4) f(n) = \sqrt{n} + \log_2 n$$

$$(5) f(n) = 2^n + 3^n$$

プログラムの漸近的評価

プログラムと漸近的評価

仮定1

プログラム内の加減算は、ある定数 C_1 時間以下で実行できる。

仮定2

プログラム内の乗除算は、ある定数 C_2 時間以下で実行できる。

仮定3

プログラム内の比較は、ある定数 C_3 時間以下で実行できる。

仮定4

プログラム内の代入は、ある定数 C_4 時間以下で実行できる。

・

・

・

プログラムでは、このように仮定できることが多い。

仮定1－4より、 $c > \max\{c_1, c_2, c_3, c_4\}$ なる C をとると、

プログラム内の4則演算、比較等はある定数時間以下で実行できる。

つよめて

プログラム内では、
繰り返し構造、
(再帰関数を含む)関数呼び出し、
以外は定数時間で実行できると仮定できることが多い。

プログラムにおける計算時間の漸近評価例

```
function1()  
{  
    for(k=0;k<n;k++)  
    {  
        ....  
    }  
}
```

forループは、この部分
だけで漸近時間計算
量が見積もれる。

この部分がn回
実行されることに
注意する。

function1の計算時間は、 $O(n)$
である。

プログラムにおける計算時間の漸近評価例2

```
function2()
```

```
{
```

```
    for(k=0;k<n;k++)
```

```
    {
```

```
        for(j=0;j<n;j++)
```

```
        {
```

```
            ....
```

```
        }
```

```
    }
```

```
}
```

この部分は n 回
実行されることに
注意する。

この部分は n^2 回
実行されることに
注意する。

この部分は n 回
実行されることに
注意する。

function2の計算時間は、 $O(n^2)$ である。

プログラムにおける計算時間の漸近評価例3

```
function3()
{
    for(k=0;k<n;k++)
    {
        for(j=0;j<k;j++)
        {
            .....
        }
    }
}
```

外側のループカウンタが、内側のループ回数に影響を与える。一見、 n と無関係に見える。

function3の計算時間 $f(n)$ を評価する。。

$$f(n) = c(1 + 2 + 3 + \dots + n) = c \frac{n(n+1)}{2} = O(n^2)$$

プログラムにおける計算時間の漸近評価例4

```
int function4(int n)
{
    if(n < 1)
    {
        return(0);
    }
    else
    {
        function4(n-1);
    }
}
```

function4の計算時間 $f(n)$ を評価する。

$$\begin{cases} f(1) = c_1 \\ f(n) = f(n-1) + c_2 \end{cases}$$

この漸化式より、

$$f(n) = O(n) \text{ である。}$$

再帰関数の時間計算量は、見た目では分かりにくい。

プログラムにおける計算時間の漸近評価例5

```
int function5(int n)
{
    if(n < 1)
    {
        return(0);
    }
    else
    {
        function5(n/2);
    }
}
```

function5の計算時間 $f(n)$ を評価してみましょう。

$$\begin{cases} f(1) = c_1 \\ f(n) = f\left(\frac{n}{2}\right) + c_2 \end{cases}$$

この漸化式より、

$f(n) = O(\log n)$ である。

プログラムにおける計算時間の漸近評価例6

```
int function6(int n)
{
    if(n<1)
    {
        return;
    }
    else
    {
        function6(n-1);
        function6(n-1);
    }
}
```

function6の計算時間 $f(n)$ を評価してみましょう。

$$\begin{cases} f(1) &= c_1 \\ f(n) &= f(n-1) + f(n-1) + c_2 \\ &= 2f(n-1) + c_2 \end{cases}$$

この漸化式より、
 $f(n) = O(2^n)$ である。

プログラムにおける計算時間の漸近評価練習1

次のプログラムの計算時間を O 記法で求めよ。
ただし、入力サイズは仮引数 n に入っている数とする。

(1)

```
exercise1(int n)
{
    for(j=0;j<n;j++)
    {
        for(k=0;k<n;k++)
        {
            .....
        }
    }
    for(l=0;l<n;l++)
    {
        x x x x
    }
}
```

(2)

```
exercise2(int n)
{
    if(n<2)
    {
        return;
    }
    else
    {
        exercise2(n-1);
        exercise2(n-2);
    }
}
```

アルゴリズムの入力について

- ・問題と問題例
- ・入力サイズ

問題と問題例

(problem and problem instances)

問題：現実の問題を定義したもの。

同じような入力と出力の関係を定めたもの。

- ・でたらめに並んだ数値を順番にならべる。
 - ソート問題(入力:でたらめな列、
出力:順序列)
- ・2つの数字の最大公約数を求める。
 - gcd問題(入力:2つの整数、
出力:1つの最大公約数)
- ・数の集合から最大値を求める
 - 最大値問題(入力:数の集合、
出力:入力中の最大値)

...

問題例：具体的に数値を与えたもの。

問題は、問題例の集合としてとらえられる。

・ソート問題例

3 4 2 8 7 → 2 3 4 7 8

1 2 9 7 3 5 6 → 1 2 3 5 6 7 9

4 2 → 2 4

7 1 3 8 → 1 3 7 8

問題

ソート問題

3 4 2 8 7

7 1 3 8

4 2

1 2 9 7 3 5 6

問題例

・最大值問題

3 4 2 8 7 → 8

1 2 9 7 3 5 6 → 9

4 2 → 4

7 1 3 8 → 8

問題

最大值問題

3 4 2 8 7

7 1 3 8

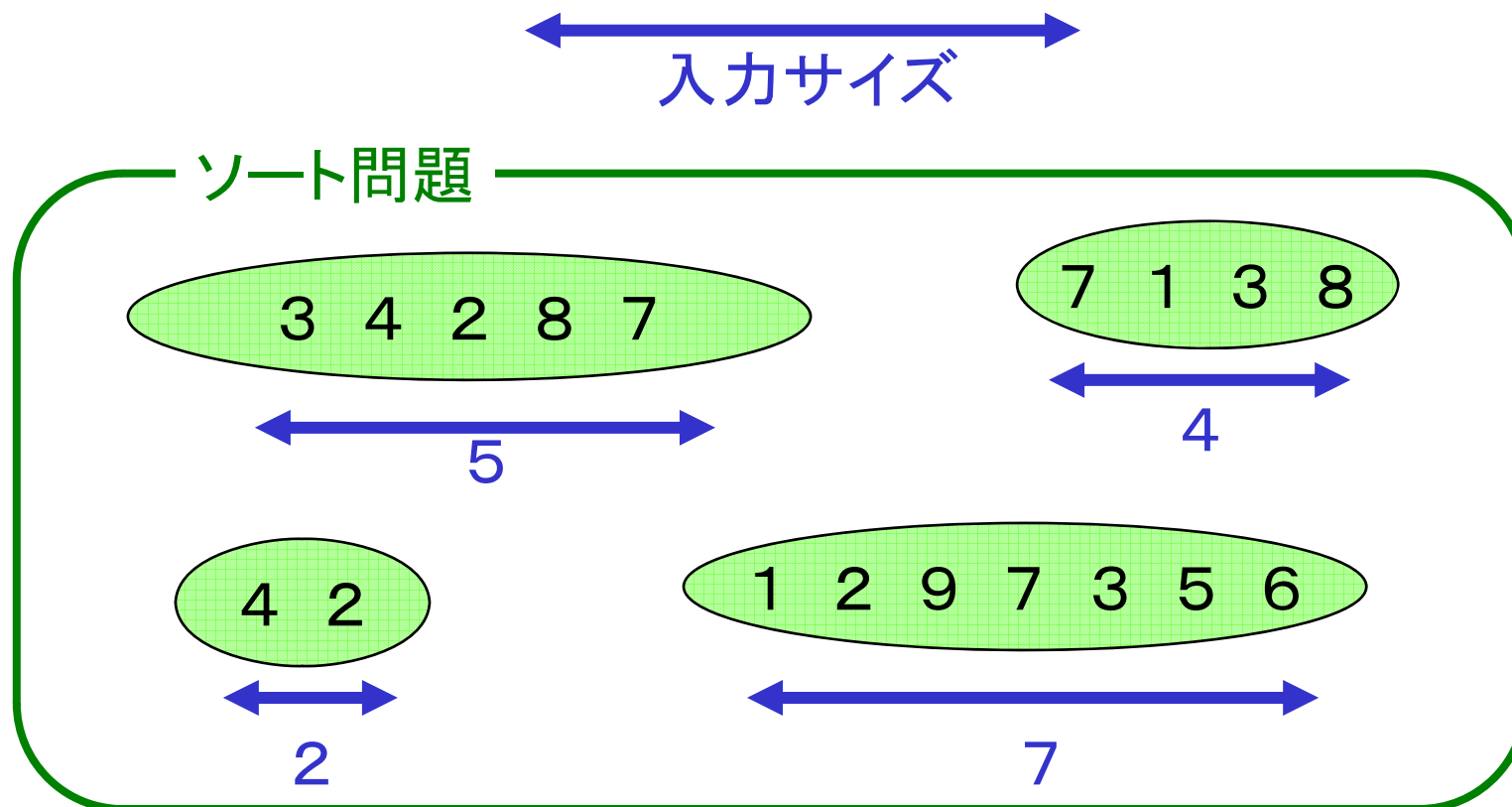
4 2

1 2 9 7 3 5 6

問題例

入力サイズ

入力を計算機で表現するときの大きさ。
一つ問題例を定めると入力サイズも定まる。



本講義では
主にこの基準を用いる

- 一様コスト基準（一様コストモデル）
どの数の計算も一定時間（定数時間）できるとき
（一つの数の入力サイズは1）
- 対数コスト基準（対数コストモデル）
数の表現を桁数まで考えて数を扱う。
桁の大きい数同士の計算は大変なので。
（数 a の入力サイズは $\log a$ ）

この基準を用いるときは、
その都度ことわる

対数コストモデルについて (計算機内での数の表現と桁数)

10進数 $d_n d_{n-1} \cdots d_0$ $d_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$



2進数 $b_m b_{m-1} \cdots b_0$ $b_i \in \{0, 1\}$

上のように相互変換されるとき、

$$m = O(n)$$

である。

n : 10進数での桁数

m : 2進数での桁数

証明

$$A \equiv d_n \times 10^n + d_{n-1} \times 10^{n-1} + \cdots + d_0 = b_m \times 2^m + \cdots + b_0$$

$2^m \leq A < 2^{m+1}$ に注意して、底2の対数をとる。

$$\log_2 2^m \leq \log_2 A < \log_2 2^{m+1}$$

$$\therefore m \leq \log_2 A < m + 1$$

また、 $10^n \leq A < 10^{n+1}$ に注意して、底2の対数をとる。

$$\log_2 10^n \leq \log_2 A < \log_2 10^{n+1}$$

$$\therefore n \log_2 10 \leq \log_2 A < (n + 1) \log_2 10$$

$c \equiv \log_2 10$ とおく。

$$m \leq \log_2 A < (n + 1) \log_2 10 = c(n + 1)$$

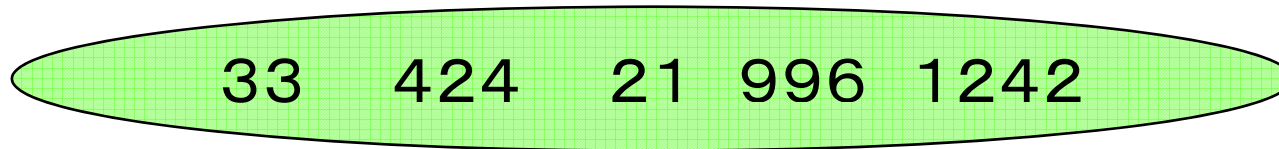
$$\therefore m < c(n + 1)$$

・最大値問題の入力サイズ

33 424 21 996 1242 → 1242

一様コスト基準
5

本講義では、
主にこちらを用いる。



$$2 + 3 + 2 + 3 + 4 \\ = 14$$

対数コスト基準