制御情報システム工学科4年
選択科目：プログラミング特論

プログラミング特論

第3回

第3章　アルゴリズムの威力

アルゴリズムの威力

第3章の概要

- ある目標を定めてそれを達成しようとするとき，
  得られる結果は同じでも，効率の良し悪しなど，
  いくつかの方法が存在．
- 古典的な例を使ってアルゴリズムについて学ぶ．
- 簡単なプログラムを作り，アルゴリズムの違いによる
  計算時間への影響について体験．
- アルゴリズム
  - 何らかの目的を達成するための一連の計算の手続き（方法）．
  - 特定のプログラミング言語には依存しない方法で記述．
- アルゴリズムの実装
  - 実際にコンピュータでアルゴリズムを動かすために
    プログラムを書くこと．

3.1　アルゴリズムと実装

- 3.1.1　最大公約数
- 3.1.2　約数を調べ上げる方法
- 3.1.3　アルゴリズムと実装
- 3.1.4　素因数分解を使った方法
- 3.1.5　ユークリッドの互除法
- 3.1.6　再帰を使った実装
- 3.1.7　アルゴリズムの比較

3.1.1　最大公約数

- 2つの自然数（1以上の整数）の公約数の中で，最大のもの．
- 2つの数が等しければ，結果はその数そのもの．
- 2つの数が互いに素ならば，結果は1．

- Pythonを使う場合
  - import math
  - math.gcd(18915, 14938)

- 関数を使わない場合，どのような手続きで計算可能か？

3.1.2　約数を調べ上げる方法

- すぐに思いつく方法は，約数を調べ上げる方法.

- アルゴリズムを記述する際によく使われる方法で書く.

アルゴリズム 3.1　最大公約数を求める方法１（自然言語）

```
1  入力： 2つの自然数a,b
2  出力：最大公約数
3  手続き：
4  1. 変数x に b を代入
5  2. a と b を x で割り，両方割り切れたら x を出力して終了
6  3. x から 1 を引き，手続き 2 へ戻る
```

- プログラムのコードにしやすい表現で書く習慣.
- 手続きの各ステップは自然言語で書かれるため，
  プログラミング言語での表現とは違い，曖昧さが残る.

- 疑似コード
  - 特定のプログラミング言語に依存しないが，
  - かなりプログラムのコードに近い形で書かれたもの．
  - アルゴリズムについての解説本では，一般的な書き方．

アルゴリズム 3.2　最大公約数を求める方法 1　（擬似

```
1  Input : two positive integers a,b
2  Output : greatest common divisor
3  Procedure :
4  1. x := b
5  2. if mod(a,x) == 0 and mod(b,x) == 0 then return x
6  3. x := x - 1 goto 2
```

コード 3.3　最大公約数を求める方法 1　（Python コード）

```
1  def simple_gcd(a, b):
2      if a < b:
3          a, b = b, a
4      x = b
5      while True:
6          if a % x == 0 and b % x == 0:
7              return x
8          x -= 1
```

  - 疑似コードの書き方にはいくつかの流儀が存在．
    - 例）変数への代入　“:=”　もしくは　“矢印”
  - 以後，アルゴリズムは，自然言語で記述，
    もしくはPythonコードで表現．

約数を数え上げる方法の計算量

- 計算量とは？
  - 問題の入力サイズに対して，
    計算のステップ数がどのように変化するかを示すもの．
  - 入力サイズは，２つの数のうち小さいほうをそのまま採用．
    ⇒　nとする．
- 計算の１ステップは？
  - コード3.3より，aとbをそれぞれxで割った余りが
    0かどうかを比較する計算の繰り返し．
    ⇒　割り算１回を１ステップと捉える．
- 最悪の場合
  - aとbが互いに素で最大公約数が1になるとき．
  - 計算のステップは合計で$2n$回繰り返し．
  - 漸近記法により定数倍は無視されるため，計算量は$O(n)$

3.1.3　アルゴリズムと実装

- アルゴリズムを設計する立場で重要なこと
  - アルゴリズムが正しく実行され（答えが正しく），
    必ず停止するかどうかを確認.
- アルゴリズムには実行のための前提条件が含まれる.
  - 例）コード3.3の場合
    2つの自然数（1以上の整数）を入力として受け取ること.
- Pythonの実装では，関数の引数にデータ型を
  明示する必要なし.
- 負の整数や小数または文字列を引数に与えて実行可能.
  ⇒　エラーの原因

- 業務や研究でアルゴリズムを実装する場面では，
  「引数に対する前提条件をコメントに書く」，
  「引数の妥当性をチェックするコードを追加する」
  などの配慮が必要.

課題1：約数を調べ上げる方法の実装

- アルゴリズム3.1をPythonで実装しなさい.
- 関数名は，simple_gcdとする.
- 注意
  - 教科書P.043の「コード3.3　最大公約数を求める方法1（Pythonコード）」を見ないこと.

アルゴリズム 3.1　最大公約数を求める方法 1 （自然言語）

```
1  入力： 2つの自然数a,b
2  出力：最大公約数
3  手続き：
4  1. 変数x に b を代入
5  2. a と b を x で割り，両方割り切れたら x を出力して終了
6  3. x から 1 を引き，手続き 2 へ戻る
```

3.1.4　素因数分解を使った方法

- 最大公約数を手計算で求めようとする場合
  - 与えられた数を素因数分解して，
    共通の素因数をすべて掛け合わせるやり方
- 素因数分解
  - 与えられた自然数を素数の席で表現
  - 例）$12 = 2^2 \times 3$

アルゴリズム 3.4　最大公約数を求める方法２（素因数分解）

1. 入力：２つの自然数
2. 出力：最大公約数
3. 手続き：
4. 1. 入力されたそれぞれの数を素因数分解する
5. 2. それぞれの数に共通する素因数を列挙する
6. 3. 手続き 2で得られた素因数をすべて掛け合わせて出力する

【手続き1】
18915 = 3 x 5 x 13 x 97
14938 = 2 x 7 x 11 x 97
【手続き2】
共通する素因数は97
【手続き3】
97を結果として出力

課題2：素因数分解を使った方法の実装

- アルゴリズム3.4を理解し，
  素因数分解を使った方法をPythonで実装しなさい．
- 関数名は，PrimeFactorization_gcdとする．

アルゴリズム 3.4　最大公約数を求める方法 2 （素因数分解）

1. 入力： 2つの自然数
2. 出力：最大公約数
3. 手続き：
4. 1. 入力されたそれぞれの数を素因数分解する
5. 2. それぞれの数に共通する素因数を列挙する
6. 3. 手続き 2で得られた素因数をすべて掛け合わせて出力する

- 似ているようで大きく違うこと
  - アルゴリズムとして記述できること
  - プログラムとして実装できること
- 例）アルゴリズム3.4の手続き1

  「入力された数を素因数分解する」
  - それほど簡単に実装できる問題ではない.
    - Pythonで素因数分解を実行したい場合，
      外部パッケージのSymPyをインストールするとsympy.f actorintが使用可能.
  - 自然言語で書くと1行で表現可能.
  - プログラムにしようとするといろいろと考慮が必要.

- 理想は，

  「簡単に実装できて，実行速度が速いアルゴリズム」

- 約数を調べ上げず，素因数分解もせずに，

  最大公約数を求める方法はないか？

3.1.5　ユークリッドの互除法

- 紀元前300年ごろに考案されたと伝えられており，
  現在知られているアルゴリズムの中でも最古のもの．

プログラムのコードとしても簡単に実装可能．

アルゴリズム 3.5　最大公約数を求める方法３（ユークリッドの互除法）

```
1  入力： 2つの自然数a,b
2  出力：最大公約数
3  手続き：
4  1. a を b で割った余りを r とする
5  2. r が 0 ならば， b を出力して終了
6  3. a に b を， b に r を代入して手続き 1 へ戻る
```

課題3：ユークリッドの互除法の実装

- アルゴリズム3.5をPythonで実装しなさい.
- 関数名は，euclidean_algorithmとする.
- 注意
  - 教科書P.045の「コード3.6　ユークリッドの互除法」を見ないこと.

アルゴリズム 3.5　最大公約数を求める方法３（ユークリッドの互除法）

```
1  入力： 2つの自然数a,b
2  出力：最大公約数
3  手続き：
4  1. a を b で割った余りを r とする
5  2. r が 0 ならば，b を出力して終了
6  3. a に b を，b に r を代入して手続き 1 へ戻る
```

ユークリッドの互除法の計算過程

- 2つの数aとbの最大公約数が，
  図では1つのブロックとして表現．
- a，bともにこのブロックの自然数倍で表現可能．
- 割り算の余りもこのブロックの自然数倍でできている．

- 計算過程
  - 黒矢印と赤矢印の順で交互に進行．
  - 最初の割り算の余りがr1で，
    以後r2，r3と続行．
  - 割り算を繰り返していくと，
    余りはどんどん縮小．
  - r3でブロック1つ分になり，
    r2をr3で割ると余りが出ない．
    ⇒　r3が求める最大公約数

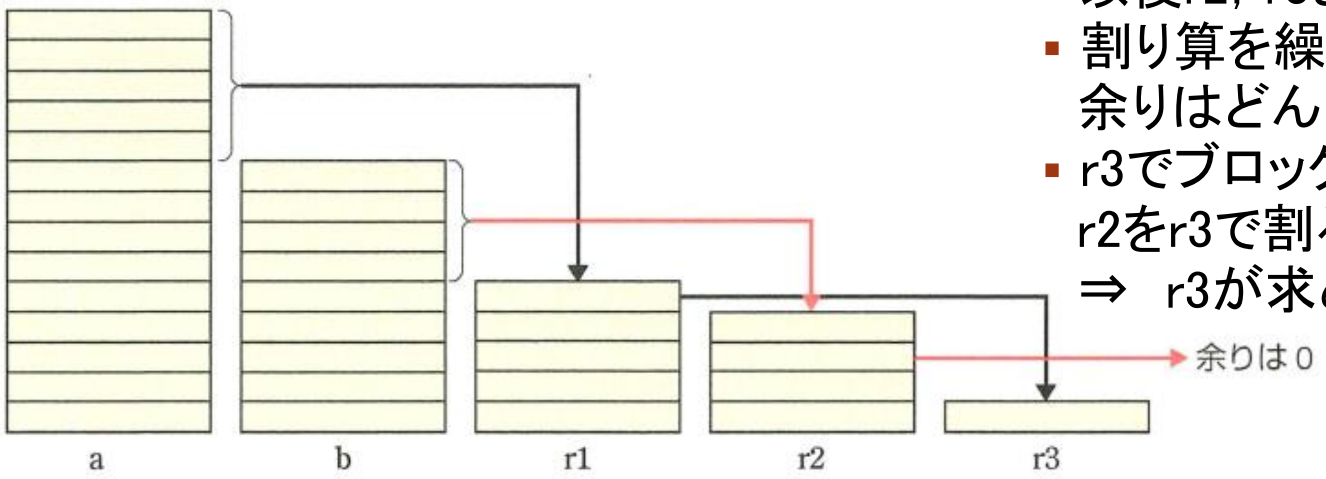


図 3.1　ユークリッドの互除法の原理

ユークリッドの互除法の計算量

- 入力のサイズは数そのものとして，便宜的にnとする．
- 計算の1ステップ
  - 割り算をして余りを求める計算．
- ユークリッドの互除法の計算
  - aの余りが小さくなっていく系列（図3.1中黒矢印）と
    bの余りが小さくなっていく系列（図中赤矢印）から構成．
- aの余りが小さくなっていく系列に注目．
  - 一般にu をvで割って得られる余りrは，$r < \frac{u}{2}$を満たす．
  - 計算が進むと，入力された数は少なくとも半分より小さくなり，
    次の計算ステップに渡される．
- 2章で学んだ2進数の計算
  - 2進数においては数が半分になる，つまり2で割るという行為は，
    桁が1つ減ることを意味．
- 数nが2進数で何桁あるかは，$\log_2 n$で表現可能．
- 最悪でも$\log_2 n$回計算すれば，最大公約数が求まる．
- ユークリッドの互除法の計算量は，$O(\log n)$

3.1.6　再帰を使った実装

- ユークリッドの互除法は，
  関数の再帰的な呼び出しを使っても実装可能.

- 似た処理を繰り返す場合，
  ループを再帰で置き換えるとコードがすっきりして
  見やすくなることがある.

- 余りが0になるまで，「関数の1つ目の引数を
  2つ目の引数で割る」という作業を繰り返す．
- この繰り返し処理をwhile文ではなく，
  自分自身を呼び出すことで表現．
- 再帰を使った関数は，値を返すreturn文の実行が必要．
  - この部分にバグが含まれていると，
    再帰呼び出しが永遠に終わらず無限ループに陥る．
- 慣れないうちは再帰を使わない実装で，
  まずプログラミングの技術を磨くことをすすめる．
- 再帰を使った実装の計算量
  - 計算量はアルゴリズム3.5と同じ，$O(\log n)$

コード3.7　ユークリッドの互除法（再帰）

```
1  def euclidean_recursion(a, b):
2      r = a % b
3      if r == 0:
4          return b
5      return euclidean_recursion(b, r)
```

3.1.7 アルゴリズムの比較

- 計算量の側面から，「約数を調べ上げる方法」と「ユークリッドの互除法」を比較．
- 横軸に入力のサイズnを，縦軸に見積もられた計算時間をプロット．

処理すべきデータサイズが計算の各ステップで半分になると，計算時間が$\log n$になる．

- 約数を調べ上げる方法（青）
  - 入力のサイズに対して線形に計算時間が増大．
  - 計算の各ステップを実行したあと，残りの計算回数が1つしか減らないため．
- ユークリッドの互除法（橙）
  - 1回の計算ステップが終了すると，次のステップに送られる数は最悪の場合でも処理前の数の約半分．
  - 入力サイズが大きくなっても，計算量の増大が抑えられる要因

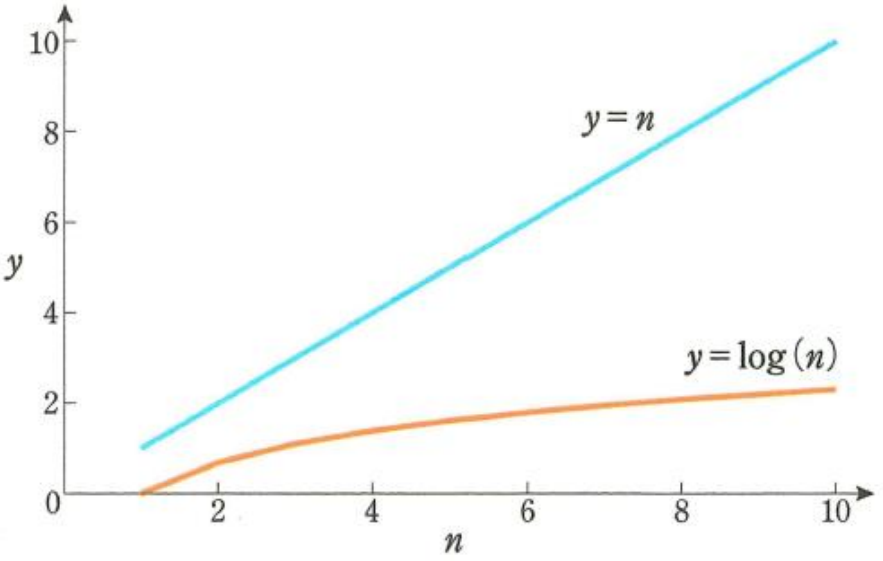


図 3.2　$\mathcal{O}(n)$ と $\mathcal{O}(\log n)$ の比較

- 図3.2 は，入力の数を1～10 まででプロット
- 入力の数を10から1×10^7(1000万)までの範囲に変更.
- 計算量が線形で増加する場合と，対数になる場合では増加のスピードがまったく違う.

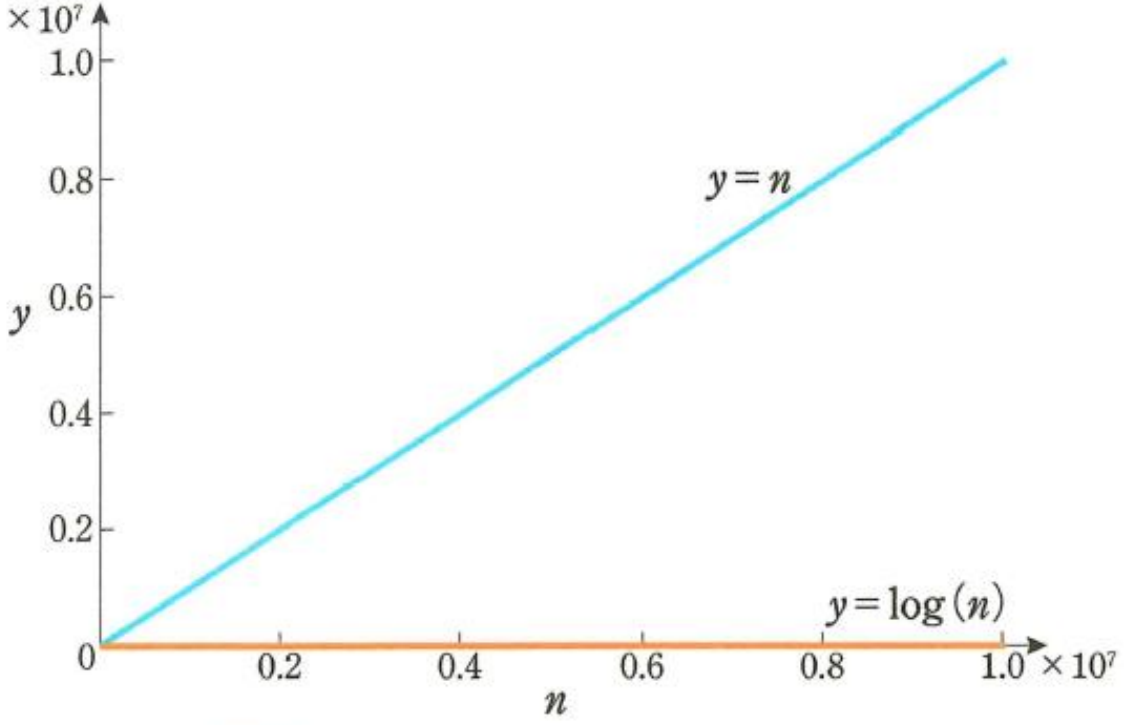


図 3.3　n を 10 から 1000 万にしたプロット

実際の計算時間を計測

- 10から1×10^7までの整数からランダムに2つを選び，
  約数を調べ上げる方法（青）とユークリッドの互除法（橙）で
  最大公約数を求める計算を200回実施．
- 横軸は入力となる数字のうち小さいほう，
  縦軸は計算にかかった時間（秒）．
- 計算は1組のデータに
  対して3回実行．

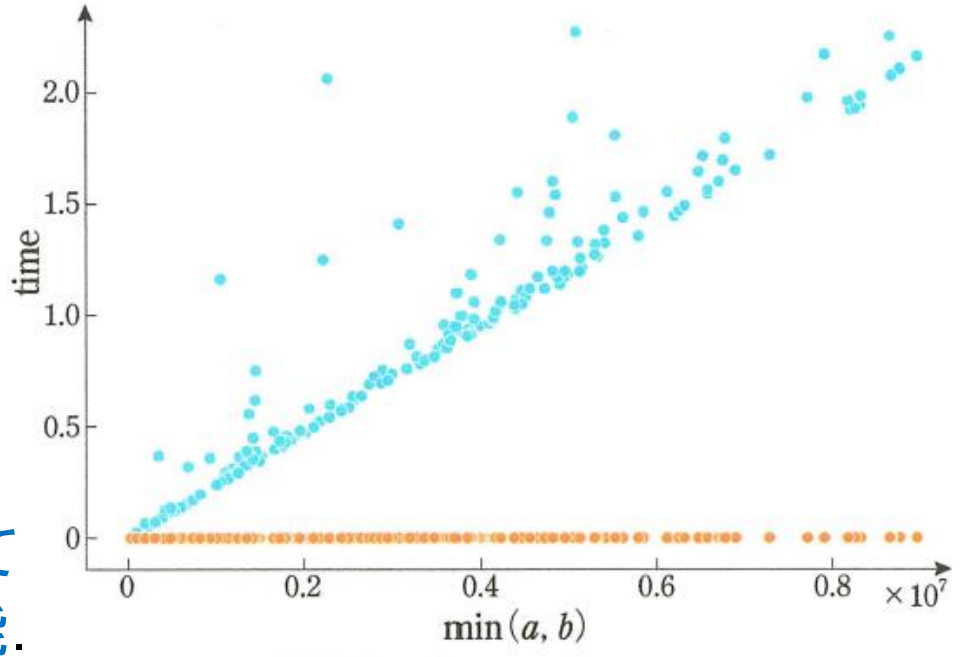


図 3.4　実際の計算時間の比較

同じ問題に対する解法でも，
アルゴリズムの違いによって
計算時間を大幅に短縮可能．

3.2　配列のソート

- 3.2.1　要素の並べ替え
- 3.2.2　選択ソート
- 3.2.3　選択ソートを実装する
- 3.2.4　選択ソートの計算量
- 3.2.5　速度の比較

整列（ソート）とは

- まとまった数の数値や文字列を順番に並べ替える作業
- 並べ替え
  - 本授業では，「整数を昇順に並べる並べ替え」のみ．
    - 理解しやすくするため．
  - 一般には，同じデータ集合でも大小の定義次第で，並べ替えた結果は変化．
- 昇順
  - 小さい値ほど先頭に近くなるように並べる方法
- 降順
  - 大きい値ほど先頭に近くなるように並べる方法
- 先頭 ＝ 左端
- Pythonなどの現代的なプログラミング言語にはすぐに使える関数が用意．
- 自前で実装したアルゴリズムとそれらを速度比較し，その圧倒的な差を体験．

【重要】整列(ソート)の定義

- 定義： ソート

入力として，全順序関係が定義されているn 個のデータ$d_0, d_1, \cdots, d_{n-1}$が
与えられたときに，そのデータを全順序関係に従って並べ換える操作

- 定義： 全順序関係(データの大小関係のこと)

すべてのデータに対して以下の性質が成り立つ場合，関係”$\leq$”は順序関係である．

反射則：すべてのxについて，$x \leq x$ が成り立つ．

推移則：すべてのx, y, zについて，$x \leq y$ かつ$y \leq z$ ならば，$x \leq z$ が成り立つ．

反対称則：すべてのx, yについて，$x \leq y$ かつ$y \leq x$ ならば，$x = y$ が成り立つ．

加えて，すべてのデータの対に対して，順序関係”$\leq$”が以下の性質をもつとき，
その順序関係を**全順序関係**であるという．

比較可能性：すべてのx, yについて，$x \leq y$，もしくは$y \leq x$ が成り立つ．

課題4（提出不要）

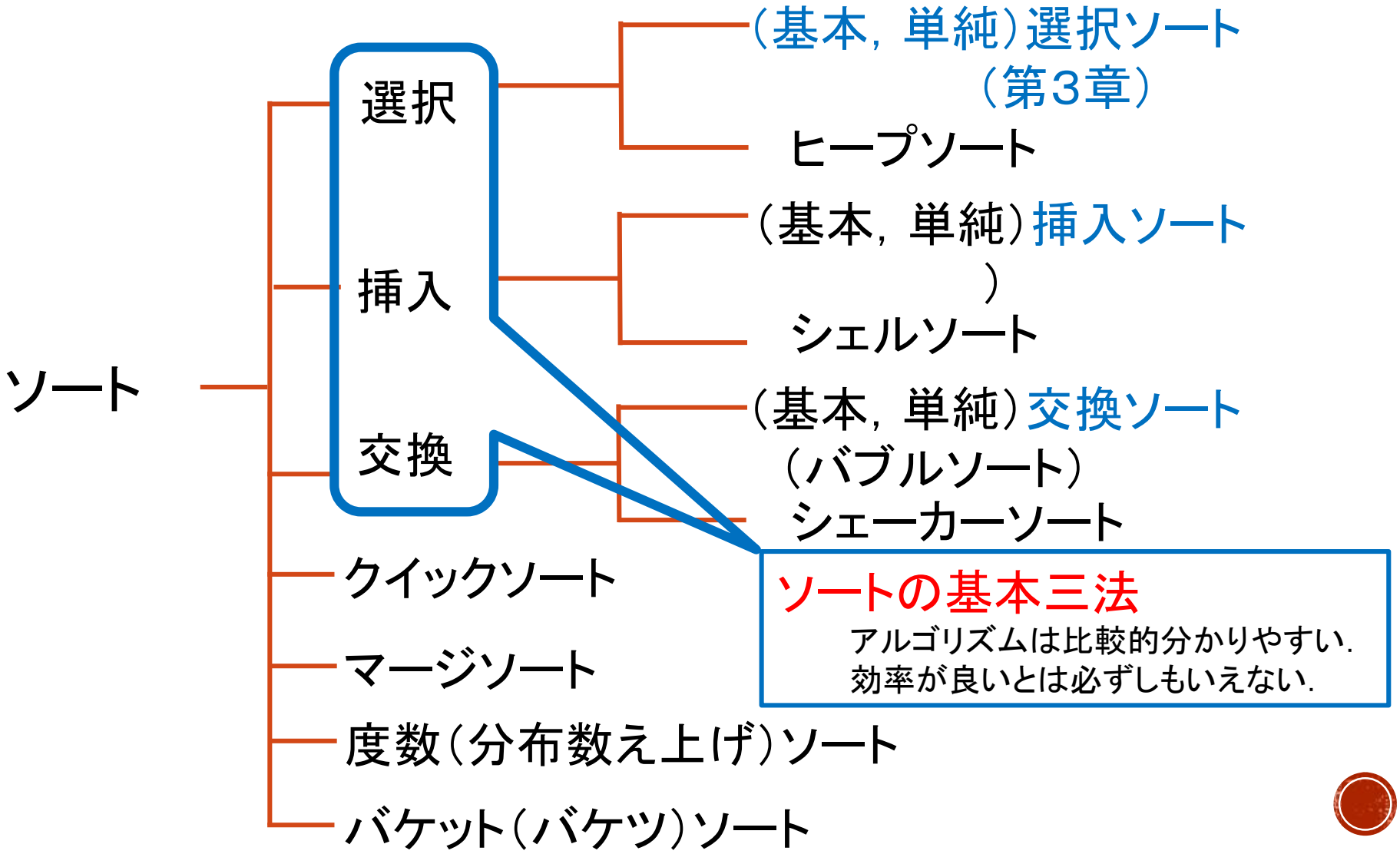
- 3.2.1　要素の並べ替え（教科書P.049）の
  プログラムを実行し，Pythonのrandomモジュールの
  使い方を確認.

- Pythonの組込み関数の動作を確認.
  - sorted

- リスト型がもつ以下のメソッドの動作を確認.
  - sort

ソートアルゴリズムの種類

順番がバラバラになっている配列の要素をどうしたら並べ替えられるか？



3.2.2　選択ソート

- 配列の要素から最も小さい数を探し出し，
  その要素を配列の前のほうへもってくる作業を繰り返す．
- 配列を何回も走査し、小さい数値が前に来るように
  順々に並び替えるという実直な方法．

アルゴリズム 3.8　選択ソート

```
1  入力：長さn の配列
2  出力：ソートされた配列
3  手続き：
4  1. i に 0 を代入
5  2. i が配列の最後の添え字なら終了
6  3. i 番目以降の要素の中から i 番目よりも小さい最小の値を見つけ，その添え字を j とする
7  4. 手続き 3で値が見つかったらi 番目の要素と j 番目の要素を交換する
8  5. i を 1 増やし手続き 2 へ戻る
```

選択ソートの実行例

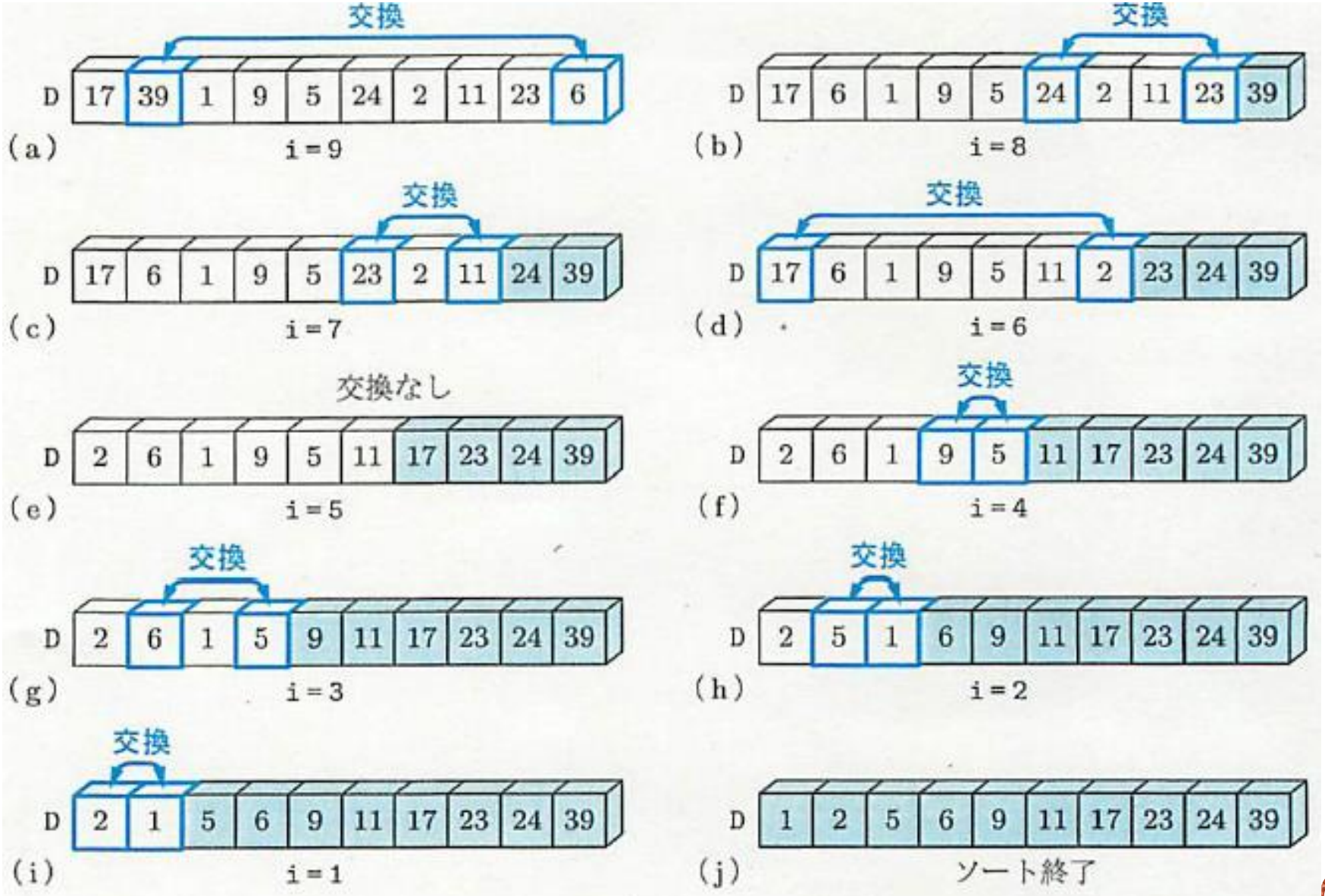


図 5.2　選択ソートの実行例

課題5　3.2.3 選択ソートを実装する

- アルゴリズム3.8をPythonで実装しなさい.
- 関数名は，selection_sortとする.
- 引数にとった配列を書き換えるのではなく，新しい配列を返す仕様.
- 注意
  - 教科書P.051の「コード3.9　選択ソートの実装」を見ないこと.

アルゴリズム 3.8　選択ソート

```
入力：長さn の配列
出力：ソートされた配列
手続き：
1. i に 0 を代入
2. i が配列の最後の添え字なら終了
3. i 番目以降の要素の中から i 番目よりも小さい最小の値を見つけ，その添え字を j とする
4. 手続き 3で値が見つかったらi 番目の要素と j 番目の要素を交換する
5. i を 1 増やし手続き 2 へ戻る
```

3.2.3 選択ソートの時間計算量

- 入力によりアルゴリズムの実行が変化しない.
  - 最良時間計算量 ＝ 最悪時間計算量
- 二重のfor文により構成
  - 外側のfor文の繰り返し実行回数: $n-1$ 回
  - 内側のfor文の繰り返し実行回数: i 回
    - 外側のfor文の変数 i に依存.
- $\sum_{i=1}^{n-1} i \times O(1) = O(1) \times \frac{n(n-1)}{2} = O(n^2)$

- ソートアルゴリズムとしては計算量が多いほう.
  - 実際のプログラム中などで使われることはほとんどない.

3.2.5 速度の比較

- 乱数で発生させて作る配列の長さを10,000とし，
  「自前で実装した選択ソート」と
  「組込み関数sorted」の実行速度を比較．
- 標準モジュールのtimeitを使い，
  コードの実行時間を計測．

```
import timeit
my_array = [random.randint(0, 99) for i in range(10000)]
timeit.timeit('selection_sort(my_array)', globals=globals(), number=1)
```

```
4.2431511878967285
```

```
my_array = [random.randint(0, 99) for i in range(10000)]
timeit.timeit('sorted(my_array)', globals=globals(), number=1)
```

```
0.0023238658905029297
```

課題6　速度の比較

- 教科書P.052,053の例のとおり,
- 「自前で実装した選択ソート」と
  「組込み関数sorted」の実行速度を比較しなさい.

- 参考3.1と参考3.2も読むこと.