Pythonによる
アルゴリズム設計

博士(工学) 神野 健哉 著

コ ロ ナ 社

ま　え　が　き

アルゴリズムという言葉はアル・フワーリズミー（al-Khuwārizmī）[†1]が基になったとされる[1)†2]。アル・フワーリズミーは代数学の祖であり，彼の著書の冒頭に Algoritmi dicti（アル・フワーリズミーに曰く）という一節があり，これがアルゴリズム（algorithm）の語源になったといわれている。

アルゴリズムは日本語では「算法」と訳され，ある問題を解くための手順や計算方法のことを指す。この手順に曖昧さがなく示されていれば，必要な時間はともかくとして誰にでも必ず問題を解くことができ，同じ答えに行き着くことができる。この性質はまさにコンピュータプログラムに必要なものであり，与えられた問題を解くための正しい手順を曖昧さがないプログラムとして記述することができれば，コンピュータは与えられた問題に対して正しく答えを出すことができる。コンピュータの命令一つひとつはコンピュータ自身に計算を指示するものであり，これをアルゴリズムに従って正しい順序で記述することでプログラムは正しく動作する。したがって正しく動くプログラムを作るためには，曖昧さのないアルゴリズムをまずは記述する必要がある。

また同じ問題を解くアルゴリズムは複数存在する場合があり，さらには同じアルゴリズムであっても，コンピュータに対する命令をどのような順で与えるかによって計算効率が変化する。プログラムは正確に，かつできるだけ高速に計算できることが重要である。コンピュータのハードウェアも日々革新が進んでおり，同じ計算の速度は非常に速くなっているが，アルゴリズムを見直すことで，ハードウェアの進展よりもはるかに速く計算結果が得られるようになることは多い。このためアルゴリズムはコンピュータプログラムで最も重要なものである。

本書では 1950 年代から 1960 年代に生まれた非常に有名なアルゴリズムを，近年非常によく用いられるようになったプログラミング言語「Python」[†3]で実装し，内容を解説する。Python はオブジェクト指向型インタープリタ言語である。インタープリタ言語は他のコンパイラ型言語と呼ばれるプログラミング言語よりも実行速度が劣る。しかしながらコンパイラ型言語と比較して非常に容易にプログラムを試し，修正することが可能である。このため最初にアルゴリズムを実際に動作させながら学ぶことに非常に適したコンピュータ言語であるといえる。また，アルゴリズムはその計算手順だけでなく，問題を解決するためのデータをどのような形式で扱うのかが非常に重要である。Python では従来のコンピュータ言語と比較してさまざまな形式

---

†1　al-Khuwārizmī（780-850 頃）：8 世紀後半から 9 世紀前半のイスラム科学者。

†2　肩つき数字は巻末の引用・参考文献を示す。

†3　`https://www.python.org/`

のデータ構造を使用することができ，これらのデータ構造を使用することで，従来のプログラムよりも簡潔に内容を記述することができる。

本書では，ある目的を解決するアルゴリズムをできるだけ複数紹介するようにした。これは冒頭で述べたように，同じ目的で同じハードウェアを使用しても，その処理速度がアルゴリズムによって大きく異なることを感じてもらうためである。特に近年，ビッグデータに注目が集まり，非常に大規模なデータを取り扱うようになった。そのような際にはアルゴリズムの良し悪しが処理時間に大きく影響を与える。各章の章末問題では処理するデータ数を極力変えられるような問題を用意し，さまざまなアルゴリズムが，問題の大きさによって処理速度がどのように変化するのかを実際に体感できるようにした。

アルゴリズムを理解するためにはまずはその概念を理解し，つぎにそれをどのように実装するのかを考えることが重要である。最初は本書に掲載しているプログラムを「写経」し，それらを改変していくことがプログラムを作成する能力のために必要である。これらの基本となるアルゴリズムを基に，機械学習などさまざまな応用につながることも意識してほしい。

本書は，東京都市大学情報工学部知能情報工学科で 1 年生に開講している専門科目「アルゴリズム設計」での講義を基に執筆している。全 14 章で構成されており 14 回の講義に対応する。

本書が一人でも多くの人の「アルゴリズム」の理解につながることを願っている。

2022 年 8 月

神野 健哉

目　　次

1. アルゴリズムとは

2. Selection sort と Bubble sort

3. Merge sort と再帰関数

4. Quick sort とリスト内包表記

5. 計　算　量

6. 検　索

7. グラフと Union-Find アルゴリズム

8. 最 小 全 域 木

9. 幅優先探索（BFS）と深さ優先探索（DFS）

10. 最短経路問題

11.　最大フロー問題

12.　最大マッチング問題・割当問題

13.　ナップサック問題

14. 敵 対 探 索

5 計　算　量

同じ処理でもコンピュータの CPU 等の性能によってその処理速度は大きく異なる。しかし，同じ目的の処理でもアルゴリズムによって CPU 等の性能差以上に処理速度が異なる。前章までに紹介した並べ替えのアルゴリズムも，そのアルゴリズムごとに並べ替え処理速度が大きく異なる。このアルゴリズムの性能を測る指標が「計算量」である。計算量には「時間計算量」と「空間計算量」があるが，本章ではおもに時間計算量について説明する。

5.1 実　行　時　間

2 章では Selection sort と Bubble sort，3 章では Merge sort，4 章では Quick sort を紹介した。これらはすべて「並べ替え」を解決するアルゴリズムであるが，実際に動作させると同じ量のデータの並べ替えでもその実行時間には大きな差が発生する。

各並べ替えアルゴリズムの実行時間を実験し，その結果を**図 5.1** および**表 5.1** に示す。

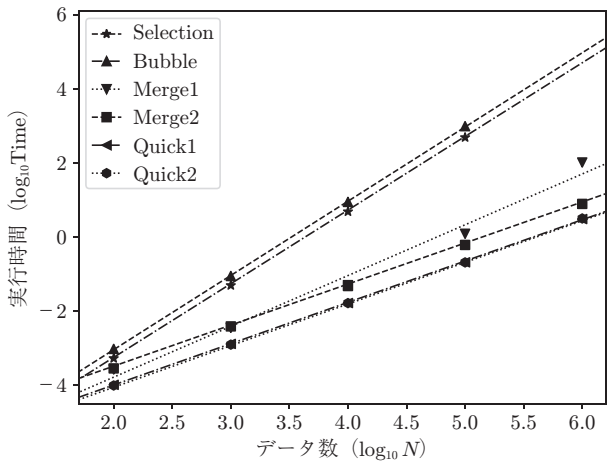


図 5.1　データサイズに対する各アルゴリズムの処理時間
（Intel Core i7 2.7 GHz）

図 5.1 および表 5.1 の結果は，MacBook Pro 2018 モデル Core i7 2.7 GHz 16 GB Memory macOS 10.14.6 上の Python 3.7.3 で各アルゴリズムを実装したプログラム 2.17，プログラム 2.18，プログラム 3.3，プログラム 3.4，プログラム 4.1，プログラム 4.2 を用いて乱数で生成

表 5.1 各並べ替えアルゴリズムの実行時間の比較

上段：施行 10 回の平均時間〔s〕，下段：標準偏差

データ数	Selection sort プログラム 2.17	Bubble sort プログラム 2.18	Merge sort プログラム 3.3	Merge sort プログラム 3.4	Quick sort プログラム 4.1	Quick sort プログラム 4.2
100	5.336e−04	9.567e−04	2.677e−04	2.854e−04	1.029e−04	9.857e−05
	1.105e−04	1.302e−04	1.764e−05	1.230e−05	6.661e−06	5.630e−06
1 000	4.972e−02	8.877e−02	3.555e−03	3.883e−03	1.307e−03	1.250e−03
	3.213e−03	3.008e−03	8.542e−05	5.271e−04	1.030e−04	5.866e−05
10 000	4.856e+00	8.876e+00	5.042e−02	4.770e−02	1.658e−02	1.650e−02
	8.572e−02	5.020e−02	7.592e−04	1.129e−03	6.427e−04	6.260e−04
100 000	4.885e+02	9.804e+02	1.227e+00	6.129e−01	2.081e−01	2.069e−01
	9.082e−01	3.998e+01	2.793e−02	1.494e−02	4.718e−03	3.761e−03
1 000 000			1.031e+02	7.849e+00	3.063e+00	3.176e+00
			2.116e+00	1.168e−01	1.069e−01	1.375e−01

MacBook Pro 2018, Core-i7 2.7 GHz 16 GB macOS 10.14.6 Python 3.7.3

した 100 個，1 000 個，10 000 個，100 000 個，1 000 000 個のデータを並べ替えた際の処理時間を測定した。1 000 000 個のデータに対しての Selection sort，Bubble sort の処理時間は測定していない。

図 5.1 は両対数軸のグラフであり，横軸はデータ数を対数で示し，縦軸は 10 回の試行の平均処理時間を対数で示している。各マーカがデータを表し，各線はこれらのデータに対する近似曲線を示す。図 5.1 に示したように，得られたデータに対する両対数グラフ上で近似曲線は直線であることから，データ数と実行時間には累乗近似関係があることがわかる。

Selection sort，Bubble sort ではデータ数が 10 倍になるごとに処理時間はおおよそ 100 倍である。これに対して Merge sort，Quick sort ではデータ数が 10 倍になるごとに処理時間はおおよそ 13 倍程度である。

表 5.2 に同様の実験を MacBook Pro 2020 モデル Apple-M1 16 GB Memory macOS 11.5.2 上の Python 3.9.7 での実行結果を示す。表 5.1 の結果と比べ，CPU の違いから処理速度が速

表 5.2 各並べ替えアルゴリズムの実行時間の比較（Apple-M1）

上段：施行 10 回の平均時間〔s〕，下段：標準偏差

データ数	Selection sort プログラム 2.17	Bubble sort プログラム 2.18	Merge sort プログラム 3.3	Merge sort プログラム 3.4	Quick sort プログラム 4.1	Quick sort プログラム 4.2
100	4.183e−04	5.185e−04	1.510e−04	1.518e−04	6.280e−05	5.813e−05
	2.053e−04	2.038e−05	4.631e−06	2.032e−06	4.476e−06	3.706e−06
1 000	3.237e−02	5.515e−02	1.731e−03	1.664e−03	8.055e−04	7.812e−04
	5.840e−04	4.941e−04	2.913e−05	9.787e−06	4.291e−05	2.409e−05
10 000	3.202e+00	5.854e+00	2.730e−02	2.148e−02	1.027e−02	1.020e−02
	2.233e−02	3.350e−02	8.096e−05	8.688e−05	2.998e−04	2.364e−04
100 000	3.281e+02	6.190e+02	4.004e+00	2.664e−01	1.269e−01	1.263e−01
	3.491e+00	3.048e+01	4.868e−02	9.824e−04	1.990e−03	1.780e−03
1 000 000			5.319e+02	3.214e+00	1.529e+00	1.527e+00
			6.389e+00	2.537e−02	2.918e−02	3.177e−02

MacBook Pro 2020, Apple-M1 16 GB Python 3.9.7

いことが確認できるが，データ数が 10 倍になった際のそれぞれのアルゴリズムの処理時間の増え方は同様である。このように CPU の違いで処理速度に差は出るものの，処理速度はアルゴリズムそのものによる影響が大きいことがわかる。そこで，各アルゴリズムの計算に要する時間を評価することを考える。

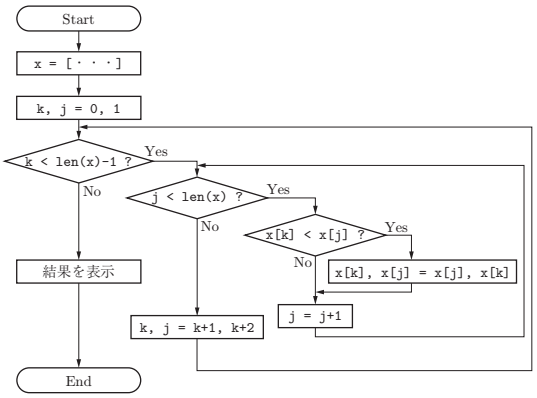
5.2 アルゴリズムの計算手順

5.2.1 Selection sort

Selection sort の計算手順を**図 5.2** に示す。ここに示した Selection sort アルゴリズムにより降順に並べ替える場合，先頭のデータから順に自身より後にある各データと大きさを比較し，後のデータのほうが大きければそれらのデータを入れ替える。データの総数を N とすると比較は以下の回数行う。

$$\sum_{k=1}^{N-1} n = \frac{N(N-1)}{2} \tag{5.1}$$

最悪の場合，すべての比較の際に入れ替えの手順が合わせて必要になる。例えば，$N = 128$ の場合，8 128 回の比較・入れ替えの手順が必要であり，$N = 1\,024$ では 523 776 回の比較・入れ替え手順が必要になる可能性がある。



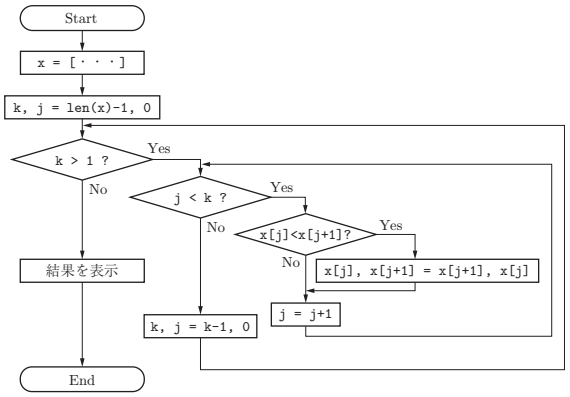

```
def selection_sort( x ):
   for k in range( len(x)-1 ):
      for j in range( k, len(x) ):
         if x[k] < x[j]:
            x[k], x[j] = x[j], x[k]
   return x
```

図 5.2　Selection sort

5.2.2 Bubble sort

Bubble sort の計算手順を**図 5.3** に示す。Bubble sort アルゴリズムでは先頭のデータから順に隣のデータと大きさを比較し，隣のデータのほうが大きければそれらのデータを入れ替えるアルゴリズムである。データの総数を N とすると，比較は以下の回数行う。

$$\sum_{k=1}^{N-1} n = \frac{N(N-1)}{2} \tag{5.2}$$



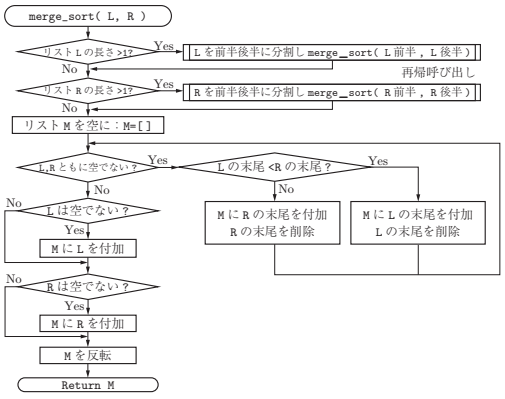

```
def bubble_sort( x ):
   for k in range( len(x)-1, 0, -1 ):
      for j in range( k ):
         if x[j] < x[j+1]:
            x[j], x[j+1] = x[j+1], x[j]
   return x
```

図 5.3 Bubble sort

最悪の場合，すべての比較の際に入れ替えが行われる。これは Selection sort と同様であり，Selection sort と Bubble sort は基本的に同じ手順数が必要であることがわかる。

5.2.3 Merge sort

Merge sort の計算手順を**図 5.4** に示す。Merge sort アルゴリズムはデータを小さな単位に分割し，それら小さな単位内で並べ替えてから，それらを元の長さになるように並べ替え結果を考慮しながら結合することで並べ替えを行うアルゴリズムである。




```
def merge_sort( X, Y ):
   if len(X) > 1:
      X = merge_sort( X[:len(X)//2], X[len(X)//2:] )
   if len(Y) > 1:
      Y = merge_sort( Y[:len(Y)//2], Y[len(Y)//2:] )
   M =[]
   k, j = 0, 0
   while len( X ) > k and len( Y ) > j :
      if X[k] > Y[j] :
         M.append( X[k] )
         k=k+1
      else:
         M.append( Y[j] )
         j=j+1
   if len( X ) > k :
      M.extend( X[k:] )
   elif len( Y ) > j :
      M.extend( Y[j:] )
   return M
```

図 5.4 Merge sort

Merge sort でデータの分割は再帰的に行われるが，手順としてはそれらを結合する際に比較・代入が必要となる。例えばデータ総数が 2 の場合の結合は，それぞれ長さ 1 の二つのデータを比較し結合するため比較回数は 1 回である。データ総数が 4 の場合の結合は，それぞれ長さ 2 の二つのデータを比較し結合するため比較回数は 3 回である。データ総数が N の場合，二つのデータを結合するための比較回数は $N-1$ 回である。これをまとめると**図 5.5** のようになる。

データ総数 N での比較回数を $T(N)$ とおく。データ総数 1 では比較回数は 0 回，データ総

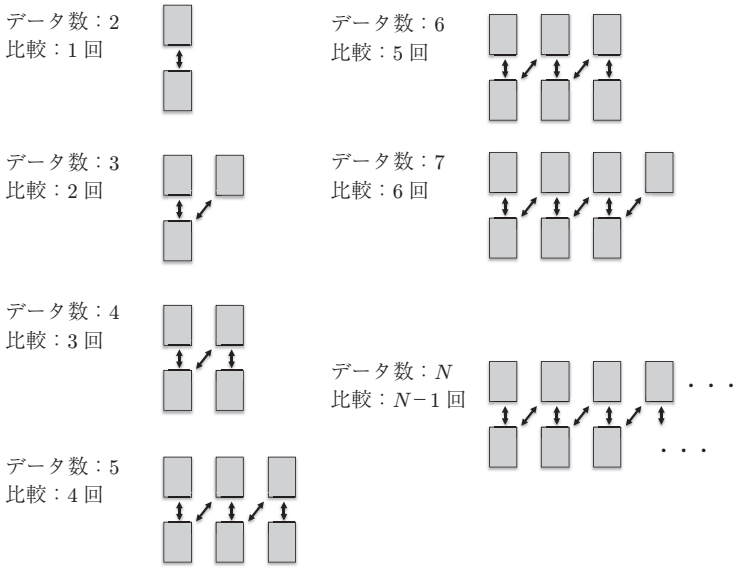


図 5.5　Merge sort の処理

数 2 では比較回数は 1 回であるので

$$T(1) = 0 \tag{5.3}$$

$$T(2) = 1 \tag{5.4}$$

である。

データ総数 3 ではデータ数 2 とデータ数 1 に，データ総数 4 ではデータ数 2 とデータ数 2 に分割する。分割後の結合は図 5.5 に示したようにデータ総数 3 では 2 回，データ総数 4 では 3 回である。したがって

$$T(3) = T(1) + T(2) + 2 = 3 \tag{5.5}$$

$$T(4) = T(2) + T(2) + 3 = 5 \tag{5.6}$$

となる。

これを一般化すると

$$T(N) = T\left(\left\lfloor \frac{N}{2} \right\rfloor\right) + T\left(\left\lceil \frac{N}{2} \right\rceil\right) + (N-1) \tag{5.7}$$

である。

$N = 2^k$ の場合について考える。$T(2^k)$ は

$$T(2^k) = 2T(2^{k-1}) + \left(2^k - 1\right) \tag{5.8}$$

である。よって

$$T(2^k) = (k-1)\,2^k + 1 \tag{5.9}$$

である。$N = 2^k$ を式 (5.9) に代入すると

$$T(N) = (\log_2 N - 1)\,N + 1 = N \log_2 N - N + 1 \tag{5.10}$$

となる。

$2^{k-1} < M < 2^k$ のとき，$N = 2^k$ とすると $M < N$ であり

$$T(M) \leqq T(N) = N \log_2 N - N + 1 \tag{5.11}$$

である。

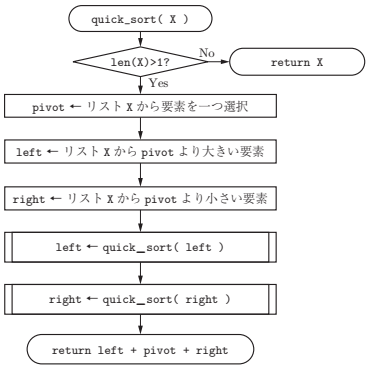
以上より，データ総数 N のときの比較回数 $T(N)$ は

$$T(N) \leqq N \log_2 N - N + 1 \tag{5.12}$$

となる。最悪の場合，すべての比較の際に入れ替えの手順が合わせて必要になる。例えば，$N = 128$ の場合 769 回の比較・入れ替えの手順が必要であり，$N = 1\,024$ では 9 217 回の比較・入れ替え手順が必要になる可能性がある。

5.2.4　Quick sort

Quick sort の計算手順を**図 5.6** に示す。Quick sort アルゴリズムはデータの中から任意の一つの要素を基準値（`pivot`）とし，これよりも大きい値のデータと小さい値のデータとにリストを分割する。分割されたリストに対して，再帰的にリストに含まれる要素が 1 個になるまで分割を繰り返すことで並べ替えを行うアルゴリズムである。




```
def quick_sort( X ):
   if len(X) > 1:
      pivot = X.pop( -1 )
      left  = [ element for element in X if element > pivot ]
      right = [ element for element in X if element < pivot ]
      mid   = [pivot] * (1+len(X)-len(left)-len(right))
      X = quick_sort(left) + mid + quick_sort(right)
   return X
```

図 5.6　Quick sort

データ総数 N での比較回数を $T(N)$ とおく。データ総数 1 では基準値のみであるため比較回数は 0 回，データ総数 2 では基準値との比較のみであるため比較回数は 1 回である。よって

$$T(1) = 0 \tag{5.13}$$

索　　引

—— 著 者 略 歴 ——

1991年　法政大学工学部電気工学科卒業
1993年　法政大学大学院工学研究科博士前期課程修了（電気工学専攻）
1996年　法政大学大学院工学研究科博士後期課程修了（電気工学専攻），博士（工学）
1996年　上智大学助手
1998年　日本工業大学助手
1999年　日本工業大学専任講師
2003年　日本工業大学助教授
2004年　関東学院大学助教授
2007年　関東学院大学准教授
2008年　ERATO 合原複雑数理モデルプロジェクト研究員
2009年　日本工業大学准教授
2010年　日本工業大学教授
2018年　東京都市大学教授
　　現在に至る

Python によるアルゴリズム設計

Algorithm Design with Python

2022 年 9 月 26 日　初版第 1 刷発行　★

検印省略

著　者　神野健哉（じんのけんや）
発行者　株式会社　コロナ社
　　代表者　牛来真也
印刷所　三美印刷株式会社
製本所　有限会社　愛千製本所

112–0011　東京都文京区千石 4–46–10
発行所　株式会社　**コロナ社**
CORONA PUBLISHING CO., LTD.
Tokyo Japan
振替 00140–8–14844・電話(03)3941–3131(代)
ホームページ　https://www.coronasha.co.jp

ISBN 978–4–339–02930–7　C3055　Printed in Japan　（齋藤）