

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-Khwārizmī)^{†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^{いわ}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ます、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に変化する。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-Khwārizmī (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

【本書ご利用にあたって】

- ・本文中に記載している会社名、製品名は、それぞれ各社の商標または登録商標です。本書では®やTMは省略しています。
- ・本書に記載の情報、ソフトウェア、URLは2022年8月現在のものを記載しています。

目 次

1. アルゴリズムとは

1.1	アルゴリズムの要件	1
1.1.1	停止性	1
1.1.2	正当性	1
1.1.3	汎用性	1
1.2	フローチャート	2
1.2.1	順次処理	2
1.2.2	選択処理	3
1.2.3	反復処理	4
1.3	最大値の探索	5
1.3.1	勝ち抜き方式	6
1.3.2	トーナメント方式	6
1.4	アルゴリズム	7
	章末問題	7

2. Selection sort と Bubble sort

2.1	Python のリスト構造	8
2.1.1	リストの生成	9
2.1.2	リストの結合	9
2.1.3	リストの比較	10
2.1.4	リストの要素のアクセス	10
2.1.5	スライスによるリストの要素のアクセス	11
2.1.6	リストの要素の置換	11
2.1.7	リストのコピー	12
2.1.8	リストの要素の追加 (<code>append</code> メソッド)	14
2.1.9	リストの要素の追加 (<code>extend</code> メソッド, 累算代入)	15
2.1.10	リストの要素の挿入 (<code>insert</code> メソッド, スライス操作)	15

2.1.11 リストの要素の削除 (<code>pop</code> メソッド, <code>del</code>)	16
2.2 最大値/最小値に着目した並べ替え	17
2.2.1 Selection sort	17
2.2.2 Bubble sort	19
章 末 問 題	20

3. Merge sort と再帰関数

3.1 関 数	22
3.1.1 関 数 の 定 義	22
3.1.2 再帰呼び出し	24
3.2 Merge sort	25
3.2.1 Merge sort のアルゴリズム	26
3.2.2 Merge sort の実装	27
3.2.3 Merge sort の実装の改良	29
3.2.4 Merge sort の <code>pop</code> を使用しない実装	31
章 末 問 題	31

4. Quick sort とリスト内包表記

4.1 Quick sort	33
4.1.1 データ分割法	33
4.1.2 実 装	34
4.2 リスト内包表記	36
4.2.1 イテラブルオブジェクト	36
4.2.2 リスト内包表記の基本形	36
4.2.3 <code>if</code> を利用した内包表記	37
4.2.4 <code>if~else</code> を利用した内包表記	37
4.2.5 複 合 型	38
4.3 リスト内包表記を利用した Quick sort	38
章 末 問 題	39

5. 計 算 量

5.1 実行時間	40
5.2 アルゴリズムの計算手順	42
5.2.1 Selection sort	42
5.2.2 Bubble sort	42
5.2.3 Merge sort	43
5.2.4 Quick sort	45
5.3 アルゴリズムの評価指標	47
5.3.1 時間計算量	47
5.3.2 O 記法	48
章末問題	49

6. 検 索

6.1 線形検索	51
6.2 二分検索	53
6.3 ハッシュ法	54
6.4 辞書	56
6.4.1 辞書の生成	56
6.4.2 辞書の情報	57
6.4.3 in 演算子	57
6.4.4 辞書の要素の置換・追加	57
6.4.5 辞書の要素の削除	58
6.5 辞書を用いた検索	58
章末問題	61

7. グラフと Union-Find アルゴリズム

7.1 グラフ	62
7.2 集合	63
7.2.1 集合の生成	63
7.2.2 in 演算子による集合の帰属性判定	64

7.2.3 集合の要素の追加・削除	64
7.2.4 集合演算	65
7.3 Union-Find アルゴリズム	66
7.3.1 Find 操作	67
7.3.2 Union 操作	67
7.4 橋の検出	68
章末問題	70

8. 最小全域木

8.1 全域木	71
8.2 クラスカル法	72
8.3 プリム法	75
章末問題	77

9. 幅優先探索 (BFS) と深さ優先探索 (DFS)

9.1 木構造データ	79
9.2 幅優先探索: BFS	80
9.2.1 幅優先探索のアルゴリズム	80
9.2.2 キュー構造	80
9.2.3 幅優先探索の実装	81
9.3 深さ優先探索: DFS	83
9.3.1 深さ優先探索のアルゴリズム	83
9.3.2 スタック構造	83
9.3.3 深さ優先探索の実装	84
9.3.4 繰り返しでの <code>break</code> 文	85
章末問題	86

10. 最短経路問題

10.1 最短経路	89
10.2 ベルマン・フォード法	90
10.2.1 ベルマン・フォード法のアルゴリズム	90

10.2.2	ベルマン・フォード法の探索の実例	90
10.2.3	ベルマン・フォード法の実装	91
10.3	ダイクストラ法	93
10.3.1	ダイクストラ法のアルゴリズム	93
10.3.2	ダイクストラ法の探索の実例	93
10.3.3	ダイクストラ法の実装	94
章 末 問 題		96

11. 最大フロー問題

11.1	フローネットワーク	98
11.2	フォード・ファルカーソン法	99
11.3	最小カット問題	101
11.4	フォード・ファルカーソン法の実装	101
章 末 問 題		104

12. 最大マッチング問題・割当問題

12.1	マ ッ チ ン グ	105
12.1.1	二 部 グ ラ フ	105
12.1.2	最大マッチング	105
12.2	最大フローによる最大マッチングの解法	106
12.3	割 当 問 題	107
12.4	実 装	108
章 末 問 題		110

13. ナップサック問題

13.1	0-1 ナップサック問題	111
13.2	貪欲法・動的計画法	112
13.2.1	貪 欲 法	112
13.2.2	動 的 計 画 法	112
13.2.3	動的計画法による探索の実例	113

13.3 動的計画法によるナップサック問題の解法の実装 115
章 末 問 題 117

14. 敵 対 探 索

14.1 ミニマックス法 118
14.2 「エイト」ゲーム 119
14.3 ミニマックス法の実装 120
章 末 問 題 123

引用・参考文献 124
章末問題解答 125
索 引 147

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。

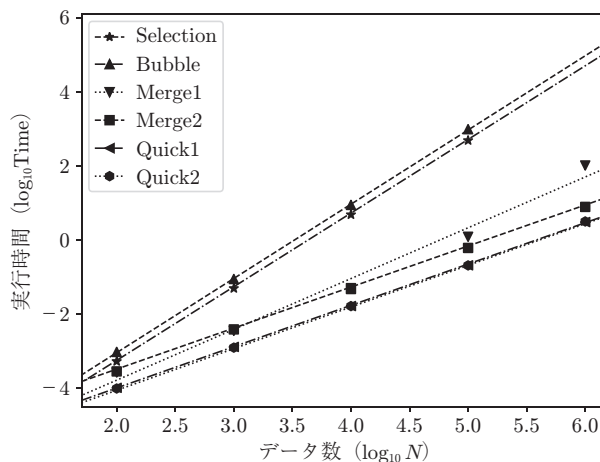


図 5.1 データサイズに対する各アルゴリズムの処理時間 (Intel Core i7 2.7 GHz)

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	Bubble sort	Merge sort		Quick sort	
	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
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	Bubble sort	Merge sort		Quick sort	
	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
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。

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

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、8128 回の比較・入れ替えの手順が必要であり、 $N = 1024$ では 523776 回の比較・入れ替え手順が必要になる可能性がある。

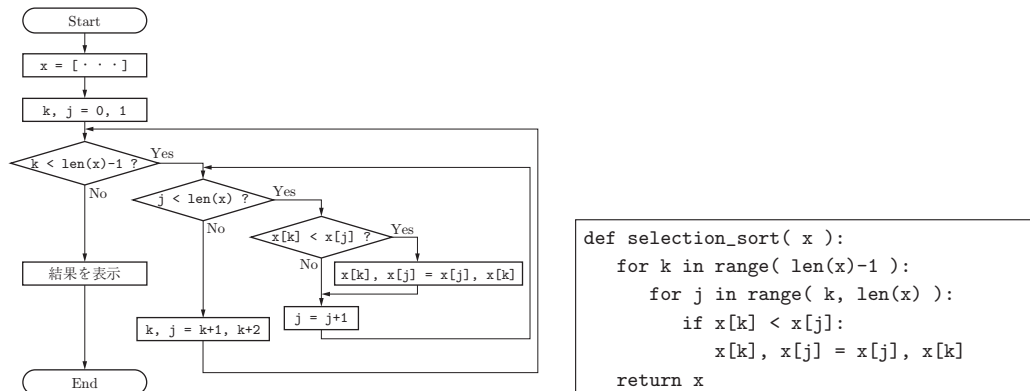


図 5.2 Selection sort

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。

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

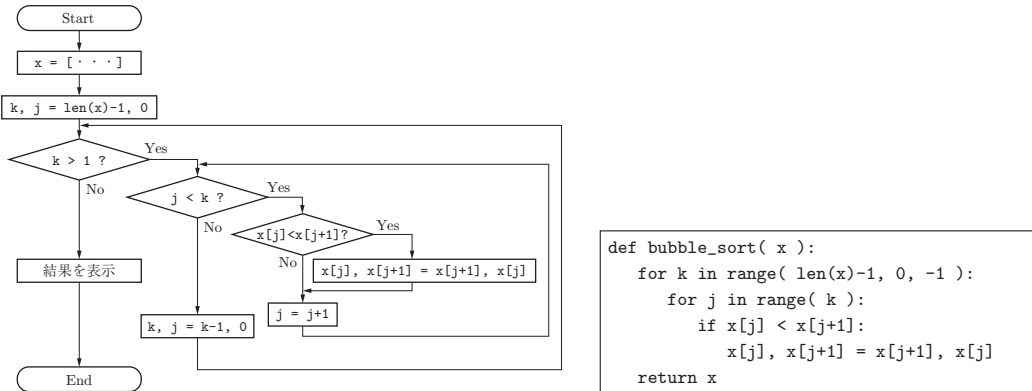


図 5.3 Bubble sort

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。

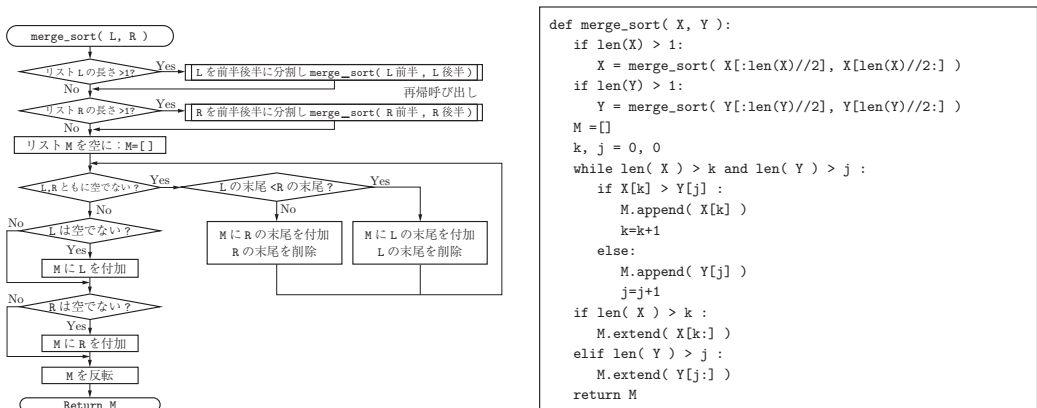


図 5.4 Merge sort

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

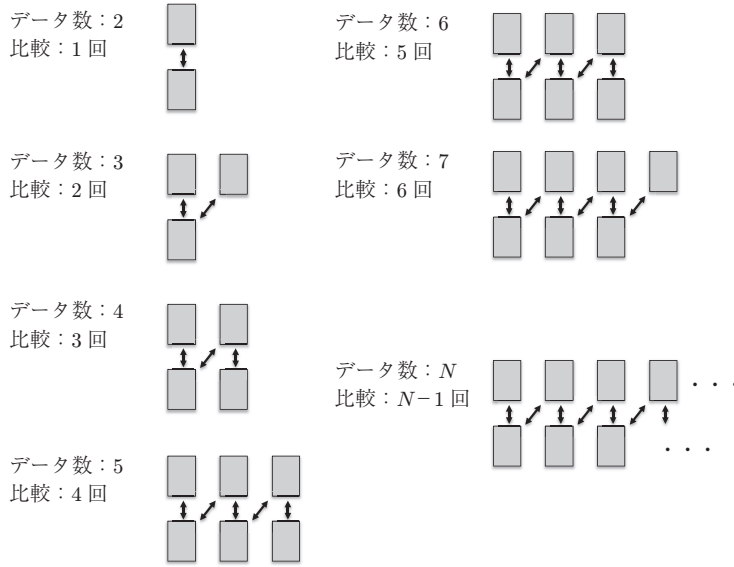


図 5.5 Merge sort の処理

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

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

である。よって

$$T(2^k) = (k - 1) 2^k + 1 \quad (5.9)$$

である。 $N = 2^k$ を式 (5.9) に代入すると

$$T(N) = (\log_2 N - 1) N + 1 = N \log_2 N - N + 1 \quad (5.10)$$

となる。

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

$$T(M) \leq T(N) = N \log_2 N - N + 1 \quad (5.11)$$

である。

以上より、データ総数 N のときの比較回数 $T(N)$ は

$$T(N) \leq N \log_2 N - N + 1 \quad (5.12)$$

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、 $N = 1024$ では 9217 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。

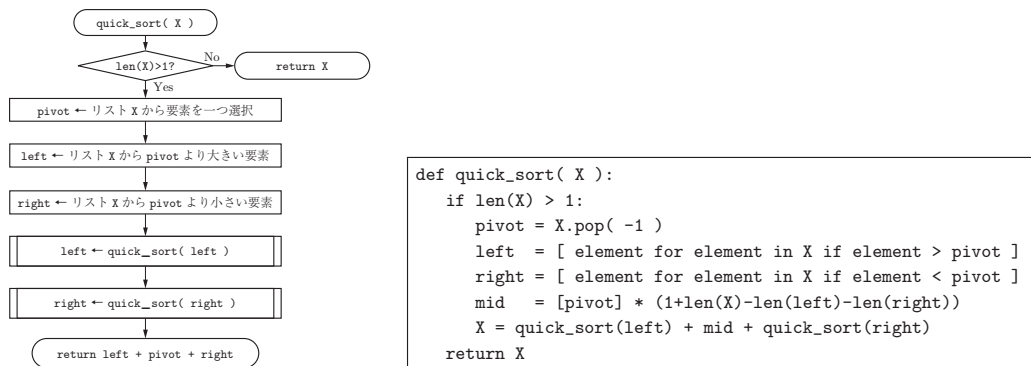


図 5.6 Quick sort

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

$$T(1) = 0 \quad (5.13)$$

索引

【あ, い】

浅いコピー	13
アサーション	1
後入れ先出し	83
後判定ループ	4
アルゴリズム	1
イテラブルオブジェクト	36

【え, お】

枝	62
オーダー	48
オープンアドレス法	56
重み付きグラフ	71
親	79

【か】

階乗値	24
カックウハッシュ法	56
カット	101
——のサイズ	101
カットエッジ	101
仮引数	23
関数	22
——の定義	22
——の呼び出し	23
完全ゲーム木	118
完全二部グラフ	105

【き】

木	71
キー	56
木構造データ	79
キュー	80

【く】

クイックソート	33
空間計算量	47
空集合	64
クラスカル法	72
グラフ	62

【け】

計算量	47
-----	----

結合	25
ゲーム木	118
検索	51

【こ】

子	79
孤立点	62

【さ】

再帰呼び出し	24
最小カット	101
最小カット問題	101
最小全域木	71
最小全域木問題	71
最大値の探索	5
最大フロー問題	98
最大マッチング	106
最短経路	89
最短経路問題	89
先入れ先出し	80
差集合	66
残余ネットワーク	99

【し】

時間計算量	47
辞書	56
——型	56
——の情報	57
——の生成	56
——の要素の削除	58
——の要素の置換	57
——の要素の追加	57
実行時間	40
——の測定	20
実引数	23
シノニム	56

集合	65
——演算	63
——型	64
——の帰属性判定	63
——の生成	64
——の要素の削除	64
——の要素の追加	64
終端頂点	79

樹形図	118
順次処理	2
衝突	56
処理時間	47

【す】

スタック	83
スライス	11

【せ】

整数ナップサック問題	112
正当性	1
積集合	66
接点	62
全域木	71
全解探索	112
線形検索	52
選択処理	2, 3
選択ソート	17
全点对最短経路問題	89

【そ】

素集合データ構造	66
ソート	8

【た】

ダイクストラ法	89, 93
対称差集合	66
ダイレクトアクセス	54
多重辺	62
単一始点最短経路問題	89
単純グラフ	62
端点	62

【ち】

チェイン法	56
頂点	62

【て】

停止性	1
敵対探索	118

【と】

動的計画法	96, 112
-------	---------

貪欲法	112				
		【な, に】			
ナップサック問題	111				
二重ハッシュ法	56				
二部グラフ	105				
二分検索	54				
		【ね】			
根	79				
根付き木	79				
		【は】			
葉	79				
橋	63				
——の検出	68				
バックトラック法	83				
ハッシュ関数	55				
ハッシュ値	55				
ハッシュ法	55				
幅優先探索	80				
パフォーマンスカウンタ	20				
バブルソート	19				
林	71				
反復処理	2, 4				
汎用性	1				
		【ひ】			
非終端頂点	79				
ビッグオー	48				
非負インデックス	10				
ピボット	33				
非連結グラフ	63				
		【ふ】			
負インデックス	10				
フォード・ファルカーソン法	99				
深いコピー	14				
深さ	79				
深さ優先探索	80, 83				
フラグ	92				
プリム法	75				
フロー	98				
フローチャート	2				
フローネットワーク	98				
分割統治法	25, 33				
		【へ, ほ】			
閉路	71				
ベルマン・フォード法	89				
辺	62				
変更可能オブジェクト	8				
歩道	71				
		【ま】			
前判定ループ	4				
マーソソート	25				
マッチング	105				
		【み, め】			
道	63				
ミニマックス法	118				
ミューダブル	8				
メモリ使用量	47				
		【ゆ】			
有向グラフ	63				
有向辺	63				
		【よ】			
容量	98				
欲張り法	112				
		【り, る】			
リスト					
——型オブジェクト	9				
——構造	8				
——内包表記	36				
——の結合	9, 15				
——のコピー	12				
——の生成	9				
——の比較	10				
——の要素のアクセス	10				
——の要素の削除	16				
——の要素の挿入	15				
——の要素の置換	11				
——の要素の追加	14, 15				
隣接行列	72				
ループ	62				
		【れ】			
連結グラフ	63				
連結リスト	8				
		【わ】			
和集合	65				
割当問題	107				

		【A】				
adjacency matrix	72		binary search	54	computational complexity	47
adversarial search	118		bipartite graph	105	connected graph	63
algorithm	1		branch	62	correctness	1
allocation problem	107		breadth first search	80	cuckoo hashing	56
all pair shortest path	89		bridge	63	cut	101
APSP	89		Bubble sort	19	cut edge	101
argument	23				cycle	71
assertion	1		【C】		【D】	
		【B】	capacity	98	deepcopy	13
Bachmann-Landau O-notation	48		chaining	56	deep copy	14
backtrack	83		child	79	depth	79
Bellman-Ford algorithm	89		circuit	71	depth first search	80
BFS	80		collision	56	DFS	80, 83
			complete bipartite graph	105	dictionary	56
			complete game tree	118	Dijkstra's algorithm	89, 93
			comprehension notation	36		

directed edge	63	LIFO	83	rooted tree	79
directed graph	63	linear search	52		
direct access	54	loop	62	[S]	
disconnected graph	63			search	51
disjoint-set data structure	66	[M]		selection	3
divide and conquer algorithm	25	matching	105	Selection sort	17
double hashing	56	maximum flow problem	98	sequence	2
dynamic programming	96, 112	maximum matching	106	set	63
		MD5	55	set difference	66
[E]		memory usage	47	SHA	55
edge	62	merge	25	shallow copy	13
end vertex	62	Merge sort	25	shortest path	89
exhaustive search	112	minimax algorithm	118	shortest path problem	89
		minimum cut	101	simple graph	62
[F]		minimum cut problem	101	single source shortest path	89
FIFO	80	minimum spanning tree	71	single-pair shortest path	
first in first out	80	minimum spanning tree		problem	89
flag	92	problem	71	size of the cut	101
flow	98	multiple edge	62	slice	11
flowchart	2			sort	8
flow network	98	[N]		space complexity	47
Ford-Fulkerson algorithm	99	negative index	10	spanning tree	71
forest	71	node	62	SSSP	89
function	22	nonterminal vertex	79	stack	83
function call	23	non-negative index	10	symmetric difference	66
		not in 演算子	57, 64	synonym	56
[G]				[T]	
game tree	118	[O]		terminal vertex	79
graph	62	O 記法	48	time complexity	47
greedy algorithm	112	open addressing	56	tree	71
		order	48	tree diagram	118
[H]				tree structure data	79
halting problem	1	[P]			
hash function	55	parameter	23	[U]	
hash search	55	parent	79	union	65
hash value	55	path	63	Union-Find アルゴリズム	66
		performance counter	20		
[I]		pivot	33	[V]	
in 演算子	57, 64	post-test loop	4	versatility	1
integer knapsack problem	112	pre-test loop	4	vertex	62
intersection	66	Prim's algorithm	75		
isolated vertex	62	processing time	47	[W]	
iterable object	36			walk	71
		[Q]		weighted graph	71
[K]		queue	80		
key	56	Quick sort	33	[Z]	
knapsack problem	111			zero-one knapsack problem	112
Kruskal's algorithm	72	[R]		[数字]	
		recursive call	24	0-1 ナップサック問題	112
[L]		repetition	4	2 頂点最短経路問題	89
last in first out	83	residual network	99		
leaf	79	root	79		

—— 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

© Kenya Jinno 2022

2022年9月26日 初版第1刷発行



検印省略

著者 神野健哉
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)



JCOPY <出版者著作権管理機構 委託出版物>

本書の無断複製は著作権法上での例外を除き禁じられています。複製される場合は、そのつど事前に、出版者著作権管理機構（電話 03-5244-5088, FAX 03-5244-5089, e-mail: info@jcopy.or.jp）の許諾を得てください。

本書のコピー、スキャン、デジタル化等の無断複製・転載は著作権法上での例外を除き禁じられています。購入者以外の第三者による本書の電子データ化及び電子書籍化は、いかなる場合も認めていません。落丁・乱丁はお取替えいたします。