ソート

山本昌志*

2007 年 7 月 24 日

概 要

ソートのアルゴリズムについて学習する．単純挿入ソート，シェルソート，クイックソートの原理と計算量，プログラム方法について学ぶ．

1 本日の学習内容

本日は，ソートと呼ばれるアルゴリズムについて学習である．教科書 [2] の pp.217–230 が範囲である．ここでの学習のゴールは以下の通りである．

- 単純挿入ソートとシェルソート，クイックソートのアルゴリズムが分かる．
- 計算量が分かる．
- プログラムが書ける．

2 コンピューターができることとソートの問題

2.1 コンピューターは何ができるのか?

ソートの問題を考える前に，コンピューター—正確には C 言語の命令—ができることを示しておく必要があるだろう．コンピューターが何ができて，何ができないかを決めないと，ソートについて議論しても仕方がない．これは，コンピューターのハードウェアーの問題に関わり，相当難しい問題であるので，証明無しで結論だけ述べよう．整列の問題を考えれるときに，コンピューターができることは次の処理だけである．

- [**比較と分岐**] 大小関係の比較が可能である．比較の結果により処理を分岐できる．
- [**代入**] 変数や配列に，値をコピーすることが可能である．
- [**繰り返し**] 同じ処理を繰り返すことが可能である．

例えば，数列の中から最大値を探すことすらできないのである．最大値を探したければ，これら 3 つの処理を組み合わせる必要がある．ソートも同様で，これらの処理を組み合わせて，高速に整数を並び替えなくてはならない．

*独立行政法人　秋田工業高等専門学校　電気情報工学科

2.2 ここで解く問題

この講義で解く問題は，配列にランダムに格納された整数を並び替える問題である．図 1 に示すように，配列 a[0]〜a[9] には整数がランダムに格納されているとする．これを先に示した 3 つの処理を使って，昇順—小さい順—に並び替えるのである．

このプリントでは，並び替える整数の個数は 10 個としているが，実際にはずーと大きな個数のソートが必要となる．例えば秋田県の人の統計処理をする場合でも，100 万程度のデータがある．

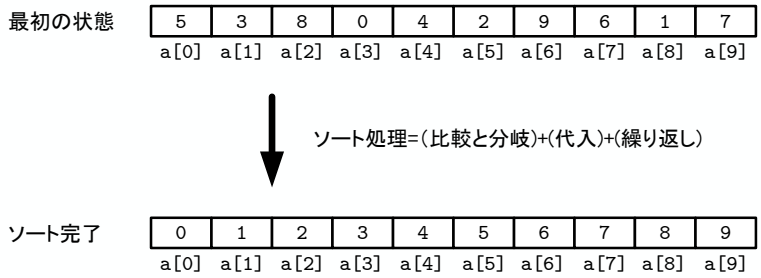

図 1: ソートのアルゴリズムで解く問題．ここでは配列に格納された整数を昇順に並べる．

3 単純挿入ソート

3.1 アルゴリズム

単純挿入ソートは，経験を積んだトランプ師がカードを並び替えるときに使う方法である [1]．まず，2 枚目をカードを取り出し 1 枚目と順序関係が正しい場所に入れる．次に 3 枚目のカードを取り出して，1〜2 枚目のカードの正しい位置に置く．これを繰り返し，最後には 52 枚目のカードを取り出し，1〜51 枚目のカードの正しい位置に置く．この 51 回の繰り返しにより，すべてのカードを正しく並べることができる．このトランプ師の整列の方法をコンピュータープログラムで行うのである．

単純挿入ソートは，a[0] から a[i-1] まで並び替えが終わっているとき，a[i] を正しい位置に入れる—ことを繰り返す方法である．その様子を図 2 に示す．処理の手順は次の通りである．

1. 処理する a[i] の値 35 よりも，48 は大きいので 35 の場所へ移動させる．
2. 次の 44 も 35 より大きいので，48 の場所へ移動させる．
3. 次の 41 も 35 より大きいので，44 の場所へ移動させる．
4. 次の 39 も 35 より大きいので，41 の場所へ移動させる．
5. 次の 32 は 35 より小さいので，それは移動させない．そして，元の 35 を 39 の場所へ移動させる．

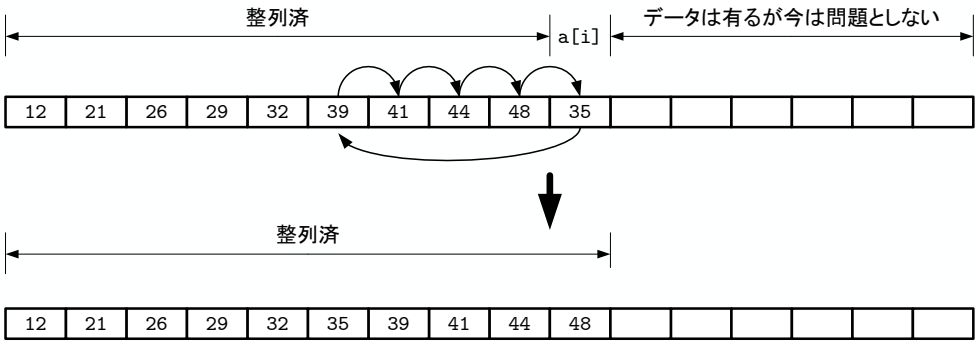

図 2: 単純挿入ソートの基本操作．a[i] を正しい位置に入れる．

3.2 プログラム

単純挿入ソートの実際のプログラム (関数) は，教科書 [2]p.219 のリスト 6.20 のように書く．整列させる整数のデータは，配列は array[] に格納されている．データの個数は num に格納されている．したがって，整列すべき整数のデータは，配列の array[0]〜array[num-1] に格納されている．

この関数がソートする様子を図 3 に示す．先に示したアルゴリズムの通りであることが分かるだろう．

この関数を呼び出すときには，次のようにする．

```
insertion_sort(a,10);
```

整列すべき整数は，a[0]〜a[9] に入っている．データの個数は 10 個である．

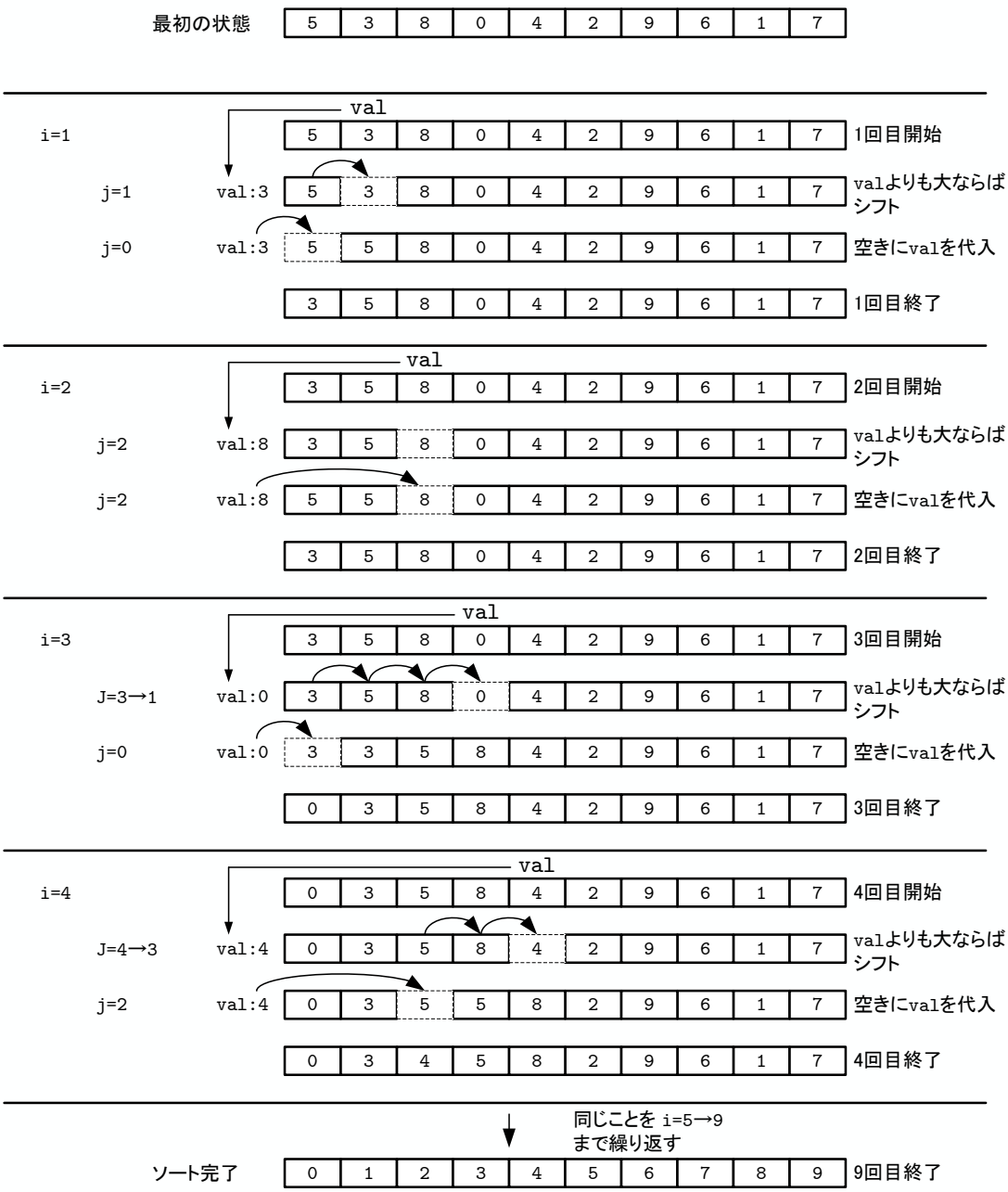

図 3: 単純挿入ソートを使った整数の整列．教科書 [2]p.219 のリスト 6.20 `insertion_sort()` 関数の動作．

3.3 計算量

単純挿入ソートの計算量は，内側の繰り返し分のループ回数で決まる．教科書 [2]p.219 のリスト 6.20 では，j の部分のループ回数である．ここでは，その計算量のオーダーを見積もる．

ソートするデータの個数を N とする．外側のループは $i = 1 \sim N-1$ までである．データがランダムの場合，内側のループは平均して $i/2$ 回繰り返される．したがって，内側のループの総繰り返し回数は，

$$\begin{aligned} \text{内側のループの繰り返し回数} &= \sum_{i=1}^{N-1} \frac{i}{2} \\ &= \frac{N^2 - N}{4} \end{aligned} \tag{1}$$

となる．データの個数が大きくなると，N^2 が支配的になる．したがって，計算量のオーダーは $O(N^2)$ となる．コンピューターでの処理の時間は，データ数の二乗 (N^2) で増加するということである．教科書 [2]p.220 の表 6.3 のランダムに並んでいる配列の場合，要素数の二乗に比例して実行時間がかかっていることが分かるであろう．

4 シェルソート

4.1 アルゴリズム

シェルソートは，は 1959 年に D.L.Shell が考案した方法で，単純挿入ソートを改良したものとなっている[1]．単純挿入ソートは，小さな値が右の方にあると，かなりの計算量を要して，左に移動する．隣同士ひとつずつ比較を行うので，多くの計算が必要となる．最初は大きな歩幅で，そしてだんだん歩幅を小さくしていけば，遠く離れても早く移動できるだろう．このような方法がシェルソートである．

単純挿入ソートは隣同士を比較したが，Shell ソートでは，大きな歩幅 h で比較する．ソートに時間のかかる大ききな数や小さな数は，一気に右や左に移動することができる．h 飛ばしで比較すると，

$$\begin{aligned} &a[0] \leqq a[h] \leqq a[2*h] \leqq a[3*h] \leqq a[4*h] \leqq a[5*h] \cdots \\ &a[1] \leqq a[h+1] \leqq a[2*h+1] \leqq a[3*h+1] \leqq a[4*h+1] \leqq a[5*h+1] \cdots \\ &a[2] \leqq a[h+2] \leqq a[2*h+2] \leqq a[3*h+2] \leqq a[4*h+2] \leqq a[5*h+2] \cdots \\ &\qquad\qquad \vdots \\ &a[h-1] \leqq a[2*h-1] \leqq a[3*h-1] \leqq a[4*h-1] \leqq a[5*h-1] \leqq a[6*h-1] \cdots \end{aligned}$$

と並び替える．この並び替えには単純挿入法をつかう．

そうして，歩幅 h をどんどん小さく，最後は $h = 1$ にすると並び替えは完了となる．この h の選び方にこつがあって，$\cdots, 121, 40, 13, 4, 1$ とする．式で表すと，

$$h_k = 3h_{k+1} + 1 \tag{2}$$

[1] この辺の説明は，www.rkmath.rikkyo.ac.jp/ kida/shellsort.htm を参考にしている．

である．これは，互いに素に近い数列である．このようにすると効率が良い．最初に実行する一番大きな h は，データの個数 N 以下にしなくてはならない．教科書では $N/3$ 以下にしている．他の文献 [1] [3] では，N 以下にしている．私だと $N/2$ 以下にするだろう．$N/2$ 以下にすると，必ずすべてのデータが一度はソートされることになる．いずれにしても，計算量にあまり大差はないであろう．

まとめると，シェルソートの手順は，次の通りである．

[**ステップ 1**] 最初の歩幅 h を決める．データの個数の半分以下で最大の h_i を最初の歩幅とする．

[**ステップ 2**] $i = 0, 1, 2, \cdots, h-1$ に対して，`a[i],a[h+i],a[2*h+i],a[3*h+i],`$\cdots$ を並び替える．これには単純挿入ソートを使う．

[**ステップ 2**] 次の `h=(h-1)/3` にして，再度 [ステップ 2] の並び替えを実行する．実際には，`h=h/3` で良い．

4.2 プログラム

単純挿入ソートの実際のプログラム (関数) は，教科書 [2]p.223 のリスト 6.21 のように書く．整列させる整数のデータは，配列は `array[]` に格納されている．データの個数は `num` に格納されている．したがって，整列すべき整数のデータは，配列の `array[0]`〜`array[num-1]` に格納されている．

この関数がソートする様子を図 4 に示す．先に示したアルゴリズムの通りであることが分かるだろう．

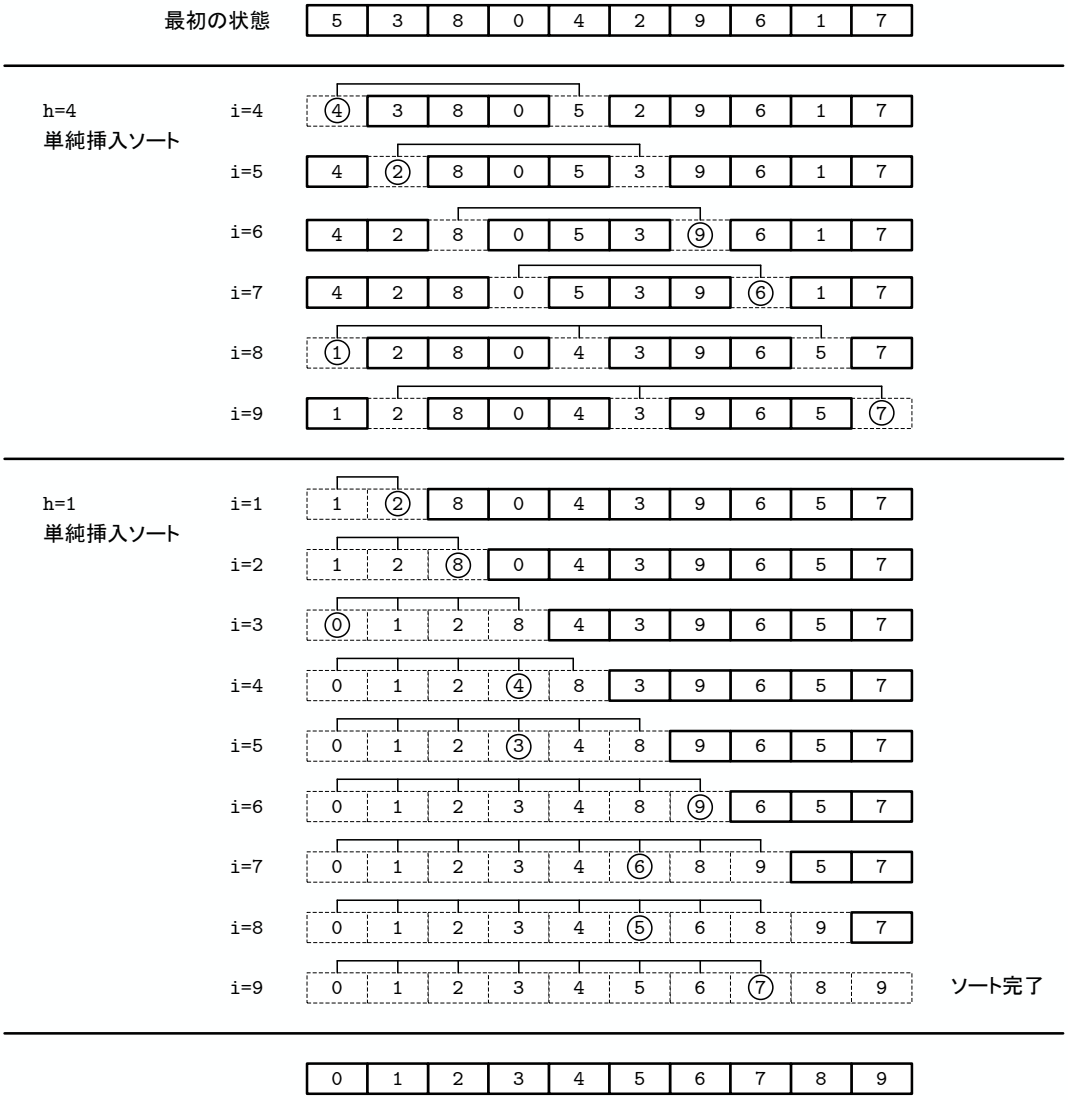

図 4: シェルソートを使った整数の整列．リストのプログラムのソート方法.

4.3 計算量

シェルソートの計算量を見積もるのは，非常に難しい．移動の歩幅 h に大きく依存するからである．式 (2) のように h を選べば，平均して $O(N^{1.25})$ となることが実験的に知られている [3].

これは，かなり良いアルゴリズムである．後で述べるクイックソートと比較してもそんなに悪くはない.

5　クイックソート

5.1　アルゴリズム

クイックソートのアルゴリズムは分割統治法の良い例である．分割統治法では，はじめに問題をいくつかの部分に分けて，それを解く．そして，解いた結果を組み合わせることにより，全体の問題の解とする方法である．部分問題も全体問題も全く同じアルゴリズムが使えるときに有効な方法である．そのような場合，再帰処理が使える．

クイックソートでは，まず適当な基準となる値を決める．そして，それより大きな値と小さな値の数列に分割する．それぞれ分割した数列で，また同じように，基準を設けて数列を分ける．これを繰り返すことにより，数列を整列させることができる．

後の説明は，教科書の通り．

5.2　プログラム

クイックソートの実際のプログラム (関数) は，教科書 [2]pp.226–227 のリスト 6.22 のように書く．整列させる整数のデータは，配列は `array[]` に格納されている．データの個数は `num` に格納されている．したがって，整列すべき整数のデータは，配列の `array[0]`〜`array[num-1]` に格納されている．

この関数がソートする様子を図 5 に示す．先に示したアルゴリズムの通りであることが分かるだろう．

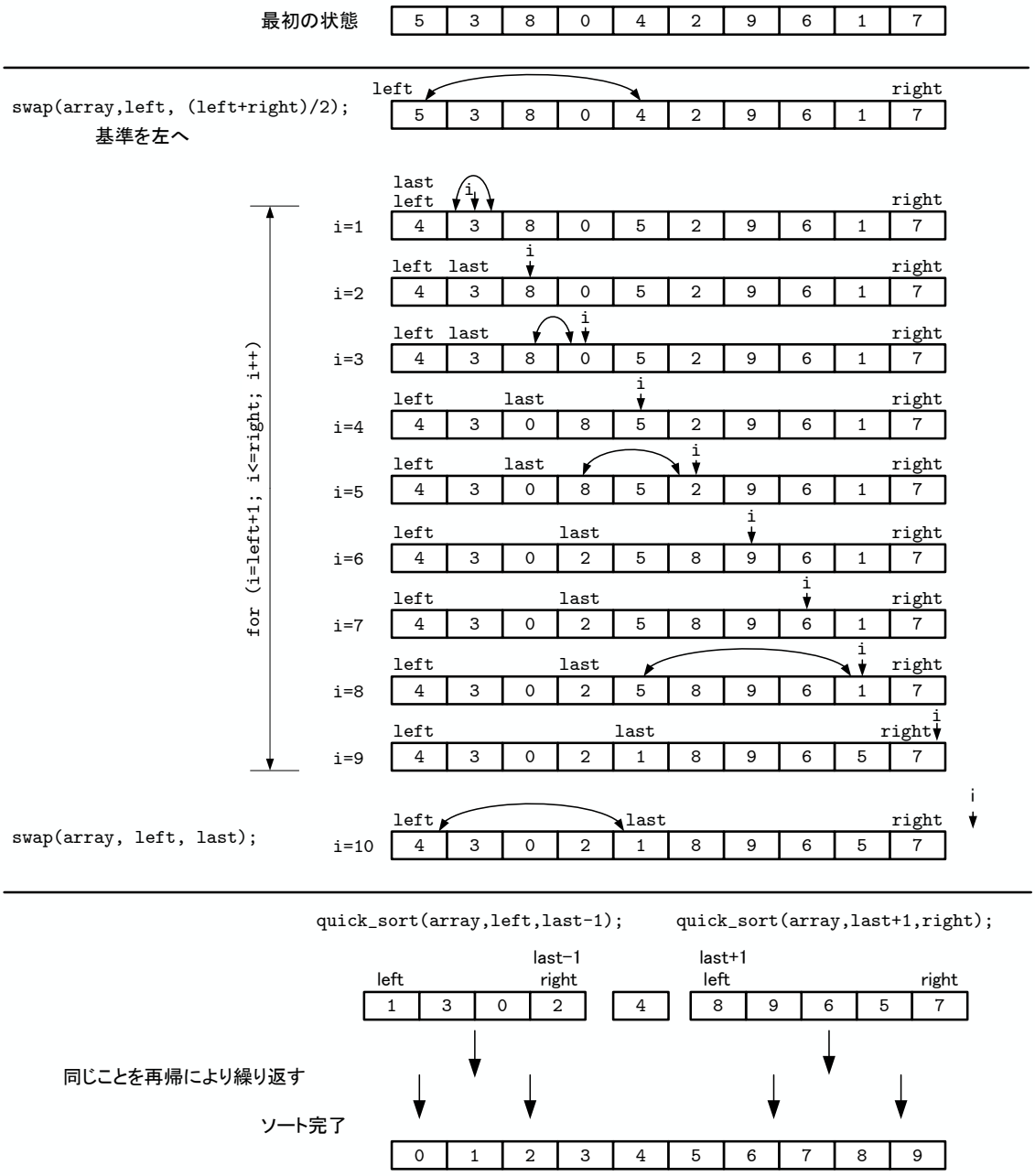

図 5: クイックソートを使った整数の整列．教科書 p.226–227 のリスト 6.22 の関数のソート方法.

5.3 計算量

クイックソートの計算量は，$O(N \log N)$ である．大きな N に対しても，$\log N$ はなかなか大きくならない．そのため，大量のデータをソートするときには，クイックソートは有利である．

参考文献

[1] Willam H. Press et al. NUMERICAL RECIPES in C [日本語版]. 技術評論社, 1996.

[2] 内田智史監修, (株) システム計画研究所編. C 言語によるプログラミング　応用編　第 2 版. (株) オーム社, 2006.

[3] 石畑清. アルゴリズムとデータ構造, 岩波講座　ソフトウェアー科学, 第 3 巻. 岩波書店, 2004.