ソート

ソートはアプリケーションを作成する際に良く使われる基本的な操作であり，いままでに多くのソートアルゴリズムが考えられてきました．今回は４つのソートアルゴリズムについて学習していきます．

ソートとは

ソートとは与えられたデータの集合をキーとなる項目の値の大小関係に基づき，一定の順序で並べ替える操作です．ソートにはキー値が小さい順に並べる昇順とその逆の降順があります．

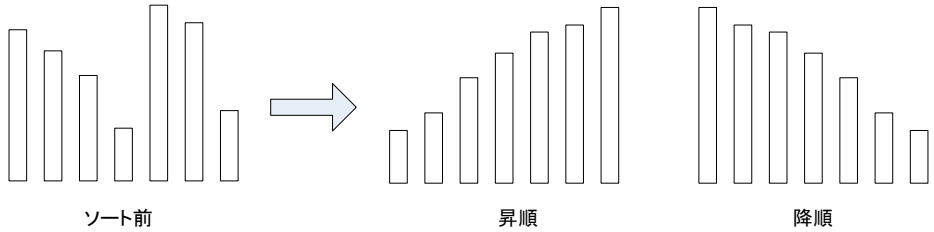

図１　昇順と降順

ソートを行った時，等しいキー値をもつデータの位置関係がソート前後においても保たれるアルゴリズムは安定なソートと呼ばれています．ソートには安定なソートと安定でないソートがあります．

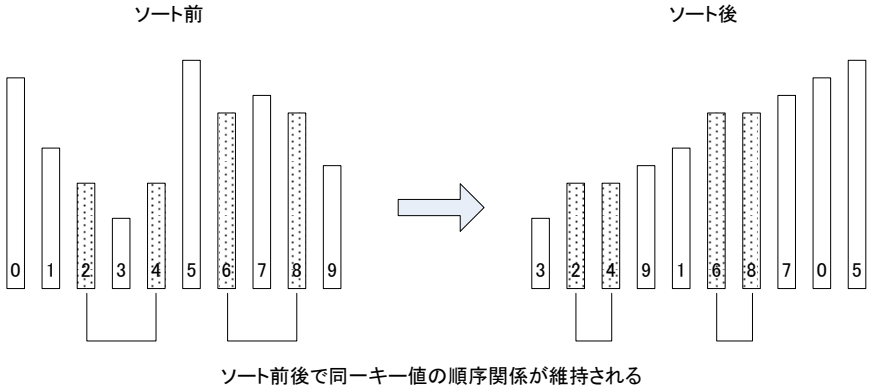

図２

ソートにはコンピュータのメモリ上に確保される配列だけでソートを行う場合と外部記憶装置上のファイルとデータのやり取りを行いながらソートを行う場合があり，前者を内部ソート，後者を外部ソートと呼びます．外部ソートは内部ソートの応用ですが，その実現には作業用ファイルを用いるなど，複雑なアルゴリズムとなります．今回は内部ソートのアルゴリズムについて学習していきます．

シェルソート

シェルソートは単純挿入ソートに工夫を加えたソートアルゴリズムです．まずは単純挿入ソートを説明します. 図 3 のように配列にデータが格納されています. 最初に 2 番目の要素である 4 に着目します. この値は 6 よりも先頭側にあるべきなので, 先頭に挿入します. この処理に伴って 6 を右にずらします. 同様にして，順に要素に着目していき，値を＜適当な位置に挿入する＞操作を繰り返すとソートが完了します.

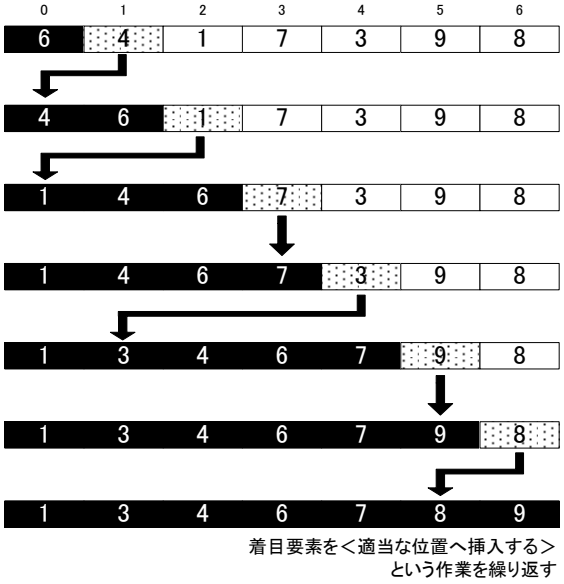

図３　単純挿入ソート

次に以下のデータの並びで単純挿入ソートを行うことを考えます．2 番目の要素 2，3 番目の要素 3，…，5 番目の要素 5 と順に着目していきますが，ここまではソート済みですので，要素の移動は発生しません．6 番目の要素 0 の挿入では 6 回の移動が必要となります．

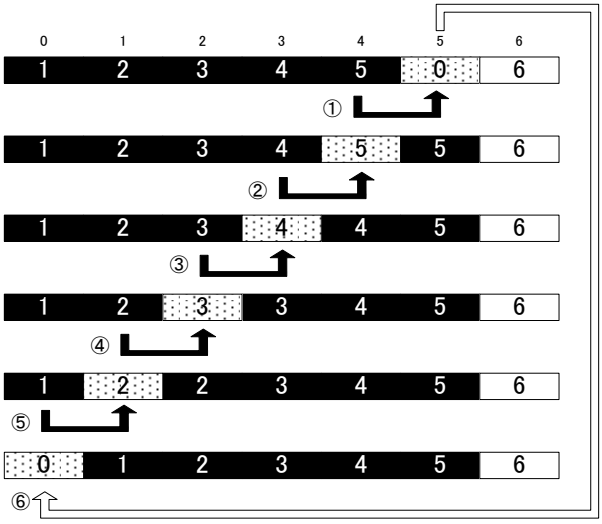

図４　単純挿入ソートにおける要素の交換

このことから，単純挿入ソートに関する以下の特徴が分かります．

[A]　ソート済みあるいはそれに近い状態では高速に動作します．
[B]　挿入先が遠く離れている場合は，移動回数が多くなります．

シェルソートでは[A]の長所を活かし，[B]の短所を補うように改良されました．挿入ソートを少し変形しただけですが非常に高速に動作します．このアルゴリズムでは最初はなるべく離れた要素をグループ化して大まかなソートを行い，そのグループを縮小しながらソートを繰り返すことによって移動回数を減らそうというアイデアに基づいています．

それでは次に示すデータの並びでシェルソートの手順を示します．まず，4 つ離れた要素を取り出し，{8,7}，{1,6}，{4,3}，{2,5}の 4 つのグループに分割し，それぞれのグループでソートを行います．シェルソートではこのような 4 つ離れた要素のソートのことを４－ソートと呼びます．

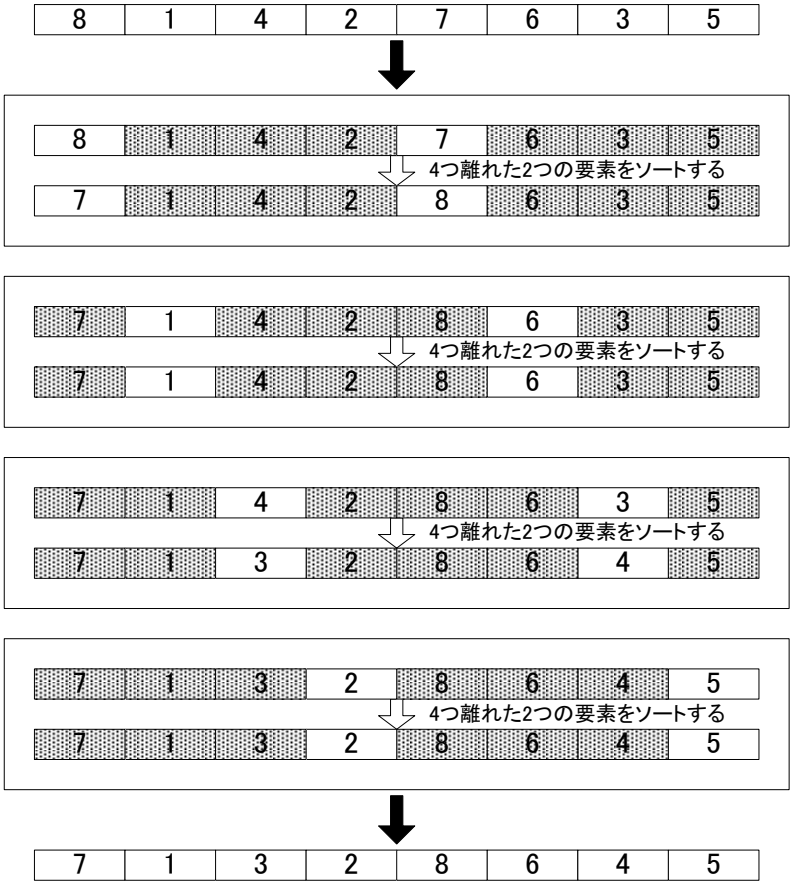

図５　シェルソートにおける４－ソート

４－ソートを行ったデータに対し，次は２つずつ離れた要素を取り出し，{7, 3, 8, 4}，{1, 2, 6, 5}のグループに分割します．２つ離れた要素のソートですので，２－ソートです．

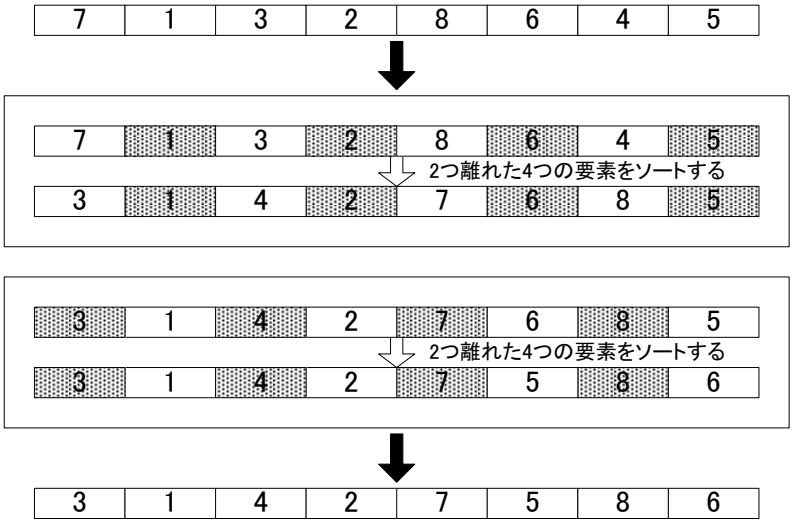

図６　シェルソートにおける２－ソート

最後に 1 つずつ離れた要素，すなわち配列全体をソートします．これを１－ソートといいます．シェルソートの過程におけるソートをｈ－ソートと呼ぶことにすると，ｈの値を 4, 2, 1 と減らしながら，以下のように合計 7 回のソートを行ったことになります．

・要素 2 に対して４－ソートを行う　…　4 組
・要素 4 に対して２－ソートを行う　…　2 組
・要素 8 に対して１－ソートを行う　…　1 組
※いずれのソートもすべて単純挿入ソートによって行います

このように，シェルソートは前準備として４－ソートや２－ソートを行い，ソート済みに近い状態にしておくことで，全体としての要素の移動回数を少なくする方法です．この例では h の値を h = 4,2,1 と変化させましたが，h はある値から減少していき，最後に 1 となればよいです．実際には十分に要素をかき混ぜた方が効率のよいソートが期待できるため，単純に作り出すことができて，よい結果を生み出すことが知られている h = …,121,40,13,4,1 という数列を利用します．1 から始めて，3 倍した値に 1 を加えるという数列です．最初に適用する h の値が大き過ぎても，あまり効果が無いため，通常は配列の要素数 n を 9 で割った値を超えないように決定します．

シェルソートは離れた要素を交換するため安定ではありません．しかし，その時間計算量は $O(n^{1.25})$ であり，非常に高速です．

クイックソート

クイックソートは最も高速なソートアルゴリズムの 1 つです．クイックソートは問題を小問題（部分問題）に分割し，小問題の解を結合して全体の解を得ようとする手法であり，これを分割統治法と呼びます．分割統治法は再帰的な手続きを用いることによって，非常に簡潔に実現できます．このアルゴリズムの概略を以下に示す 8 人のグループを身長順にソートする例を用いて説明します．

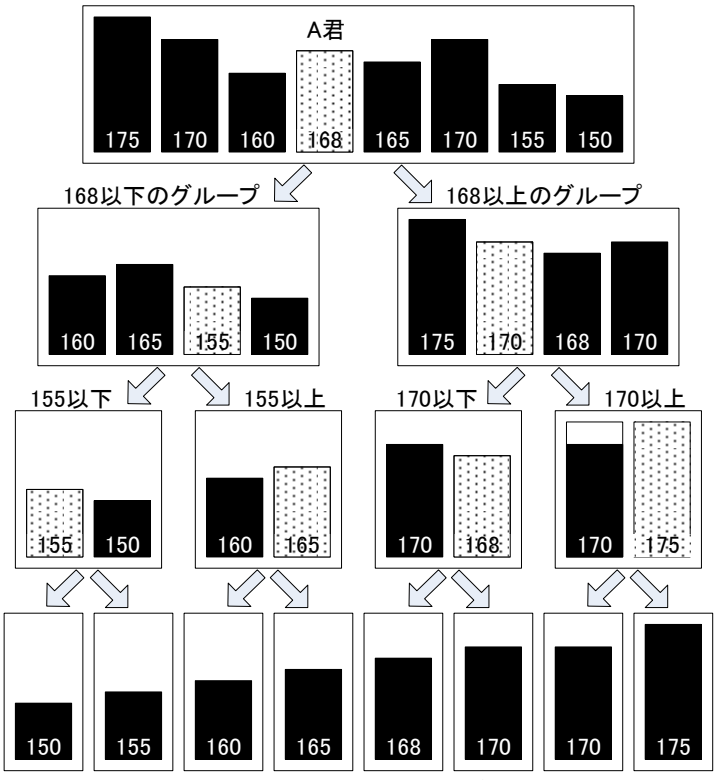

図７　クイックソートの考え方

図の最上段において，身長が 168cm の A 君に着目すると，A 君以上のグループと A 君以下のグループに分けることができます．この A 君の身長のようにグループ分けの基準とする要素を枢軸と呼びます．引き続き 2 つになったグループに対して，新たに枢軸を選択し，分ける作業を繰り返します．最終的にすべてのグループが 1 人になったとき，ソートが完了していることになります．

枢軸の選択は任意のルールに基づいて決定しますが，最も単純な方法は分割する配列の左端，中央，右端のいずれかの要素を枢軸として採用することで，プログラムの実現も簡単です．しかし，枢軸の選択には次のような問題があります．以下に示す配列のクイックソートの場合，枢軸として左端の要素を

採用するとどうなるでしょうか.

8	7	6	5	4	3	2	1

この配列は枢軸 8 だけのグループとそれ以外のグループとに分割されます．このような偏った分割を繰り返すような場合には高速なソートは期待できません．理想的にはデータの中央値を枢軸にすれば，配列は偏ることなく半分の大きさに分割されます．しかし，中央値を求める問題はそれなりの処理が必要であり，その部分に計算時間をかけるのは本末転倒です．そこで，最悪の場合をさけることが期待できる次に示す方法が比較的よく用いられます.

◆分割すべき配列の要素数が 3 以上であれば，先頭の要素，中央の要素，
末尾の要素の三値の中央値をもつ要素を枢軸として選択する.

クイックソート時間計算量は O(n log n)です．しかし，上述の例のように，ソートする配列の要素の初期値や枢軸の選択法によっては，高速なはずのクイックソートも，遅くなってしまう場合があります．例えば，ただ 1 つの要素とそれ以外のグループというように分割を繰り返すと，n 回の分割が必要となって $O(n^2)$ となってしまい．単純ソートと変わらなくなります.

マージソート

マージソートは再帰的に配列を前半部と後半部の 2 つに分け，それをマージ（併合）することによってソートを行う方法で，分割統治アルゴリズムの一種です.

まずは図８を用いて，2 つのソート済み配列をマージする方法について説明します．各配列の先頭から順に比較を行い，小さい方の値をもつ要素を先に取り出して別の配列に格納する操作を繰り返すと，ソート済み配列ができあがります．簡単なアルゴリズムで実現されており，このマージの操作の時間計算量は O(n)です．マージの手続きは⊗記号で表すことにします.

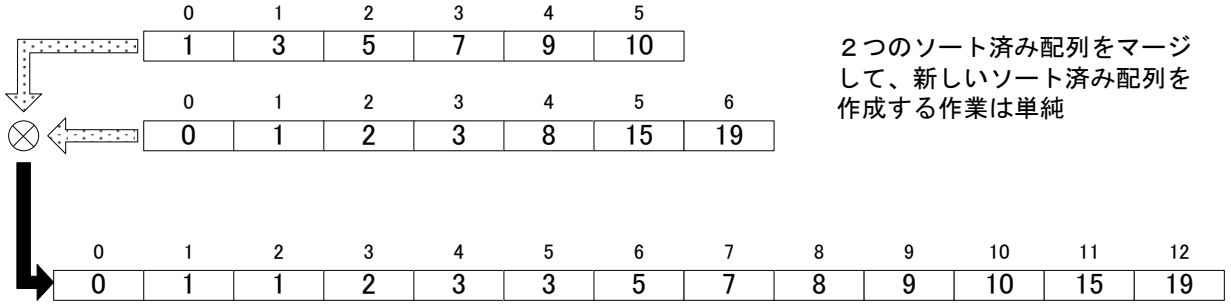

図８　ソート済み配列のマージ

マージソートでは配列を 2 つに分割し，それぞれをソートした後でマージすると全体のソートが完了します．以下に手順を示します.

配列の要素数が 2 以上であれば，以下の手続きを適用する
■配列の前半部をマージソートする
■配列の後半部をマージソートする
■配列の前半部と後半部をマージする

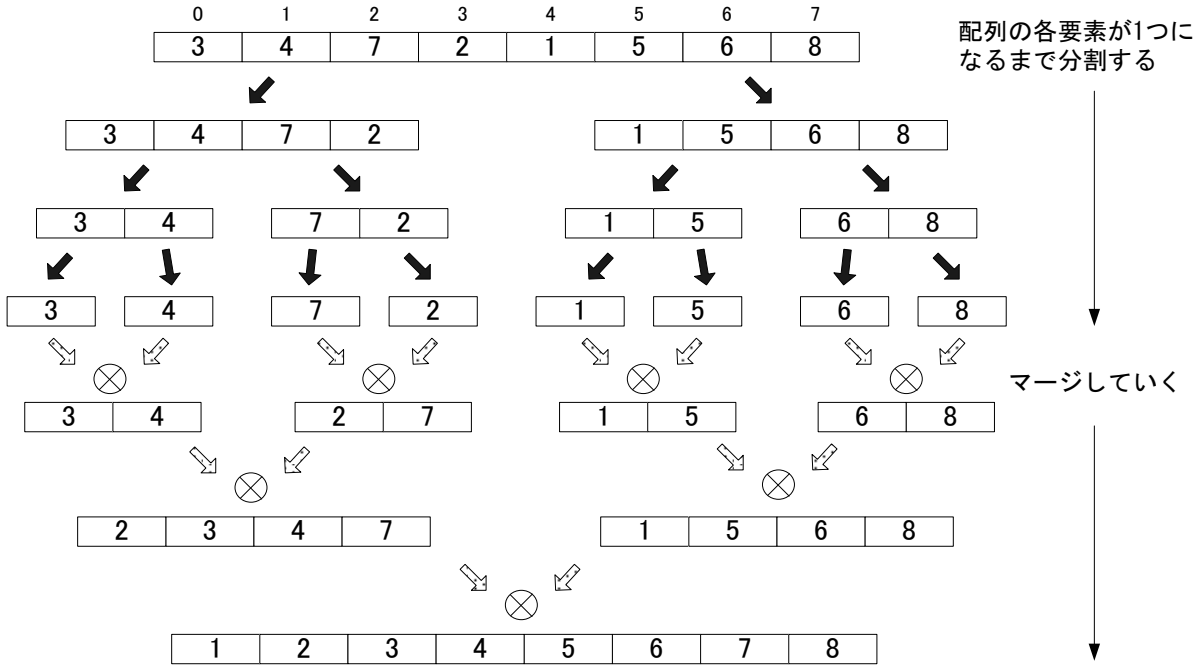

図９　マージソート

配列のマージの操作そのものは時間計算量が O(n)ですが，データの要素数が n のとき，マージソートの階層としては log n の深さが必要ですので，全体の時間計算量は O(n log n)です．

度数ソート

これまでのソートアルゴリズムは何らかの形で 2 つの要素のキー値を比較するものでした．ここで説明する度数ソートは比較の必要がないという特徴があります．例として，10 点満点のテストの学生 9 人分のデータを対象とします．データは以下に示す配列 a に格納されています．

	0	1	2	3	4	5	6	7	8
a	5	7	0	2	4	10	3	1	3

■Step1　度数分布表の作成

配列 a をもとに＜各点数の学生が何人いるか＞を表す度数分布表を作成します．点数は 0 から 10 点の 11 種類ありますので，その格納先は要素数が 11 ある配列 f です．配列 a のデータを元に，配列 f を完成させると以下のようになります．

	0	1	2	3	4	5	6	7	8	9	10
f	1	1	1	2	1	1	0	1	0	0	1

■Step2　累積度数分布表の作成

次に，＜0 点からその点数までに何人の学生がいるか＞を表す累積度数分布表を作成します．この作業は配列 f の 2 番目以降の要素に対して，1 つ手前の要素の値を加えていくだけです．その結果，配列 f は以下となります．

	0	1	2	3	4	5	6	7	8	9	10
f	1	2	3	5	6	7	7	8	8	8	9

たとえば，f[4]の値は 6 ですが，これは 0 点から 4 点までに累計 6 人いることを表します．

■Step3　目的配列の作成

残る作業は原配列の各要素の値と累積度数分布表をつきあわせ，ソート済みの配列を作成することです．そのためには，原配列 a と同じ要素数をもった作業用の配列が必要です．ここではその配列を b とします．原配列 a の要素を末尾側から走査しながら，つきあわせを行います．末尾要素 a[8]の値は 3 です．累積度数を表す配列 f[3]の値を調べてみると 5 ですので，0 点から 3 点までに 5 人いることが分かります．そこで，目的配列 b[4]に 3 を格納します．

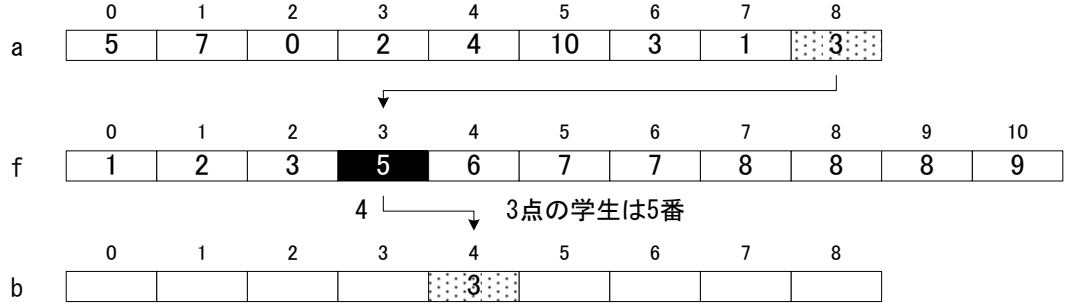

図１０　目的配列の作成（その 1）

プログラムではこの作業を行う際に f[3]の値を 5 から 4 にデクリメントします．これは同じ値が複数あった場合に格納先がぶつからないようにするための配慮です．次の要素 a[7]の値は 1 です．累積度数を表す配列 f[1]の値は 2 で，0 点から 1 点までに 2 人いることが分かります．したがって，作業用の目的配列 b[1]に 1 を格納します．

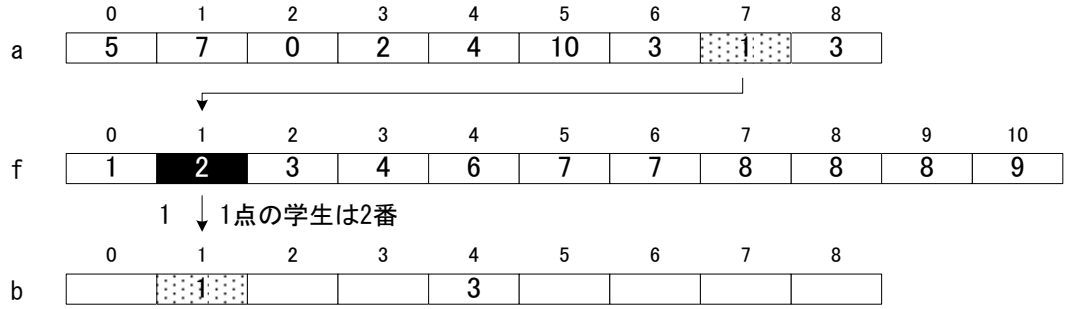

図１１　目的配列の作成（その 2）

さらに走査を続けます．次に着目する a[6]の値は 3 です．3 点の学生の格納を行うのは 2 回目となり，先ほど a[8]の値 3 を目的配列に格納する際に，f[3]の値はデクリメントを行い，5 から 4 にしていました．したがって，今回は目的配列の 4 番目の要素 b[3]に格納することになります．

図１２　目的配列の作成（その 3）

以上の作業を a[0]まで行うとソートが完了し，その後，配列 b の全要素を配列 a の要素にコピーすることで操作が終了します．このアルゴリズムはテストの点数のように，整数値でデータの最小値と最大値があらかじめ分かっている場合にしか適用できません．そのかわり，データの比較や交換の作業が不要なので，非常に高速にソートを実行できます．また，各ステップでは配列の要素を飛び越えることなく順に走査するので,ソートアルゴリズムは安定です.Step3 で配列 a の操作を末尾側から行いましたが，これを先頭側から行うと，安定ではなくなるので注意が必要です.

課題３

Kadai3-1

List6-7 はシェルソートを用いてデータを昇順にソートするプログラムです．このプログラムを降順にソートするように変更してください.

Kadai3-2

List6-9 はクイックソートを用いてデータを昇順にソートするプログラムです．このプログラムを降順にソートするように変更してください．また，枢軸の選択方法を以下に示す方法に変更してください.

「分割すべき配列の要素数が 3 以上であれば，先頭の要素，中央の要素，末尾の要素の三値の中央値をもつ要素を枢軸として選択する．分割された集合の個数が偶数となり，中央の要素の候補が 2 つある場合には，中央の要素は先頭側に近い方が選択されるものとする.」

Kadai3-3

List6-14 はマージソートを用いてデータを昇順にソートするプログラムです．このプログラムを降順にソートするように変更してください.

Kadai3-4

List6-17 は度数ソートを用いてデータを昇順にソートするプログラムです．このプログラムを降順にソートするように変更してください.

※結果の配列を逆順にしないでください．計算時間が増加してしまいます.
ヒント：累積度数分布表を大きい方から作ります.