第9章　C言語 応用編：ソーティング

9.1　ソーティング

コンピュータは、大量のデータを高速かつ正確に処理することを得意としますが、その中でも、データの検索・追加・削除・並び替えといった操作はデータ処理の基本となります。この章では、並び替え、すなわち**ソーティング** (sorting) に絞って、そのアルゴリズムを紹介します。

ソーティングは、与えられた n 個のデータの列 $\{a_1,\ a_2,\ \cdots,\ a_n\}$ をある基準に従って順に並び替えることです[1]。並べ替える基準には、データを $a'_1 \leq a'_2 \leq \cdots \leq a'_n$ のように単調増加的に並べる**昇順ソート** (ascending sort) と、データを $a'_1 \geq a'_2 \geq \cdots \geq a'_n$ のように単調減少的に並べる**降順ソート**(descending sort) があります。

有名なソーティングアルゴリズムには、

- **9.2 節**　**選択ソート** (selection sort)　* **最小値選択法**とも呼ばれる。
- **9.3 節**　**バブルソート** (bubble sort)
- **9.4 節**　**挿入ソート** (insertion sort)　* 挿入ソートを拡張した**シェルソート** (shell sort) がある。
- **9.5 節**　**クイックソート** (quick sort)
- **9.6 節**　**ヒープソート** (heap sort)
- (**9.7 節**　**マージソート** (merge sort)　* **併合ソート**とも呼ばれる。)

[1] ここで言うある基準とは、データの大小の比較によってソーティングを行なうことです。なお、本テキストでは紹介しませんが、**ビンソート** (bin sort)・**分布数え上げソート** (distribution counting sort)・**基数ソート** (radix sort) など、比較によらないソーティング方法もあります。

があります。

なお、これらアルゴリズムはその性格によって、データ列を主記憶装置 (メモリ) に置いてソーティングするのに適した**内部ソート** (internal sort; **内部整列**) と、データ列をテープやハードディスクなどの外部装置に置いてソーティングするのに適した**外部ソート** (external sort; **外部整列**) に分類されます[2]。上記に挙げたアルゴリズムの中では、唯一、マージソートが外部整列に属しています。

以後、各アルゴリズムの解説を行いますが、データ列には $\{4,\ 10,\ 5,\ 2,\ 1,\ 7,\ 8,\ 6,\ 3,\ 9\}$ を具体例として使用し、

$$\{4,\ 10,\ 5,\ 2,\ 1,\ 7,\ 8,\ 6,\ 3,\ 9\} \quad \Longrightarrow \quad \{1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10\}$$

のように昇順にソーティングされて行く過程を観察します。ただし、C 言語によるプログラムでは、添え字は0 から始めるのが普通ですが理解しやすいように添え字を1 から始めてあります (配列を「`int a[10+1]={0,4,10,5,2,1,7,8,6,3,9};`」と宣言してあります)。また、各ソーティングアルゴリズムについて、少し安直ですが比較回数を基準に**計算量**のオーダを導き、アルゴリズムの良し悪しについても考察します (1 回の比較を 1 演算として換算します)。

9.2　選択ソート

選択ソート (最小値選択法) は、未整列部分のデータ列の最小値を選択し、未整列部分の先頭と最小値を交換する操作を繰り返し実行することでソーティングを行ないます。アルゴリズムは、

Step 1	$a_i = \min(a_1,\ a_2,\ \cdots,\ a_n)$ を探し、a_i と a_1 を交換する。a_1 が決定する。
	* $\min(a_1,\ a_2,\ \cdots,\ a_n)$ を決定するために $n-1$ 回の比較が必要となる。
Step 2	$a_j = \min(a_2,\ a_3,\ \cdots,\ a_n)$ を探し、a_j と a_2 を交換する。a_2 が決定する。
	* $\min(a_2,\ a_3,\ \cdots,\ a_n)$ を決定するために $n-2$ 回の比較が必要となる。
⋮	⋮
Step $n-1$	$a_k = \min(a_{n-1}, a_n)$ を探し、a_k と a_{n-1} を交換する。a_{n-1} と a_n が決定する。
	* $\min(a_{n-1},\ a_n)$ を決定するために 1 回の比較が必要となる。

となります。従って、データ列 $\{4,\ 10,\ 5,\ 2,\ 1,\ 7,\ 8,\ 6,\ 3,\ 9\}$ を選択ソートによってソーティングすると表 9.1 のようになります。

なお、選択ソートでは、Step 1 で $n-1$ 回、Step 2 で $n-2$ 回、$\cdots$、Step $n-1$ で 1 回の比較が必要となるので、計算量は

$$(n-1)+(n-2)+\cdots+1 = \frac{\{1+(n-1)\}\cdot(n-1)}{2} = \frac{n^2-n}{2}$$

となります。更に、オーダ記法で表すと $O(n^2)$ となります。

[2]データ列が主記憶装置に収まらない場合は、データ列を主記憶装置に収まるように分け、分けた各データ列を内部整列でソーティングし、外部整列で各データ列を結合します。現実的には、このようにアルゴリズムを組み合わせてソーティングするのが一般的です。

START	{4, 10, 5, 2, 1, 7, 8, 6, 3, 9}	
Step 1	{4, 10, 5, 2, 1, 7, 8, 6, 3, 9}	未整列部分から最小値を探し、
	{1, 10, 5, 2, 4, 7, 8, 6, 3, 9}	最小値 1 と a_1 を交換する。
	{1, 10, 5, 2, 4, 7, 8, 6, 3, 9}	a_1 が決定する。
Step 2	{1, 10, 5, 2, 4, 7, 8, 6, 3, 9}	未整列部分から最小値を探し、
	{1, 2, 5, 10, 4, 7, 8, 6, 3, 9}	最小値 2 と a_2 を交換する。
	{1, 2, 5, 10, 4, 7, 8, 6, 3, 9}	a_2 が決定する。
Step 3	{1, 2, 3, 10, 4, 7, 8, 6, 5, 9}	以下、同様の手順を繰り返す。
Step 4	{1, 2, 3, 4, 10, 7, 8, 6, 5, 9}	
Step 5	{1, 2, 3, 4, 5, 7, 8, 6, 10, 9}	
Step 6	{1, 2, 3, 4, 5, 6, 8, 7, 10, 9}	
Step 7	{1, 2, 3, 4, 5, 6, 7, 8, 10, 9}	
Step 8	{1, 2, 3, 4, 5, 6, 7, 8, 10, 9}	
Step 9	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}	
END	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}	

表 9.1: 選択ソートによるソーティングの例

問題 1 「選択ソートによるソーティングプログラム」を作成しなさい。

9.3 バブルソート

バブルソート[3]は、隣り合う 2 つの項を比較しながらデータ列を左から右へ走査することでソーティングを行ないます。ただし、隣り合う 2 つの項を比較したとき 2 つのデータが並べ替えの基準を満たさなければデータを交換するという作業を行います。結果的には、データ列の右から順にソーティングが完了して行きます。アルゴリズムは、

Step 1	a_i と a_{i+1} を比較し、$a_i > a_{i+1}$ ならば交換する $(i = 1, 2, \cdots, n-1)$。
	a_n が決定する。　* $n-1$ 回の比較が必要となる。
Step 2	a_i と a_{i+1} を比較し、$a_i > a_{i+1}$ ならば交換する $(i = 1, 2, \cdots, n-2)$。
	a_{n-1} が決定する。　* $n-2$ 回の比較が必要となる。
⋮	⋮
Step $n-1$	a_i $(= a_1)$ と a_{i+1} $(= a_2)$ を比較し、$a_1 > a_2$ ならば交換する $(i = 1)$。
	a_1 と a_2 が決定する。　* 1 回の比較が必要となる。

となります。従って、データ列 {4, 10, 5, 2, 1, 7, 8, 6, 3, 9} をバブルソートによってソーティングすると表 9.2 から表 9.3 までのようになります。

[3] ソーティングによりデータが確定していく様子が、泡が下から上昇している様子に似ていることからバブルソートと呼ばれます。

START	{4, 10, 5, 2, 1, 7, 8, 6, 3, 9}	
Step 1	{4, 10, 5, 2, 1, 7, 8, 6, 3, 9}	a_1 と a_2 を比較し、
	{4, 10, 5, 2, 1, 7, 8, 6, 3, 9}	$a_2 \leq a_3$ なのでそのまま。
	{4, 10, 5, 2, 1, 7, 8, 6, 3, 9}	a_2 と a_3 を比較し、
	{4, 5, 10, 2, 1, 7, 8, 6, 3, 9}	$a_2 > a_3$ なので交換する。
	{4, 5, 2, 10, 1, 7, 8, 6, 3, 9}	・
	{4, 5, 2, 1, 10, 7, 8, 6, 3, 9}	・
	{4, 5, 2, 1, 7, 10, 8, 6, 3, 9}	・
	{4, 5, 2, 1, 7, 8, 10, 6, 3, 9}	・
	{4, 5, 2, 1, 7, 8, 6, 10, 3, 9}	・
	{4, 5, 2, 1, 7, 8, 6, 3, 10, 9}	・
	{4, 5, 2, 1, 7, 8, 6, 3, 9, 10}	(a_{10} が決定する。)
Step 2	{4, 5, 2, 1, 7, 8, 6, 3, 9, 10}	以下、Step 1 と同様の手順を繰り返す。
	{4, 2, 5, 1, 7, 8, 6, 3, 9, 10}	
	{4, 2, 1, 5, 7, 8, 6, 3, 9, 10}	
	{4, 2, 1, 5, 7, 8, 6, 3, 9, 10}	
	{4, 2, 1, 5, 7, 8, 6, 3, 9, 10}	
	{4, 2, 1, 5, 7, 6, 8, 3, 9, 10}	
	{4, 2, 1, 5, 7, 6, 3, 8, 9, 10}	
	{4, 2, 1, 5, 7, 6, 3, 8, 9, 10}	(a_9 が決定する。)
Step 3	{2, 4, 1, 5, 7, 6, 3, 8, 9, 10}	
	{2, 1, 4, 5, 7, 6, 3, 8, 9, 10}	
	{2, 1, 4, 5, 7, 6, 3, 8, 9, 10}	
	{2, 1, 4, 5, 7, 6, 3, 8, 9, 10}	
	{2, 1, 4, 5, 6, 7, 3, 8, 9, 10}	
	{2, 1, 4, 5, 6, 3, 7, 8, 9, 10}	
	{2, 1, 4, 5, 6, 3, 7, 8, 9, 10}	(a_8 が決定する。)
Step 4	{1, 2, 4, 5, 6, 3, 7, 8, 9, 10}	
	{1, 2, 4, 5, 6, 3, 7, 8, 9, 10}	
	{1, 2, 4, 5, 6, 3, 7, 8, 9, 10}	
	{1, 2, 4, 5, 6, 3, 7, 8, 9, 10}	
	{1, 2, 4, 5, 3, 6, 7, 8, 9, 10}	
	{1, 2, 4, 5, 3, 6, 7, 8, 9, 10}	(a_7 が決定する。)

表 9.2: バブルソートによるソーティングの例 (前半)

Step 5	{1, 2, 4, 5, 3, 6, 7, 8, 9, 10}	
	{1, 2, 4, 5, 3, 6, 7, 8, 9, 10}	
	{1, 2, 4, 5, 3, 6, 7, 8, 9, 10}	
	{1, 2, 4, 3, 5, 6, 7, 8, 9, 10}	
	{1, 2, 4, 3, 5, 6, 7, 8, 9, 10}	(a_6 が決定する。)
Step 6	{1, 2, 4, 3, 5, 6, 7, 8, 9, 10}	
	{1, 2, 4, 3, 5, 6, 7, 8, 9, 10}	
	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}	
	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}	(a_5 が決定する。)
	↑ この Step でソーティングが終了している。	
Step 7	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}	
	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}	
	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}	(a_4 が決定する。)
Step 8	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}	
	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}	(a_3 が決定する。)
Step 9	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}	(a_2 と a_1 が決定する。)
END	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}	

表 9.3: バブルソートによるソーティングの例 (後半)

なお、バブルソートでは、Step 1 で $n-1$ 回、Step 2 で $n-2$ 回、$\cdots$、Step $n-1$ で 1 回の比較が必要となるので、計算量は

$$(n-1)+(n-2)+\cdots+1=\frac{\{1+(n-1)\}\cdot(n-1)}{2}=\frac{n^2-n}{2}$$

となります。更に、オーダ記法で表すと $O(n^2)$ となります。

ところで、表 9.3 をよく見るとソーティングは Step 6 で終了しています。すなわち、以降のステップを省略することができます (終了条件は次のステップで交換が起こらなければ、その時点でソーティングを終了します)。特に、完全に整列済みのデータ列であれば Step 1 で終了条件を直ちに満たすため比較回数は $n-1$ 回となり、計算量は $n-1$ となります (オーダ記法で表すと $O(n)$ となります)。また、ほとんど整列済みのデータ列の場合も (Step i ($\ll n$) で終了条件を満たすとすると)、計算量は

$$(n-1)+(n-2)+\cdots+(n-i)=\frac{\{(n-1)+(n-i)\}\cdot i}{2}=\frac{(2i)\cdot n-i^2-i}{2}$$

となり、オーダ記法で表すと $O(n)$ となります[4] ($\because i$ は具体的に与えられた小さな数であるから、$O(i\cdot n)\Longrightarrow O(n)$ となる)。この様に、バブルソートはデータ列に依存したアルゴリズムです。

問題 1 「バブルソートによるソーティングプログラム」を作成しなさい。

[4]ほとんど整列済みのデータ列でも、$\{2,3,4,5,6,7,8,9,10,1\}$ のような場合は $O(n^2)$ となります。ただし、アルゴリズムを「a_i と a_{i+1} を比較し、$a_i>a_{i+1}$ ならば交換する ($i=n-1,n-2,\cdots,1$)」と変更すれば $O(n)$ となります。