9.4　挿入ソート

挿入ソートは、ある位置まで整列済みのデータ列に未整列の要素を適切な位置に挿入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　整列済みのデータ列 $\{a_1\}$ に未整列の要素 a_2 を適切な位置に挿入する。
* 挿入位置を決めるため最小で 1 回、最大で 1 回の比較が必要となる。

Step 2　整列済みのデータ列 $\{a_1,\ a_2\}$ に未整列の要素 a_3 を適切な位置に挿入する。
* 挿入位置を決めるため最小で 1 回、最大で 2 回の比較が必要となる。

⋮　⋮

Step $n-1$　整列済みのデータ列 $\{a_1,\ a_2, \cdots,\ a_{n-1}\}$ に未整列の要素 a_n を適切な位置に挿入する。
* 挿入位置を決めるため最小で 1 回、最大で $n-1$ 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} を挿入ソートによってソーティングすると表 9.4 のようになります。

START	{4, 10, 5, 2, 1, 7, 8, 6, 3, 9}	
Step 1	{{4}, 10, 5, 2, 1, 7, 8, 6, 3, 9}	整列済みデータ列に未整列要素を
	{{4, 10}, 5, 2, 1, 7, 8, 6, 3, 9}	適切な位置に挿入する。
Step 2	{{4, 10}, 5, 2, 1, 7, 8, 6, 3, 9}	整列済みデータ列に未整列要素を
	{{4, 5, 10}, 2, 1, 7, 8, 6, 3, 9}	適切な位置に挿入する。
Step 3	{{4, 5, 10}, 2, 1, 7, 8, 6, 3, 9}	以下、同様の手順を繰り返す。
	{{2, 4, 5, 10}, 1, 7, 8, 6, 3, 9}	
Step 4	{{2, 4, 5, 10}, 1, 7, 8, 6, 3, 9}	
	{{1, 2, 4, 5, 10}, 7, 8, 6, 3, 9}	
Step 5	{{1, 2, 4, 5, 10}, 7, 8, 6, 3, 9}	
	{{1, 2, 4, 5, 7, 10}, 8, 6, 3, 9}	
Step 6	{{1, 2, 4, 5, 7, 10}, 8, 6, 3, 9}	
	{{1, 2, 4, 5, 7, 8, 10}, 6, 3, 9}	
Step 7	{{1, 2, 4, 5, 7, 8, 10}, 6, 3, 9}	
	{{1, 2, 4, 5, 6, 7, 8, 10}, 3, 9}	
Step 8	{{1, 2, 4, 5, 6, 7, 8, 10}, 3, 9}	
	{{1, 2, 3, 4, 5, 6, 7, 8, 10}, 9}	
Step 9	{{1, 2, 3, 4, 5, 6, 7, 8, 10}, 9}	
	{{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}}	
END	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}	

表 9.4: 挿入ソートによるソーティングの例

なお、挿入ソートでは、挿入位置を決定するために、Step 1 で最大 1 回、Step 2 で最大 2 回、$\cdots$、Step $n-1$ で最大 $n-1$ 回の比較が必要となるので、計算量は

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

となります。更に、オーダ記法で表すと $O(n^2)$ となります。

挿入ソートもデータ列によって計算量が変動するアルゴリズムです。最も良いデータ列は、挿入位置を決定するのにどの Step でも比較回数が 1 回で済む場合 (非常に稀なケース) で、その計算量は $n-1$ となります。一般には、挿入位置が平均的に (Step i において等確率 $\frac{i}{2}$ 回の比較で) 見つかると仮定して、計算量

$$\frac{1}{2}+\frac{2}{2}+\cdots+\frac{n-1}{2}=\frac{\left(\frac{1}{2}+\frac{n-1}{2}\right)\cdot(n-1)}{2}=\frac{n^2-n}{4}$$

を求めると、やはりオーダは $O(n^2)$ となります。

問題 1 「挿入ソートによるソーティングプログラム」を作成しなさい。

問題 2 $1, 2, \cdots, 10$ で構成されるデータ列の内、挿入ソートを用いると計算量が最大となるデータ列と最小となるデータ列をそれぞれ求めなさい。

9.5 クイックソート

クイックソートは、データ列から適切な基準値を選び[5]、その基準値より小さな要素で構成されたデータ列と大きな要素で構成されたデータ列に分割することを繰り返し実行することでソーティングを行ないます (分割は、分割されたデータ列の要素の個数が 1 個になるまで再帰的に行なう)。アルゴリズムは、

Step 1 $\{a_1,\ a_2,\ \cdots,\ a_n\}$ から適切な基準値を選ぶ (ここでは a_1 を選ぶ)。基準値より小さな要素 $(a'_i < a_1,\ i=1,\cdots,j)$ で構成されたデータ列 $\{a'_1,\ a'_2,\ \cdots,\ a'_j\}$ と大きな要素 $(a''_i \geq a_1,\ i=1,\cdots,k)$ で構成されたデータ列 $\{a''_1,\ a''_2,\ \cdots,\ a''_k\}$ に分割する $(j+k=n)$。
* 基準値と最大で n 回の比較が必要となる (図 9.2 参照)。

Step 2 $\{a'_1,\ a'_2,\ \cdots,\ a'_j\}$ から基準値 a'_1 を選び、Step 1 と同様に分割する。
$\{a''_1,\ a''_2,\ \cdots,\ a''_k\}$ から基準値 a''_1 を選び、Step 1 と同様に分割する。
* 基準値と最大で $n-1$ 回の比較が必要となる (図 9.2 参照)。

$\vdots$ $\vdots$

Step m 分割された各データ列の要素が全て 1 個になれば終了。
* 基準値と最大で 2 回の比較が必要となる (図 9.2 参照)。

注意：分割されたデータ列の先頭要素を基準値に選んでいます。

注意：Step 数は、再帰的方法による深さを表し、m は最大で $n-1$ となります (図 9.2 参照)。

[5]分割するデータ列の長さを丁度半分にするような基準値の選び方が、基準値の良い選び方です。

となります。従って、データ列 {4, 10, 5, 2, 1, 7, 8, 6, 3, 9} をクイックソートによってソーティングすると表 9.5 のようになります (**注意：下記の「クイックソートによるソーティングプログラム」でソーティングした場合、表 9.5 のソーティング結果とは異なる**)。

START	{4, 10, 5, 2, 1, 7, 8, 6, 3, 9}	
Step 1	{{4, 10, 5, 2, 1, 7, 8, 6, 3, 9}}	4 を基準値に選び、
	{{3, 1, 2, 5, 10, 7, 8, 6, 4, 9}}	並び替え分割する。
Step 2	{{3, 1, 2}, {5, 10, 7, 8, 6, 4, 9}}	3, 5 を基準値に選び、
	{{2, 1, 3}, {4, 10, 7, 8, 6, 5, 9}}	並び替え分割する。
Step 3	{{2, 1}, 3, 4, {10, 7, 8, 6, 5, 9}}	以下、同様の手順を繰り返す。
	{{1, 2}, 3, 4, {9, 7, 8, 6, 5, 10}}	
Step 4	{1, 2, 3, 4, {9, 7, 8, 6, 5}, 10}	
	{1, 2, 3, 4, {5, 7, 8, 6, 9}, 10}	
Step 5	{1, 2, 3, 4, {5, 7, 8, 6}, 9, 10}	
	{1, 2, 3, 4, {5, 7, 8, 6}, 9, 10}	
Step 6	{1, 2, 3, 4, 5, {7, 8, 6}, 9, 10}	
	{1, 2, 3, 4, 5, {6, 8, 7}, 9, 10}	
Step 7	{1, 2, 3, 4, 5, 6, {8, 7}, 9, 10}	
	{1, 2, 3, 4, 5, 6, {7, 8}, 9, 10}	
END	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}	

表 9.5: クイックソートによるソーティングの例

また、Step 1 において、基準値 4 ($= a_1$) より小さな要素で構成されたデータ列と大きな要素で構成されたデータ列に並べ替えを行なう過程を表 9.6 に挙げておきます。

	i								j	
{4,	10,	5,	2,	1,	7,	8,	6,	3,	9}	
{3,	10,	5,	2,	1,	7,	8,	6,	4,	9}	交換する。
	i			j						
{3,	10,	5,	2,	1,	7,	8,	6,	4,	9}	
{3,	1,	5,	2,	10,	7,	8,	6,	4,	9}	交換する。
		i	j							
{3,	1,	5,	2,	10,	7,	8,	6,	4,	9}	
{3,	1,	2,	5,	10,	7,	8,	6,	4,	9}	交換する。
		j	i							
{3,	1,	2 ,	5,	10,	7,	8,	6,	4,	9}	
{3,	1,	2},	{5,	10,	7,	8,	6,	4,	9}	分割する。

表 9.6: Step 1 における並べ替えの過程

注意：基準値がデータ列の最小値の場合は、基準値とそれ以外の要素に分割されます (表 9.5 の Step 5)。

● クイックソートによるソーティングプログラム quicksort.c

```
 1: #include <stdio.h>
 2:
 3: #define N 10
 4:
 5: void quicksort(int a[], int start, int end);
 6: void printarray(int a[], int start, int end);
 7:
 8: int main(void)
 9: {
10:     int a[N + 1] = {0, 4, 10, 5, 2, 1, 7, 8, 6, 3, 9};
11:
12:     quicksort(a, 1, N);
13:
14:     return 0;
15: }
16:
17: void quicksort(int a[], int start, int end)
18: {
19:     int i, j, x, tmp;
20:
21:     i = start;
22:     j = end;
23:     x = a[start];
24:
25:     printarray(a, start, end);
26:     while (1) {
27:         while (a[i] < x) i++;
28:         while (a[j] > x) j--;
29:         if (i >= j) break;
30:         tmp = a[i];
31:         a[i] = a[j];
32:         a[j] = tmp;
33:         i++; j--;
34:         printarray(a, start, end);
35:     }
36:     printf("¥n");
37:     if (start < i - 1) quicksort(a, start, i - 1);
38:     if (j + 1 < end) quicksort(a, j + 1, end);
39: }
```

```
40:
41: void printarray(int a[], int start, int end)
42: {
43:     int i;
44:
45:     for (i = 1; i <= N; i++) {
46:         if (i == start) printf("{");
47:         printf("%3d ", a[i]);
48:         if (i == end) printf("}");
49:     }
50:     printf("¥n");
51: }
```

注意：関数「quicksort()」は再帰的に呼び出されるため、表 9.5 のような Step 順ではソーティングされません。

なお、クイックソートでは、基準値と Step 1 で最大 n 回、Step 2 で最大 $n-1$ 回、$\cdots$、Step $m=n-1$ (図 9.1 のように分割される場合) で最大 2 回の比較が必要となるので、計算量は

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

となります。更に、オーダ記法で表すと $O(n^2)$ となります。

しかしながら、**適切な基準値を取れば**図 9.1 のようになることは稀で、Step 回数 m は n に比べて非常に小さくなります。図 9.2 は比較回数が最小の場合で、$n=2^m$ ($\Longleftrightarrow m=\log n\cdot(\log 2)^{-1}$) が成り立ちます。各 Step の比較回数は n ですから、その計算量は

$$n\cdot m=(\log 2)^{-1}\cdot n\log n$$

となり、オーダ表記すると $O(n\log n)$ となります。

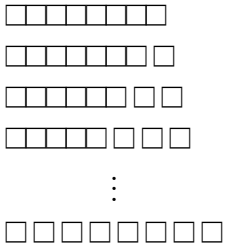

図 9.1: 比較回数が最大となるデータ列

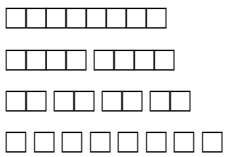

図 9.2: 比較回数が最小となるデータ列

問題 1　「クイックソートによるソーティングプログラム (quicksort.c)」を検証しなさい。