第8章　C言語 応用編

8.1　ソーティング問題

ソーティング(sorting) とは、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。並べ替える基準には、データを単調増加的に並べるソートを**昇降ソート**(ascending sort) と単調減少的に並べるソートを**降順ソート**(descending sort) があります。有名なソーティングとしては、**最小値選択法**・**バブルソート**・**挿入法**・**クイックソート**・**ヒープソート**などがあります。データの列 $\{4, 10, 5, 2, 1, 7, 8, 6, 3, 9\}$ を例に挙げ、各ソーティングについて詳しく見ていきましょう。

なお、ソーティング問題の**計算量**についても、比較の回数を基準に考察していきます。

8.2　最小値選択法

最小値選択法 (selection sort) は、データ列の最小値を選択し未整列部分の先頭に順次繰り返すソーティング方法です。手順としては、

Step 1　$\min(a_1, a_2, \cdots, a_n)$ と a_1 の値を交換する。
Step 2　$\min(a_2, a_3, \cdots, a_n)$ と a_2 の値を交換する。
$\vdots$
Step n-1　$\min(a_{n-1}, a_n)$ と a_{n-1} の値を交換する。

となります。従って、データの列 $\{4, 10, 5, 2, 1, 7, 8, 6, 3, 9\}$ を最小選択法でソーティングすると表 8.1 となります。

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

表 8.1: 最小選択法によるソーティングの例

なお、表 8.1 より、比較の回数は $(9+1)9/2 = 9+8+7+6+5+4+3+2+1$ ですから、n 個のデータの列では常に $n(n-1)/2$ 回の比較が必要となり、n^2 に比例していることがわかります。このことは、データの個数が 100 倍になると計算量が 10000 倍になることを意味するので、仮に前者のソーティングが 1 秒で終わったとすると後者は 2.8 時間もかかり、効率があまり良くないと言えます。

問題 1　例のデータ列を最小選択法によってソーティングするプログラムを作成しなさい。

8.3　バブルソート

バブルソート[1](bubble sort) は、隣り合う 2 つの項を比較ながらデータ列を左から右へ走査します。このとき、比較した 2 つのデータが並べ替えの基準を満たさなければ値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されます。手順としては、

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。
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。
⋮
Step n-1　a_1 と a_2 を比較し、$a_1 > a_2$ ならば交換する $(i = 1)$。a_2 が決定する。

となります。従って、データの列 $\{4, 10, 5, 2, 1, 7, 8, 6, 3, 9\}$ をバブルソートでソーティングすると表 8.2 となります。

[1]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。

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

表 8.2: バブルソートによるソーティングの例

表 8.2 より、比較の回数は $(9+1)9/2 = 9+8+7+6+5+4+3+2+1$ となり計算量は最小値選択法と同じになります。ところが、表 8.2 を注意深く観察するとソーティングは Step 6 で終了しています。従って、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が終了したことになり、以降のステップを省略することができます。このことより、ほとんど整列済みのデータ列では、計算量は n に比例することがわかります (完全に整列済みであれば n-1 回の比較でよい)。このように、バブルソートはデータに依存したアルゴリズムといえます。

問題 1 例のデータ列をバブルソートによってソーティングするプログラムを作成しなさい。

8.4　挿入法

挿入法 (insertion sort) は、データ列のある位置までを整列済みであると仮定して、未整列の要素を整列済みのデータ列の適切な位置に挿入する方法です。この方法は、私達がトランプゲームを行うとき手札を並べ替えるときなどに利用しています。手順としては、

Step 1　a_1 が整列済みと仮定し、a_2 を挿入する ($\{a'_1, a'_2\}$)。

Step 2　$\{a_1, a_2\}$ が整列済みと仮定し、a_3 を挿入する ($\{a'_1, a'_2, a'_3\}$)。

$$\vdots$$

Step n-1　$\{a_1, a_2, \cdots, a_{n-1}\}$ が整列済みと仮定し、a_n を挿入する ($\{a'_1, a'_2, \cdots, a'_n\}$)。

となります。従って、データの列 $\{4, 10, 5, 2, 1, 7, 8, 6, 3, 9\}$ を挿入法でソーティングすると表 8.3 となります。

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

表 8.3: 挿入法によるソーティングの例

挿入法では、比較回数は整列済みのデータの挿入する位置によって比較回数が変化します。従って、最良の場合は $(n-1) \times 1$ 回で済みますが、最悪の場合は $(n-1)n/2$ 回となります。平均の場合は、i ステップで $(i-1)/2$ 回の等確率で挿入位置が見つかるものとすると、計算量は $(n-1)n/4$ となります。

問題 1　例のデータ列を挿入法によってソーティングするプログラムを作成しなさい。

問題 2　挿入法で比較回数が最大となるデータ列を考えなさい。

問題 3　挿入法で比較回数が最小となるデータ列を考えなさい。

8.5 クイックソート

クイックソート (quick sort) は、内部ソートの中では最も速いアルゴリズムで、データの列から基準値を選び、その値より小さなデータを左に大きな値を右に分割し、分割されたデータが 1 個になるまで再帰的に繰り返し行う方法です。手順としては、

Step 1 $\{a_1, a_2, \cdots, a_n\}$ から基準値 a_1 を選ぶ。基準値より小さな値は配列の左へ移し大きな値は右へ移す ($\{a'_1, a'_2, \cdots, a'_i\}, \{a'_{i+1}, \cdots, a'_n\}$)。

Step 2 $\{a'_1, a'_2, \cdots, a'_i\}$ から基準値 a'_1 を選び、Step 1 の操作を繰り返す。
$\{a'_{i+1}, \cdots, a'_n\}$ についても同様の操作を行う。

⋮

Step n 分割されたデータが全て 1 個になれば終了。

となります。従って、データの列 $\{4, 10, 5, 2, 1, 7, 8, 6, 3, 9\}$ をクイックソートでソーティングすると表 8.4 となります。

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

表 8.4: クイックソートによるソーティングの例

なお、クイックソートの平均計算量は $n \log n$ に比例しています (詳しく述べないが、各自で調べておくこと)。実際、クイックソートのプログラムは次のようになります。

● クイックソート 8-1.c

```
1: #include <stdio.h>
2:
3: int quicksort(int a[],int start,int end);
4:
5: main()
```

```
 6: {
 7:     int a[10]={4,10,5,2,1,7,8,6,3,9};
 8:
 9:     quicksort(a,0,9);
10: }
11:
12: int quicksort(int a[],int start,int end)
13: {
14:     int i,j,k,x,tmp;
15:
16:     i=start;
17:     j=end;
18:     x=a[start];
19:     while(1){
20:         while(a[i]<x) i++;
21:         while(a[j]>x) j--;
22:         if(i>=j) break;
23:         tmp=a[i]; a[i]=a[j]; a[j]=tmp;
24:         i++; j--;
25:     }
26:     if(start<i-1) quicksort(a,start,i-1);
27:     if(j+1<end) quicksort(a,j+1,end);
28: }
```

問題 1　プログラム 8-1.c を検証しなさい。

8.6　ヒープソート

ヒープソート (heap sort) は、図 8.1 のような **2 分木**(binary tree) というデータ構造を用い、ヒープ化とダウンヒープという 2 つの作業によってソーティングされます。計算量は $n \log n$ で、クイックソートのようにデータの良し悪しに左右されませんが、平均的な計算速度はクイックソートに劣ります。

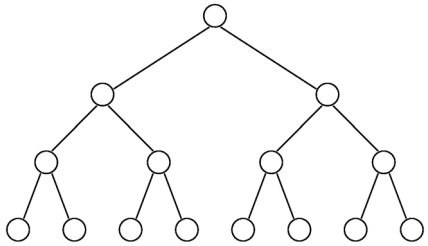

図 8.1: 2 分木

最初に行うヒープ化という作業は、親の持つデータはその子のデータより大きいというヒープ条件を満たすように 2 分木を構成します。例えば、データの列 $\{4, 10, 5, 2, 1, 7, 8, 6, 3, 9\}$ のヒープ化された 2 分木は図 8.2 のようになります。また、この木の根 (root) はデータの列の最大値ですから $a[n]$ が決定します。

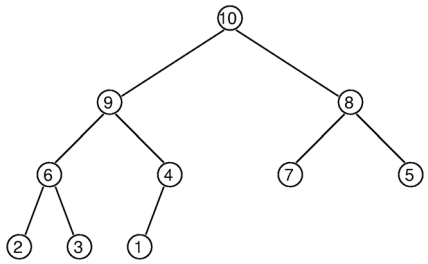

図 8.2: ヒープ化

次に、図 8.3 のように根の最大値を取り除き、代わりに一番深いところにある子の値を代入しヒープ化すると新しい木ができます。この作業をダウンヒープと呼びます。この新しい木の根は $a[n-1]$ を決定し、さらに、この作業を繰り返せばソーティングが完了することがわかります。

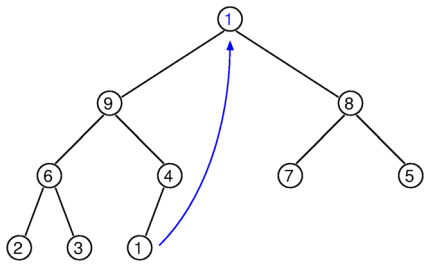

図 8.3: ダウンヒープ

ヒープソートのプログラムは次のようになります。

● ヒープソート 8-2.c

```
1: #include <stdio.h>
2:
```

```
 3: main()
 4: {
 5:     int a[11]={0,4,10,5,2,1,7,8,6,3,9},n=10;
 6:     int i,j,k,x;
 7:
 8:     for(k=n/2;k>=1;k--){
 9:         i=k; x=a[i];
10:         while((j=2*i)<=n){
11:             if(j<n && a[j]<a[j+1]) j++;
12:             if(x>=a[j]) break;
13:             a[i]=a[j]; i=j;
14:         }
15:         a[i]=x;
16:     }
17:     while(n>1){
18:         x=a[n]; a[n]=a[1]; n--;
20:         i=1;
21:         while((j=2*i)<=n){
22:             if(j<n && a[j]<a[j+1]) j++;
23:             if(x>=a[j]) break;
24:             a[i]=a[j]; i=j;
25:         }
26:         a[i]=x;
27:     }
28: }
```

問題 1　プログラム 8-2.c を検証しなさい。

問題 2　プログラム 8-2.c から、2 分木と配列の関係を調べなさい。