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。

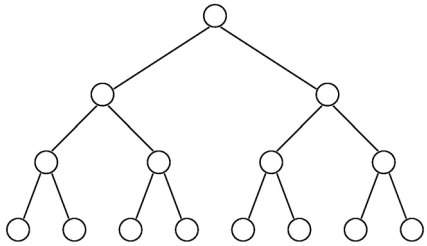

図 8.1: 2 分木

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。

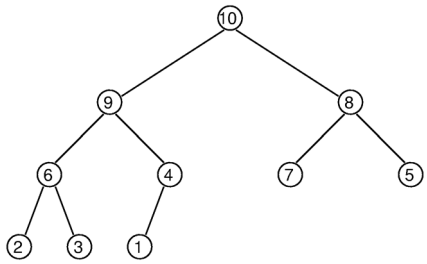

図 8.2: ヒープ化

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。

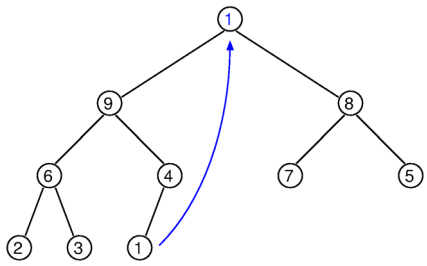

図 8.3: ダウンヒープ

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。