再帰関数Ⅲ

０．目次

１１．ソート

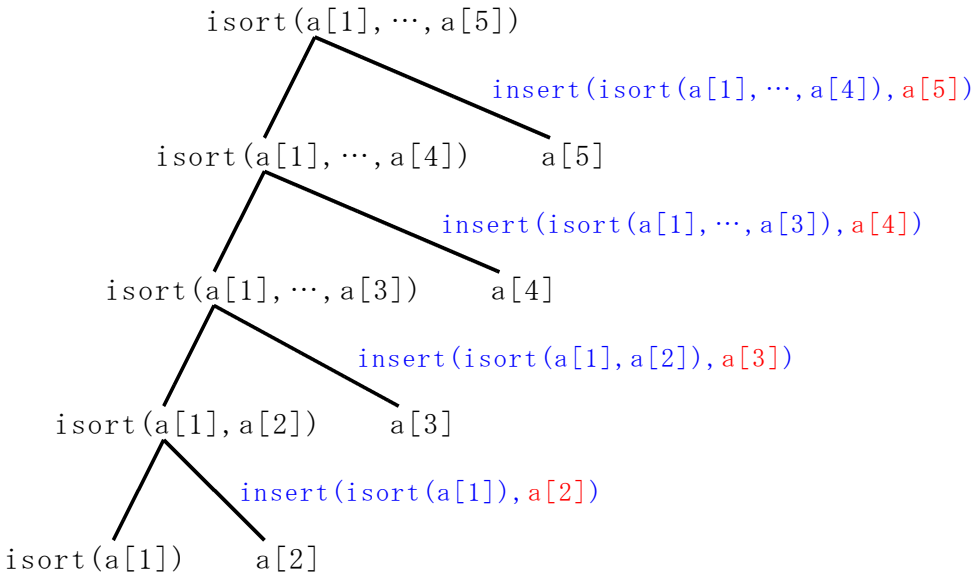
１１．１　挿入ソート

挿入ソートを再帰関数isortを用いて書く。整列しているデータ(a[1]からa[n-1]まで)にa[n]を挿入する操作を繰り返す。

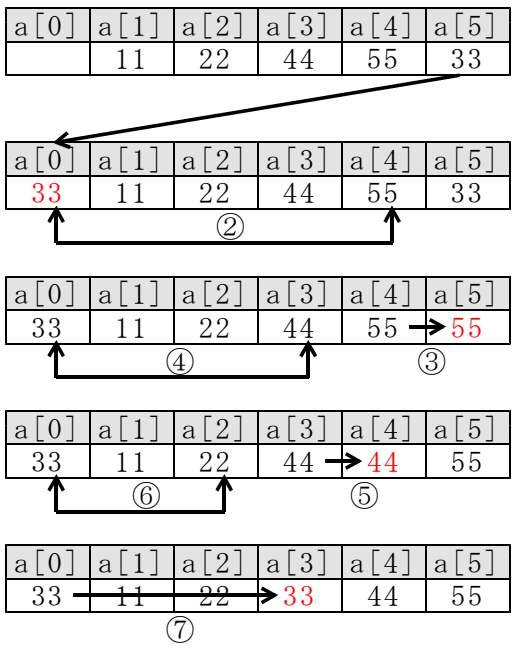
●再帰的定義

```
isort(a[1],…,a[n]) = insert(isort(a[1],…,a[n-1]),a[n])    (n>1)
                   = a[1]                                  (n=1)
```

n=5の場合



●動作例　insert(isort(a[1],…,a[4]),a[5])



① a[5]をa[0]に代入。
② a[0]とa[4]を比較。
③ a[4]をa[5]に代入。
④ a[0]とa[3]を比較。
⑤ a[3]をa[4]に代入。
⑥ a[0]とa[2]を比較。
⑦ a[0]をa[3]に代入。

再帰的定義に基づくプログラムは次のように書ける。

●プログラム(rf1111.c)

```
1  /* << rf1111.c >> */
2  #include <stdio.h>
3  #define N 9999 /* データの最大個数。*/
4  int a[N+1]; /* データを格納する配列。*/
5  void isort(int k);
6
7  int main() {
8    int i,
9        n; /* データの個数。*/
10
11   while( 1 ) {
12     /* データの個数の読み込み。*/
13     scanf("%d",&n);
14     if( (n <= 0)||(n > N) ) { break; }
15     for( i=1; i<=n; i++ ) { scanf("%d",&a[i]); }
16
17     /* 整列前データの出力。*/
18     printf("整列前 : ");
19     for( i=1; i<=n; i++ ) { printf("%d ",a[i]); }
20     printf("¥n");
21
22     /* 並べ替え。*/
23     isort(n);
24
25     /* 整列後データの出力。*/
26     printf("整列後 : ");
27     for( i=1; i<=n; i++ ) { printf("%d ",a[i]); }
28     printf("¥n");
29   }
30 }
31 /* 整列しているデータ(a[1]からa[k-1]まで)にa[k]を挿入する関数。*/
32 void isort(int k) {
33   int i;
34   if( k > 1 ) {
35     isort(k-1);
36     /* 挿入操作。*/
37     a[0] = a[k];
38     i = k-1;
39     while( a[i] > a[0] ) {
40       a[i+1] = a[i];
41       i--;
42     }
43     a[i+1] = a[0];
44   }
45 }
```

実行結果

```
% cc rf1111.c
% ./a.out
6
55 22 33 11 66 44
整列前：55 22 33 11 66 44
整列後：11 22 33 44 55 66
0
```

●プログラム(rf1111.c)において、nが与えられたとき、配列a[i](1≦i≦n)がつぎのように定義されている。

```
a[i] = (11*i)%64
```

n=10の場合、11,22,33,44,55,2,13,24,35,46。
関数isortが呼び出される回数

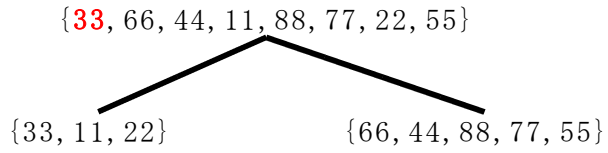
n	呼び出される回数
10	10
20	20
30	30
40	40

１１．２　クイックソート

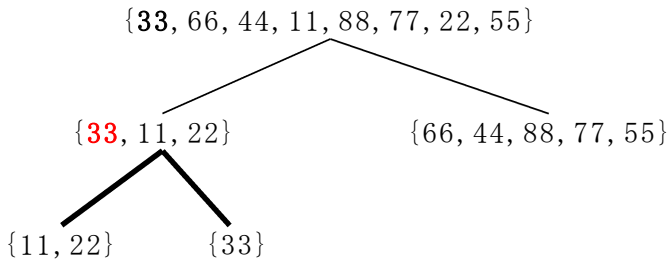
n個のデータの並びa[1], a[2], …, a[n]を適当な値を基準として、２つに分割する。前者の並びのどの要素も、後者の並びのどの要素よりも小さいか等しくし、後者の並びのどの要素も前者の並びのどの要素よりも大きいか等しくする。
あとは、同様の分割をそれぞれの並びに対して、並びに含まれる要素の個数が1個以下になるまで行う。
このような考え方でデータを昇順に並べ換える方法をクイックソートという。

並び{33, 66, 44, 11, 88, 77, 22, 55}で具体的に示す。ここで、並びの先頭の要素を適当な値とする。

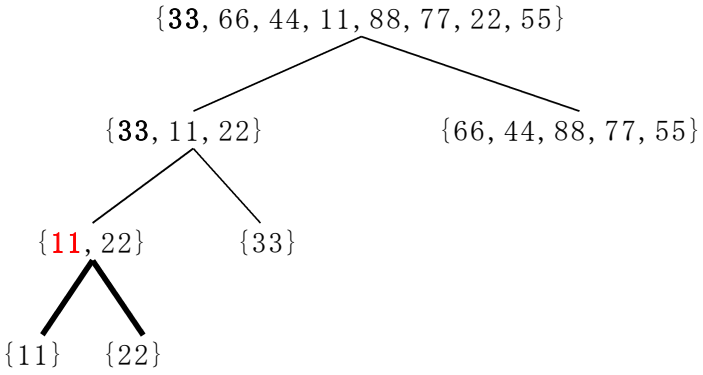
① 集合{33, 66, 44, 11, 88, 77, 22, 55}から適当な値33を選び、
33以下の集合{33, 11, 22}と33以上の集合{6, 44, 88, 77, 55}に分割する。



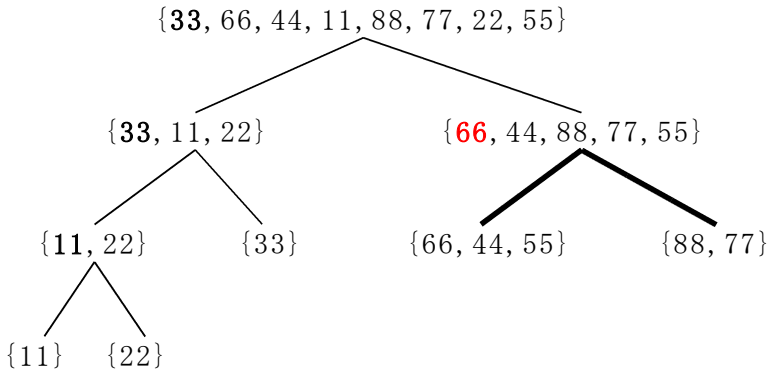
② 集合{33, 11, 22}から適当な値33を選び、
33以下の集合{11, 22}と33以上の集合{33}に分割する。



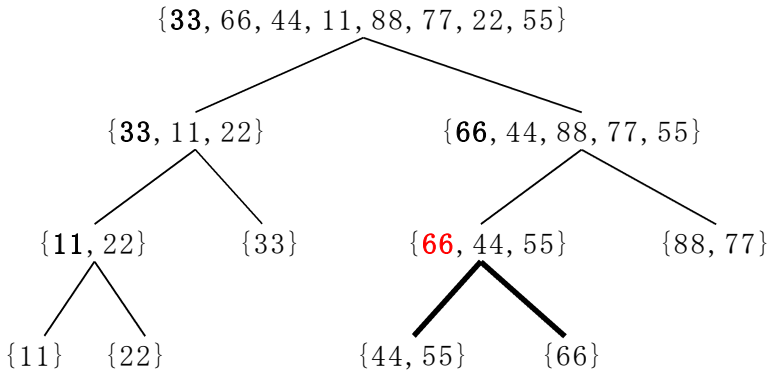
③ 集合{11, 22}から適当な値11を選び、
11以下の集合{11}と11以上の集合{22}に分割する。



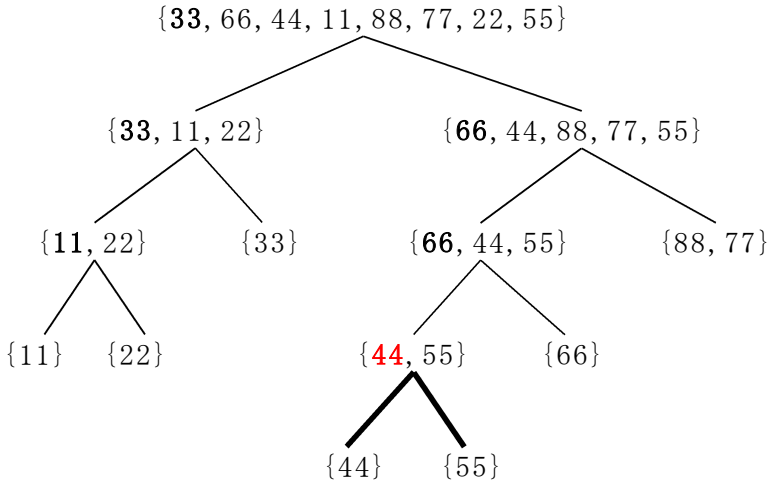
④ 集合{66, 44, 88, 77, 55}から適当な値66を選び、
66以下の集合{66, 44, 55}と66以上の集合{88, 77}に分割する。



⑤ 集合{66, 44, 55}から適当な値66を選び、
66以下の集合{44, 55}と66以上の集合{66}に分割する。

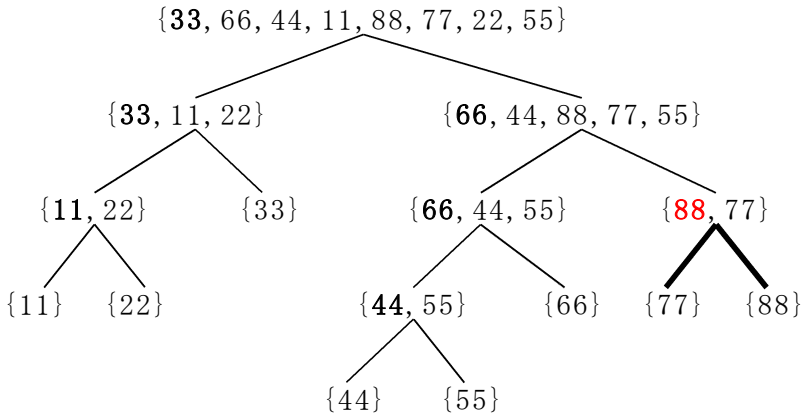


⑥ 集合{44, 55}から適当な値44を選び、
44以下の集合{44}と44以上の集合{55}に分割する。

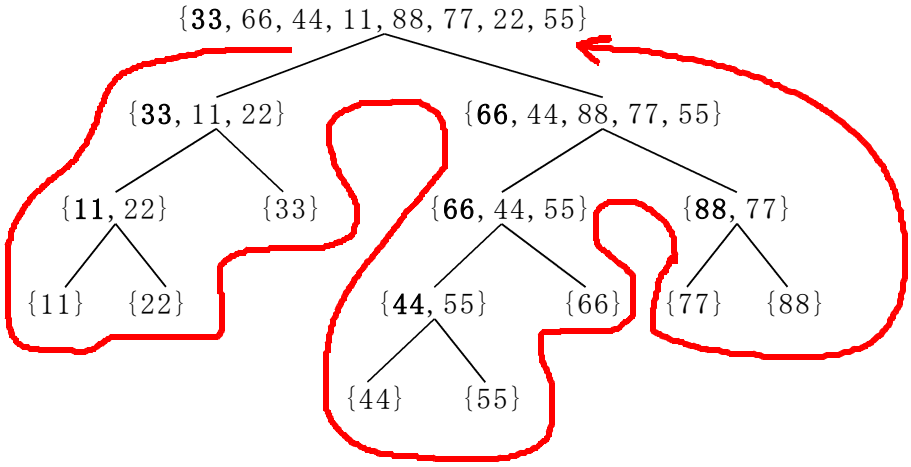


⑦ 集合{88, 77}から適当な値88を選び、

88以下の集合{77}と88以上の集合{88}に分割する。



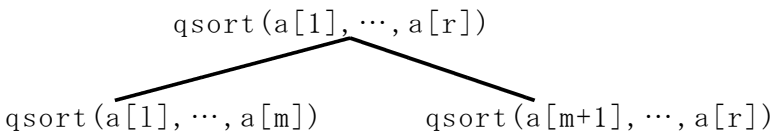
⑧ このように分割された結果を枝分かれに沿って、反時計回りにたどっていき、1個の要素からなる集合を出力する。



●プログラム(rf1120.c)

```
1  /* << rf1120.c >> */
2  #include <stdio.h>
3  #define N 9999 /* データの最大個数。*/
4  int a[N+1]; /* データを格納する配列。*/
5  void qsort(int l, int r);
6
7  int main() {
8    int i,
9        n; /* データの個数。*/
10
11   while( 1 ) {
12     /* データの個数の読み込み。*/
13     scanf("%d",&n);
14     if( (n <= 0)||(n > N) ) { break; }
15     for( i=1; i<=n; i++ ) { scanf("%d",&a[i]); }
16
```

```
17      /* 整列前データ出力。*/
18      printf("整列前 : ");
19      for( i=1; i<=n; i++ ) { printf("%d ",a[i]); } printf("¥n");
20
21      /* 並べ替え。*/
22      qsort(1,n);
23
24      /* 整列後データ出力。*/
25      printf("整列後 : ");
26      for( i=1; i<=n; i++ ) { printf("%d ",a[i]); } printf("¥n");
27    }
28  }
29
30  /* クイックソート。*/
31  void qsort(int l, int r) {
32    int m,w;
33    if( l < r ) {
34      /* wを適当な値にして、配列a[l]～a[r]を
35         a[l]～a[m]とa[m+1]～a[r]に分割。*/
36      qsort(l,m); qsort(m+1,r);
37    }
38    return;
39  }
```



●解法１：作業用配列を使う

配列a[l]～a[r]を、この中のある値wを基準に、**w未満とw以上**に分割する。
このとき、作業用配列b[]を用いる。

●動作例　l=3, r=7　　a[l]からa[r]までを　w=a[l]=55　で分割

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
22	11	55	66	44	77	33	99	88

w=a[3]=55とする

↓

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
22	11	55	66	44	77	33	99	88

wとa[4]を比較する。

b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]	b[8]	b[9]
						66		

a[4]をb[7]に代入する。

↓

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
22	11	55	66	44	77	33	99	88

wとa[5]を比較する。

b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]	b[8]	b[9]
		44				66		

a[5]をb[3]に代入する。

↓

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
22	11	55	66	44	77	33	99	88

wとa[6]を比較する。

↓

b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]	b[8]	b[9]
		44			77	66		

a[6]をb[6]に代入する。

↓

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
22	11	55	66	44	77	33	99	88

wとa[7]を比較する。

b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]	b[8]	b[9]
		44	33		77	66		

a[7]をb[4]に代入する。

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
22	11	55	66	44	77	33	99	88

wをb[5]に代入する。

b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]	b[8]	b[9]
		44	33	55	77	66		

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
22	11	44	33	55	77	66	99	88

b[3]～b[7]をa[3]から
a[7]に代入する。

b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]	b[8]	b[9]
		44	33	55	77	66		

解法１に基づくプログラムは次のように書ける。

●プログラム(rf1121.c)

```
/* << rf1121.c >> */
#include <stdio.h>
#define N 9999 /* データの最大個数。*/
int a[N+1], /* データを格納する配列。*/
    b[N+1]; /* データを格納する配列。*/
void qsort(int l, int r);

int main() {
  int i,
      n; /* データの個数。*/

  while( 1 ) {
    /* データの個数とデータの読み込み。*/
    scanf("%d",&n);
    if( (n <= 0)||(n > N) ) { break; }
    for( i=1; i<=n; i++ ) { scanf("%d",&a[i]); }

    /* 整列前データの入力。*/
    printf("整列前 : ");
    for( i=1; i<=n; i++ ) { printf("%d ",a[i]); }
    printf("¥n");

    /* 並べ替え*/
    qsort(1,n);

    /* 整列後データの出力。*/
    printf("整列後 : ");
    for( i=1; i<=n; i++ ) { printf("%d ",a[i]); }
    printf("¥n");
  }
}

/* クイックソート。*/
void qsort(int l, int r) {
  int i,j,k;
  if( l < r ) {
    i = l; j = r;
    /* a[l]を適当な値にして分割。*/
    for( k=l+1; k<=r; k++ ) {
      if( a[k] < a[l] ) {
        b[i] = a[k];
        i++;
      } else {
        b[j] = a[k];
        j--;
```

```
46        }
47      }
48      b[i] = a[l];
49      /* b[l]～b[i-1]がw未満、b[i]=a[l]、b[j+1]～b[r]がw以上。*/
50      for( k=l; k<=r; k++ ) {
51        a[k] = b[k];
52      }
53      qsort(l,i-1);
54      qsort(j+1,r);
55    }
56    return;
57  }
```

実行結果

```
% cc rf1121.c
% ./a.out
5
11 11 11 11 11
整列前：11 11 11 11 11
整列後：11 11 11 11 11
5
11 22 33 44 55
整列前：11 22 33 44 55
整列後：11 22 33 44 55
6
55 22 33 11 66 44
整列前：55 22 33 11 66 44
整列後：11 22 33 44 55 66
0
```

●解法２：作業用配列を使わない

配列a[l]～a[r]を、この中のある値wを基準に、**w以下とw以上**に分割する。このとき、配列a内でつぎの交換を繰り返し、配列の左にw以下の要素を集め、配列の右にw以上の要素を集める。

交換：配列aに左方からw以上の要素xを見つけ、右方からw以下の要素yを見つける。要素xとyを交換する。

●動作例　l=3, r=7

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
22	11	55	66	44	77	33	99	88
		i				j		

w=a[3]=55とする
iはw以上の要素の位置
jはw以下の要素の位置

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
22	11	55	66	44	77	33	99	88
		i				j		

a[3]とa[7]を交換する。

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
22	11	33	66	44	77	55	99	88
			i		j			

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
22	11	33	66	44	77	55	99	88
			i	j				

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
22	11	33	66	44	77	55	99	88
			i	j				

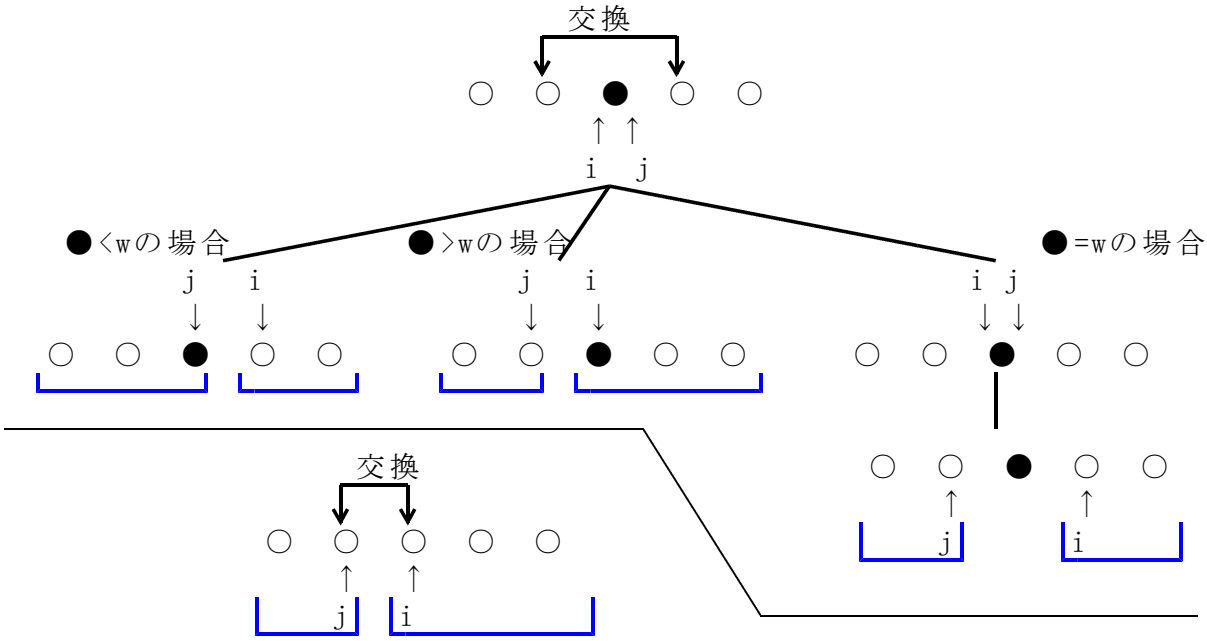
a[4]とa[5]を交換する。

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
22	11	33	44	66	77	55	99	88
			j	i				

配列a[3]～a[4]の分割、配列a[5]～a[7]の分割へ進む。

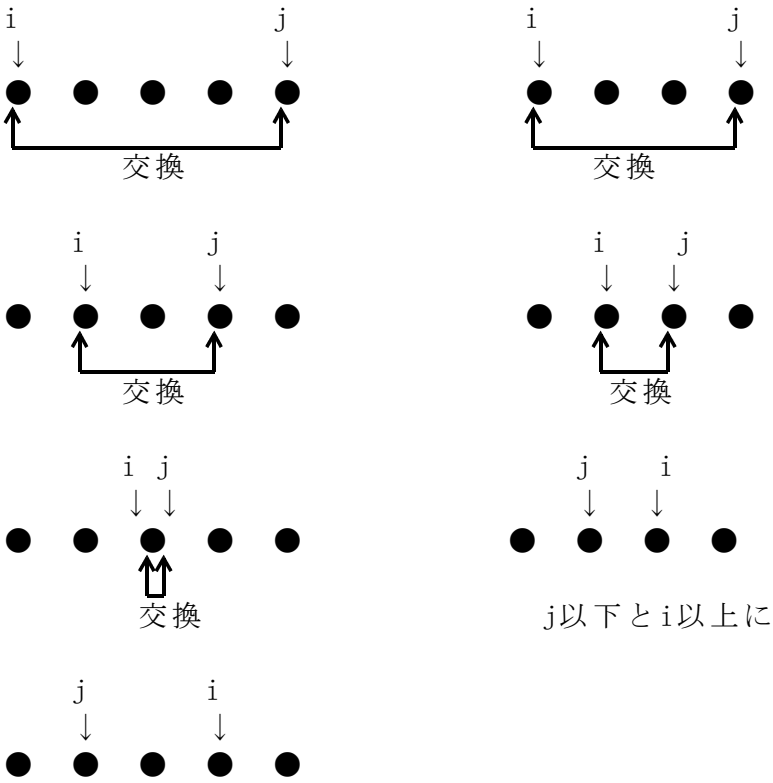
●交換操作

・一般に起こる場合



いずれも、j以下とi以上に分割される。

・分割する範囲がすべて同じ値の場合



j以下とi以上に分割される。

j以下とi以上に分割される。

解法２に基づくプログラムは次のように書ける。

●プログラム(rf1122.c)

```
/* << rf1122.c >> */
#include <stdio.h>
#define N 9999 /* データの最大個数。*/
int a[N+1]; /* データを格納する配列。*/
void qsort(int l, int r);

int main() {
  int i,
      n; /* データの個数。*/

  while( 1 ) {
    /* データの個数とデータの読み込み。*/
    scanf("%d",&n);
    if( (n <= 0)||(n > N) ) { break; }
    for( i=1; i<=n; i++ ) { scanf("%d",&a[i]); }

    /* 整列前データの出力。*/
    printf("整列前 : ");
    for( i=1; i<=n; i++ ) { printf("%d ",a[i]); }
    printf("¥n");

    /* 並べ替え。*/
    qsort(1,n);

    /* 整列後データ出力。*/
    printf("整列後 : ");
    for( i=1; i<=n; i++ ) { printf("%d ",a[i]); }
    printf("¥n");
  }
}

/* クイックソート。*/
void qsort(int l, int r) {
  int i,j,w,z;
  if( l < r ) {
    w = a[l]; /* wを適当な値a[l]にして分割。*/
    i = l; j = r;
    do {
      while( a[i] < w ) { i++; }
      while( a[j] > w ) { j--; }
      if( i <= j ) {
        z = a[i]; a[i] = a[j] ; a[j] = z;
        i++; j--;
      }
    } while( i <= j );
    qsort(l,j);
    qsort(i,r);
  }
  return;
}
```

実行結果

```
% cc rf1222.c
% ./a.out
5
11 11 11 11 11
整列前：11 11 11 11 11
整列後：11 11 11 11 11
5
11 22 33 44 55
整列前：11 22 33 44 55
整列後：11 22 33 44 55
6
55 22 33 11 66 44
整列前：55 22 33 11 66 44
整列後：11 22 33 44 55 66
0
```

●プログラム(rf1122.c)において、nが与えられたとき、配列a[i](1≦i≦n)がつぎのように定義されている。

```
a[i] = (11*i)%64
```

n=10の場合、11,22,33,44,55,2,13,24,35,46。
関数qsort(1,n)が呼び出される回数

n	呼び出される回数
10	19
20	39
30	59
40	79

（考察）プログラム(rf1122.c)の３９行目を以下のように変更して実行すると、エラーになる。

```
while( a[j] >= w ) { j--; }
```

実行結果

```
% cc rf1122.c
% ./a.out
5
11 11 11 11 11
整列前：11 11 11 11 11
Segmentation Fault (core dumped)
```

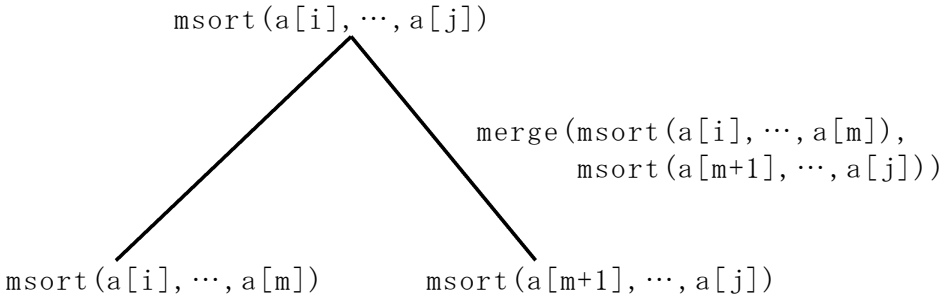
１１．３　マージソート

２つのソートされたリスト({a[i],…,a[k]} と {a[k+1],…,a[j]})を１つのソートされたリストにまとめる関数をmerge(a[i],…,a[j]) とする。
関数mergeを繰り返し使うソート法は、マージソートと呼ばれ、リスト{a[i],…,a[j]}をソートする関数msortは、次のように定義される。

●再帰的定義

msort(a[i],…,a[j])

= merge(msort(a[i],…,a[m]),msort(a[m+1],…,a[j]))　(i<j, i≦m<j)
= a[i]　(i=j)



●動作例　merge(a[3],a[4],a[5],a[6],a[7]), i=3,j=7,m=(i+j)/2=5

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
22	11	55	66	44	77	33	99	88

a[3]からa[5]をソート。a[6]からa[7]をソート。

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
22	11	44	55	66	33	77	99	88

a[3]からa[5]とa[6]からa[7]をマージ。

b[1]	b[2]	b[3]	b[4]	b[5]	b[6]	b[7]	b[8]	b[9]
		33	44	55	66	77		

作業用配列bから配列aに代入。

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
22	11	33	44	55	66	77	99	88

再帰的定義に基づくプログラムは次のように書ける。

●プログラム(rf1131.c)

```
/* << rf1131.c >> */
#include <stdio.h>
#define N 9999 /* データの最大個数。*/
int a[N+1], /* データを格納する配列。*/
    b[N+1]; /* 作業用配列。*/
void msort(int i, int j);

int main() {
  int i,
      n; /* データの個数。*/

  /* データの個数nとデータの読み込み。*/
  scanf("%d",&n);
  for( i=1; i<=n; i++ ) { scanf("%d",&a[i]); }

  /* 整列前データの出力。*/
  printf("整列前 : ");
  for( i=1; i<=n; i++ ) { printf("%3d",a[i]); }
  printf("¥n");

  /* 並べ替え。*/
  msort(1,n);

  /* 整列後データ出力。*/
  printf("整列後 : ");
  for( i=1; i<=n; i++ ) { printf("%3d",a[i]); }
   printf("¥n");
}

/* マージソート。a[i]からa[j]を並べ替える。*/
void msort(int i, int j) {
  int f, /* 前半部の先頭位置。*/
      g, /* 後半部の先頭位置。*/
      k, /* 併合部の格納位置。*/
      m;
  if( i >= j ) { return; }
  /* ほぼ中央の位置mを求める。*/
  m = (i+j)/2;
  /* 前半部(a[i]～a[m])と後半部(a[m+1]～a[j])をマージソート。*/
  msort(i,m);
  msort(m+1,j);
  /* 前半部と後半部をマージ。*/
  f = i; g = m+1;
  k = f - 1;
  while( 1 ) {
    /* 前半部が空になったときの処理。*/
```

```
47      if( f > m ) {
48        while( g <= j ) {
49          k++; b[k] = a[g]; g++;
50        };
51        break;
52      }
53      /* 後半部が空になったときの処理。*/
54      if( g > j ) {
55        while( f <= m ) {
56          k++; b[k] = a[f]; f++;
57        };
58        break;
59      }
60      k++;
61      if( a[f] <= a[g] ) {
62        b[k] = a[f]; f++;
63      } else {
64        b[k] = a[g]; g++;
65      }
66    }
67    /* 配列bに保存された結果を配列aに戻す。*/
68    for( k=i; k<=j; k++ ) { a[k] = b[k]; }
69  }
```

実行結果

```
% cc rf1131.c
% ./a.out
2 11 22
整列前： 11 22
整列後： 11 22
% ./a.out
2 22 11
整列前： 22 11
整列後： 11 22
% ./a.out
3 11 22 33
整列前： 11 22 33
整列後： 11 22 33
% ./a.out
3 33 22 11
整列前： 33 22 11
整列後： 11 22 33
% ./a.out
9 99 11 88 22 77 33 66 44 55
整列前： 99 11 88 22 77 33 66 44 55
整列後： 11 22 33 44 55 66 77 88 99
```

●プログラム(rf1131.c)において、nが与えられたとき、配列a[i](1≦i≦n)がつぎのように定義されている。

```
a[i] = (11*i)%64
```

n=10の場合、11,22,33,44,55,2,13,24,35,46。
関数msortが呼び出される回数

n	呼び出される回数
10	19
20	39
30	59
40	79