ソート(整列)

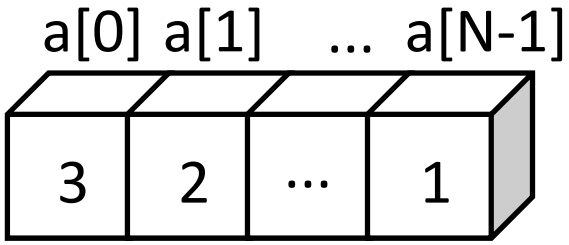
ソート(整列)とは

- データの列を受け取って、それを値の大きさの順に並べ替えること
  - 例 : 10 20 55 4 15 12
  - -> 4 10 12 15 20 55
- 整列の対象 : 数や文字列
- ソートを必要とする場面は非常に多い
  - 名簿、辞書
  - 成績の順位

問題の定義

- 整列すべきデータの個数をN
- 整列の対象のデータは、配列の中に入っていると仮定

```
#define N 100
int a[N];
```



- データ型に比較演算子(>や<)が適用できると仮定

- 小さな値を前に、大きな値を後に：昇順
- 大きな値を前に、小さな値を後に：降順

単純な整列アルゴリズム

```
#include <stdio.h>
#define N 10
int a[N];

void swap(int i, int j){
    int tmp;
    tmp = a[i]; a[i] = a[j]; a[j] = tmp;
}
```

a[i]とa[j]を交換

```
void simplesort(){
    int i,j;
    for(i=0;i<N-1;i++)
        for(j=i+1;j<N;j++)
            if(a[i]>a[j]) swap(i,j);
}
```

整列

以後のスライドではこの部分だけ書く

```
int main(void){
    int i;
    for(i=0;i<N;i++){
        scanf("%d",&a[i]);
    }
    simplesort();
    for(i=0;i<N;i++){
        printf("%d ",a[i]);
    }
    printf("¥n");
}
```

データの入力

整列結果の表示

simplesortの動作

- 内側のfor文では、jをi+1からnまで動かして、a[j]とa[i]を比較し、a[j]の方が小さければ交換
- 終わった時には、a[i],a[i+1],...,a[n]の最小値がa[i]に残る

初期状態	10	20	55	4	15	12
1	4	20	55	10	15	12
2	4	10	55	20	15	12
3	4	10	12	55	20	15
4	4	10	12	15	55	20
5	4	10	12	15	20	55

simplesortの計算量

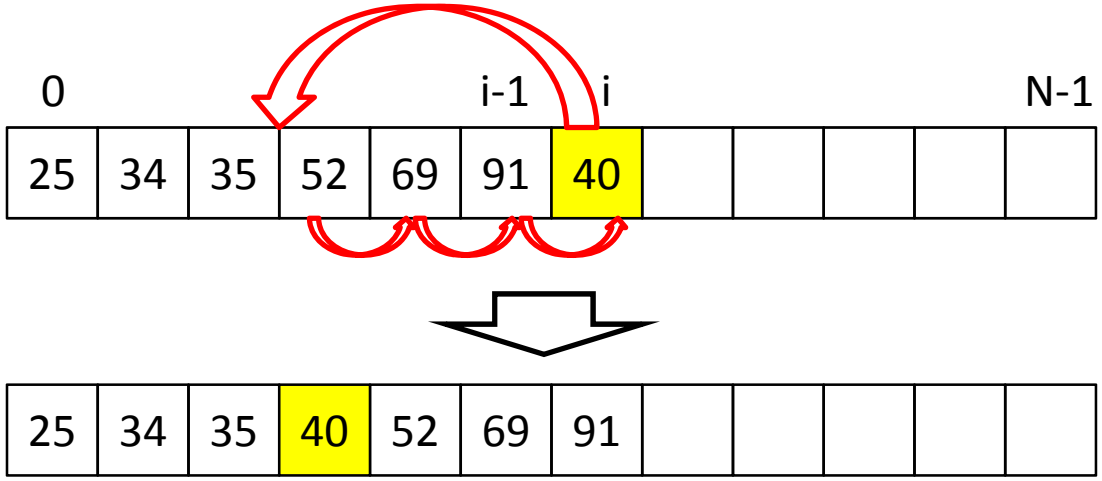
- データ同士の比較の回数に注目
- i=kの時に、内側のfor文ではN-k-1回の比較を行うから、全体の比較の回数は

$$(N-1)+(N-2)+\ldots+1=\frac{1}{2}N(N-1)$$

- このアルゴリズムの計算量は$O(N^2)$
  - bubble sortも同様
- データの交換の回数は
  - 入力データが予め整列されたものであれば0回
  - 逆順に整列されたものであれば$\frac{1}{2}N(N-1)$回

insertion sort

- a[0]からa[i-1]までが整列されているとき、この中の適切な位置にa[i]を挿入する操作を繰り返す。



insertion sortのプログラム

```
void insertionsort(){
    int i,j,w;
    for(i=1;i<N;i++){
        w = a[i];
        j = i-1;
        while((j >= 0) && (w < a[j])){
            a[j+1] = a[j];
            j = j-1;
        }
        a[j+1]=w;
    }
}
```

insertion sortの計算量

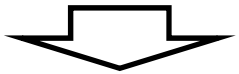
- 内側のループを回る回数は、a[i]がa[0],...,a[i-1]のどの位置に入るかによる
  - a[i]がa[i-1]より大なら0回
  - a[i]がa[0]より小ならi回

  平均してi/2回
- 全体では $\sum_{i=1}^{N-1} \frac{i}{2} = \frac{1}{4} N(N-1)$
- 最悪の場合は $\sum_{i=1}^{N-1} i = \frac{1}{2} N(N-1)$
- 計算量は $O(N^2)$
  - もとのデータがほとんど正しい順序ならほぼ $O(N)$

shell sort

- insertion sortの内側のループを変更

```
while((j >= 0) && (w < a[j])){
        a[j+1] = a[j];
        j = j-1;
}
```



```
while((j >= 0) && (w < a[j])){
        a[j+h] = a[j];
        j = j-h;
}
```

hは1以上の整数(h=1ならinsertion sort)

hおきに取ったデータが整列される
a[0] <= a[h] <= a[2h] <= ...
a[1] <= a[1+h] <= a[1+2h] <= ...
a[2] <= a[2+h] <= a[2+2h] <= ...

最初にhの値を大きくしておき、だんだん小さくしていく

互いに素なhの値を選ぶと効率的

shell sortのプログラム

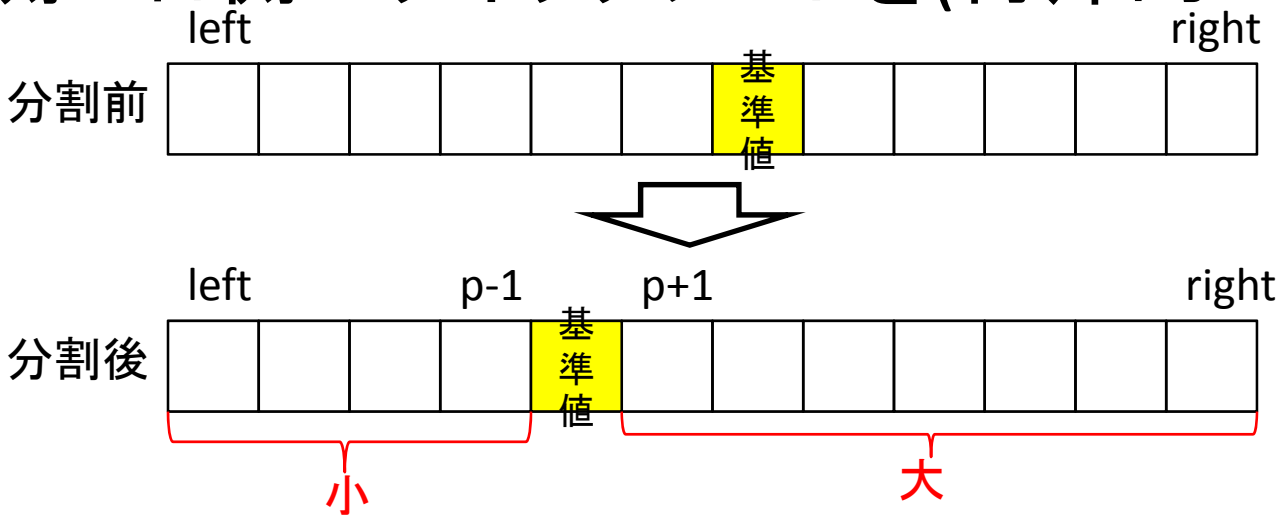
```
void shellsort(){
   int i,j,h,w;
   h=1;
   while(h < N){
      h = h * 3 + 1;
   }
   while(h > 1){
      h = h / 3;
      for(i=h+1;i<N;i++){
         w = a[i];
         j = i - h;
         while((j >= 0) && (w < a[j])){
            a[j+h] = a[j];
            j = j - h;
         }
         a[j + h] = w;
      }
   }
}
```

「互いに素」に近いhとして...,121,40,13,4,1,を取る($h_{k+1}=3h_k+1$の逆順)

計算量は$O(N^{1.25})$〜$O(N^{1.5})$程度

quick sort

- 整列する範囲の中から適当な値を一つ選ぶ(基準値)
- 範囲内の要素を一つずつ調べ、基準値より小さなデータを左側、大きなデータを右側に集める(分割)
- 左側と右側にクイックソートを(再帰的に)適用



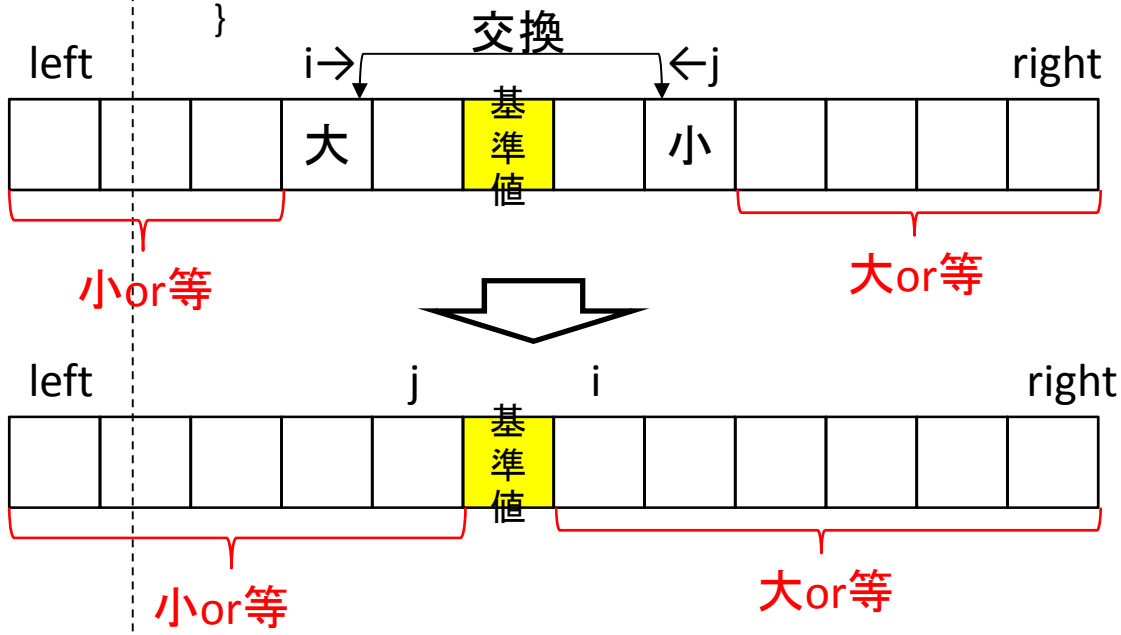
クイックソートについて

- 問題をいくつかの部分に分解してそれぞれを解き、その結果を組み合わせて全体の解を得る→
- 高速な整列アルゴリズム(計算量 $O(N \log N)$)
  - 基準値より小さなデータと大きなデータがいつも半々の場合が理想的
  - 基準値としていつも最大(または最小)の値を取るのが最悪(計算量$O(N^2)$)

quick sortのプログラム

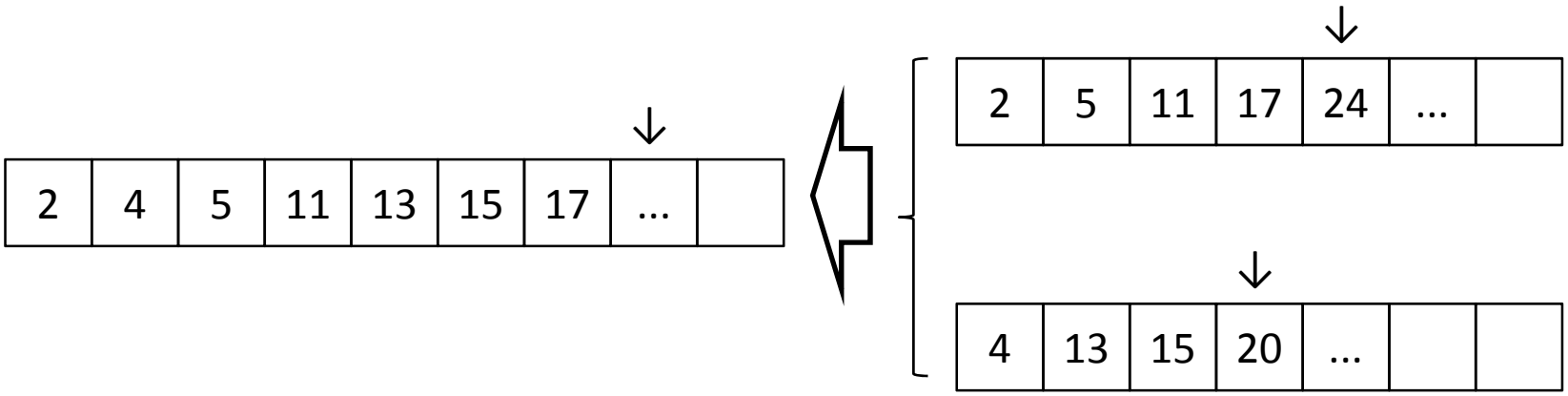
```
void qsort(int left, int right){
    int i,j,somewhere,pivot;
    if (left < right){
        somewhere = (left + right)/2;
        pivot = a[somewhere];
        i = left; j = right;
        do {
            while (a[i] < pivot) i = i + 1;
            while (a[j] > pivot) j = j - 1;
            if (i <= j){
                swap(i,j);
                i = i + 1; j = j - 1;
            }
        } while (i <= j);
        qsort(left,j);
        qsort(i,right);
    }
}
```

```
void quicksort(){
    qsort(0,N-1);
}
```



merge sort

- 整列された複数の列を併合

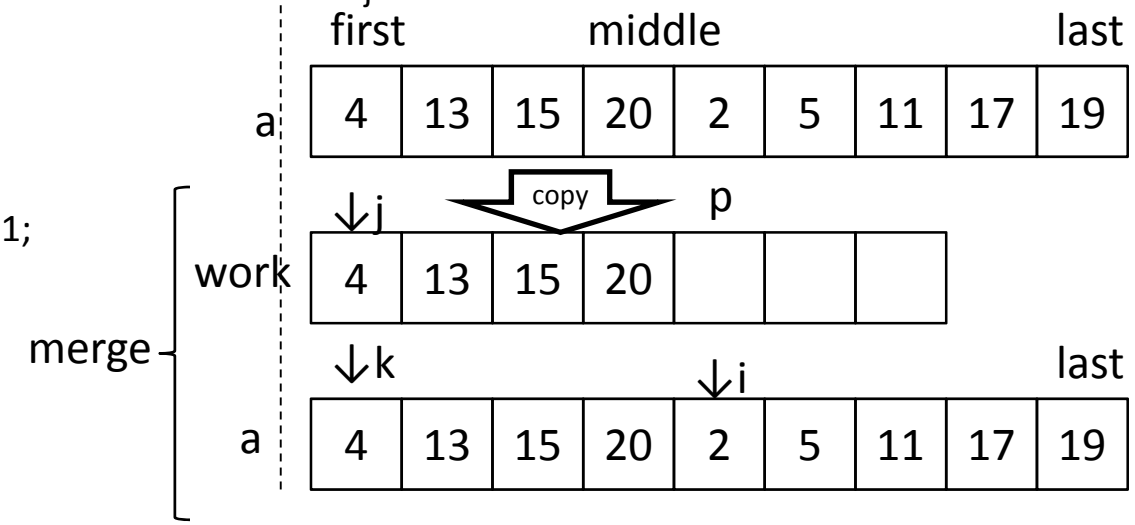


- 計算量は$O(N \log N)$

merge sortのプログラム

```
void msort(int first, int last){
   int i,j,k,p,middle;
   if (first < last){
      middle = (first + last) / 2;
      msort(first, middle);
      msort(middle+1, last);
      p = 0;
      for (i=first;i<=middle;i++){
         work[p]=a[i];
         p=p+1;
      }
      i=middle+1;j=0;k=first;
      while((i<=last) && (j<p)){
         if (work[j]<=a[i]){
            a[k]=work[j];k=k+1;j=j+1;
         } else {
            a[k]=a[i];k=k+1;i=i+1;
         }
      }
      while (j<p){
            a[k] = work[j];
            k=k+1;
            j=j+1;
         }
      }
   }
   void mergesort(){
      msort(0,N-1);
   }
```

・aの前半をworkにコピー
・aの後半とworkをmergeして
aに格納



整列アルゴリズムまとめ

- どんな整列アルゴリズムでも、$O(N \log N)$ の計算量が必要
- insertion sortは$O(N^2)$
- shell sortは$O(N^{1.25})$〜$O(N^{1.5})$ 程度
- quick sortは平均$O(N \log N)$、最悪$O(N^2)$
- merge sortは$O(N \log N)$