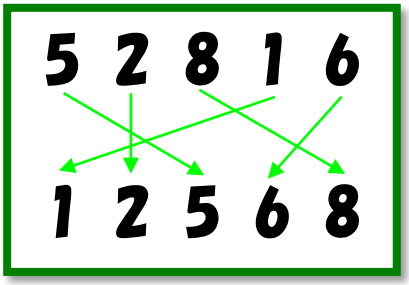
担当：長尾 智晴 (nagao@ynu.ac.jp)
濱上 知樹 (hamagami@ynu.ac.jp)

## 「アルゴリズムとデータ構造」講義日程

~~1. 基本的データ型~~
~~2. 基本的制御構造~~
~~3. 変数のスコープルール．関数~~
~~4. 配列を扱うアルゴリズムの基礎(1)．最大値，最小値~~
~~5. 配列を扱うアルゴリズムの基礎(2)．重複除去，集合演算，ポインタ~~
~~6. ファイルの扱い~~
7. 整列(1)．単純挿入整列・単純選択整列・単純交換整列 ← 本日の内容
8. 整列(2)．ヒープ整列・マージ整列・クイック整列
9. 再帰的アルゴリズムの基礎．再帰におけるスコープ．ハノイの塔など．
10. バックトラックアルゴリズム．8 王妃問題など．
11. 線形リストを扱うアルゴリズム(1 回)
12. 木構造を扱うアルゴリズム(1) 基礎
13. 木構造を扱うアルゴリズム(2) 挿入，削除，バランスなど．
14. ハッシング
15. その他のアルゴリズム

# 第 7 回「整列（ソーティング）(1)」

## ☆ 準備　ー　構造体（復習）




### ◎ 構造体って？

- 任意の型の要素を，複数，内部にもつ複合的な型．
- 複合的な情報…「項目」と「値」のペアの集合
  - 例 1　2 次元座標
    x 座標 = 1
    y 座標 = 2
  - 例 2　個人情報
    名前 = 横浜太郎
    大学 = 横浜国大
    本籍 = 神奈川県
- C 言語では struct という構造体宣言のためのキーワードを用いる．
  例)

名前は自由につけてよい

```
struct   point {  /* point という名前の構造体を宣言．構造体タグという */
     int x;       /* 構造体の項目すなわちメンバその 1．x はそのメンバ名 */
     int y;       /* 構造体のメンバその 2．y はそのメンバ名 */
};
```

※ 構造体宣言は形だけを定義する（point という名前の構造体は x という名前の"記入欄"と y という名前の"記入欄"をもつという特徴だけを述べている）．そのため，構造体テンプレートと呼ばれることもある．実際のデータはその型を持つ変数などとして別途，宣言する．

- **構造体変数**

  ある構造体テンプレートの形をもつ実際のデータを保持するための変数.

  例）struct point pt;

  あらかじめ定義した構造体テンプレート point の形を持つデータを保持できる pt という名前の変数を宣言．名前は他の変数と同様に自由につければよい.

※ 「構造体テンプレート」も「構造体変数」も単に「構造体」と呼ばれることが多いのでどちらを指しているかに注意.

- **初期化**

  x の値を 1 に，y の値を 2 に初期化した pt を用意する場合.

  例）struct point pt = {1,2};

- **構造体の各メンバへのアクセス** … **メンバ演算子**"."を使う.

  メンバ自身も変数のように扱える → 値参照，代入

```
例）i = pt.x;       /* 変数 i に，構造体変数 pt のメンバ x の値を代入 */
    pt.x = i;       /* 構造体変数 pt のメンバ x に，i の値を代入 */
```

## ☆ 整列（ソーティング sorting）って？

- 与えられたものの集まりをある特定の「順番」に並べ直すこと.
- 形式的には，項目の集合

  a(1)，a(2)，a(3)，…, a(n)

  が与えられた時，ある順序

  a(k1), a(k2), a(k3), …, a(kn)

  に並べ変える．ただし，**順序づけ関数** f に関して次の式を満足する.

  f(a(k1)) ≦ f(a(k2)) ≦ f(a(k3))≦ … ≦ f(a(kn))

- 通常，順序づけ関数の値は特定の計算規則によって値を求めるのではなく，項目内の特定の"成分"（C 言語の構造体に対応させるならメンバ）の値を直接用いる．この成分（の値）を**「キー（key）」**という.

- 以降，以下の構造体を仮定.

```
struct item {
    int key;                /* メンバ key がキー．型は整数 */
    他のメンバの定義
};
```

※ キーが整数（int）型であるので，**関係演算子**がそのまま利用できる.

## ☆ 整列の種類

### 内部ソーティングと外部ソーティング

#### ◎ 内部ソーティング

- 配列上にデータがあり，それを整列.
- 内部記憶上にランダムアクセス可能なデータがある場合.
- 一般に小規模なデータ向き
- 例 … **単純挿入整列**，**単純選択整列**，**単純交換整列**，**クイックソート**

#### ◎ 外部ソーティング

- 順ファイル（sequential file）上にデータがあり，それを整列.
- 外部記憶（磁気ディスクなど）上にデータがある場合．順番に読むことができるが，任意の場所にアクセスすることは不可能/遅い.
- 一般に大規模なデータ向き
- 例 … **マージソート**

## ☆ 内部ソーティング

#### ◎ 注目すべき点

- 記憶場所（＝配列）を経済的に使用する．→ その場所（一つの配列）で入れ換える．
  （この意味において以下述べるアルゴリズムは同じ「記憶空間の効率（記憶空間計算量)」をもつといえる）
- 「計算の（時間的な）手間（時間計算量）」
  キーの比較回数 C と項目間の移動（置換）回数 M が尺度（いずれも項目数 n の関数になる）.

#### ◎ 類別

- 挿入によるソーティング
  未整理の項目の集合中の（任意の）一つの項目を，整列済みの列の正しい位置に**「挿入」**
- 選択によるソーティング
  未整理の項目の集合から序列によってきまる適切な項目を一つ**「選択」**し，整列済みの列の最後に付加
- 交換によるソーティング
  未整理の項目の集合を（未整列の）列にし，その中の二つの項目を序列により比較，**「交換」**する.

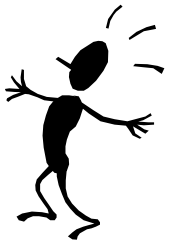
## ◎ 単純挿入整列

- 考え方

  a[1]～a[i-1]まで整列しているとき，a[i]について

  a[j] ＜a[i]＜a[j+1]

  なる j を探しだし，a[j]の次に a[i]を挿入すれば a[1]～a[i]までが整列したことになる．これを i=2～n まで繰り返す(n は最大の添字).



- 「挿入」

  a[i]を a[j]の次に挿入するには，

  1) a[i]の内容を別のところに保存.
  2) a[j+1],...,a[i-1]を a[j+2],...,a[i]にコピーする.
  3) a[j+1]に 1)で保存しておいた内容をコピー.

- 例

```
５ | ２ ３ ４ １        “|”はそこまで整列できていることを表す.
                       ２に注目．列「５」への挿入場所を見つける.
２ ５ | ３ ４ １        ２を列「５」に挿入.
                       ３に注目．列「２ ５」への挿入場所を見つける.
２ ３ ５ | ４ １        ３を列「２ ５」に挿入．以下同様.
２ ３ ４ ５ | １
１ ２ ３ ４ ５ |
```

- 効率

  キーの比較回数 C（平均） = $(n^2+3n-4)/4$ （ = $O(n^2)$ ）

  項目間の移動（置換）回数 M（平均） = $(n^2+11n-12)/4$ （ = $O(n^2)$ ）

```
/*************************************************************
    アルゴリズムとデータ構造
    サンプルプログラム insort.c
    <<単純挿入整列>>
    Copyright (c) 1995,96,97   T.Mori <mori@forest.dnj.ynu.ac.jp>
 *************************************************************/
#include <stdio.h>
#define CHARLEN 20
/* 1 データ分を表す構造体 */
struct item {
    int key;
    char name[CHARLEN];
};
void printitem(struct item a[ ], int n);
void insort(struct item a[ ], int n);

int main(void)
{
    #define N 9
    struct item table[N] = {
        {  0, ""},  /* 番人用 */
        { 65, "国語"},{ 90, "数学"},{ 85, "理科"},{ 70, "社会"},
        { 86, "英語"},{ 92, "体育"},{ 63, "音楽"},{ 85, "美術"}
    };
    int m = N;
```

```
26
27      printf("table[ ]: ");
28      printitem(table,m);
29      printf("insort(table,m)¥n");
30      insort(table,m);
31      printf("table[ ]: ");
32      printitem(table,m);
33      return 0;
34  }
35
36  /* 配列内のデータを印刷する */
37  void printitem( struct item a[ ], int n )
38  {
39      int i;
40
41      for( i=1; i<n; i++ )     /* a[0] は番人用なので印刷しない */
42          printf("<%d,%s> ", a[i].key, a[i].name);   /* %s は文字列を印字 */
43      printf("¥n");
44  }
45
46  /* 単純挿入法による整列 */
47  void insort( struct item a[ ], int n )
48  {
49      int i, j, m;
50      struct   item   x;
51
52      for(i=2; i<n; i++){
53          x = a[i];
54          a[0] = x; /* 番人 */
55          /* すでに整列されている a[1]～a[i-1]を調べて，x(= a[i]) を挿入すべ
56           * き位置を見つける．a[j] >= x なる j が見つかったとすれば，a[j+1]
57           * ～ a[i-1] までを一つずつ添字が大きくなる方向，つまり，a[j+2] ～
58           * a[i] に移動し，空いた a[j+1] に x を挿入すればよい．
59           * この移動は実際には j を見つける過程で同時に行なわれる．*/
60          j = i-1;
61          while (x.key < a[j].key) {
62              a[j+1] = a[j];
63              j--;
64          }
65          a[j+1] = x;
66      }
67  }
```

【実行結果】

```
table[ ]: <65,国語> <90,数学> <85,理科> <70,社会> <86,英語> <92,体育>
<63,音楽> <85,美術>
insort(table,m)
table[ ]: <63,音楽> <65,国語> <70,社会> <85,理科> <85,美術> <86,英語>
<90,数学> <92,体育>
```

## ◎ 単純選択整列

- 考え方

  a[0]～a[i-1]まで整列しているとき，a[i]～a[n]の中からキーが最小になる項目 a[j]を見つけ出し（=「選択」），a[i]の内容と a[j]の内容を入れ換えれば，a[1]～a[i]までが整列したことになる．これを i=0～n-1 まで繰り返す（n は最大の添字）．

- 例

| | | | | | | |
|---|---|---|---|---|---|---|
| \| | ５ | ２ | ３ | ４ | １ | 列「５　２　３　４　１」から最小値を求める． |
| １ | \| | ５ | ２ | ３ | ４ | １であるので，整列済み列の最後に加える．<br>列「５　２　３　４」から最小値を求める． |
| １ | ２ | \| | ５ | ３ | ４ | ２であるので，整列済み列「１」の最後に加える．<br>列「５　３　４」から最小値を求める． |
| １ | ２ | ３ | \| | ５ | ４ | ３であるので，整列済み列「１　２」の最後に加える． |
| １ | ２ | ３ | ４ | \| | ５ | 以下同様． |
| １ | ２ | ３ | ４ | ５ | \| | |

- 効率

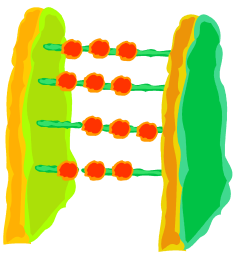
  キーの比較回数 C = $(n^2-n)/2$（ = $O(n^2)$ ）

  項目間の移動（置換）回数 M（近似平均）= n ×(ln n + g + 1)（= $O(n \times \ln n)$）

  g：オイラー定数（0.577...）

```
/*************************************************************
    アルゴリズムとデータ構造
    サンプルプログラム selsort.c
    <<単純選択整列>>
    copyright (c) 1995,96,97   T.Mori <mori@forest.dnj.ynu.ac.jp>
*************************************************************/
#include <stdio.h>
#define CHARLEN   20
/* 1 データ分を表す構造体 */
struct item {
    int key;
    char name[CHARLEN];
};
void printitem(struct item a[ ], int n);
void selsort(struct item a[ ], int n);

int main(void)
{
    #define N 8
    struct  item  table[N] = {
        { 65, "国語"},{ 90, "数学"},{ 85, "理科"},{ 70, "社会"},
        { 86, "英語"},{ 92, "体育"},{ 63, "音楽"},{ 85, "美術"}
    };
    int m = N;
```



```
25
26      printf("table[ ]: ");
27      printitem(table,m);
28      printf("selsort(table,m)¥n");
29      selsort(table,m);
30      printf("table[ ]: ");
31      printitem(table,m);
32      return 0;
33  }
34
35  /* 配列内のデータを印刷する */
36  void printitem( struct item a[ ], int n )
37  {
38      int i;
39
40      for( i=0; i<n; i++ )
41          printf("<%d,%s> ", a[i].key, a[i].name);   /* %s は文字列を印字 */
42      printf("¥n");
43  }
44
45  /* 単純選択法による整列 */
46  void selsort( struct item a[ ], int n )
47  {
48      int i, j, m;
49      struct item x;
50
51      for( i=0; i<n-1; i++ ){
52          /* すでに a[0]～a[i-1]が整列されているとする.
53           * この時，a[i]～a[n-1] の間のうちキーが最小のものを求める.
54           * その時の添字を m とする.
55          */
56          m = i;
57          for( j=i+1; j<n; j++ )
58              if ( a[j].key < a[m].key )
59                  m = j;
60          /* a[m] と a[i] の内容を入れ換える */
61          x = a[m]; a[m]=a[i]; a[i]=x;
62          /* この時点で，a[0]～a[i]が整列したことになる． */
63      }
64  }
65
66
67  【実行結果】
68  table[ ]: <65,国語> <90,数学> <85,理科> <70,社会> <86,英語> <92,体育>
69  <63,音楽> <85,美術>
70  selsort(table,m)
71  table[ ]: <63,音楽> <65,国語> <70,社会> <85,理科> <85,美術> <86,英語>
72  <90,数学> <92,体育>
```

## ◎ 単純交換整列

- 考え方（バブルソート）

  a[i]と a[i+1]を比較し，a[i]>a[i+1]ならそれらの要素を入れ換える操作を i=0～n-1 まで行なうと a[n-1]が最大になる．この操作を上限値を n-1 から順に 1 まで減らしながら行なうことで全体を整列させる．

- 例

| | |
|---|---|
| 5 2 3 4 1 \| | もとの数字列 |
| 2 3 4 1 5 \| | 最大値（５）が最も右側に移動する． |
| 2 3 1 4 \| 5 | 最大値（４）が最も右側に移動する． |
| 2 1 3 \| 4 5 | 最大値（３）が最も右側に移動する． |
| 1 2 \| 3 4 5 | 最大値（２）が最も右側に移動する． |

- 効率

  キーの比較回数 C = $(n^2-n)/2$（ = $O(n^2)$ ）

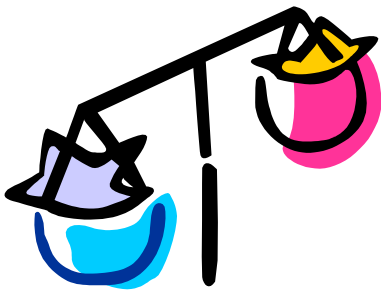
  項目間の移動（置換）回数 M（近似平均）= $n\times(n-1)/4$（ = $O(n^2)$ ）

```
/*************************************************************
    アルゴリズムとデータ構造
    サンプルプログラム exsort.c
    <<単純交換整列（バブルソートの例）>>
*************************************************************/
#include <stdio.h>
#define CHARLEN   20
/* １データ分を表す構造体 */
struct item {
    int key;
    char name[CHARLEN];
};
void printitem(struct item a[ ], int n);
void exsort(struct item a[ ], int n);

int main(void)
{
    #define N 8
    struct   item   table[N] = {
        { 65, "国語"},{ 90, "数学"},{ 85, "理科"},{ 70, "社会"},
        { 86, "英語"},{ 92, "体育"},{ 63, "音楽"},{ 85, "美術"}
    };
    int m = N;

    printf("table[ ]: ");
    printitem(table,m);
    printf("exsort(table,m)¥n");
    exsort(table,m);
    printf("table[ ]: ");
```

```
30        printitem(table,m);
31        return 0;
32    }
33
34    /* 配列内のデータを印刷する */
35    void printitem( struct item a[ ], int n )
36    {
37        int i;
38
39        for( i=0; i<n; i++ )
40            printf("<%d,%s> ", a[i].key, a[i].name);   /* %s は文字列を印字 */
41        printf("¥n");
42    }
43
44    /* 単純交換（バブルソート）による整列 */
45    void exsort( struct item a[ ], int n )
46    {
47        int i, j;
48        struct item x;
49
50        for( i=n-1; i>0; i-- ){
51            for( j=0; j<i; j++ ){
52                /* 隣同士を比較して，大小関係が異なれば入れ換える．
53                これにより，最も右側の要素が最大になる．これをくり
54                返すことで小さい順に並べ替えている．*/
55                if ( a[j].key > a[j+1].key ){
56                    x = a[j];   a[j] = a[j+1];   a[j+1] = x;
57                }
58            }
59        }
60    }
61
62    【実行結果】
63    table[ ]: <65,国語> <90,数学> <85,理科> <70,社会> <86,英語> <92,体育>
64    <63,音楽> <85,美術>
65    exsort(table,m)
66    table[ ]: <63,音楽> <65,国語> <70,社会> <85,理科> <85,美術> <86,英語>
67    <90,数学> <92,体育>
```



# おまけ：ソートアルゴリズムの一覧　（wikipedia より引用）

配列に格納された $n$ 個のデータをソートする場合について，各アルゴリズムの性能を示す．計算時間の表記に用いている記号 O については**ランダウの記号**を参照．以下の表で，$n$ はソートすべきデータ要素数である．平均実行時間と最悪実行時間は**時間計算量**を示している．このとき，ソートキーの長さは一定と仮定しており，比較や交換といった操作は定数時間で行われるとする．メモリ使用量は，入力データの格納域以外に必要となる領域を示している．これらは，いずれも**比較ソート**である．

| 名称 | 平均計算時間 | 最悪計算時間 | メモリ使用量 | 安定性 | 手 法 | 備　考 |
|---|---|---|---|---|---|---|
| バブルソート | — | $O(n^2)$ | $O(1)$ | ○ | 交換 | |
| シェーカーソート | — | $O(n^2)$ | $O(1)$ | ○ | 交換 | |
| コムソート | $O(n \log n)$ | $O(n \log n)$ | $O(1)$ | × | 交換 | コードサイズが小さくて済む。 |
| ノームソート | — | $O(n^2)$ | $O(1)$ | ○ | 交換 | |
| 選択ソート | $O(n^2)$ | $O(n^2)$ | $O(1)$ | × | 選択 | 安定ソートとしても実装可能 |
| 挿入ソート | $O(n + d)$ | $O(n^2)$ | $O(1)$ | ○ | 挿入 | $d$ は置換群の反転数で、$O(n^2)$ |
| シェルソート | — | $O(n \log^2 n)$ | $O(1)$ | × | 挿入 | |
| 2 分木ソート | $O(n \log n)$ | $O(n \log n)$ | $O(n)$ | ○ | 挿入 | 平衡 2 分探索木を使った場合 |
| ライブラリソート | $O(n \log n)$ | $O(n^2)$ | $O(n)$ | ○ | 挿入 | |
| マージソート | $O(n \log n)$ | $O(n \log n)$ | $O(n)$ | ○ | マージ | |
| In-place マージソート | $O(n \log n)$ | $O(n \log n)$ | $O(1)$ | ○ | マージ | 実装例はこちら: [1] |
| ヒープソート | $O(n \log n)$ | $O(n \log n)$ | $O(1)$ | × | 選択 | |
| スムースソート | — | $O(n \log n)$ | $O(1)$ | × | 選択 | |
| クイックソート | $O(n \log n)$ | $O(n^2)$ | $O(\log n)$ | × | パーティショニング | 単純な実装ではメモリ使用量が $O(n)$ になる。ピボット値として中央値を使えば、最悪時間が $O(n \log n)$ |
| イントロソート | $O(n \log n)$ | $O(n \log n)$ | $O(\log n)$ | × | 混成 | STL の実装によく使われている. |
| ペイシェンスソート | — | $O(n^2)$ | $O(n)$ | × | 挿入 | $O(n \log n)$ 以内に全ての最長増加部分列を探す。 |
| ストランドソート | $O(n \log n)$ | $O(n^2)$ | $O(n)$ | ○ | 選択 | |
| 奇偶転置ソート | — | $O(n^2)$ | $O(1)$ | ○ | 交換 | |
| シェアソート | — | $O(n^{1.5})$ | $O(1)$ | × | 交換 | |

たくさんありますが，代表的なアルゴリズム（本講義でサンプルプログラムが出ているもの）を覚えておけば，とりあえずは充分です．