C プログラミング

— ソート —

早稲田大学

本日の目標

- ソートを理解し，アルゴリズムを実装する
  - バブルソート
  - クイックソート
  - qsort 関数

ソート

ソートとは …

- 与えられた集合に対して，その要素を順序関係に基づいて並べ替えること．
- 順序関係
  - 任意の二つの要素 x，y について，どちらが先に並ぶべきかを表したもの．
  - 等号・不等号記号を流用して，$x < y$，$x = y$，$x > y$ と表現したりする．
  - $x < y$，$y < z$，$z < x$ のように矛盾があってはならない．
  - $x = y$ の場合は同順位を表す．この場合に，ソート前の順序を保持するアルゴリズムを安定ソート，必ずしも保持しないものを不安定ソートと呼ぶ．
  - 整数や実数については，数学的な大小関係をそのまま使うことが可能．
  - 文字列も，五十音順，アルファベット順などの順序関係を定義すればソート可能（例: 辞書，名簿など）．
- 計算量はアルゴリズムによって大きく異なり，要素数を N とすると，$\mathcal{O}(N^2)$，$\mathcal{O}(N \log N)$，$\mathcal{O}(N)$ のものがよく使われる．

ソート

- 代表的なアルゴリズム
  - バブルソート（隣接交換法）
  - 単純挿入法（基本挿入法）
  - 直接選択法
  - シェルソート
  - ヒープソート
  - クイックソート
  - マージソート
  - 基数ソート

バブルソート

- 隣接したデータの交換だけで実現される簡便なソート．隣接交換法とも呼ばれる．
- 安定ソートである．
- 計算量は $\mathcal{O}(N^2)$ と多いため，使用するのは，要素数が 10 個程度まで，または，ソートにかける時間の割合が非常に小さい場合のみに限定した方がよい．

バブルソートのアルゴリズム

要素数 N のデータを昇順にソートするとする．

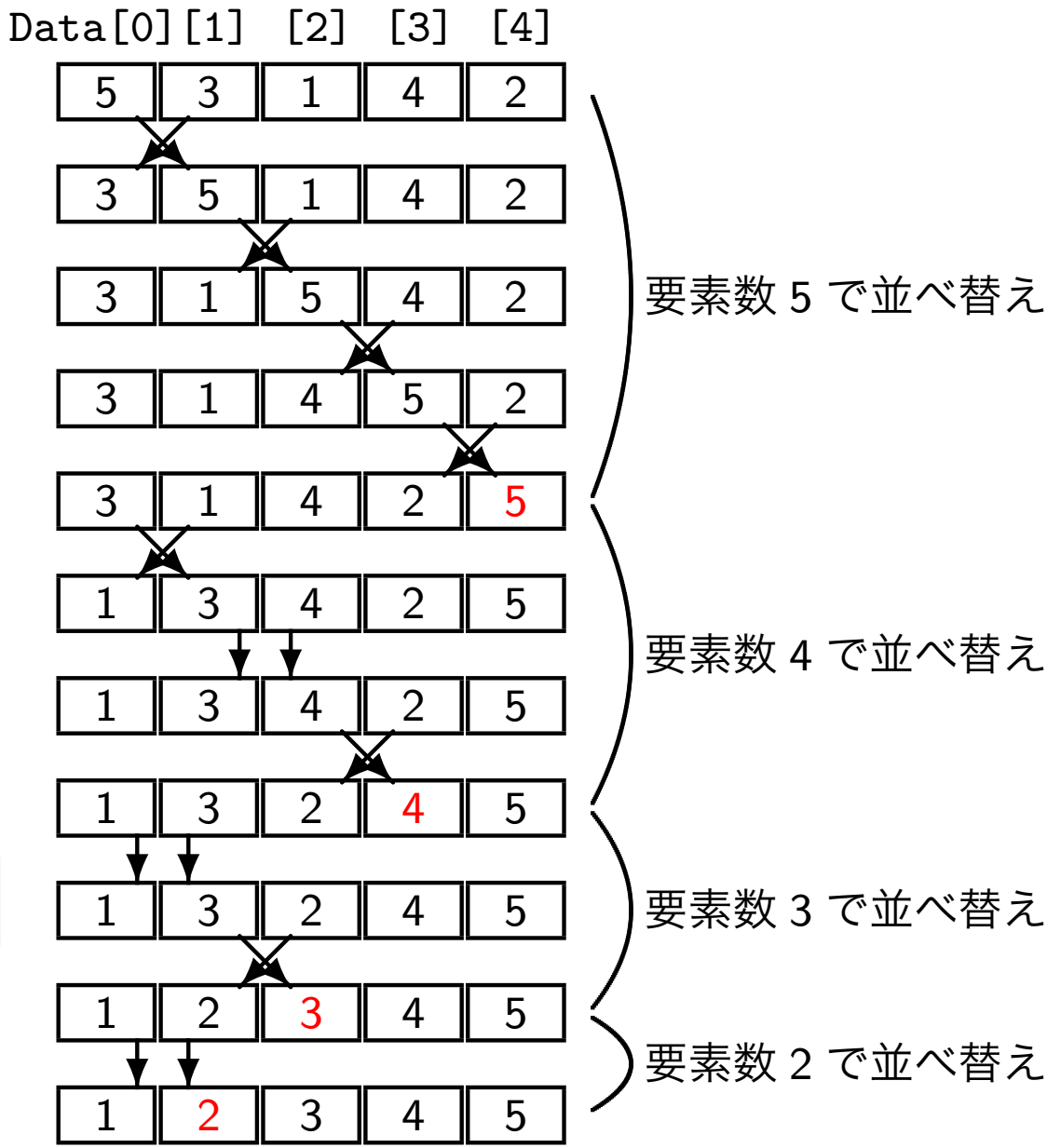
Step 1. Step 2 の操作を，先頭から N 個の要素について，$N-1$ 個の要素について，... と，2 個の要素についてまで繰り返して終了する．

Step 2. 先頭から順に隣接する二つの要素間で順序関係が正しくなるよう交換を行う．
（この操作により，最後に来るべき要素は最後に正しく移動する．）

バブルソート

- 要素数 $N = 5$ でのバブルソート例
- 最初は要素数 $i = 5$ で並べ替える.
  具体的には 0 番目と 1 番目の要素の比較入替, 1 番目と 2 番目の要素の比較入替,... 3 番目と 4 番目の要素の比較入替を行う.
- 次に要素数 $i = 4$ だと思って同様に並べ替える
- 上記操作を $i = 2$ まで繰り返す

上記操作を一般化してアルゴリズムを実装する.



演習問題

ファイル名: bubblesort.c

double 型配列に 5, 3, 1, 4, 2 がこの順番で格納されているとする.
このデータを昇順（小さいものから順に大きくなるように並べる）にバブルソートするプログラムを作成せよ.

- 表示はソート前と毎回の比較交換操作後に行うこと.

```
5, 3, 1, 4, 2,
3, 5, 1, 4, 2,
3, 1, 5, 4, 2,
3, 1, 4, 5, 2,
3, 1, 4, 2, 5,
1, 3, 4, 2, 5,
1, 3, 4, 2, 5,
1, 3, 2, 4, 5,
1, 3, 2, 4, 5,
1, 2, 3, 4, 5,
1, 2, 3, 4, 5,
```

- バブルソートの部分は次ページのような引数，戻り値を持つ関数として作成すること.

演習問題

プログラムの概形は次のようにせよ．配列のサイズ N を計算で求めて，引数として渡している点に注意．

```
#include <stdio.h>

void PrintData(double *Data, int N) {
  ...
}
void swap(double *x, double *y) {
  ...
}
void BubbleSort(double *Data, int N) {
  ...
}
int main(void) {
  double Data[] = {                    /*要素数を与えていないことに注意*/
    5, 3, 1, 4, 2
  };
  int N = sizeof(Data) / sizeof(Data[0]); /*次頁を参考にせよ．*/
  PrintData(Data, N);
  BubbleSort(Data, N);
  return 0;
}
```

sizeof() 演算子

`sizeof(型)` という形で使用する．指定された型がメモリ上で使用するバイト数を返す演算子．コンパイル時に定数として扱われる．変数を指定しても，その型について計算される．

- 使用例と意味する数字
  - `sizeof(char)` $\leftarrow 1$
  - `sizeof(int)` $\leftarrow 4$
  - `sizeof(double)` $\leftarrow 8$
- 次のように宣言されていたとすると，

```
int i;
int A[5];
```

次のような使い方もできる．

- `sizeof(i)` $\leftarrow 4$
- `sizeof(A[0])` $\leftarrow 4$
- `sizeof(A)` $\leftarrow 20$

静的確保配列のサイズ取得

- 静的確保配列は，初期化を行うことでサイズ指定を省略できる．
  次の書き方は，

  ```
  double A[] = { 5, 3, 1, 4, 2 };
  ```

  初期化子の要素数が 5 なので，次のように解釈される．

  ```
  double A[5] = { 5, 3, 1, 4, 2 };
  ```

- このことを用いると，配列のサイズを自動取得できる．
  次のコードでは，N は 5 となる．

  ```
  double A[] = { 5, 3, 1, 4, 2 };
  int N = sizeof(A) / sizeof(A[0]);
  ```

  配列の要素数を色々と変えて試したい場合などに便利．

解答例

```
#include <stdio.h>

void PrintData(double *Data, int N) {
  /* Data の要素を順に表示する */
  int i;
  for(i=0;i<N;i++){
    printf("%.0f,",Data[i]);
  }
  printf("\n");
}

void swap(double *x, double *y) {
  /* (*x) と (*y) を入れ替える */
  double z;
  z = *x; *x = *y; *y = z;
}
```

```
void BubbleSort(double *Data, int N) {
  /* Data の要素を昇順にバブルソートする */
  int i,j;
  for(j=N;j>=2;j--){
    for(i=0;i<j-1;i++){
      if(Data[i]>Data[i+1]){
        swap(&Data[i],&Data[i+1]);
      }
      PrintData(Data,N);
    }
  }
}

int main(void) {
  double Data[] = {
    5, 3, 1, 4, 2
  };
  int N = sizeof(Data) / sizeof(Data[0]);
  PrintData(Data, N);
  BubbleSort(Data, N);
  return 0;
}
```

解答例

```
#include <stdio.h>

void PrintData(double *Data, int N) {
  /* Data の要素を順に表示する */
  int i;
  for(i=0;i<N;i++){
    printf("%.0f,",Data[i]);
  }
  printf("\n");
}

void swap(double *x, double *y) {
  /* (*x) と (*y) を入れ替える */
  double z;
  z = *x; *x = *y; *y = z;
}
```

```
void BubbleSort(double *Data, int N) {
  /* Data の要素を昇順にバブルソートする */
  int i,j;
  for(j=N;j>=2;j--){
    for(i=0;i<j-1;i++){
      if(Data[i]>Data[i+1]){
        swap(&Data[i],&Data[i+1]);
      }
      PrintData(Data,N);
    }
  }
}

int main(void) {
  double Data[] = {
    5, 3, 1, 4, 2
  };
  int N = sizeof(Data) / sizeof(Data[0]);
  PrintData(Data, N);
  BubbleSort(Data, N);
  return 0;
}
```

解答例

```
#include <stdio.h>

void PrintData(double *Data, int N) {
  /* Data の要素を順に表示する */
  int i;
  for(i=0;i<N;i++){
    printf("%.0f,",Data[i]);
  }
  printf("\n");
}

void swap(double *x, double *y) {
  /* (*x) と (*y) を入れ替える */
  double z;
  z = *x; *x = *y; *y = z;
}
```

```
void BubbleSort(double *Data, int N) {
  /* Data の要素を昇順にバブルソートする */
  int i,j;
  for(j=N;j>=2;j--){
    for(i=0;i<j-1;i++){
      if(Data[i]>Data[i+1]){
        swap(&Data[i],&Data[i+1]);
      }
      PrintData(Data,N);
    }
  }
}

int main(void) {
  double Data[] = {
    5, 3, 1, 4, 2
  };
  int N = sizeof(Data) / sizeof(Data[0]);
  PrintData(Data, N);
  BubbleSort(Data, N);
  return 0;
}
```

解答例

```c
#include <stdio.h>

void PrintData(double *Data, int N) {
  /* Data の要素を順に表示する */
  int i;
  for(i=0;i<N;i++){
    printf("%.0f,",Data[i]);
  }
  printf("\n");
}

void swap(double *x, double *y) {
  /* (*x) と (*y) を入れ替える */
  double z;
  z = *x; *x = *y; *y = z;
}
```

```c
void BubbleSort(double *Data, int N) {
  /* Data の要素を昇順にバブルソートする */
  int i,j;
  for(j=N;j>=2;j--){
    for(i=0;i<j-1;i++){
      if(Data[i]>Data[i+1]){
        swap(&Data[i],&Data[i+1]);
      }
      PrintData(Data,N);
    }
  }
}

int main(void) {
  double Data[] = {
    5, 3, 1, 4, 2
  };
  int N = sizeof(Data) / sizeof(Data[0]);
  PrintData(Data, N);
  BubbleSort(Data, N);
  return 0;
}
```

クイックソート

- 比較的高速とされる標準的な並べ替えアルゴリズム.
- 不安定ソートである.
- 計算量は平均的には $\mathcal{O}(N \log N)$ であるが，最悪の場合は $\mathcal{O}(N^2)$ となる．一般的に大規模データに使用して問題ないが，こだわる場合は他手法も考慮すること.
- クイックソートの概念
  - 要素の中から，中央値に近いものをピボットとし，そのピボット以下のグループとピボット以上のグループに分割する.
  - 上記操作を，分割後のグループに対しても再帰的に繰り返す.
  - グループの要素数が 2 未満になれば，並べ替えの必要がないので終了.
- 上記操作の補足
  - 中央値に近いものをどうやって選ぶか？
    → ここでは単純化のため，単に真ん中に位置するものを選ぶ．この場合，最初からある程度並べ替わっているデータの場合は計算量が少なくてすむ.
  - ピボットや，ピボットと同じ値の要素は，どちらのグループに分類されるのか？
    → どちらに分類してもよい．また，どちらにも分類せず，二つのグループの間に置いておいてもよいことにする.

クイックソート

クイックソートのアルゴリズム

データ $x_L, x_{L+1}, ..., x_R$ を昇順にソートする.

Step 1. $l = L$, $r = R$, ピボットを $p = x_{(L+R)/2}$（割り算は端数切捨て）とする.

Step 2. $l > r$ ならば Step 8 に移る.

Step 3. $x_l < p$ ならば, $l \leftarrow l + 1$ として Step 3 を繰り返す.

Step 4. $x_r > p$ ならば, $r \leftarrow r - 1$ として Step 4 を繰り返す.

Step 5. $l > r$ ならば Step 8 に移る.

Step 6. $l < r$ ならば x_l と x_r の値を入れ替える.

Step 7. $l \leftarrow l + 1$, $r \leftarrow r - 1$ として, Step 2 に戻る.

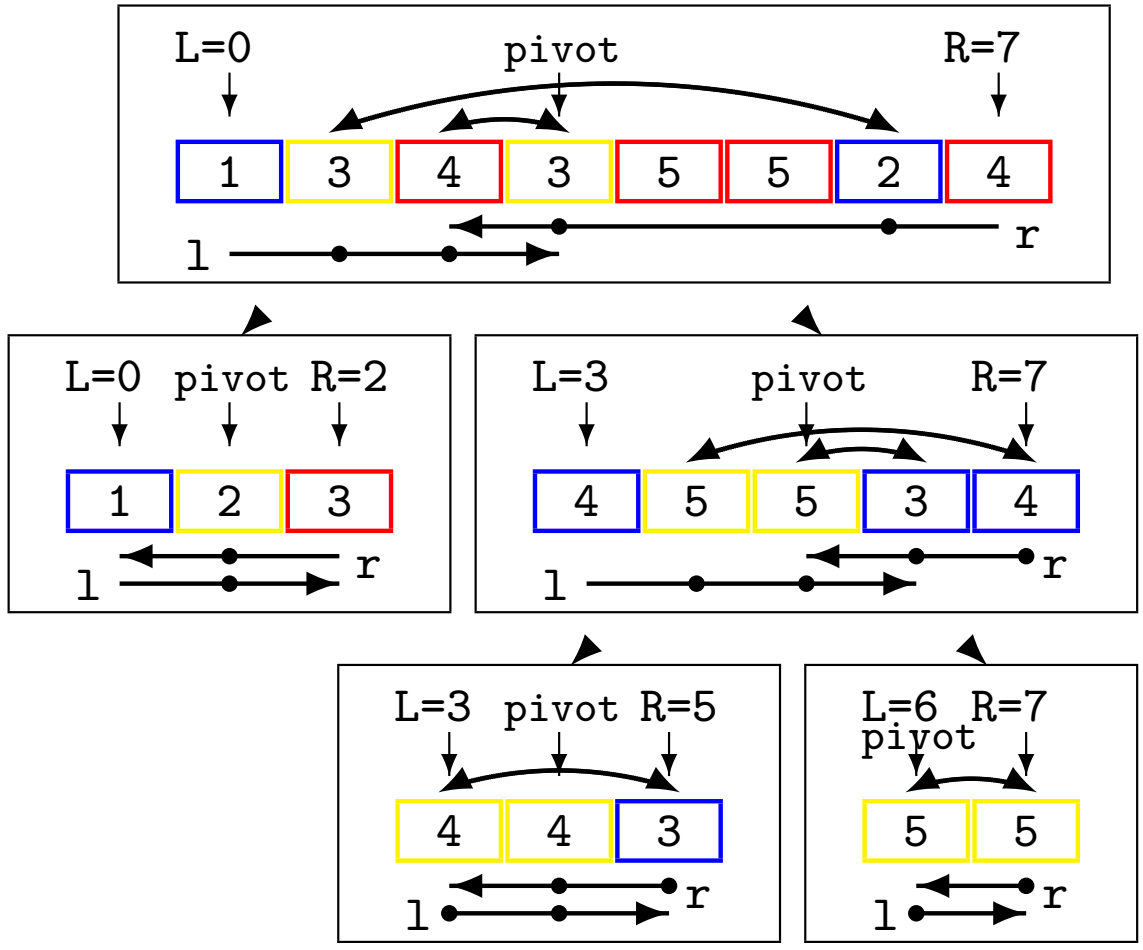
Step 8. $L < r$ ならば, $x_L, x_{L+1}, ..., x_r$ のデータについて本アルゴリズムを再適用.

Step 9. $l < R$ ならば, $x_l, x_{l+1}, ..., x_R$ のデータについて本アルゴリズムを再適用.

- Step 8, 9 の条件は, 要素数が 2 以上であることを意味している.
- 要素数 N の配列全体をソートするには, $L = 0$, $R = N - 1$ として上記アルゴリズムを適用する.

クイックソート

- ピボットと同じ値を黄色，小さい値を青色，大きい値を赤色として例示した．
- l は左端からピボット以上の値を探して停止．r は右端からピボット以下の値を探して停止．
- 停止位置を黒丸で例示．停止位置が l<r であればデータを入れ替え．
- l>r なら終了．各グループが 2 個以上データがあれば再帰的に上記操作を行う．



ソート関数

配列をソートする関数：qsort

- qsort：

```
void qsort(void *data,
           size_t num,
           size_t size,
           int (*func)(const void *, const void*))
```

- stdlib.h をインクルードする
- void data：配列 data の先頭ポインタ
- size_t num：配列の要素数
- size_t size：配列の要素一つあたりのサイズ
- int func： int 型の比較関数（自身で作成する）

ソート関数使用例

```
#include <stdio.h>
#include <stdlib.h>

int compare(const void *a, const void *b){ /* 自作の比較関数 */
    return *(int*)a-*(int*)b;
}

int main(void){
    int i;
    int data[] = {6,3,5,4,2,1};
    int N = sizeof(data) / sizeof(data[0]);
    printf("Before sorting the list is:");
    for (i=0; i<N; i++){
        printf("%3d",data[i]);
    }
    printf("\n");
    printf("After sorting the list is:");
    qsort(data,N,sizeof(int),compare);
    for (i=0; i<N; i++){
        printf("%3d",data[i]);
    }
    printf("\n");
    return 0;
}
```

ソート関数使用例

```
#include <stdio.h>
#include <stdlib.h>

int compare(const void *a, const void *b){ /* 自作の比較関数 */
    return *(int*)a-*(int*)b;
}

int main(void){
    int i;
    int data[] = {6,3,5,4,2,1};
    int N = sizeof(data) / sizeof(data[0]);
    printf("Before sorting the list is:");
    for (i=0; i<N; i++){
        printf("%3d",data[i]);
    }
    printf("\n");
    printf("After sorting the list is:");
    qsort(data,N,sizeof(int),compare);
    for (i=0; i<N; i++){
        printf("%3d",data[i]);
    }
    printf("\n");
    return 0;
}
```

ソート関数使用例の説明

```
int compare(const void *a, const void *b){
        return *(int*)a-*(int*)b;
}
```

- $*a < *b$ の場合は負の整数を返す.
- $*a = *b$ の場合は 0 を返す.
- $*a > *b$ の場合は正の整数を返す.
- 戻り値によってソートを行う（上記関数は昇順にソートされる）.

```
qsort(data,N,sizeof(int),compare);
```

- data：ソートする配列
- 配列のサイズは N.
- sizeof(int) で int 型のサイズ（4byte）が得られる.
  例えば sizeof（double）では double 型のサイズ (8byte) が得られる.
- 自作した比較関数（compare）を書く.

本日のまとめ

- ソートを理解し，アルゴリズムを実装する
  - バブルソート
  - クイックソート
  - qsort 関数