

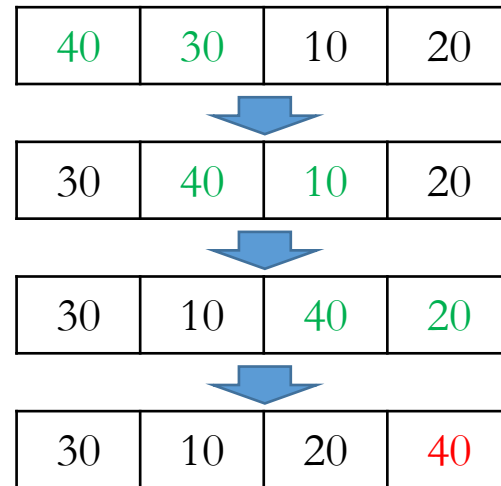
バブルソート

考え方

- 隣接する二つのデータを比較し、データの大小関係が逆のとき、二つのデータの入れ替えを行って整列を行う

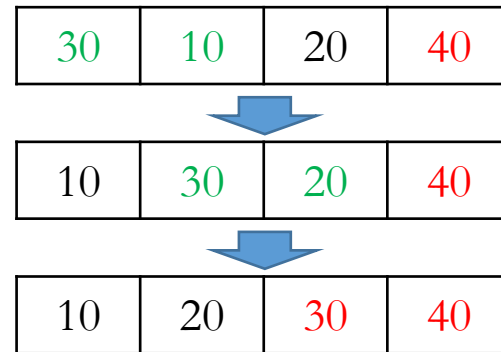
ソート例: (40, 30, 10, 20)

Step 1: データ全体に対し、隣同士の比較・交換を行い、最大値を最後に移動



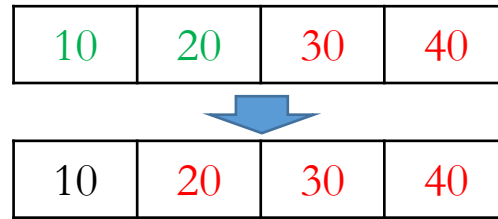
ソート例: (40, 30, 10, 20)

Step 2: 最後を除いたデータに対し、隣同士の比較・交換を行い、最大値を「最後」に移動



ソート例: (40, 30, 10, 20)

Step 3: 最後2個を除いたデータに対し、隣同士の比較・交換を行い、最大値を「最後」に移動



データは1個しか残っていないので、処理は終了

手順

データ: 配列 $a[0], a[1], \dots, a[n-1]$

- Step 1 配列要素の隣同士 $a[0]$ と $a[1]$, $a[1]$ と $a[2], \dots, a[n-2]$ と $a[n-1]$ を比較・（大小が逆であれば）交換していき、その結果、最大値が $a[n-1]$ に入る
- Step 2 配列要素の隣同士 $a[0]$ と $a[1]$, $a[1]$ と $a[2], \dots, a[n-3]$ と $a[n-2]$ を比較・（大小が逆であれば）交換していき、その結果、（2番目の）最大値が $a[n-2]$ に入る。
- ⋮
- Step $n-1$ 配列要素の隣同士 $a[0]$ と $a[1]$ を比較・（大小が逆であれば）交換し、その結果、（ $n-1$ 番目の）最大値が $a[1]$ に入る

Question

- n 個のデータのソートには合計何回の比較が必要か？
- 交換は？

ちなみに、選択ソートの計算量

- ソートは比較と交換からなる
- 比較の回数:

$$(n - 1) + (n - 2) + \dots + 1 = \frac{(n-1+1)}{2} \times (n - 1) = O(n^2)$$

- 交換の回数:

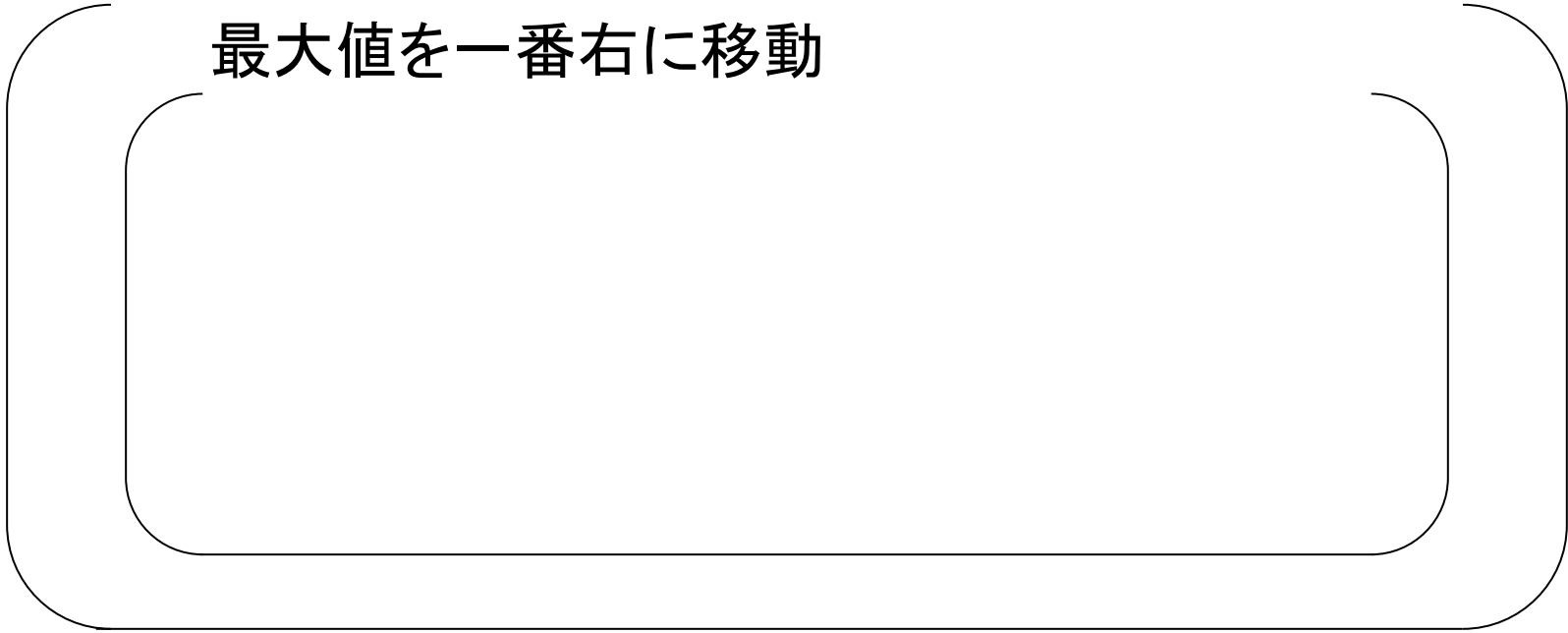
最大でも(あるいは判定を使わなければ) $n-1$ 回で、 $O(n)$



- $O(n^2)$

大まかなループ

データ範囲を縮小しながら
最大値を一番右に移動



最大値を一番右に移動させるループ

配列: $a[0], a[1], \dots, a[i]$

繰り返し変数: j

1. $j=?$ から $??$ まで、以下を繰り返す

1.1 $?$ であれば $a[?]$ と $a[?]$ を交換する

Question

- 次のバブルソートのアルゴリズム（基本版）を完成させなさい。ただし、中の「**最大値を一番右に移動させるループ**」は、その対象データの範囲はステップごとに縮小していくことに注意

バブルソートアルゴリズム(基本版)

入力: 配列 $a[0], \dots, a[n-1]$

出力: ソート済みの配列 $a[0], \dots, a[n-1]$

補助: ステップ用 i , 配列の要素番号用 j

(データ範囲縮小に対応するループ)

1. $i=?$ から $??$ まで、次の処理を繰り返す

(最大値を一番右に移動させるループ)

1.1 {比較、交換} $j=?$ から $??$ まで、次の処理を繰り返す

1.1.1 もし $?>??$ であれば、 $?$ と $??$ を交換する

バブルソート(基本版)の問題点

- 問題1: 処理範囲内がすべて整列済みのデータであっても、最初から最後まで比較が行われる
 - たとえば1,3,5,8,10や10,1,3,5,8
- 問題2: ステップごとに、最大のデータを右に移動し、確定しているが、果たして確定できるのが1個のみか

バブルソートの改良

- 改良1: ステップ内で交換が一度も起こらなかったら処理を打ち切る
 - これにより、問題1を解決
- 改良2: ステップ内で可能なら複数のデータを確定する
 - その「複数のデータ」が全データなら、改良1に等しい
 - 次のステップのデータ範囲はその「複数のデータ」分縮小するので、効率がよくなる
 - つまり、改良2は問題1,2を同時に解決
- シェーカーソート
 - 改良2の双方向版

改良1

- ステップ内で交換あったか／なかったかについての変数(フラッグ)を導入すると簡単に実現できる

Question

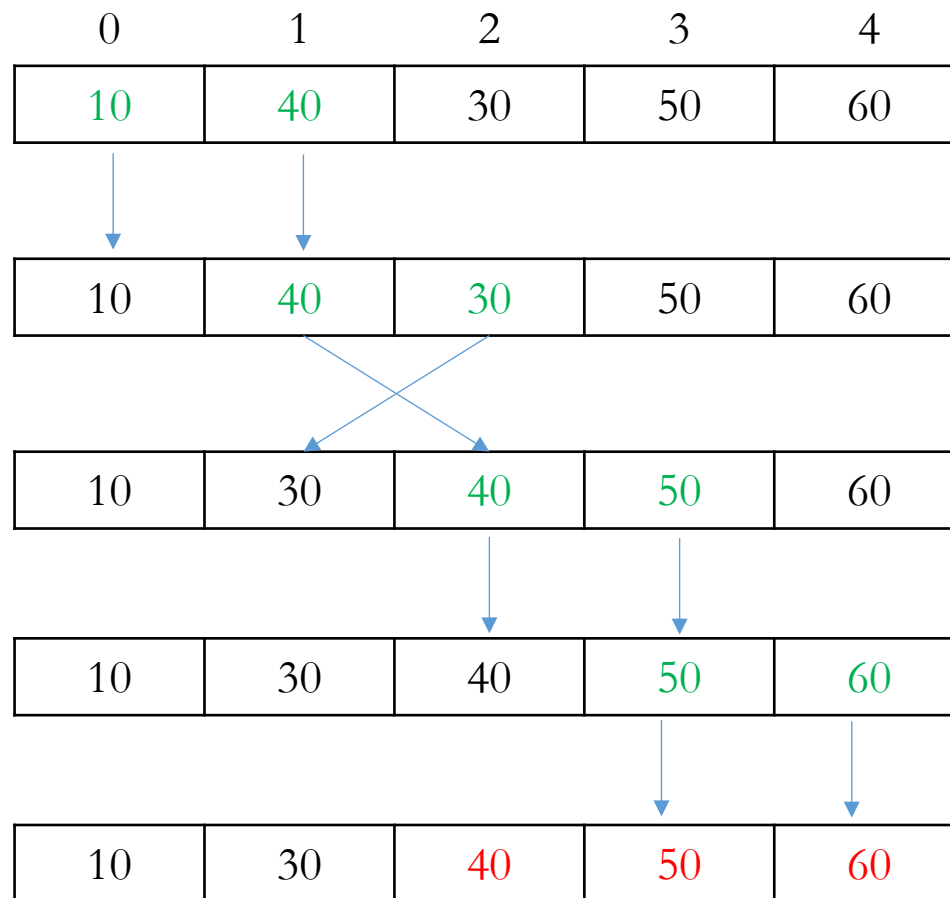
- アルゴリズム(基本版)をどう直せばアルゴリズム(改良1)になるか？

改良2

- ステップ内において、可能であれば複数のデータを一括してソート済みとして確定する
 - この改良は改良1を含む

バブルソート(改良2)の例

Step 1



要素番号2以降のデータ間に交換がなかったため、これらのデータをソート済みとして確定

バブルソート(改良2)の例

Step 2

0	1	2	3	4
10	30	40	50	60
↓	↓			
10	30	40	50	60

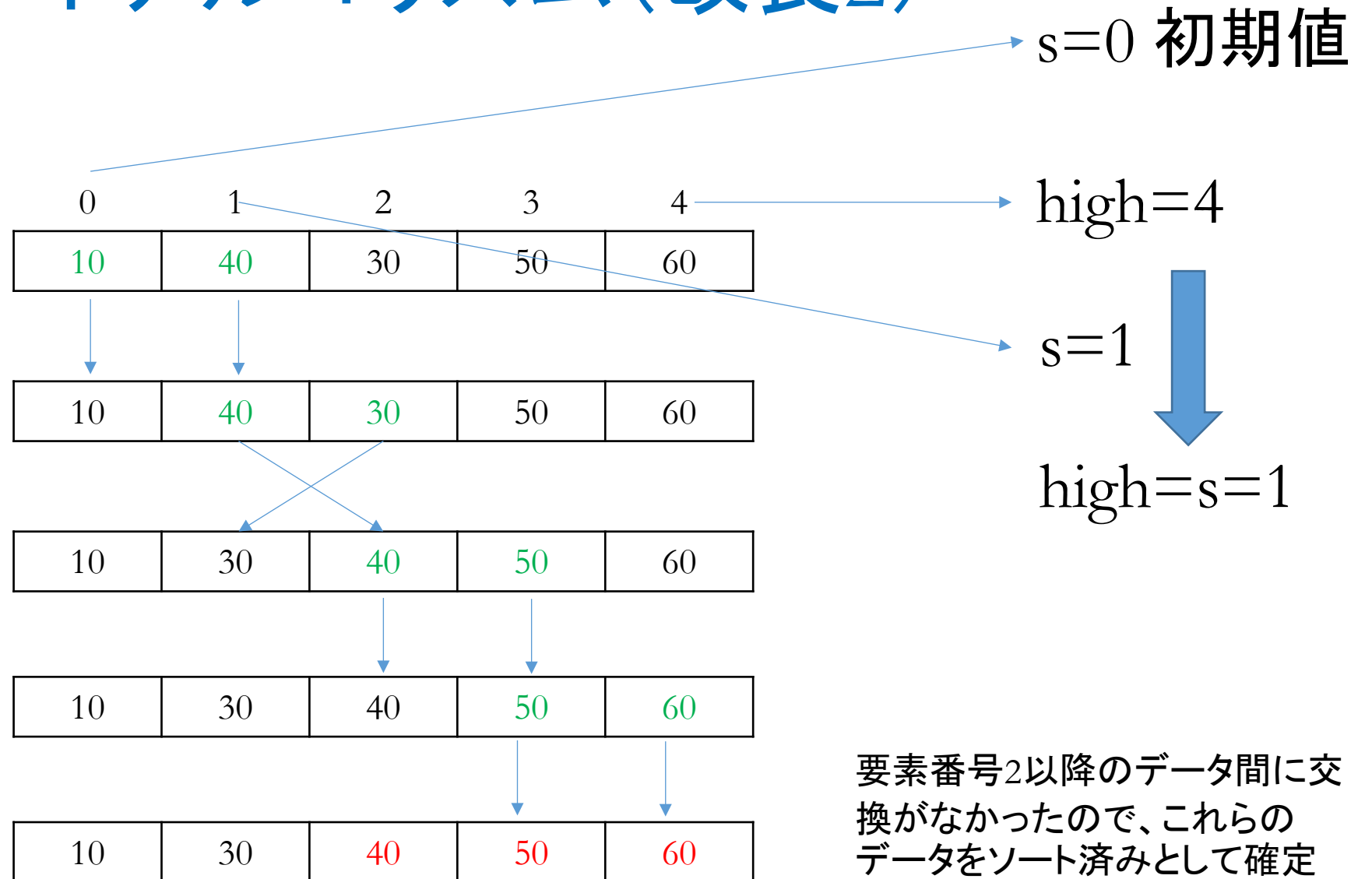
要素番号0,1のデータ間に交換がないので、ソート済みとして確定
ソート終了

Question

- 上記例は、改良2であれば、5個のデータを2ステップでソートすることができたことを意味する。さて、上記データを基本版でソートするとなれば、何ステップ必要？

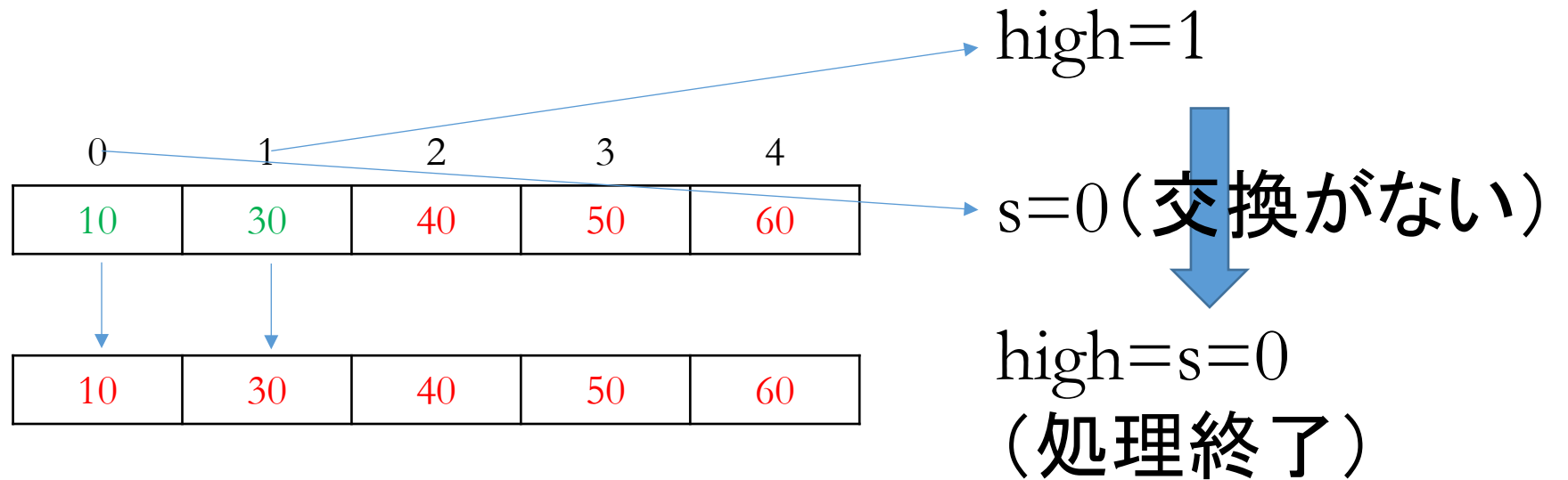
バブルソートアルゴリズム(改良2)

Step 1



バブルソートアルゴリズム(改良2)

Step 2



要素番号0,1のデータ間に交換がないので、ソート済みとして確定
ソート終了

プログラミングのポイント

- **最後に**交換のあったデータの位置を次のステップの右の範囲として用いるとよい
- プログラム的には以下のような構造を用いるとよい

```
high=n-1;           // 右の範囲
while(high>0){       // 各ステップに対応
    s=0;
    for(j=0; j<high; j++){ // ステップ内の比較交換に対応
        交換あればs=j;    // 交換ある度にsを更新
    }
    high=s;
}
```

バブルソートアルゴリズム(改良2)

- バブルソートアルゴリズム(基本版)を、前のスライドの「プログラミングのポイント」で修正するとよい

バブルソートアルゴリズム(改良2)

入力: 配列 $a[0], \dots, a[n-1]$

出力: ソート済みの配列 $a[0], \dots, a[n-1]$

補助: 配列の要素番号用 j

交換はどこまであったか用 s , 右の範囲用 $high$

1. $high = ?$ とおく
2. 条件 $high > ?$ が満たされる間、次の操作を繰り返す
 - 2.1 $s = ?$ とおく
 - 2.2 $j = 0$ から $high - 1$ まで、次の操作を繰り返す
 - 2.2.1 もし $? > ?$ であれば、 $? と ?$ を交換し、 $s = ?$ とおく
 - 2.3 $high = ?$ とおく

計算量について

- 基本版(既出)

- 比較の回数:

$$(n-1) + (n-2) + \dots + 1 = \frac{(n-1+1)}{2} \times (n-1) = O(n^2)$$

- 交換の回数: 比較回数以下

- 計算量は $O(n^2)$

- 改良版

- 整列済みの部分が多い場合に比較回数を減らすことが可能

- データの交換の回数を減らすことはできない

- データの交換に時間がかかる場合は時間をあまり短縮できない

- 計算量は $O(n^2)$

Question

- 選択ソートとバブルソートの共通点と相違点を列挙しなさい
- どっちが速そう？

第3回演習課題

1. まず、バブルソートアルゴリズム(基本版)を実現する関数 `void BSort(int n, int a[])` を作成し、その動作を確認できるプログラム (`ex03-bsort.c`) を作成しなさい。ただし、`n` はデータの個数、`a[]` はデータ配列である。上記完成後、比較回数と交換回数をグローバル変数で宣言し、`void BSort(int n, int a[])` の中で比較回数と交換回数をカウントするように関数を改良する。求めた比較回数を `main` 関数で出力するように動作確認プログラム (`ex03-bsortC.c`) を改良しなさい。なお、グローバル変数については本講義資料の最後のページを参照するとよい。

注意: 提出は `ex03-bsortC.c` の1回のみです。

第3回演習課題

2. アルゴリズム(改良1)を実現する関数 `void BSortI(int n, int a[])` を作成し、その動作を確認できるプログラム (`ex03-bsortI.c`) を作成しなさい。ただし、課題1と同様、比較と交換の回数が出力できるようにすること。
3. アルゴリズム(改良2)を実現する関数 `void BSortII(int n, int a[])` を作成し、その動作を確認できるプログラム (`ex03-bsortII.c`) を作成しなさい。ただし、課題1と同様、比較と交換の回数が出力できるようにすること。

ex03-bsortI.cの実行例

Input the number of data

4

Input the data

4 1 2 3

Sorted data

1 2 3 4

Number of Comparing and Swapping: 5 3

Input the number of data

6

Input the data

1 2 3 4 5 6

Sorted data

1 2 3 4 5 6

Number of Comparing and Swapping: 5 0

グローバル変数について

- `main()` や他の関数の外で宣言する変数で、その影響する範囲はプログラム全範囲となる(あまり厳密な言い方ではないが)。
- プログラム例

```
#include <stdio.h>
int Z=10;    // グローバル変数
int proc(int y)
{
    return Z+y;
}
int main(void)
{
    int x=5;
    printf("%d %d %d\n", x, Z, proc(x));
    return 0;
}
```