

アルゴリズムとデータ構造

第4回 単純なソート

第4回のキーワード

2

アルゴリズム関係

- ソート, 並び替え, 整列
- バブルソート/単純交換法 (bubble sort)
- 選択ソート/単純選択法 (selection sort)
- 挿入ソート/単純挿入法 (insertion soft)
- 連 (run)
- $O(n^2)$
- マージ (merge)

Java関係

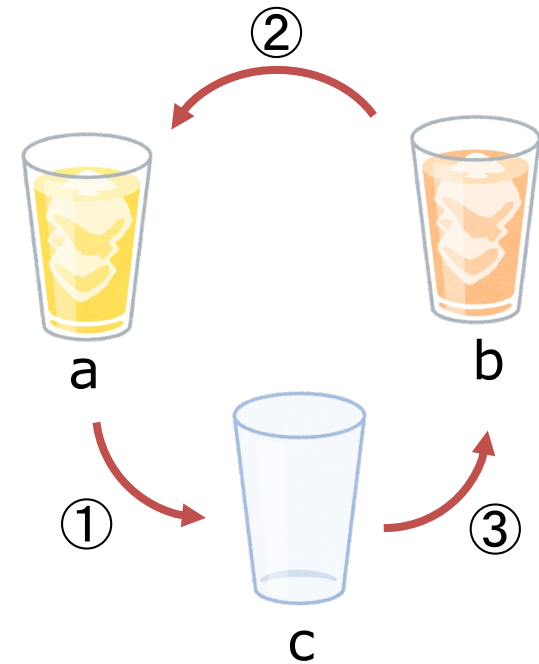
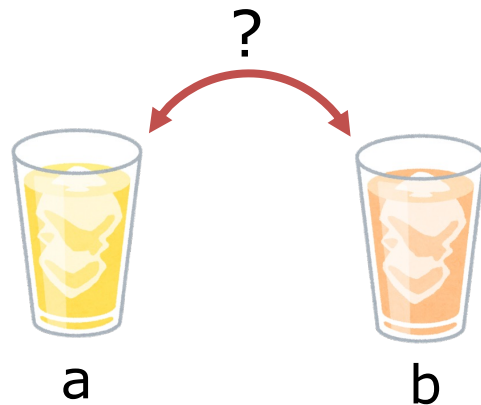
- 配列の要素を交換する
- 配列から最小値を選ぶ
- 配列の要素をずらす
- `a[i++]`

今回からデータが動きます

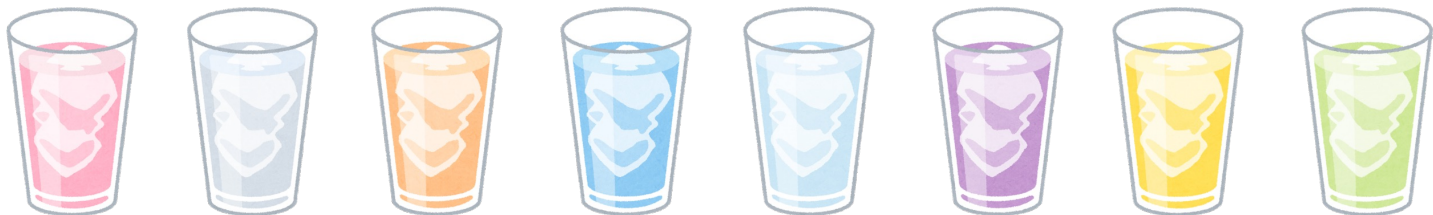
3

□ 内容を破壊しない手順が必要

■ 例: 変数の値の交換



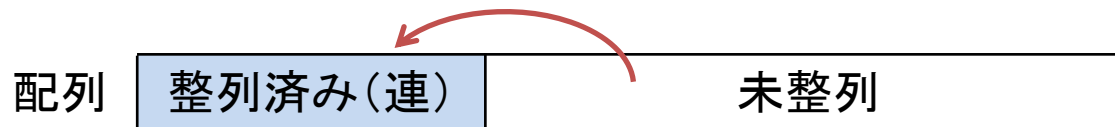
■ 配列の内容がどんどん入れ替わっていく



ソート(整列)

4

- ソート(整列, 並び替え)
 - ▣ 配列の内容を何らかの順序で並び替える
 - ▣ 例: 小さい順(昇順), 大きい順(降順), アルファベット順など
 - ▣ 要素には全順序の関係が必要(どの2要素も前後関係がある)
- 単純なソートの基本戦略
 - ▣ 配列の前半に整列済みの列(「連」という)を作り, 残りの未整列の部分から1つずつ要素を移して, 連を伸ばしていく



- ▣ そのためを使う基本アルゴリズム: 配列の要素同士を交換 / 配列から最小値を選択 / 配列の内容をずらして挿入

バブルソート

5

□ アルゴリズム

- 未整列の範囲の末尾から先頭に向かって、順にとなり合う要素を比較し、小さい順でなければ要素を**交換**していく
- すると、最小値が先頭に来るので、整列済みの範囲を1要素分広げ、残りの未整列の範囲に対して同様の処理を行う
- 以上の手順を繰り返すと、最終的に全範囲が整列済みになる

□ 実は、ダメな方法

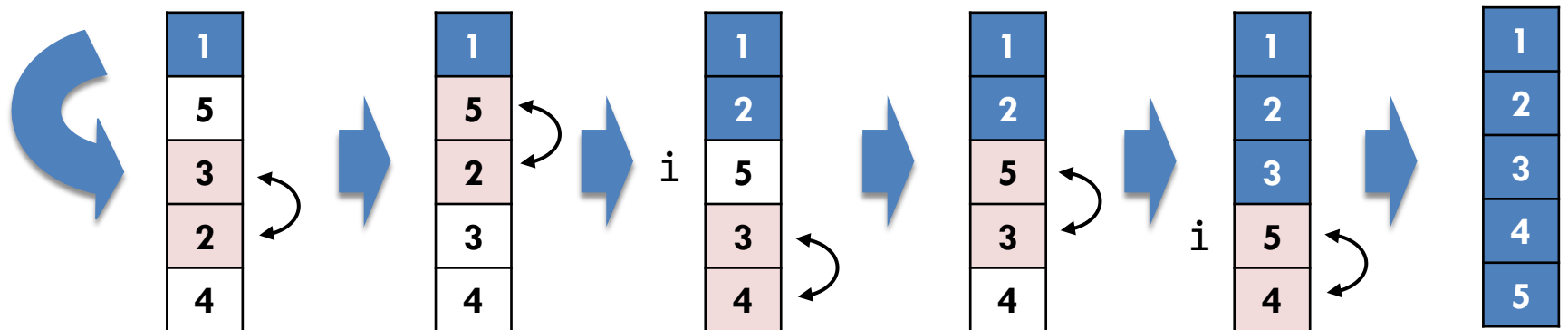
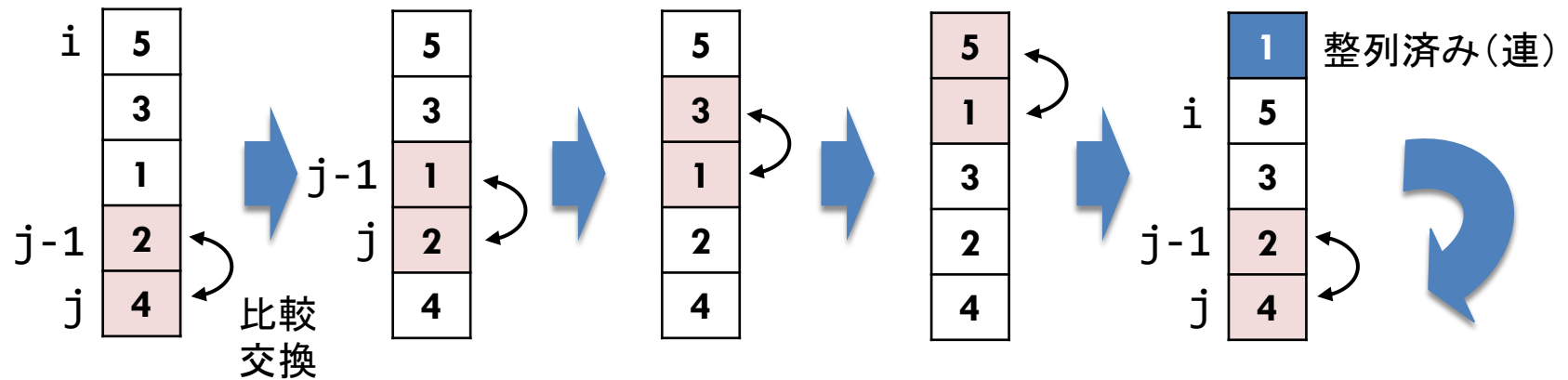
- 世界初の整列アルゴリズム
- どんな場合でも、2重のforループをすべて回る
- 効率が悪いので使われない

```
for (i = 0; i < n - 1; i++) {  
    for (j = n - 1; j > i; j--) {  
        if (a[j-1] > a[j]) {  
            a[j-1]とa[j]を交換  
        }  
    }  
}
```

バブルソートの例

6

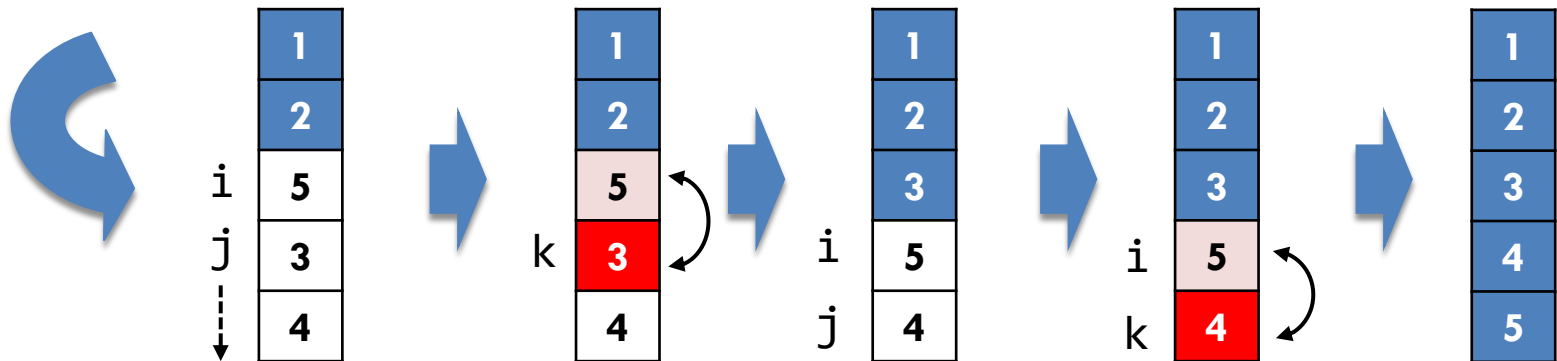
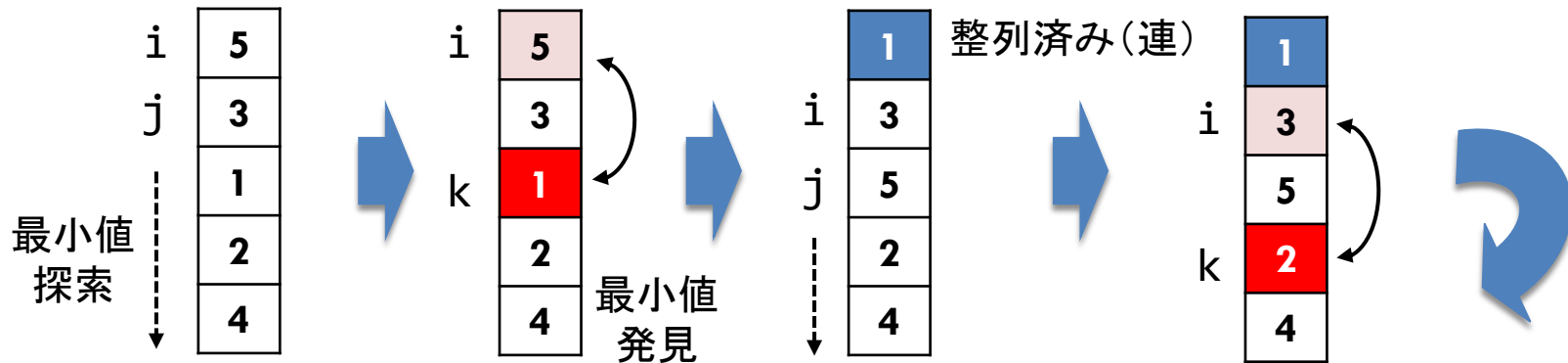
- 後ろから順に, となり同士を比較して交換していく



選択ソートの例

8

- 未整列範囲から最小値を選択し，先頭に並べていく



確認問題

9

- 最小値の位置を見つける処理
 - ▣ 下記のプログラムの空欄を適切に埋めよ
 - ▣ さらに, この部分の比較回数の最大と平均を考えよ

```
// a[i]~a[n-1]の範囲から最小値a[k]を見つける
int k = i; // 最初の要素a[i]を仮の最小値とする
for (int j = i + 1;           ; j++) {
    if (                       ) {
        k = j; // kは添字 (場所) であることに注意
    }
}
// 最小値が入っている要素a[k]を見つけた
```

挿入ソート

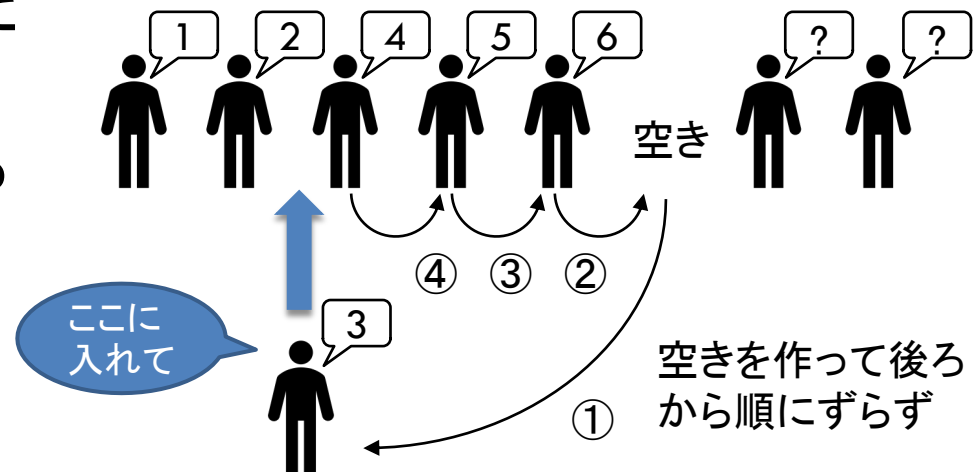
10

□ アルゴリズム

- まず、配列の先頭の要素は1個で整列済みとする
- 未整列範囲の先頭の要素を取り出し、その位置を空きとみなす
- 空きを利用して整列済みの要素を後ろから順に1つずつ後ろにずらし、取り出した要素を適切な大小関係の位置に**挿入**する
- 以上の手順を繰り返すと、最終的に全範囲が整列済みになる

□ 配列への挿入のしかた

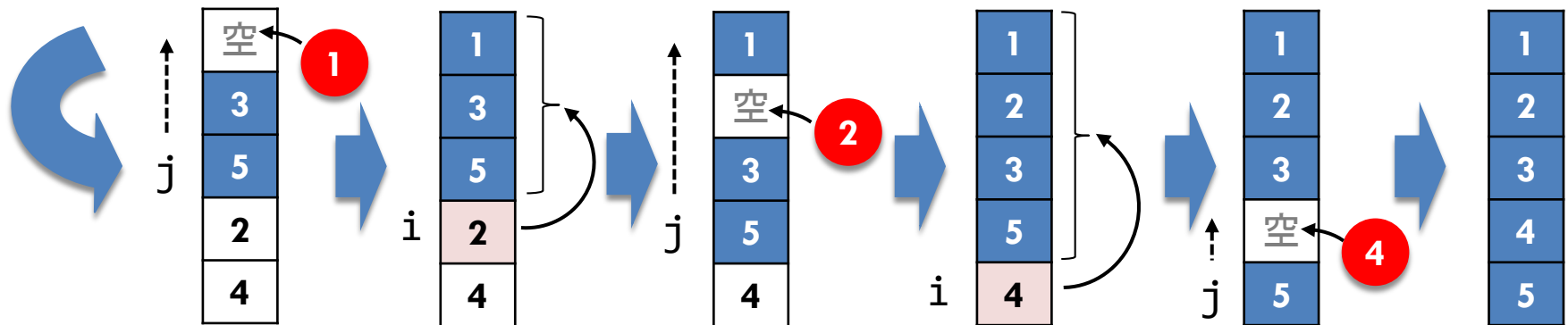
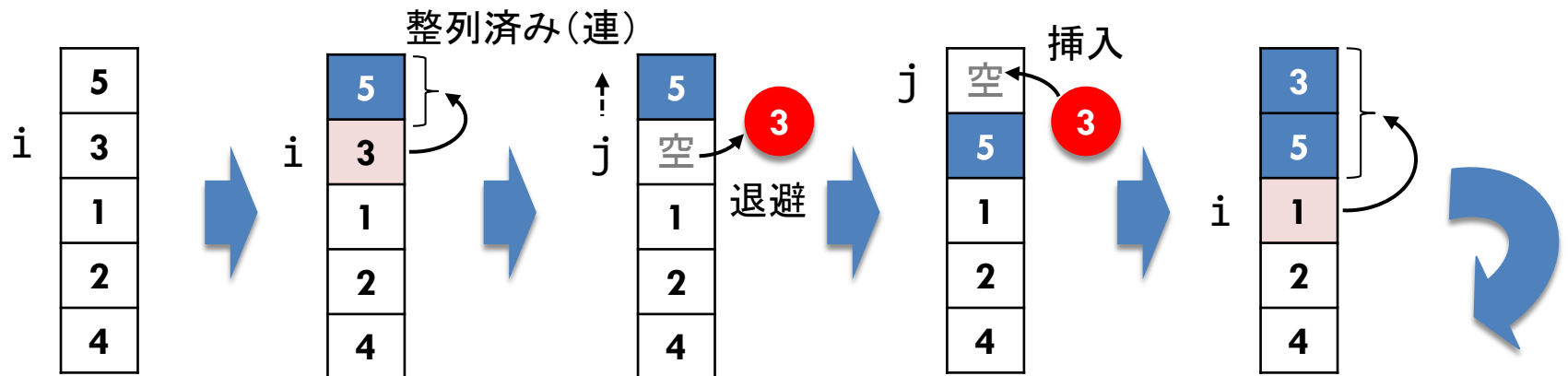
- 列に割り込ませようとしても前後が詰まっている
- 要素の中身を**後ろから順に**ずらして、「空き」を挿入位置まで持ってくる



挿入ソートの例

11

- 未整列の値を整列済み範囲に挿入していく



確認問題

12

- 配列の内容を後ろにずらす処理
 - 下記のプログラムの空欄を適切に埋めよ
 - さらに, この部分の比較回数の最大と平均を考えよ

```
double t = a[i]; // a[i]の中身をtに退避して“空き”とした
int j = i; // jを使って空き位置を指す

// 整列済みのa[0]~a[i]の範囲で, tより大きい要素を後ろにずらす
while (j >=    ) {
    if (a[j-1]    t) break; // 大小関係が適切な挿入位置を発見

    j--; // 空き位置は前に1つずれたことになる
}
// a[j]は空いているものとみなせる
```

確認問題

13

□ 単純なソートの比較

- 下記の配列aの内容を小さい順に並べ替えたい

a

4	1	3	2
---	---	---	---

- 「バブルソート」を用いた場合について、配列aの内容の変化の過程を図示し、値の比較と交換の回数を述べよ
- 「選択ソート」を用いた場合について、配列aの内容の変化の過程を図示し、値の比較と交換の回数を述べよ
- 「挿入ソート」を用いた場合について、配列aの内容の変化の過程を図示し、値の比較と移動の回数を述べよ

単純なソートの計算量

14

- 比較と交換の回数
 - ▣ ソートでは, 要素の比較回数は, 交換(移動)回数よりも多い
 - ▣ 計算オーダーについては, 比較回数だけを考慮すればよい
 - ▣ ただし, 交換の方が時間がかかるので実用上は注意が必要

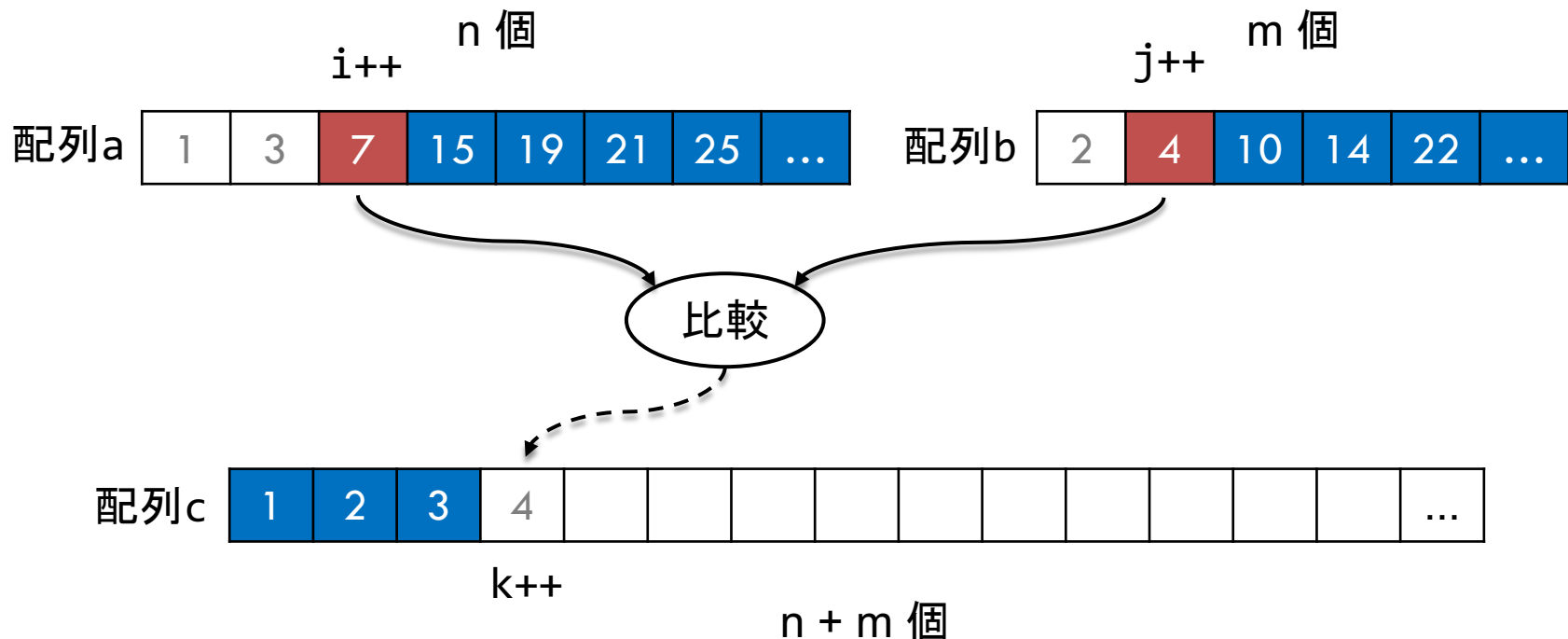
- バブルソートと選択ソート
 - ▣ 2重ループを全部回るので, 比較回数は $n(n-1)/2 \Rightarrow O(n^2)$
 - ▣ その上で, 交換回数が劇的に少ない選択ソートの方が高速

- 挿入ソート
 - ▣ 最悪の場合は, ループを全部回り, 比較回数は $n(n-1)/2$
 - ▣ 最善の場合は, n 回の比較だけで全く要素を移動せずに完了

配列のマージ

15

- 整列済みの配列を併合する
 - ▣ 先頭同士を比較して、先に来るものを取り出して並べる
 - ▣ ただし、各配列の中に残りがあることを先にチェックする



C系言語のよくある書き方

16

- **{ } カッコの省略**
 - ▣ ブロックの中身が1文の場合は, カッコ { } は省略できる
 - ▣ というより, { } は複数の文をまとめて1文として扱う記号
 - ▣ 例: `if (x < 0) x = 0; // これは if (x < 0) { x = 0; }`と同じ

- **a[i++], a[++i], a[i--], a[--i] など**
 - ▣ 配列の要素に順にアクセスする簡潔な記法
 - i++(後置インクリメント)は, 先に式を評価してから, iに1を加える
 - ++i(前置インクリメント)は, 先にiに1を加えてから, 式を評価する
 - ▣ 配列のコピーで `a[i++] = b[j++]` のような書き方をよく使う
 - ▣ 例: `while (i < a.length && j < b.length)`
`a[i++] = b[j++]; // { a[i] = b[j]; i++; j++; }`と同じ