

第4回のキーワード

1

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)

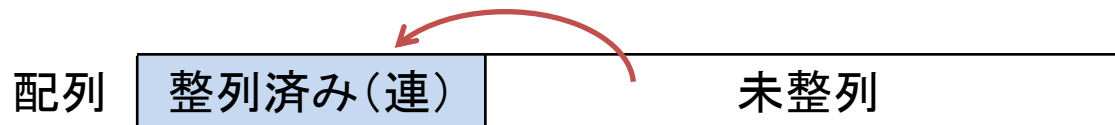
Java関係

- 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++]`

ソート(整列)

2

- 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

3

□ 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

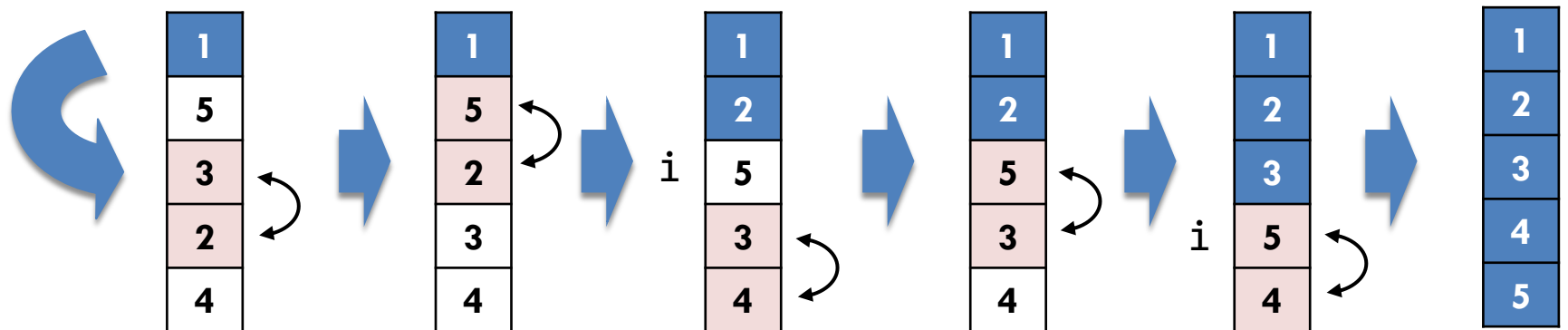
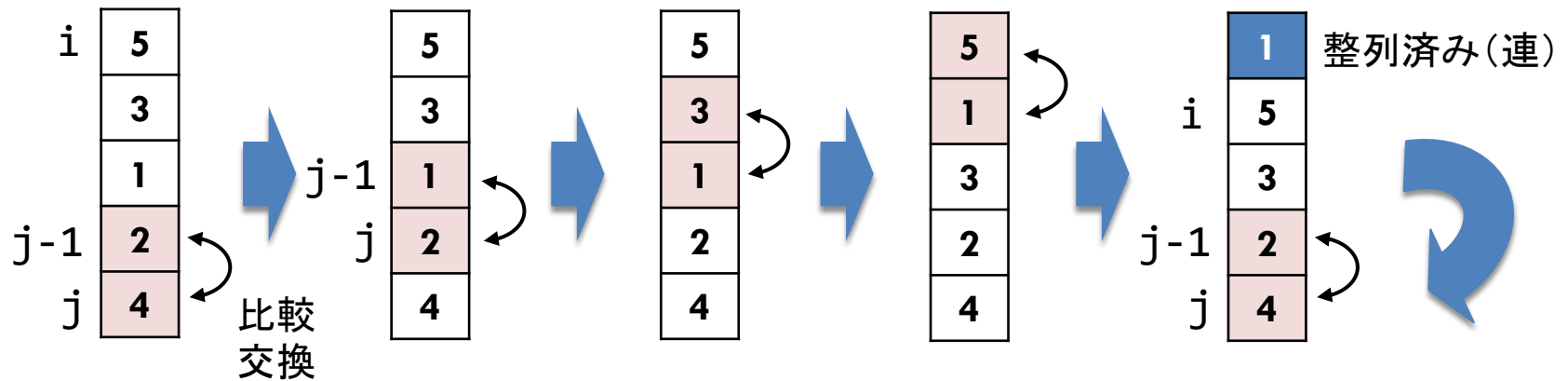
- 世界初の整列アルゴリズム
- どんな場合でも、2重のforループをすべて回る
- 効率が悪いので使われない

```
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

4

- 後ろから、と隣り同士を比較して交換していく



選択ソート

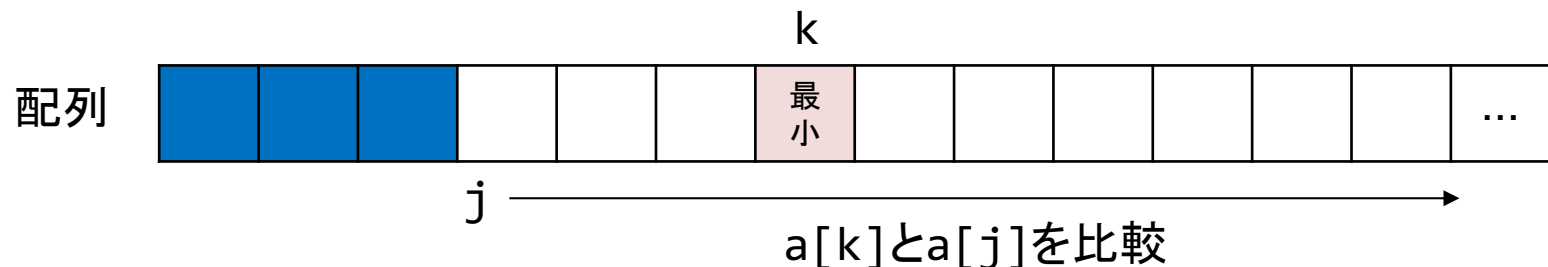
5

□ アルゴリズム

- 未整列の範囲の要素を順に調べて最小の要素を**選択**し、未整列の範囲の先頭の要素と交換する
- すると、整列済みの範囲が1要素分広がるので、残りの未整列の範囲に同様の処理を行う
- 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

□ 最小値の“位置”の求め方

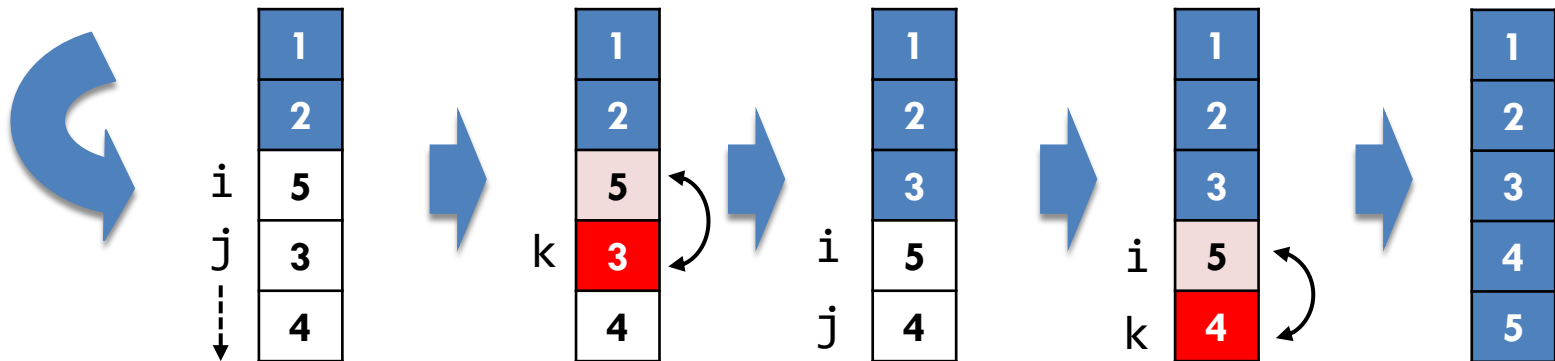
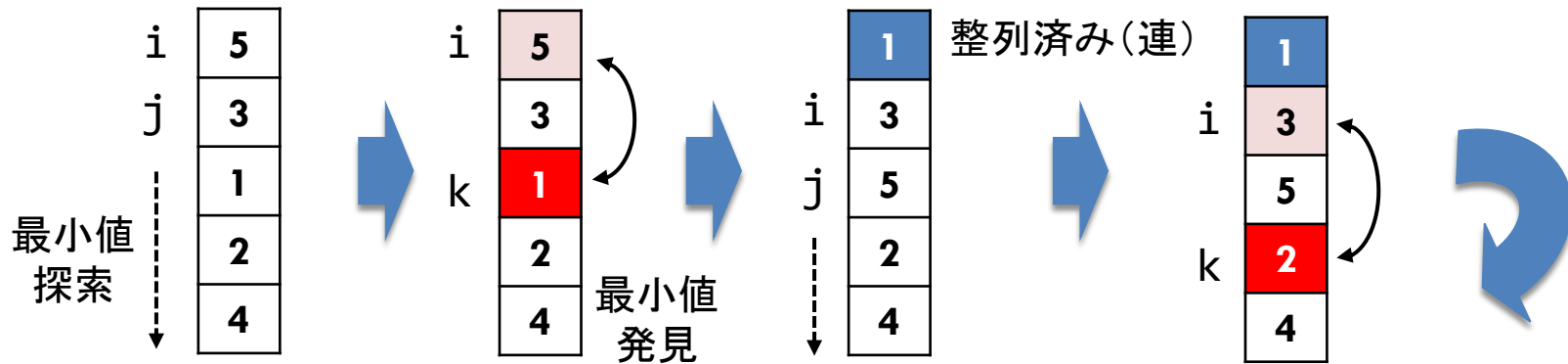
- 仮の最小値の位置を k として、 $a[k]$ と範囲内の各位置 j の要素 $a[j]$ を比較し、 $a[j]$ のほうが小さければ、 k の値を j に更新する



選択ソートの例

6

- 未整列範囲から最小値を選択し，先頭に並べていく



挿入ソート

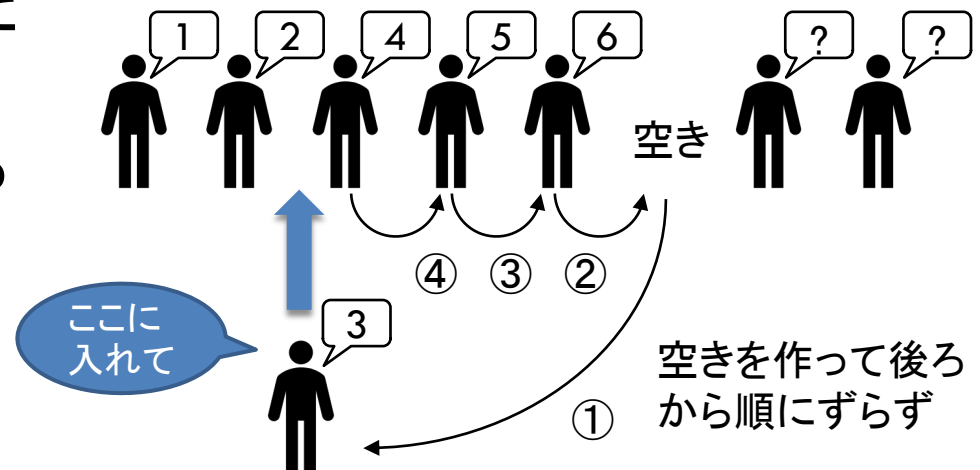
7

□ アルゴリズム

- まず、配列の先頭の要素は整列済みとする
- 未整列の範囲の先頭の要素を取り出すとその位置は空く
- 空きを利用して整列済みの要素を末尾から1つずつ後ろにずらすことで、取り出した要素を大小関係が適切な位置に**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

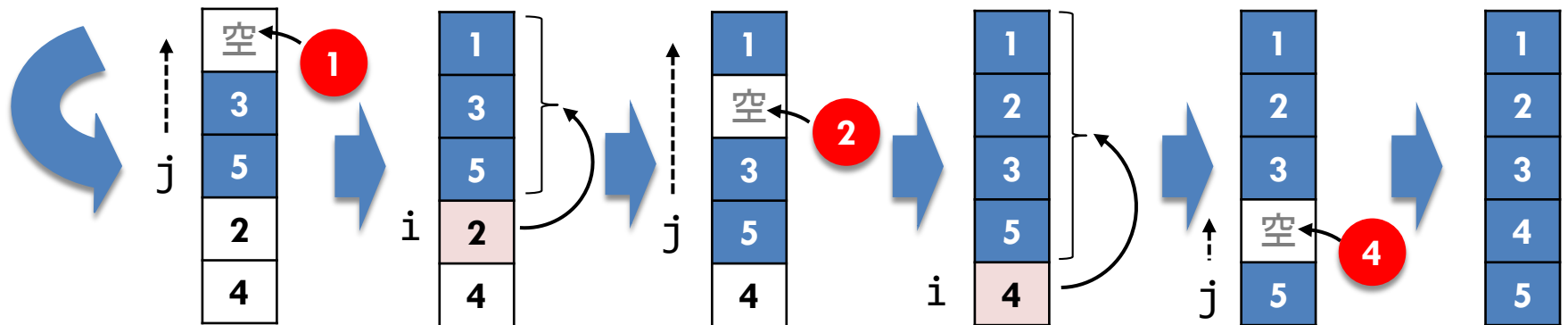
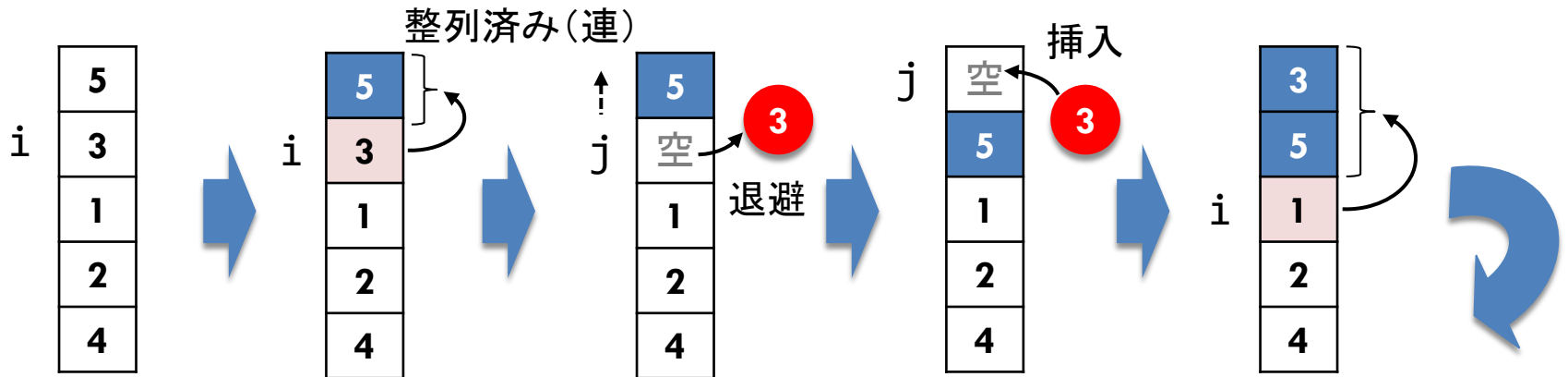
- 列に割り込ませようとしても前後が詰まっている
- 要素の中身を**後ろから順に**ずらして、「空き」を挿入位置まで持ってくる



挿入ソートの例

8

- 未整列の値を整列済み範囲に挿入していく



単純なソートの計算量

9

- 比較と交換
 - ▣ ソートでは, 要素同士の比較回数は, 交換回数よりも多い
 - ▣ 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

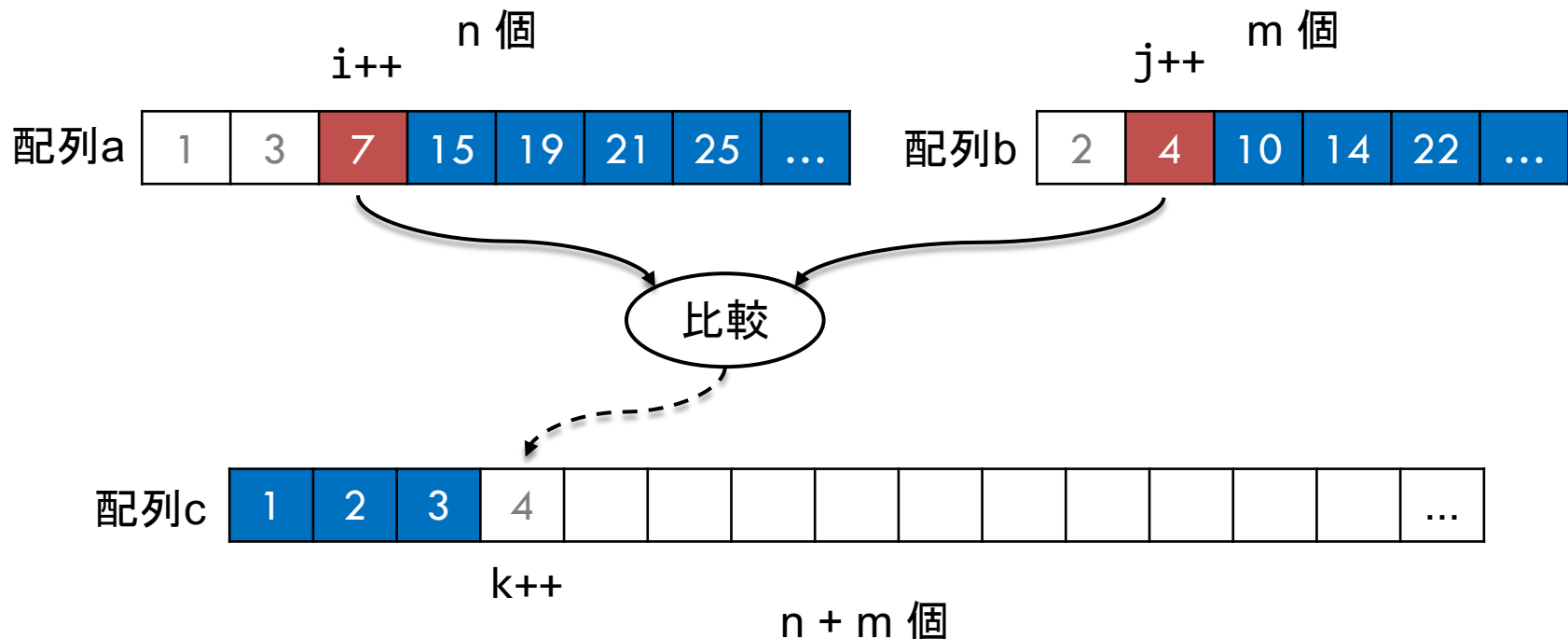
- バブルソートと選択ソート
 - ▣ 2重ループを全部回るので, 比較回数は $n(n-1)/2 \Rightarrow O(n^2)$
 - ▣ その上で, 交換回数が劇的に少ない選択ソートの方が高速

- 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交換せずに完了

配列のマージ

10

- 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

11

- **{ } 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`