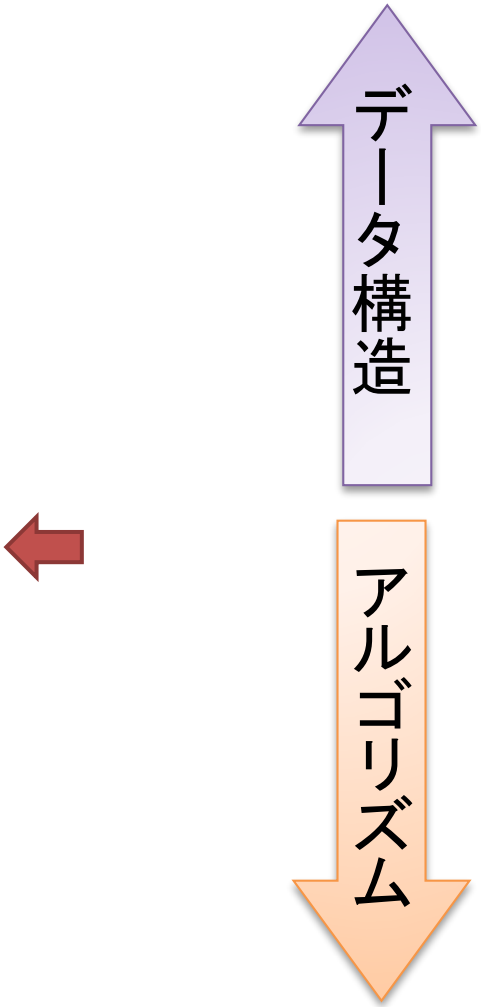
データ構造とアルゴリズム
第8週

掛下　哲郎
kake@is.saga-u.ac.jp

講義スケジュール

週	講義計画
1－2	導入
3	探索問題
4－5	基本的なデータ構造
6	動的探索問題とデータ構造
7	アルゴリズム演習（第1回）
8－9	データの整列
10－11	グラフアルゴリズム
12	文字列照合のアルゴリズム
13	アルゴリズム演習（第2回）
14	アルゴリズムの設計手法
15	計算困難な問題への対応



今日の内容

- データの整列（sorting; ソーティング）
  - データを順番に並べるアルゴリズム
- トピック
  - バブルソート(Bubble Sort)
  - 選択ソート（Selection Sort）
  - 挿入ソート（Insertion Sort）
  - シェルソート（Shell Sort）

データの整列

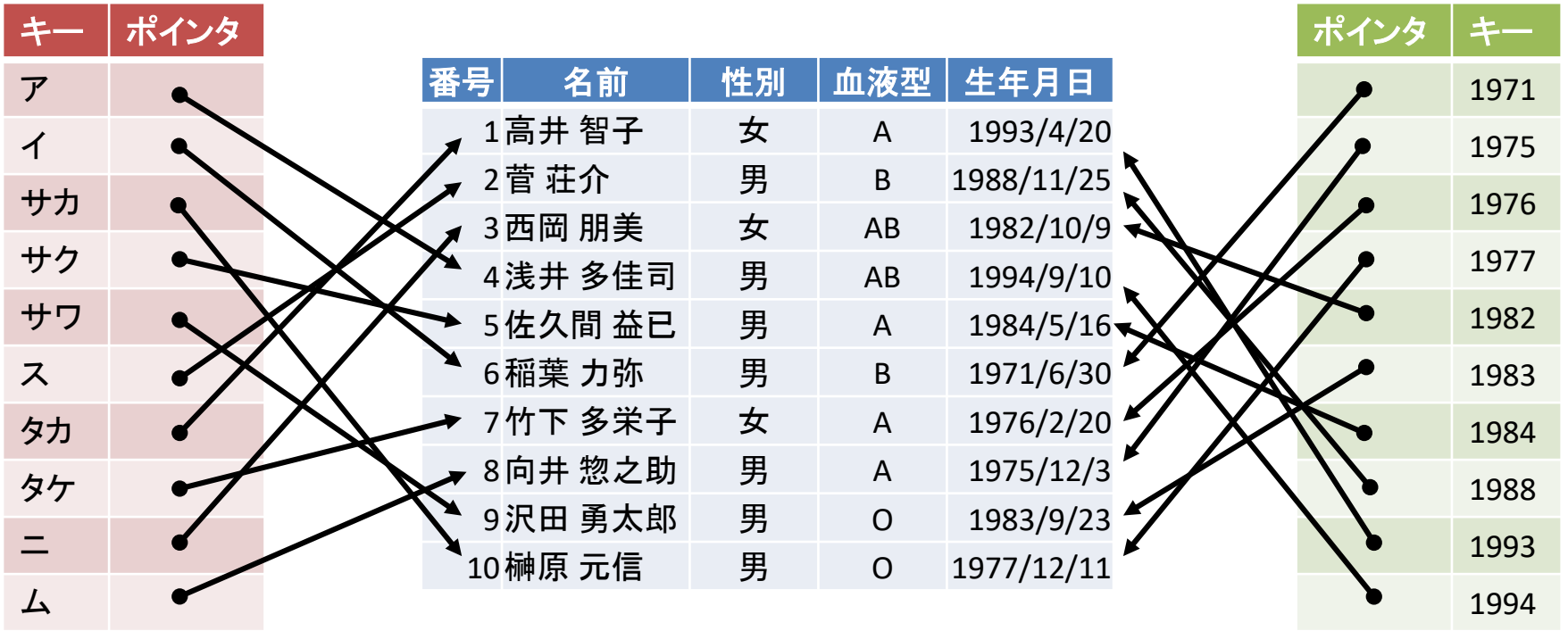
- 整列(sorting；ソーティング)
  n個のデータが与えられたとき，
  それらをある順序に並べ替える処理

※ 以下，整数データを小さい順に並べ替えるとして議論

- もっとも重要で，基本的な問題の一つ
  - データが順番に並ぶ → 2分探索が使える

索引(Index)データ構造

ソートを活用したデータ構造



キー	ポインタ
ア	
イ	
サカ	
サク	
サワ	
ス	
タカ	
タケ	
ニ	
ム	

番号	名前	性別	血液型	生年月日
1	高井 智子	女	A	1993/4/20
2	菅 荘介	男	B	1988/11/25
3	西岡 朋美	女	AB	1982/10/9
4	浅井 多佳司	男	AB	1994/9/10
5	佐久間 益已	男	A	1984/5/16
6	稲葉 力弥	男	B	1971/6/30
7	竹下 多栄子	女	A	1976/2/20
8	向井 惣之助	男	A	1975/12/3
9	沢田 勇太郎	男	O	1983/9/23
10	榊原 元信	男	O	1977/12/11

ポインタ	キー
	1971
	1975
	1976
	1977
	1982
	1983
	1984
	1988
	1993
	1994

氏名の索引

- 同一データを異なる順序で管理できる
- 実データを動かさずにソートできる
- 索引上で二分探索ができる
- 索引だけを用いたデータ処理ができる

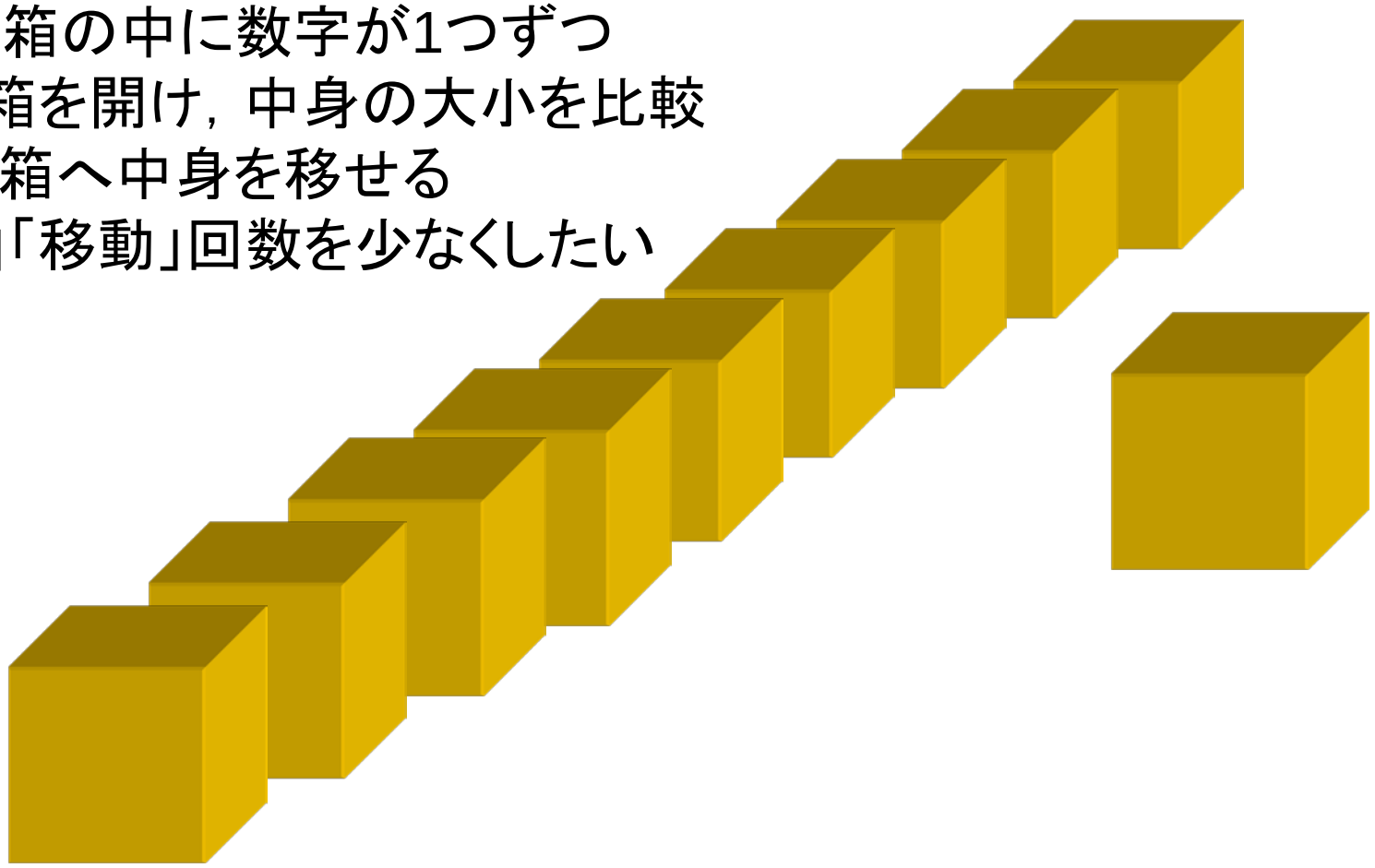
生年月日の索引

要書き込み

どんな整列アルゴリズムを考える？

イメージ

- 並んだ箱の中に数字が1つずつ
- 2つの箱を開け，中身の大小を比較
- 箱から箱へ中身を移せる
- 「比較」「移動」回数を少なくしたい



整列アルゴリズム(その1)

バブルソート(Bubble Sort)

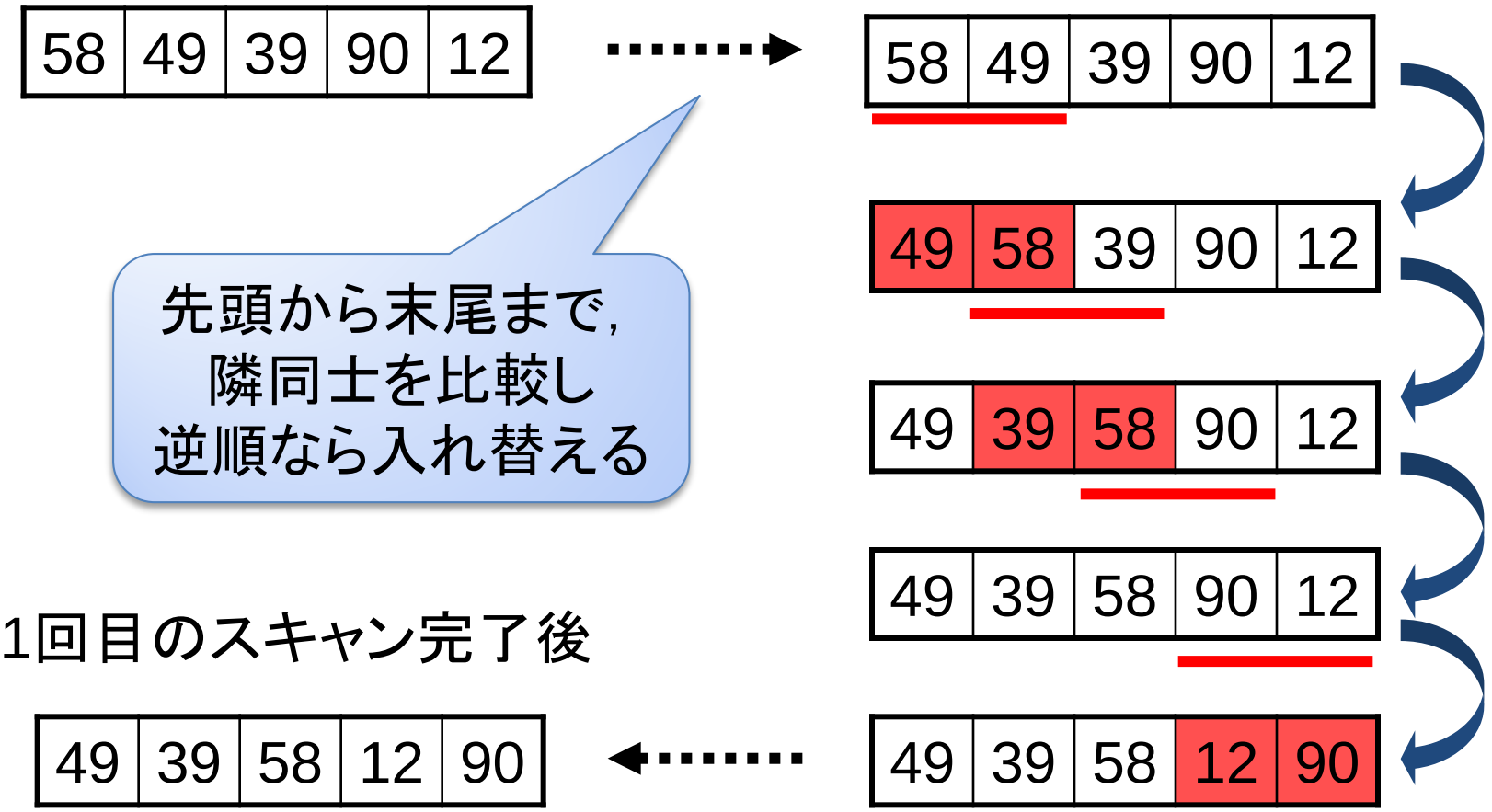
- もっとも単純
  - 比較$O(n^2)$, コピー$O(n^2)$
  - プログラミングが容易
- アイデア
  - 先頭から末尾までチェックしながら
    隣同士を比較し逆順なら入れ替える,
    という操作を何度も繰り返す.

データ数が少ない場合には十分

要書き込み

バブルソートの動作例

ソートしたいデータ. (n=5)



1回目のスキャン完了後

Q

スキャン（左から右までチェックし必要なら入れ替え）を，さらに2回行って下さい

49	39	58	12	90

(2回目スキャン)

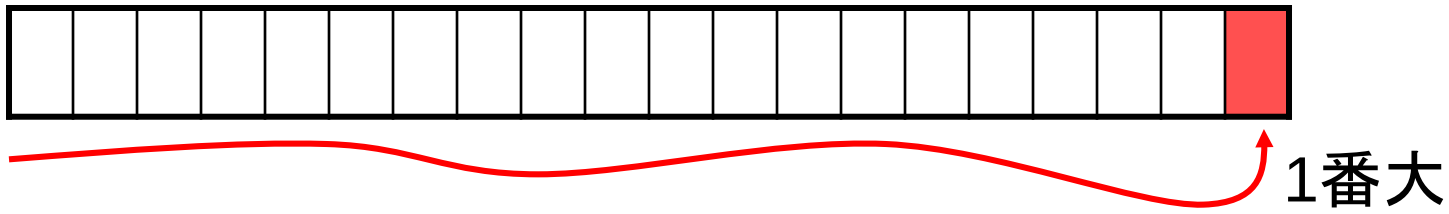
49	39	58	12	90
39	49	58	12	90
39	49	58	12	90
39	49	12	58	90
39	49	12	58	90

(3回目スキャン)

39	49	12	58	90
39	49	12	58	90
39	12	49	58	90
39	12	49	58	90
39	12	49	58	90

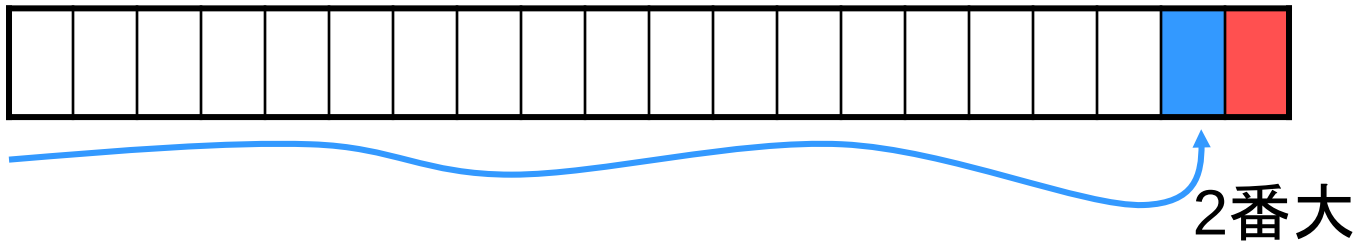
バブルソートの動作

- 1回目のスキャン後



泡（バブル）がプクプクと浮き上がるイメージ

- 2回目のスキャン後



- k回目のスキャン後
  - k番目に大きい値が，右端からk番目の位置へ

バブルソートのアルゴリズム

昇順

1. ソート済み位置を配列の末尾+1とする.
2. ソート済み位置が配列の先頭+1になるまで以下の処理を繰り返す.
    2.1 現在位置を配列の先頭からソート済み位置-2まで変えながら以下の処理を繰り返す.
        2.1.1 現在位置とその1つ先の要素を比較する.
        2.1.2 現在位置の要素が大きいならば, 2つの要素を交換する.
    2.2 ソート済み位置を1つ前の位置にする.

バブルソートの効率

- ステップ2の繰り返し回数
  - ソート済み位置:「配列の末尾+1」⇒「配列の先頭+1」
  - ステップ2を繰り返すたびにソート済み位置は1ずつ減少.

n回

- ステップ2.1の繰り返し回数
  - 現在位置:「配列の先頭」⇒「ソート済み位置-2」
  - ステップ2.1を繰り返すたびに現在位置は1ずつ進む.

ソート済み位置がiの時i-2回

- 比較回数:ステップ2.1.1の実行回数

  $(n-1) + (n-2) + \cdots +1+0 = n(n-1)/2 = O(n^2)$

- コピー回数:ステップ2.1.2の実行回数

  最悪時は $O(n^2)$

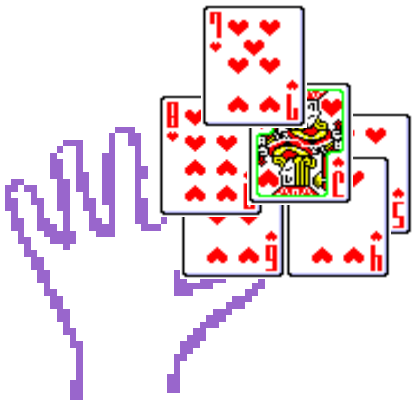
要書き込み

整列アルゴリズム(その2)

選択ソート(Selection Sort)

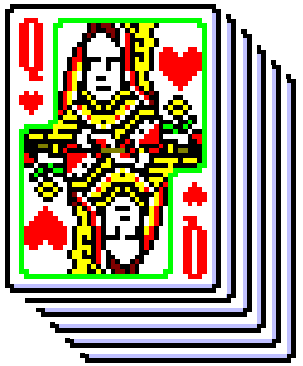
- コピー回数を低減：比較O(n^2)，コピーO(n)
- 各要素のサイズが大きい場合に有利
- アイデア
  - 「ソート済」と「未ソート」の2つの部分に分ける
  - 「未ソート」部分から最小値を見つけ，それを取り出して「ソート済」部分へ追加していく

(未ソート)



小さい順に
1枚ずつ取り出し，
並べていく

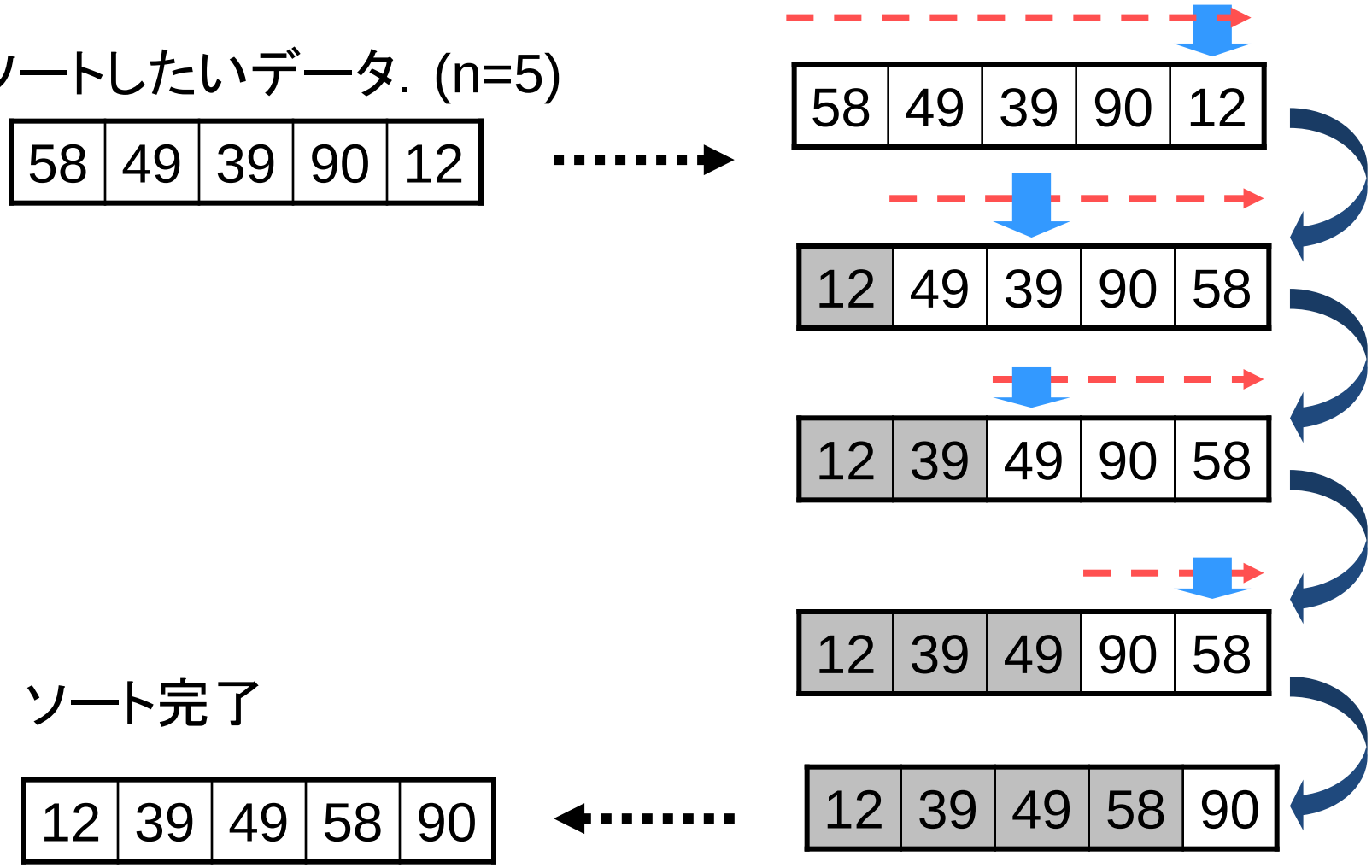
(ソート済)



要書き込み

選択ソートの動作例

ソートしたいデータ. (n=5)



選択ソートのアルゴリズム

昇順

1. ソート済み位置を配列の先頭-1とする.
2. ソート済み位置が配列の末尾-1になるまで以下の処理を繰り返す.
    2.1　ソート済み位置+1の位置に格納されている要素を最小値とし，その値と位置を格納する.
    2.2 現在位置をソート済み位置+2から配列の末尾まで変えながら以下の処理を繰り返す.
        2.2.1 現在位置の要素と最小値を比較する.
        2.2.2 現在位置の要素が小さいならば，その値と位置を最小値と最小値の位置に格納する.
    2.3 最小値の要素とソート済み位置+1の要素を交換する.
    2.4 ソート済み位置を1進める.

選択ソートの効率

- ステップ2の繰り返し回数
  - ソート済み位置:「配列の先頭-1」⇒「配列の末尾-1」
  - ステップ2を繰り返すたびにソート済み位置は1ずつ進む.

n回

- ステップ2.2の繰り返し回数
  - 現在位置:「ソート済み位置+2」⇒「配列の末尾」
  - ステップ2.2を繰り返すたびに現在位置は1ずつ進む.

ソート済み位置がiの時i-2回

- 比較回数:ステップ2.2.1の実行回数

  $(n-1) + (n-2) + \cdots +1+0 = n(n-1)/2 = O(n^2)$

- コピー回数:ステップ2.3の実行回数

  最悪時は $O(n)$

ステップ2の繰り返し回数と一致

要書き込み

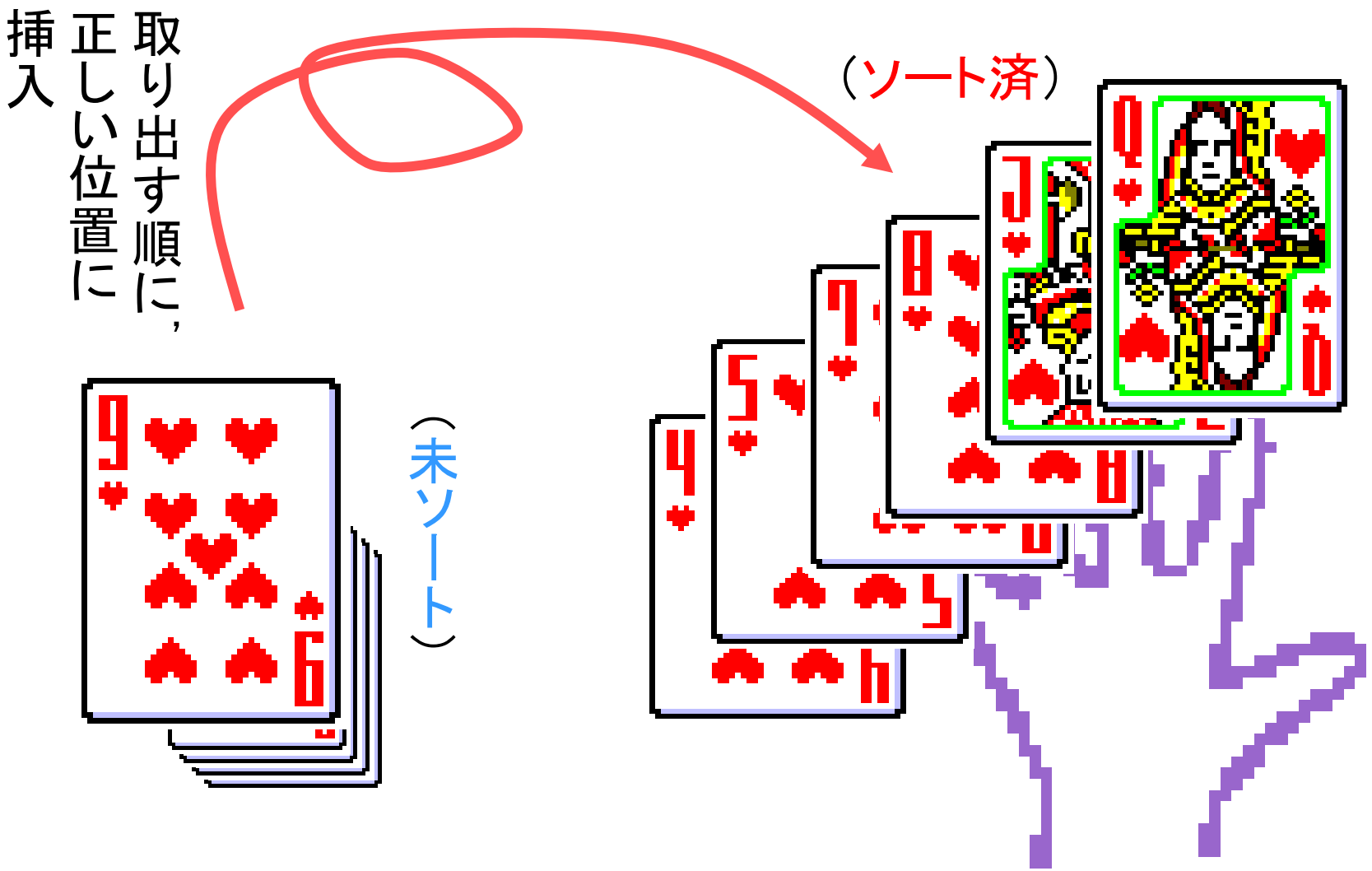
整列アルゴリズム（その3）

挿入ソート(Insertion Sort)

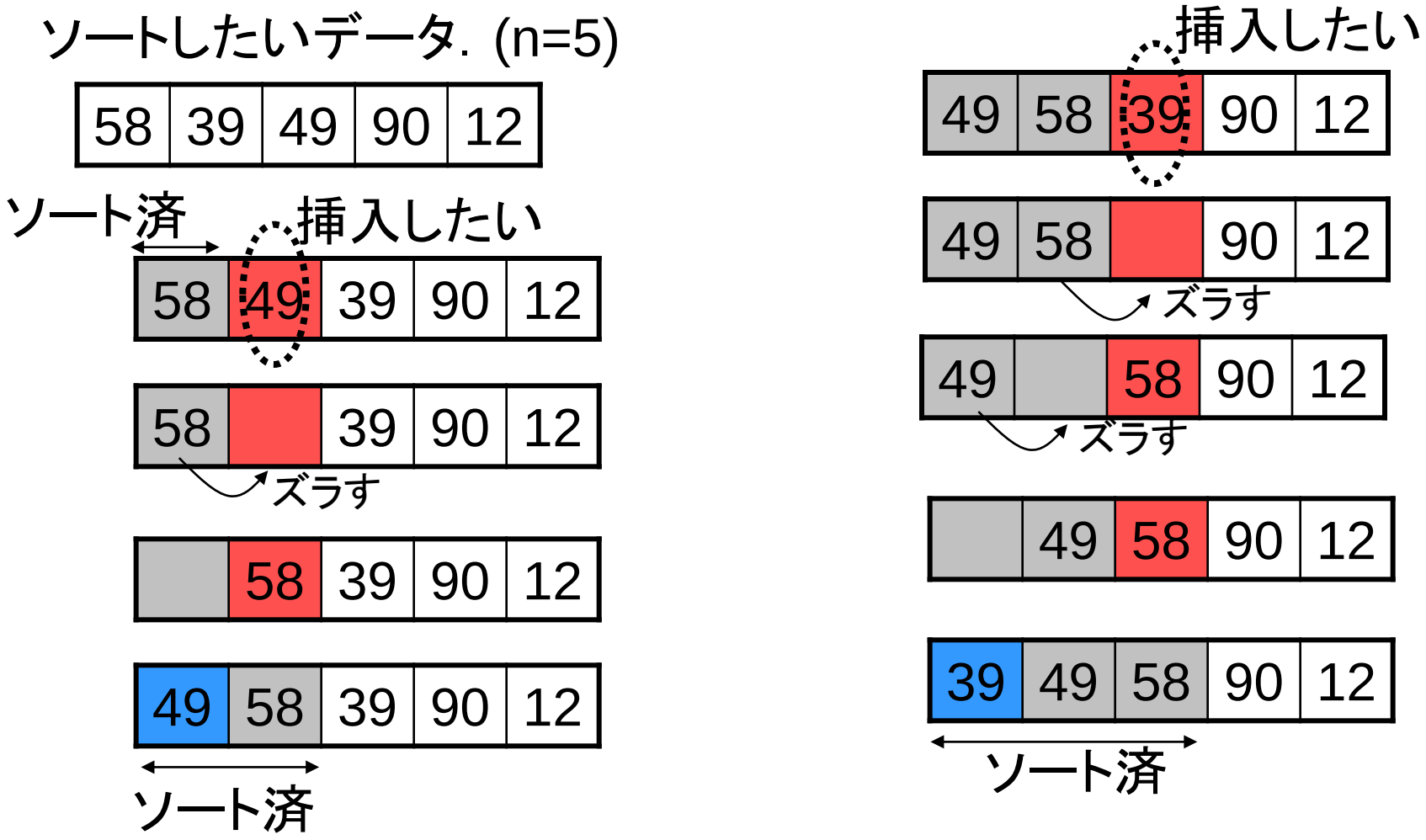
- ほぼ整列済みのデータを整列するのに適している（比較・コピー，ともにO(n)）
  - ただし，最悪時は$O(n^2)$
- アイデア：
  - 「ソート済」と「未ソート」の2つの部分に分ける
  - 「未ソート」部分から「ソート済」部分へ1つずつ挿入していく（※いちいち最大値を求めない点に注意）

要書き込み

挿入ソートのイメージ



挿入ソートの動作例


ソートしたいデータ. (n=5)
58 39 49 90 12
ソート済
挿入したい
58 49 39 90 12
58 39 90 12
ズラす
58 39 90 12
49 58 39 90 12
ソート済
挿入したい
49 58 39 90 12
49 58 90 12
ズラす
49 58 90 12
ズラす
49 58 90 12
39 49 58 90 12
ソート済

挿入ソートのアルゴリズム

昇順

1. ソート済み位置を配列の先頭とする.
2. ソート済み位置が配列の末尾-1になるまで以下の処理を繰り返す.
   2.1 ソート済み位置+1の位置に格納されている要素を挿入要素とする.
   2.2 ソート済み位置+1を挿入位置とする.
   2.3 現在位置をソート済み位置とする.
   2.4 現在位置の要素が挿入要素よりも大きい限り以下の処理を実行する.
      2.4.1 当該要素を1つ後の場所に格納する.
      2.4.2 挿入位置を現在位置とする.
      2.4.3 現在位置を配列の先頭方向に1つ戻す.
   2.5 挿入要素を挿入位置に格納する.
   2.6 ソート済み位置を1つ進める.

挿入ソートの効率

- ステップ2の繰り返し回数
  - ソート済み位置:「配列の先頭」⇒「配列の末尾-1」
  - ステップ2を繰り返すたびにソート済み位置は1ずつ進む.

n-1回

- ステップ2.4の繰り返し回数
  - 現在位置:「ソート済み位置」⇒「配列の先頭」
  - ステップ2.4を繰り返すたびに現在位置は1ずつ減少.

ソート済み位置がiの時 i回

- 比較回数:ステップ2.4の繰り返し回数

  最悪時:$1 + 2 + \cdots + (n-2) + (n-1) = n(n-1)/2 = O(n^2)$

  最良時:$1 + 1 + \cdots + 1 = n-1 = O(n)$

- コピー回数:ステップ2.3の実行回数

  最悪時: $O(n^2)$

  最良時:$1 + 1 + \cdots + 1 = n-1 = O(n)$

ほぼソート済みのデータに対してはO(n)で動作

要書き込み

整列アルゴリズム(その4)

シェルソート(Shell Sort)

- 挿入ソートの問題点
  - コピー回数が$O(n^2)$と多い.
- アイデア
  - 飛び飛びの要素に対して挿入ソートを行う.
  - 例:ギャップh=4の場合
    - 項目0, 4, 8, 12, … のソート
    - 項目1, 5, 9, 13, … のソート
    - 項目2, 6, 10, 14, … のソート
    - 項目3, 7, 11, 15, … のソート

挿入ソートを4回繰り返す

  - これを通じて配列全体が大まかにソートされる.
  - ギャップの間隔hを徐々に縮める.(Knuthの方法)
  - 最後に通常の挿入ソートを実行する. ⇒ O(n)に近づく

要書き込み

Knuthの方法

- $h_k = 3*h_{k-1}+1, h_0 = 1$

k	0	1	2	3	4	5	6	7	8
h	1	4	13	40	121	364	1,093	3,280	9,841

- 最初の h_k の値は $h_k < n \leqq h_{k+1}$ を満足するように選択
  - 例：n=1000個のデータをソートする場合
  - 最初はギャップ 364 でソート（364回繰り返し）
  - 2回目はギャップ 121 でソート（121回繰り返し）
  - 3回目はギャップ 40 でソート（40回繰り返し）
    :
  - 最後はギャップ 1 でソート（通常の挿入ソート）

シェルソートのアルゴリズム

1. ギャップの間隔 h を 1 とする.
2. 3h ≦ nである限り 3h+1 で h を置き換える.
3. h ＞ 0 である限り以下の処理を繰り返す.

   3.1 上限を h から n まで1ずつ増やしながら以下の処理を繰り返す.

      3.1.1 上限の位置にある要素を挿入要素とする.

      3.1.2 挿入要素を上限，上限-h，上限-2h，…，上限-k･hの適切な場所に挿入する.

   3.2 (h-1)/3 で h を置き換える.

挿入ソート･ステップ2.3～2.5と同様の手順

シェルソートの特徴

- 実用性が高い
  - 挿入ソートよりも高速
  - クイックソートよりも単純
- 一般的な性能解析は困難
  - ギャップhの決定法やデータの初期状態に依存
    - 教科書の方法，Knuthの方法など
  - 実験に基づく推測値：$O(n^{3/2})$～$O(n^{7/6})$
    - さまざまな評価結果がある．

まとめ

	比較回数		コピー回数		余分な作業領域
	最良ケース	最悪ケース	最良ケース	最悪ケース	
バブルソート	$O(n^2)$	$O(n^2)$	$O(n)$	$O(n^2)$	不要
選択ソート	$O(n^2)$	$O(n^2)$	$O(n)$	$O(n)$	不要
挿入ソート	$O(n)$	$O(n^2)$	$O(n)$	$O(n^2)$	不要
シェルソート	$O(n^{3/2})$ ～$O(n^{7/6})$		$O(n^{3/2})$ ～$O(n^{7/6})$		不要
ヒープソート	$O(n \log n)$		$O(n \log n)$		必要（ヒープ）

ソートアルゴリズムの特徴

ソートアルゴリズム	特徴
バブルソート	• アルゴリズムが最も単純で実現が容易 • 比較回数，コピー回数が$O(n^2)$必要．
選択ソート	• アルゴリズムが比較的単純 • コピー回数が$O(n)$で済むため，各要素のサイズが大きい場合に効率が良い．
挿入ソート	• データの交換（代入3回）が不要．移動（代入1回）で済む． • 平均比較回数がバブルソート等の半分で済む． • データがほぼソートされている場合には，ほぼ$O(n)$時間で実行できる．
シェルソート	• 高々$O(n^{3/2})$時間でソートできる． • 多くの場合，挿入ソートより高速．クイックソートより単純．
ヒープソート	• 最悪の場合でも$O(n \log n)$時間でソートできる． • ヒープを保存するために，$O(n)$の余分な作業領域が必要

確認テスト（第8回）

- 以下のデータ列を、各種ソーティングアルゴリズムでソートした場合の変更過程を示せ。

7	3	6	1	2	5	4	8

  - バブルソート
  - 選択ソート
  - 挿入ソート
  - シェルソート