大阪電気通信大学 情報通信工学部 光システム工学科 2年次配当科目

コンピュータアルゴリズム

整列アルゴリズム (1)

第4講: 平成20年10月24日 (金) 4限 E252教室

中村 嘉隆(なかむら よしたか)
奈良先端科学技術大学院大学 助教
y-nakamr@is.naist.jp
http://narayama.naist.jp/~y-nakamr/

第 1 講の復習

- アルゴリズムの定義
  - 入力と出力
  - 正当性, 決定性, 有限性, 停止性
- ユークリッドの互除法
- フローチャートの描き方
- 擬似言語の書き方

第 2 講の復習

- アルゴリズムの評価は時間計算量で行う
  - 領域計算量もある
- 計算量の評価にはオーダー記法を使う
  - 並んでいる計算量は足し算
  - 繰り返しに含まれる計算量は掛け算
  - 係数は省略する
- 多項式オーダーと指数オーダー
  - 指数オーダーのアルゴリズムは使い物にならない
- 再帰プログラム

第 3 講の復習

- アルゴリズムとデータ構造
  - アルゴリズムに適したデータ構造の選択が必要
- 基本的なデータ構造(抽象データ型)
  - 配列
    - 参照は $O(1)$, 挿入・削除は $O(n)$
  - リスト
    - 参照は $O(n)$, 挿入・削除は $O(1)$
  - キュー (待ち行列)
    - 先入れ先出し, ランダム参照不可, 追加・取出しは $O(1)$
  - スタック
    - 後入れ先出し, ランダム参照不可, 追加・取出しは $O(1)$
  - 木
    - 根(root), 親, 子, 葉(終端頂点), 非終端頂点, 高さ, 深さ

今後の方針

- 準備はそろそろ終わって, 実際のアルゴリズムを見ていこう
  - 整列アルゴリズム
    - 数値の列の並べ替え
    - $O(n^2)$ のアルゴリズムから $O(n\log n)$ のアルゴリズムまで
      - 特定の場合には $O(n)$ になるアルゴリズムも
  - 探索アルゴリズム
    - 数値の列から目的の数値の位置を見つける
    - $O(n)$ のアルゴリズムから $O(\log n)$ のアルゴリズム
      - 特定の場合には $O(1)$ になるアルゴリズムも

今日の講義内容

- 整列アルゴリズム
  - 整列: 並べ替え, ソーティングともいう
- 今日紹介する整列アルゴリズム
  - 選択ソート
  - バブルソート
  - 挿入法
  - マージソート(次週)
  - クイックソート(次週)

整列（ソーティング）問題とは

- ソーティング：Sorting，整列，並べ替え
  - n 個のデータ列をある基準に従って順に並べ替える処理
- 昇順ソート（Ascending Sort）
  - 単調増加に整列（小さいもの順に整列）
  - 一般的にソートといえばこちらを指す
- 降順ソート（Descending Sort）
  - 単調減少に整列（大きいもの順に整列）
- 昇順と降順は比較に用いる不等号を逆にする
- ソーティングにおける時間計算量は比較の回数を基準として考える
  - if 文を用いた大小比較

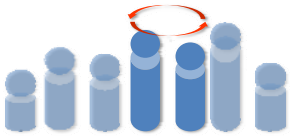
整列問題の分類

- 安定性
  - 安定ソート
    - ソートの際，同等なデータには元の並び順が保存されているソート法
      例）元々学籍番号順に並んでいたデータをその成績順にソートしたとき，同じ点数の生徒は学籍番号順に並んでいるようなソート法
- 記憶領域
  - 外部ソート
    - ソートの際，定数個より多くの外部記憶領域を必要とするソート法
      例）現在操作中の配列の他に，その長さに応じた別のデータ格納用の配列が必要なソート法
  - 内部ソート
    - ソートの際，定数個の記憶領域で十分なソート法
      例）現在操作中の配列の内部の入れ替えだけで十分なソート法

準備

- 入力は長さ n の数値の列
  - $\{a1, a2, a3, a4, \ldots, an\}$ で表す
- 数値の大小関係には推移律が成り立つ
  - $a < b$ で $b < c$ なら $a < c$
- Swap 手続き
  - 配列中の 2 つの要素の値を入れ替える手続き
  - 実際には以下のようにテンポラリ（一時的）の変数を準備して入れ替えする

```
swap (a, b) {
    temp ← a ;
    a ← b ;
    b ← temp ;
}
```

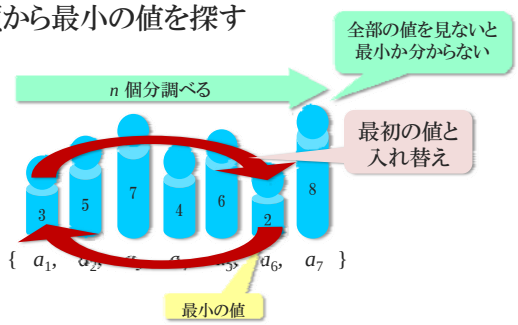


選択ソート：概要

- 最小値選択法：Selection Sort
- 直接選択法：Straight Selection
- アルゴリズム
  1. 未整列部分から最小値を選択
  2. 未整列部分の先頭に置く
  3. 以上を未整列部分がなくなるまで繰り返す

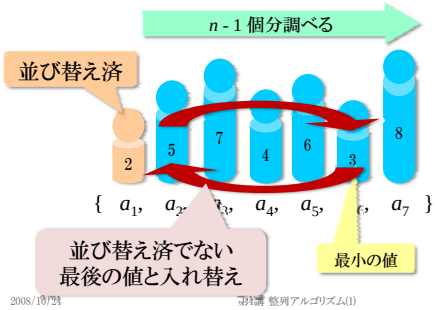
選択ソート：概念図

- 先頭から最小の値を探す



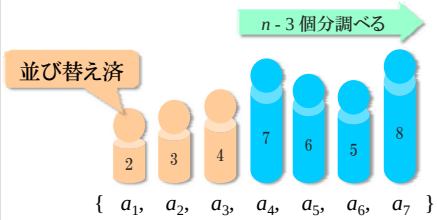
選択ソート：概念図

- 並び替え済みでないもので繰り返す



選択ソート：概念図

- 並び替え済みでないもので繰り返す……



選択ソート：プログラム

- 最小の要素を示す添え字 min を使う

```
for ( i ← 1; i < n; i ← i + 1 ) {
    min ← i;
    for ( j ← i + 1; j <= n; j ← j + 1 ) {
        if ( a_j < a_min ) {
            min ← j;
        }
    }
    swap( a_i, a_min ) ;
}
```

ちなみに配列の表記を使って a_i を a[i] と書いても良い

選択ソート：計算量

- 計算量：$O(n^2)$
  - 詳しく見てみると……
    - 第 i 回目の繰り返しでは $n - i$ 回の比較が必要
    - 全体の比較回数は

$$\sum_{i=1}^{n-1}(n-i)=\sum_{i=1}^{n-1}n-\sum_{i=1}^{n-1}i=(n-1)n-\frac{(n-1)n}{2}=\frac{n(n-1)}{2}$$

    - 各繰り返しでは最小値だけ求めており，その他の値の大小関係の比較結果が次の繰り返しに反映されていない

選択ソート：演習

- 途中経過と結果を書く

最小値

入力列	4	6	5	2	8	7	1
1回目後	1	6	5	2	8	7	4
2回目後	1	2	5	6	8	7	4
3回目後	1	2	4	6	8	7	5
4回目後	1	2	4	5	8	7	6
5回目後	1	2	4	5	6	7	8
最終結果	1	2	4	5	6	7	8

並べ替え済み

選択ソートのまとめ

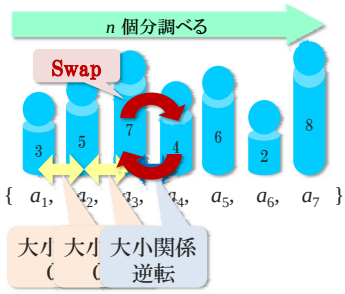
- 計算量 $O(n^2)$
  - 最良も最悪も $O(n^2)$
- 安定ソートではない
  - 最小値と先頭値を交換するので元の順番が崩れる
- 内部ソート
  - 最初の配列以外の記憶領域を使わない
- 最悪計算時間が $O(n^2)$ と遅いが，アルゴリズムが単純で実装が容易なため，しばしば用いられる

バブルソート：概要

- バブルソート：Bubble Sort
- アルゴリズム
  1. 隣接する 2 要素を比較
  2. 右側（要素番号の大きい側）の値が大きくなるように，比較結果によって値を交換
  3. 以上の操作をデータ列の左側（要素番号の小さい側）から右側へ向けて繰り返す

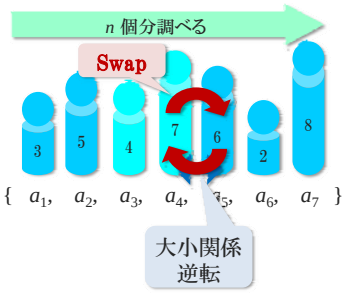
バブルソート: 概念図

- 先頭から隣り合う 2 つずつを比べていく



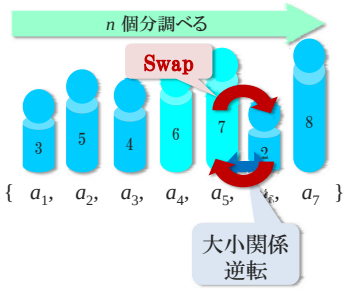
バブルソート: 概念図

- 先頭から隣り合う 2 つずつを比べていく



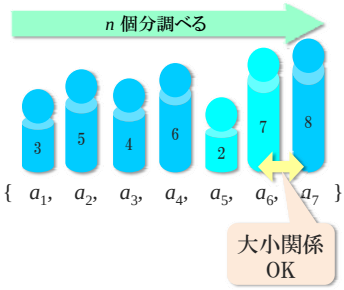
バブルソート: 概念図

- 先頭から隣り合う 2 つずつを比べていく



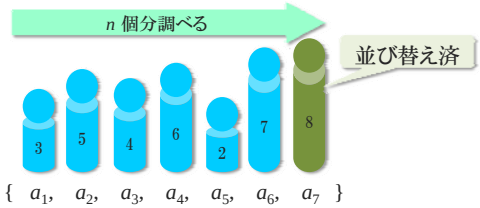
バブルソート: 概念図

- 先頭から隣り合う 2 つずつを比べていく



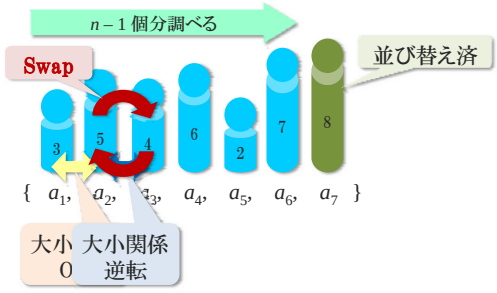
バブルソート: 概念図

- 先頭から隣り合う 2 つずつを比べていく



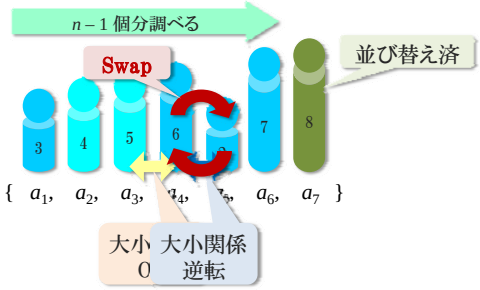
バブルソート: 概念図

- 並べ替え済みでない部分で繰り返す



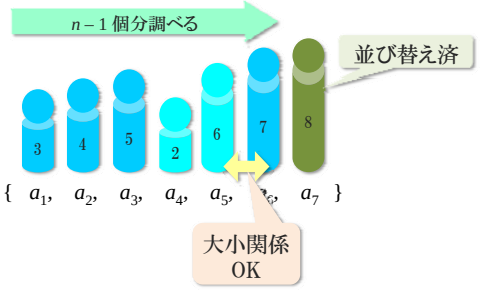
バブルソート: 概念図

- 並べ替え済みでない部分で繰り返す



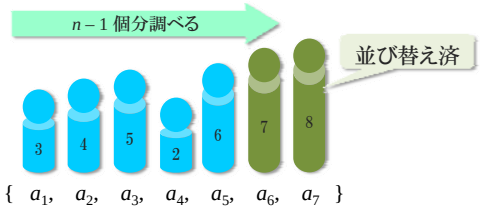
バブルソート: 概念図

- 並べ替え済みでない部分で繰り返す



バブルソート: 概念図

- 並べ替え済みでない部分で繰り返す



バブルソート: プログラム

- 隣り合う要素を大小比較

```
for ( i ← n; i > 2; i ← i - 1 ) {
    for ( j ← 1; j < i; j ← j + 1 ) {
        if ( a_j > a_{j+1} ) {
            swap( a_j, a_{j+1} ) ;
        }
    }
}
```


バブルソート: 計算量

- 計算量: $O(n^2)$
  - 選択ソートと同じだけ比較を行う
  - Swap 回数が多いバブルソートの方が遅い
  - ちなみに最大値がアブクのように沸き上がってくるので**バブル**ソートと呼ばれる
  - 繰り返しの中で一度も Swap が発生しなければ，そこでソートを完了してもよい

バブルソート: 演習

- 途中経過と結果を書く

小さいものが 1 つ前に出る
（結果的に大きいものが後ろへ行く）

入力列	4	6	5	2	8	7	1
1回目後	4	5	2	6	7	1	8
2回目後	4	2	5	6	1	7	8
3回目後	2	4	5	1	6	7	8
4回目後	2	4	1	5	6	7	8
5回目後	2	1	4	5	6	7	8
最終結果	1	2	4	5	6	7	8

並べ替え済み

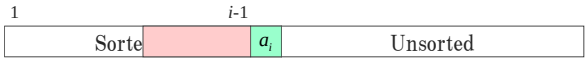
バブルソートのまとめ

- 計算量 $O(n^2)$
  - 最良は入力列がソートされているときで，1 回目の繰り返しで打ち切れば $O(n)$
  - 平均・最悪はもちろん $O(n^2)$
- 安定ソート
  - データの値が同じ場合，元の順番が保存される
- 内部ソート
  - 最初の配列以外の記憶領域を使わない

- 選択ソートと同じく最悪計算時間が $O(n^2)$ と遅いが，アルゴリズムが単純で実装が容易な上，安定ソートであるのでしばしば用いられる

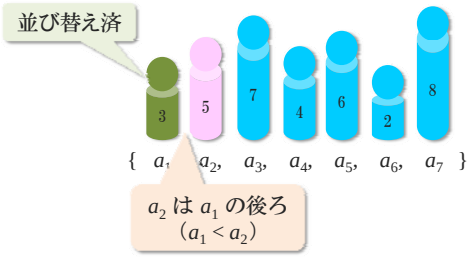
挿入法：概要

- 挿入法：Insertion Sort
- アルゴリズム
  - a_1 から a_{i-1} までがソート済みと仮定
  - a_i をしかるべき位置に挿入
    - 先頭から探す
  - 以上を i を増加させつつ繰り返す



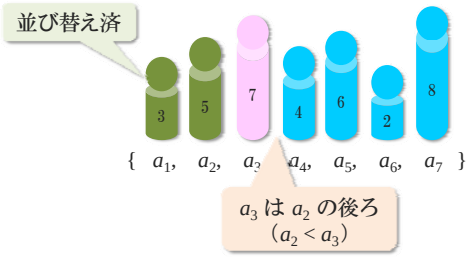
挿入法：概念図

- a_1 はソート済みとする（当たり前）
- a_2 が a_1 のどこに入るか調べる



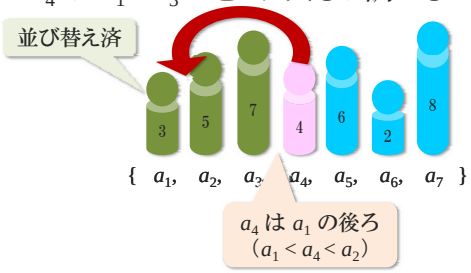
挿入法：概念図

- $a_1 \sim a_2$ はソート済み
- a_3 が $a_1 \sim a_2$ のどこに入るか調べる



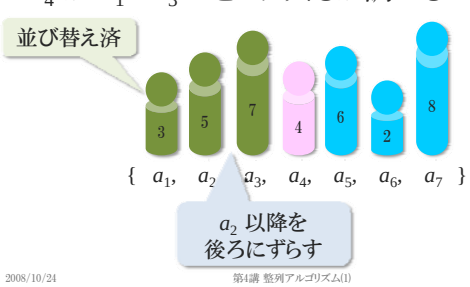
挿入法：概念図

- $a_1 \sim a_3$ はソート済み
- a_4 が $a_1 \sim a_3$ のどこに入るか調べる



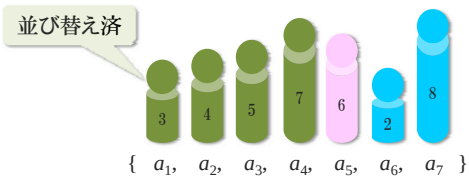
挿入法：概念図

- $a_1 \sim a_3$ はソート済み
- a_4 が $a_1 \sim a_3$ のどこに入るか調べる



挿入法：概念図

- a_1～a_4 はソート済み
- a_5 が a_1～a_4 のどこに入るか調べる……



挿入法：プログラム

- ソート済みの部分の後ろから順にずらしていく

```
for ( i ← 2; i <= n; i ← i + 1 ) {
    w ← a_i;
    j ← i;
    while ( a_{j-1} < w and j > 1 ) {
        a_j ← a_{j-1};
        j ← j − 1;
    }
    a_j ← w;
}
```


挿入法：計算量

- 計算量：$O(n^2)$
  - 最悪の場合，ソート済みの全ての要素を 1 つずつずらさないといけない
  - 最悪の入力は，逆順にソートされている入力
  - 最良の入力は，ソートされている入力で $O(n)$
  - 平均は？？

挿入法：平均計算量

- プログラム

```
for ( i ← 2; i <= n; i ← i + 1 ) {
    w ← a_i;                          O(1)
    j ← i;
    while ( a_{j-1} < w and j > 1 ) {
        a_j ← a_{j-1};                ここは平均どれだけ回る？？
        j ← j − 1;
    }
    a_j ← w;                          O(1)
}
```

中の `while` ループは平均 $j/2$ 回 $= i/2$ 回 $= (n/2)/2 = n/4$ 回 $= O(n)$

全体では $O(n) \times \{O(1) + O(n) + O(1)\} = O(n) \times O(n) = O(n^2)$

挿入法：演習

- 途中経過と結果を書く

入力列	4	(6)	5	2	8	7	1
1回目後	4	6	(5)	2	8	7	1
2回目後	4	5	6	(2)	8	7	1
3回目後	2	4	5	6	(8)	7	1
4回目後	2	4	5	6	8	(7)	1
5回目後	2	4	5	6	7	8	(1)
最終結果	1	2	4	5	6	7	8

並べ替え済み　　( ) が入る位置

挿入法のまとめ

- 計算量 $O(n^2)$
  - 最良は入力列がソートされているときで $O(n)$
  - 最悪は入力例が逆順にソートされているときで $O(n^2)$
  - 平均も $O(n^2)$
- 安定ソート
  - データの値が同じ場合，元の順番が保存される
- 内部ソート
  - 最初の配列以外の記憶領域を使わない
- 前述のソートと同じく最悪計算時間が $O(n^2)$ と遅いが，アルゴリズムが単純で実装が容易な上，安定ソートであるのでしばしば用いられる

第 4 講のまとめ

- 整列アルゴリズム
  - ソーティング，並べ替え
  - $O(n^2)$ のアルゴリズム
    - 選択ソート
    - バブルソート
    - 挿入法
- 次回は $O(n \log n)$ の整列アルゴリズムを紹介