算法数理工学
第1回

定兼　邦彦

目的・成績評価・参考書

- アルゴリズム理論による設計は，大規模なデータを高速で処理する際に特に有効
- 部品 (データ構造) を組み合わせてプログラムを作るだけでなく，部品の中身を理解する

- 成績評価は試験とレポート
- メール： sada@mist.i.u-tokyo.ac.jp
- ホームページ： http://researchmap.jp/sada/
- 居室: 工学部6号館341

- 参考書：アルゴリズムイントロダクション(近代科学社)など
プログラミング問題集 AIZU ONLINE JUDGE
問題セットの Introduction to Algorithms and Data Structures
http://judge.u-aizu.ac.jp/onlinejudge/finder.jsp?course=ALDS1

アルゴリズムの概念
～ オーダーと計算量 ～

- 算法 = アルゴリズム (algorithm)
- アルゴリズムの概念
- オーダーの定義と計算量

アルゴリズム

- アルゴリズムとは
  - 入力 (input): ある値(の集合)
  - 出力(output): ある値(の集合)
  - 明確に定義された計算手続き
- 明確に定義された計算問題を解くための道具
- 問題の記述とは
  - 望むべき入出力関係を指定すること

ソーティング問題の形式的定義

- 入力: n 個の数の列 $\langle a_1, a_2, ..., a_n \rangle$
- 出力: $a'_1 \le a'_2 \le \dots \le a'_n$ であるような入力列の置換 $\langle a'_1, a'_2, ..., a'_n \rangle$

- 入力例 (具体例, instance)
  - 入力 ⟨31, 41, 59, 26, 41,58⟩
  - 出力 ⟨26, 31, 41, 41, 58, 59⟩

ソーティング

- 計算機科学における基本的な操作
- 多くのアルゴリズムが開発されている
- 入力例によって優劣が異なる
  - ソートすべきデータの数
  - どの程度までデータがすでにソートされているか
  - 用いる記憶装置の種類 (主記憶, ディスク, テープ)

アルゴリズムの正しさ

- アルゴリズムが正しい (correct)
  ⇒全ての具体例に対して正しい出力とともに停止する
  – 与えられた計算問題を解く (solve) という.
- 正しくないアルゴリズム
  – ある具体例に対して望ましい答えを出力せずに停止
  – ある具体例に対して全く停止しない
- 確率的アルゴリズムも存在
  – モンテカルロ法 (高い確率で正しい答えを出す)
    - 円周率の計算, 素数判定
  – ラスベガス法 (高い確率で早く停止する)
    - ランダムクイックソート

挿入ソート

- 入力: 長さ n の配列 A[0..n-1]
- 出力: ソートされた配列 A[0..n-1]

```
void INSERTION-SORT(data *A, int n)
{
    data key;
    int i, j;
    for (j=1; j < n; j++) {
        key = A[j];
        //   A[j] をソート列 A[0..j-1] に挿入する
        i = j – 1;
        while (i >= 0 && A[i] > key) {
                A[i+1] = A[i];
                i = i -1;
        }
        A[i+1] = key;
    }
}
```

5 2 4 6 1 3 入力

2 5 4 6 1 3

2 4 5 6 1 3

2 4 5 6 1 3

1 2 4 5 6 3

1 2 3 4 5 6 出力

C言語の場合

```
#include <stdio.h>
#include <stdlib.h>

typedef int data;


void INSERTION_SORT(data *A, int n)
{
 data key;
 int i, j;
 for (j=1; j < n; j++) {
  key = A[j];
  //  A[j] をソート列 A[0..j-1] に挿入する
  i = j - 1;
  while (i >= 0 && A[i] > key) {
   A[i+1] = A[i];
   i = i - 1;
  }
  A[i+1] = key;
 }
}
```

```
int main(int argc, char *argv[])
{
 data A[14] =
{27,17,3,16,13,10,1,5,7,12,4,8,9,0};
 int i,n;

 n = 14;
 INSERTION_SORT(A,n);
 for (i=0;i<n;i++) printf("%d ",A[i]);
 printf("¥n");

}
```

アルゴリズムの解析

- アルゴリズムの実行に必要な資源を予測したい
  - メモリ量
  - 通信バンド幅, 論理ゲート
  - 計算時間
- 解析を行うにはモデルを仮定する必要がある
- RAM (random access machine) モデル
  - 命令は1つずつ実行
  - どのメモリ番地も一定の時間で読み書き可

実行時間の解析

- 実行時間はアルゴリズムと入力に依存する.
- アルゴリズム A に入力 I を与えた時の実行時間を $T(A, I)$ と表すとする.
  - A: 挿入ソート
  - I: 数列 $\langle a_1, a_2, ..., a_n \rangle$
- 入力サイズの定義
  - ソーティング, 離散フーリエ変換など: データ数
  - 整数の積の計算など: 入力のビット数
  - グラフの問題: グラフの頂点と辺の数

実行時間の定義

- 実行される基本的な演算の回数
- プログラムの第 i 行の実行に c_i 時間かかるとする (c_i は定数)
- 注: サブルーチン呼び出しは定数時間ではない

	コスト	回数
void INSERTION-SORT(data *A, int n)		
{		
data key;		
int i, j;		
for (j=2; j <= n; j++) {	c1	n
key = A[j];	c2	n-1
// A[j] をソート列 A[1..j-1] に挿入する		
i = j – 1;	c4	n-1
while (i > 0 && A[i] > key) {	c5	$\sum_{j=2}^{n} t_j$
A[i+1] = A[i];	c6	$\sum_{j=2}^{n} (t_j - 1)$
i = i -1;	c7	$\sum_{j=2}^{n} (t_j - 1)$
}		
A[i+1] = key;	c8	n-1
}		
}		

t_j: while ループが j の値に対して実行される回数

入力 I から決まる

INSERTION-SORTの実行時間

$$T(A,I) = c_1 n + c_2(n-1) + c_4(n-1)$$

$$+ c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

t_j の値は入力によって変化する.
最良の場合 = 配列が全てソートされている場合 (I_1 とする)
$t_j = 1$

$$T(A,I_1) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$

$$= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

$$= an + b$$

n の線形関数 (linear function)

最悪の場合

- 配列が逆順にソートされている場合(I_2 とする)
  - $t_j = j$

$$T(A, I_2) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right)$$

$$+ c_6\left(\frac{n(n-1)}{2} - 1\right) + c_7\left(\frac{n(n-1)}{2} - 1\right) + c_8(n-1)$$

$$= an^2 + bn + c$$

n の2次関数 (quadratic function)

時間計算量 (Time Complexity)

- アルゴリズムの実行時間は入力に依存する
- 実行時間を簡単に見積もりたい
  - 入力サイズ n の関数にしたい
  - $T(A, I) = T'(A, n) \quad (n = |I|)$
- 主に2つのやり方がある
  - 最悪時間解析
  - 平均時間解析

最悪時間解析

- アルゴリズム A の入力として，サイズ n の全ての入力を考える
  - 集合 I_n とする
- 全ての入力に対する最悪実行時間は n の関数になる

$$T_{\mathrm{worst}}(A,n)=\max_{I\in I_n}\{T(A,I)\}$$

- 最悪時の実行時間を保証できる

平均時間解析

- アルゴリズム A の入力として，サイズ n の全ての入力が等確率で現れるとして，実行時間の平均を求める
- やはり n の関数になる

$$T_{\mathrm{average}}(A,n)=\mathrm{E}\left[T(A,I)\right] \qquad (I\text{: 確率変数})$$

$$=\sum_{I\in I_n}\Pr\left[I\right]\cdot T(A,I)=\sum_{I\in I_n}\frac{T(A,I)}{\left|I_n\right|}$$

増加のオーダ

- 実行時間の解析を容易にするための抽象化
  - 各行の実行時間 (コスト) を定数 c_i とする
  - 最悪の実行時間を an^2+bn+c と表す
  - 実行時間の増加率をみるには主要項 an^2 で十分
  - 定数係数も無視
  - $\Theta(n^2)$ と表す
- 挿入ソートは $\Theta(n^2)$ という最悪実行時間を持つ
- あるアルゴリズムが他より効率がよい

  $\Leftrightarrow$最悪実行時間の増加率が低い

関数のオーダ

- アルゴリズムの効率を実行時間のオーダで特徴付け，相対的な比較を行う
- 入力サイズ n が大きいときの挙動を知りたい
- アルゴリズムの漸近的 (asymptotic) な効率を調べる

漸近記号

- Θ-記法
- O-記法 (オーダ)
- Ω-記法
- *o*-記法 (リトルオー)
- *ω*-記法

漸近記号

Θ-記法

- ある関数 $g(n)$ に対し, $\Theta(g(n))$ は次のような集合と定義する

$\Theta(g(n)) = \{f(n)$: 全ての $n \geq n_0$に対して
$0 \leq c_1\, g(n) \leq f(n) \leq c_2\, g(n)$ となるような
正の定数c_1, c_2, n_0 が存在$\}$

$$\Theta(g(n)) = \{f(n) : \exists c_1 > 0 \exists c_2 > 0 \exists n_0 > 0 \forall n \geq n_0$$
$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

- $f(n) = \Theta(g(n))$ は $f(n) \in \Theta(g(n))$ を意味する
- $2n^2+3n+1 = 2n^2 + \Theta(n)$ という表現も用いる

$\frac{1}{2}n^2 - 3n = \Theta(n^2)$を示す

全ての$n \geq n_0$に対して$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$となればいい

n^2で割ると$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$

$c_2 \geq \frac{1}{2}$なら$n \geq 1$に対して右辺の不等号が成立

$c_1 \leq \frac{1}{14}$なら$n \geq 7$に対して左辺の不等号が成立

よって$c_1 = \frac{1}{14}, c_2 = \frac{1}{2}, n_0 = 7$とすれば成立

$6n^3 \neq \Theta(n^2)$を背理法で示す

全ての$n \geq n_0$に対して$6n^3 \leq c_2 n^2$であるような

定数c_2, n_0が存在したとする

任意の大きなnに対して$6n \leq c_2$となり矛盾

$$f(n) = an^2 + bn + c = \Theta(n^2)$$

任意のd次多項式$p(n) = \sum_{i=0}^{d} a_i n^i$に対し$(a_d > 0)$

$p(n) = \Theta(n^d)$

O-記法

- ある関数 $g(n)$ に対し, $\mathrm{O}(g(n))$ は次のような集合と定義する

$\mathrm{O}(g(n)) = \{f(n)$: 全ての $n \geq n_0$に対して
$0 \leq f(n) \leq c\, g(n)$ となるような
正の定数c, n_0 が存在$\}$

$$\mathrm{O}(g(n)) = \{f(n) : \exists c > 0 \exists n_0 > 0 \forall n \geq n_0 \quad 0 \leq f(n) \leq cg(n)\}$$

- $f(n) = \Theta(g(n))$ ならば $f(n) = \mathrm{O}(g(n))$
- $n = \mathrm{O}(n^2)$ も正しい表現

- 「実行時間が$\mathrm{O}(n^2)$である」という命題の意味

$$\max_{I \in I_n} \left\{ T\left(A, I\right) \right\} = T_{\text{worst}}\left(A, n\right) = \mathrm{O}\left(n^2\right)$$

  - サイズ n のどんな入力に対しても $\mathrm{O}(n^2)$ 時間
  - $\Theta(n^2)$ 時間かかる入力があるとは言っていない
- 最悪実行時間について言っている
  - 実行時間は入力データに依存する
  - 最悪実行時間はデータ数 n のみに依存
- 挿入ソートの実行時間は$\mathrm{O}(n^2)$…正解
- 挿入ソートの実行時間は$\Theta(n^2)$…間違い
  - ソートされた入力に対しては$\Theta(n)$

Ω-記法

- ある関数 $g(n)$ に対し, $\Omega(g(n))$ は次のような集合と定義する

$\Omega\ (g(n)) = \{f(n)$: 全ての $n \ge n_0$に対して

$0 \le c\ g(n) \le f(n)$ となるような

正の定数c, n_0 が存在$\}$

$$O(g(n)) = \{f(n) : \exists c > 0 \exists n_0 > 0 \forall n \ge n_0$$
$$0 \le cg(n) \le f(n)\}$$

- $f(n)$, $g(n)$ に対して $f(n) = \Theta(g(n))$ であるための必要十分条件は

$f(n) = O(g(n))$ かつ $f(n) = \Omega(g(n))$

- Ω-記法は下界を表す
- 挿入ソートの最良の実行時間はΩ(n)とは
  - このアルゴリズムではどのような入力に対しても n に比例した時間が必ず必要という意味

o-記法 (リトルオー)

- ある関数 $g(n)$ に対し, $o(g(n))$ は次のような集合と定義する

$o(g(n)) = \{f(n)$: 任意の定数 $c > 0$ に対し

ある定数 n_0 が存在し,

全ての $n \geq n_0$に対して

$0 \leq f(n) \leq c\, g(n)\}$

$$o(g(n)) = \{ f(n) : \forall c > 0 \exists n_0 > 0 \forall n \geq n_0 \quad 0 \leq f(n) \leq cg(n) \}$$

$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$が成り立つ

- ω-記法

$\omega(g(n)) = \{f(n)$: 任意の定数 $c > 0$ に対し

ある定数 n_0 が存在し,

全ての $n \geq n_0$に対して

$0 \leq c\, g(n) \leq f(n)\}$

- $n^2/2 = \omega(n)$
- $n^2/2 \neq \omega(n^2)$

$$f(n) \in \omega(g(n)) \Leftrightarrow g(n) \in o(f(n))$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$ が成り立つ

アルゴリズムの設計

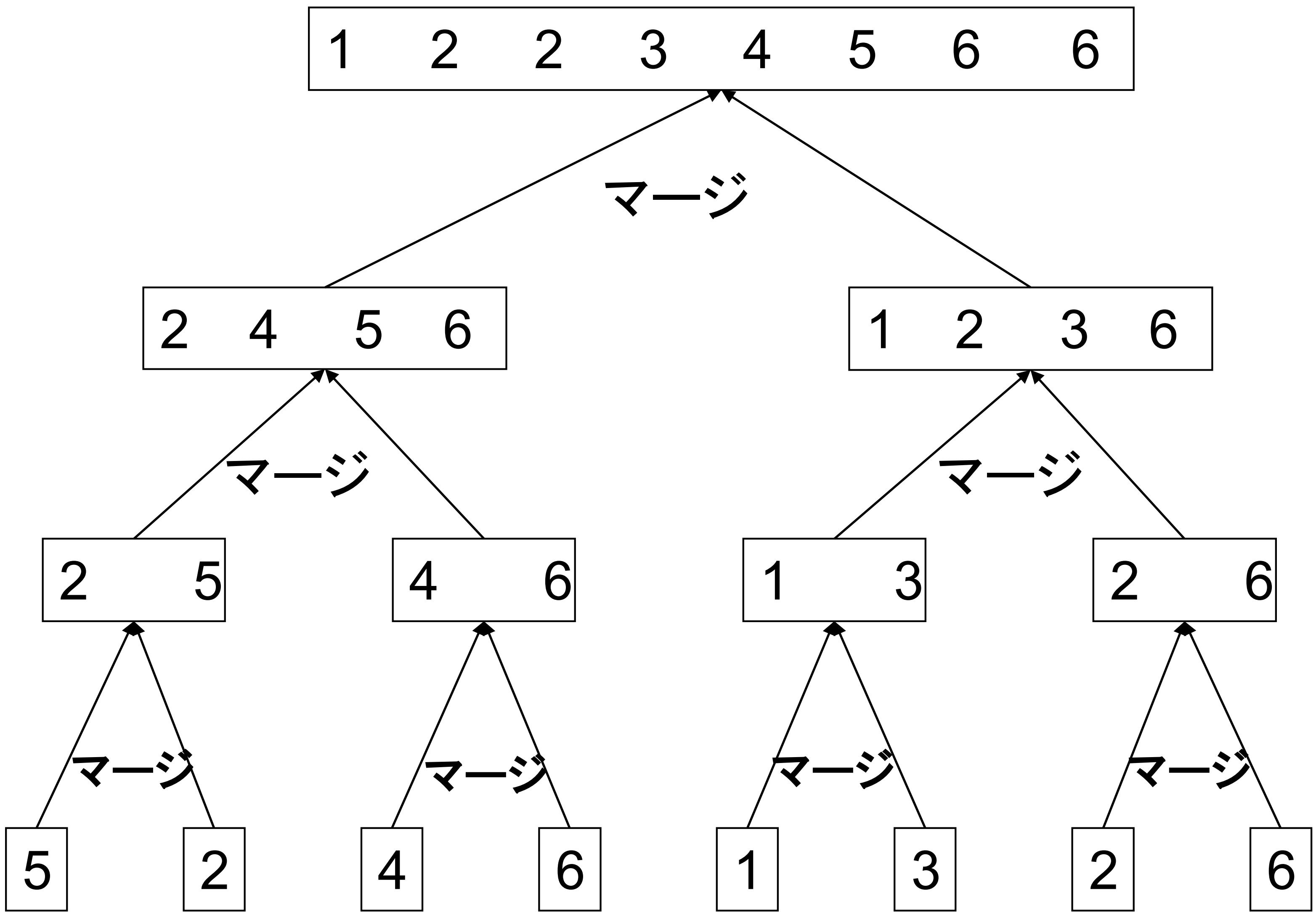
- 挿入ソート: 逐次添加法 (incremental approach)
- 分割統治法 (divide-and-conquer) に基づく方法

  →マージソート

  – 実行時間の解析が容易であることが多い

分割統治法

- 問題の再帰的な (recursive) 構造を利用
  - 分割 : 問題をいくつかの小さな部分問題に分割
  - 統治 : 各部分問題を再帰的に解く
  - 統合 : それらの解を組合わせて元の問題の解を構成
- マージソートでは
  - 分割 : n 要素の列を $n/2$ 要素の2つの部分列に分割
  - 統治 : マージソートを用いて2つの部分列をソート
  - 統合 : 2つのソートされた部分列を統合して答を得る


1 2 2 3 4 5 6 6
マージ
2 4 5 6
1 2 3 6
マージ
マージ
2 5
4 6
1 3
2 6
マージ
マージ
マージ
マージ
5
2
4
6
1
3
2
6

マージソート

```
void MERGE_SORT(data *A, int p, int r, data *B)     // A[p..r] をソート
{
  int q;

  if (p < r) {                                      // p==r ならソートする必要なし
    q = (p+r)/2;
    MERGE_SORT(A, p, q, B);   //  A[p..q] を再帰的にソート
    MERGE_SORT(A, q+1, r, B); //  A[q+1..r] を再帰的にソート
    MERGE(A, p, q, r, B);  // ソートされた部分列 A[p..q] とA[q+1..r] を統合
  }
}
```

マージ

- 一時的な配列 $B[0,n\text{-}1]$ を用いる

```
void MERGE(data *A, int p, int q, int r, data *B)
{      // ソートされた部分列 A[p..q] と A[q+1..r] を統合
  int i,j,k;
  data t;

  k = p;   i = p;   j = q+1;
  while (k <= r) {
    if (j > r) t = A[i++];                // 前半のみにデータがある
    else if (i > q)  t = A[j++];          // 後半のみにデータがある
    else if (A[i] <= A[j]) t = A[i++];  // 前半のほうが小さい
    else t = A[j++];                      //  後半のほうが小さい
    B[k++] = t;                           // 一時的な配列に保存
  }
  for (i=p; i<=r; i++) A[i] = B[i];  // 元の配列に書き戻す
}
```

```
void MERGE_SORT(data *A, int p, int r, data *B)     // A[p..r] をソート
{
  int q;
  if (p < r) {              // p==r ならソートする必要なし
    q = (p+r)/2;
    MERGE_SORT(A, p, q, B);   //  A[p..q] を再帰的にソート
    MERGE_SORT(A, q+1, r, B); //  A[q+1..r] を再帰的にソート
    MERGE(A, p, q, r, B);  // ソートされた部分列 A[p..q] とA[q+1..r] を統合
  }
}

int main(int argc, char *argv[])
{
  data A[14] = {27,17,3,16,13,10,1,5,7,12,4,8,9,0};
  data B[14];
  int i,n;

  n = 14;
  MERGE_SORT(A,0,n-1,B);
  for (i=0;i<n;i++) printf("%d ",A[i]);
  printf("¥n");

}
```

分割統治アルゴリズムの解析

- 全体の実行時間は問題のサイズに関する漸化式 (recurrence) で記述できることが多い
- サイズ n の問題に関する実行時間を $T(n)$ とする
- $n \le c$ (ある定数) ならば定数時間($\Theta(1)$)
- 問題を a 個の部分問題に分割し, それぞれが元のサイズの $1/b$ 倍になったとすると

$$T(n) = \begin{cases} \Theta(1) & n \le c \text{のとき} \\ aT(n/b) + D(n) + C(n) & \text{それ以外} \end{cases}$$

$D(n)$, $C(n)$: 問題の分割, 統合にかかる時間

マージソートの解析

- n は2のべき乗と仮定する
- $n = 1$のとき $T(n) = \Theta(1)$
- $n > 1$のとき
  – 分割: $D(n) = \Theta(1)$
  – 統治:再帰的にサイズ$n/2$の部分問題を解く
  $2T(n/2)$
  – 統合:MERGEは $C(n) = \Theta(n)$

$$T(n) = \begin{cases} \Theta(1) & n = 1\text{のとき} \\ 2T(n/2) + \Theta(n) & n > 1\text{のとき} \end{cases}$$

$T(n) = \Theta(n \lg n)$ となる

アルゴリズムの重要性

- コンピュータが速くても, 実行時間のオーダが大きいアルゴリズムは役に立たない
- スーパーコンピュータで挿入ソートを実行
  - 1秒間に1億命令実行
  - $2n^2$ 命令必要
- パーソナルコンピュータでマージソートを実行
  - 1秒間に100万命令実行
  - $50\ n \lg n$ 命令必要

- 100万個の数の配列のソート
- スーパーコンピュータで挿入ソート

$$\frac{2\cdot(10^6)^2\text{命令}}{10^8\text{命令}/\text{秒}} = 20{,}000\text{秒} \approx 5.56\text{時間}$$

- パーソナルコンピュータでマージソート

$$\frac{50\cdot10^6\lg10^6\text{命令}}{10^6\text{命令}/\text{秒}} = 1{,}000\text{秒} \approx 16.67\text{分}$$

- オーダの低いアルゴリズムの開発が重要