アルゴリズムとデータ構造の基本概念

計算科学研究センター
システム情報工学研究科CS専攻
北川博之

アルゴリズムとは

- コンピュータを利用した問題解決のためには，ソフトウェアの作成，すなわちプログラミングが必要．
- プログラミング
  - 問題をコンピュータ上で扱えるように定式化し，定式化した問題を解決するための手順とその際に利用するデータを，コンピュータが理解できる形式に書き下す作業．
- アルゴリズム
  - 問題を解くための方法について定めた機械的な手順（指示の系列）．
  - 入力からどのようにして出力を生成するかについての手順

例題1：最大公約数

- 入力：2つの正の整数nとm
- 出力：nとmの最大公約数
- 最も素朴な手順
  ① 1, 2, 3, …の順にnを割り切れるか，mを割り切れるかをチェックする．
  ② これをnとmのうちの小さい方の数に到達するまで行い，nとmの両方を割り切れることができた数のうち，最大のものを出力する．

例題1:最大公約数

リスト 1.1 最大公約数を求める素朴なアルゴリズム

```
// 2 つの整数 n と m の最大公約数
gcd(n, m) {
  if (n > m)
    n と m を交換;                    // 小さい方の数を n とする
  i = 1;
  while (i <= n) {
    if (n%i == 0 && m%i == 0)
      gcd = i;
    i = i+1; }
  return gcd;
}
```

例題2：多項式の値の計算

- 入力：
  - 多項式：$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$
  - 数値：x_0
- 出力：$p(x_0)$
- 最も素朴な手順

  ① pを0に初期化.

  ② i=0からi=nまで$a_i x^i$をpに加算する.

例題2: 多項式の値の計算

リスト 1.2 多項式の値の計算

```
// 配列 A[i] を係数とする n 次の多項式の変数 x に
// x0 を代入したときの値
polynomial(A, x0) {
  p = 0;
  for (i = 0; i <= n; i = i+1) {
    x = 1;
    for (j = 0; j < i; j = j+1)
      x = x*x0;
    p = p+A[i]*x;
  }
  return p;
}
```

例題3：配列データの探索

- 入力：
  - n個のデータの集合を格納した配列A[0], …, A[n−1]
  - データ：d
- 出力：配列A[0], …, A[n−1]内にデータdが存在する(TRUE)か否(FALSE)か
- 最も素朴な手順（線形探索）
  - i=0からi=n−1までA[i]=dが成り立つかをチェックする.

例題3:配列データの探索

リスト 1.3 線形探索による配列データの探索

```
// 配列 A の中からデータ d を探索
linear_search(A, d) {
  for(i = 0; i < n; i = i+1)
    if (A[i] == d) return TRUE; // データ d が存在
  return FALSE;                 // データ d が存在しない
}
```

アルゴリズムの正しさ

- 部分正当性
  - 条件１：いかなる入力に対しても，間違った答えを出力しない．
- 完全正当性
  - 条件１
  - 条件２：いかなる入力に対しても，有限時間で停止する．

例題1：最大公約数（別アルゴリズム）

- 入力：2つの正の整数nとm
- 出力：nとmの最大公約数
- ユークリッドの互除法
  - nとmの最大公約数はmとr（nをmで割った余り）の最大公約数に等しいという事実を反復的に利用.
- 例：31640と13673の最大公約数

  31640÷13673=2…4294
  13673÷4294=3…791
  4294÷791=5…339
  791÷339=2…113
  339÷113=3…0

例題1 : ユークリッドの互除法

リスト 1.4 ユークリッドの互除法

```
// 2 つの整数 n と m の最大公約数
euclid(n, m) {
  if (n < m) n と m を交換;      // 大きい方の数を n とする
  while (m != 0) {
    r = n%m;
    n = m;
    m = r;
  }
  return n;
}
```

例題2：多項式の値の計算（別アルゴリズム）

- 入力：
  - 多項式：$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$
  - 数値：x_0
- 出力：$p(x_0)$
- ホーナーの方法（Horner's method）
  - $p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$
    $= (\cdots((a_n x + a_{n-1})x + a_{n-2})x + \cdots + a_1)x + a_0$
  - 上記の式に従って括弧の内側から足し算と掛け算の組合せを繰り返し行う.

例題2:ホーナーの方法

リスト 1.5 ホーナーの方法による多項式の値の計算

```
// 配列 A[i] を係数とする n 次の多項式の変数 x に
// x0 を代入したときの値
horner(A, x0) {
  p = 0;
  for (i = n; i > 0; i = i-1)
    p = (p+A[i])*x0;
  p = p+A[0];
  return p;
}
```

例題3：配列データの探索（別アルゴリズム）

- 2分探索(binary search)
  - A[0]<=A[1]<=･･･<=A[n−1]を仮定．（そうなっていない場合は並べ替える（整列，ソーティング）．）
  - 探索範囲の下限をl，上限をhとする．最初は，l=0, h=n−1.
  - lとhの中間m= ⌊(l+h)/2⌋ に対し，A[m]をチェック.
    - A[m]=d：dが見つかった．TRUEを返して終了.
    - A[m]<d：l=m+1
    - A[m]>d：h=m−1
    - l<hならば上記を繰り返す.
  - A[l]=dならばTRUEを，そうでないならば，FALSEを返して終了.
- ⌊x⌋：床(floor)関数：x以下の最大の整数（切捨て）
- ⌈x⌉：天井(ceiling)関数：x以上の最小の整数（切上げ）

例題3:2分探索

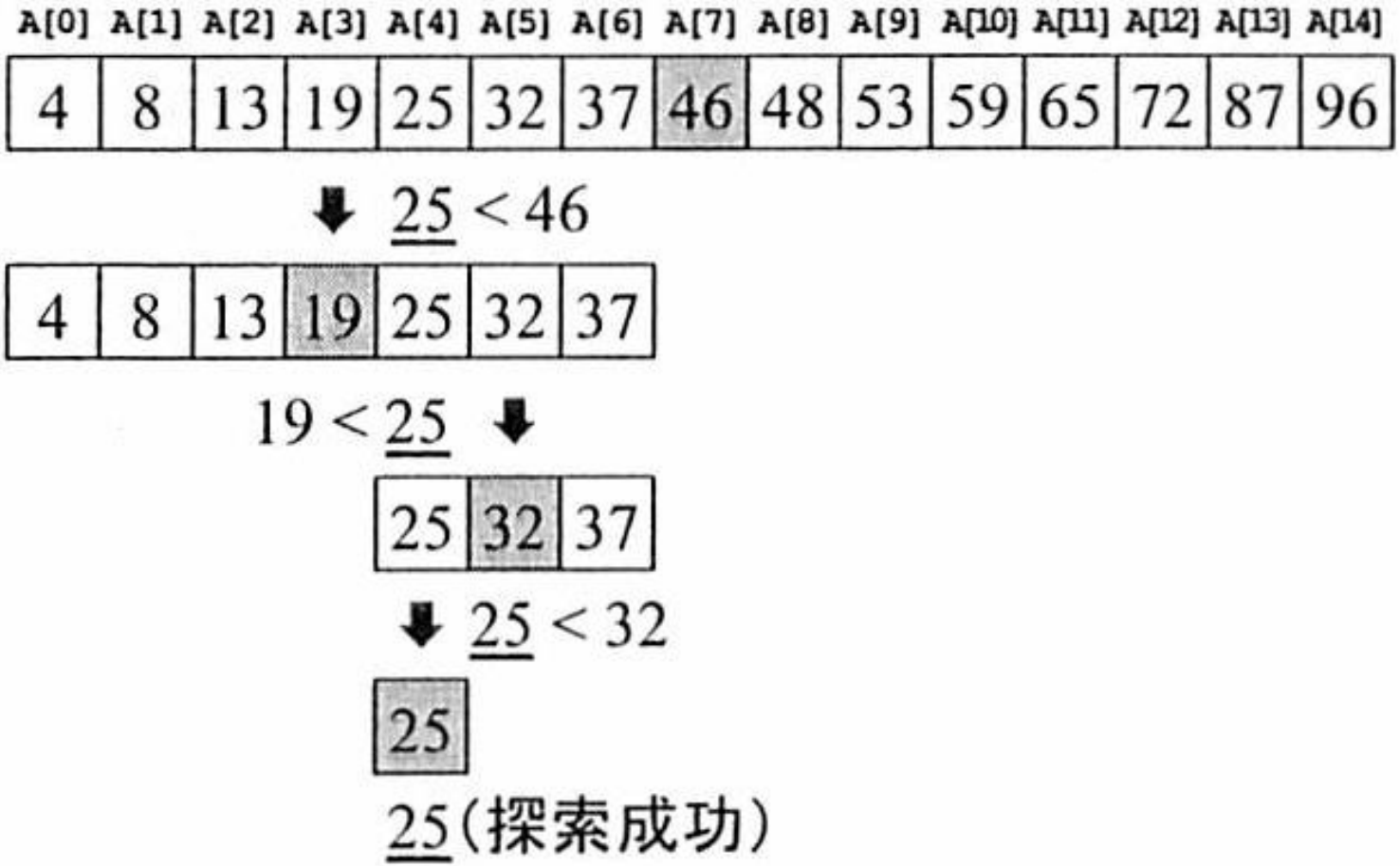


図 1.1　2 分探索のイメージ（25 の探索）

例題3:2分探索

リスト 1.6 2分探索による配列データの探索

```
// 配列 A の中からデータ d を探索
binary_search(A, d) {
  l = 0; h = n-1;
  while (l < h) {
    m = ⌊(l+h)/2⌋;
    if (A[m] == d) return TRUE;  // データ d が存在
    if (d < A[m]) h = m-1; else l = m+1;
  }
  if (A[l] == d) return TRUE;    // データ d が存在
  return FALSE;                  // データ d が存在しない
}
```

アルゴリズムの良し悪し

- 代表的な評価基準
  - 速く解を求められるほど良い
  - 少ない記憶領域で解を求められるほど良い
- アルゴリズム計算量
  - 時間計算量（time complexity）：単に計算量といった場合はこちら.
  - 領域計算量（space complexity）
- 時間計算量を求める例
  - 実際の実行時間は，ハードウェアや言語処理系に依存するので共通的な評価基準としては適切でない.
  - 通常は，問題サイズに応じたアルゴリズム中の基本操作回数に基づき見積もる.
  - 最大公約数を求める素朴なアルゴリズム
    - $f(n)=T_1+T_2n$
    - T_1はwhile文以外の部分，T_2はwhileループ内の必要時間

漸近的計算量

- 問題の規模が大きくなった時にどのように計算量が増加するかに注目
- O−記法（O−notation，ビッグオー記法等とも呼ぶ）
  - 計算量の漸近的上界を与える
  - 問題の規模nに対する計算量をf(n)．あるn_0と正定数cが存在し，$n>n_0$なる全てのnに対してf(n)<=cg(n)が成り立つ時，f(n)=O(g(n))と表記し，「アルゴリズムのオーダーはg(n)である」，「アルゴリズムはg(n)のオーダーである」という．
- 簡単な例
  - 最大公約数を求める素朴なアルゴリズム
    - $f(n)=T_1+T_2n \rightarrow O(n)$
  - $f(n)=3n^2+4n+2\log_2 n+3$
    - $O(n^2)$
- f(n)中でnに関して最も速く増加する項以外を削除し，その係数を無視したもの．

漸近的計算量

- Ω−記法（Ω−notation）
  - 計算量の漸近的下界を与える
  - 問題の規模nに対する計算量をf(n)．ある正定数cが存在し，無限個のnに対してf(n)>=cg(n)が成り立つ時，f(n)=Ω(g(n))と表記する．
- 通常は，計算量の上界が注目されるので，特に断りのない場合はO−記法による計算量を意味する．
- アルゴリズムによっては，同じサイズの入力データでもその内容によって計算量が変化することがある．
  - 最悪計算量（worst case complexity）
  - 平均計算量（average case complexity）

これまで示したアルゴリズムの計算量

- 最大公約数
  - 時間計算量
    - 素朴なアルゴリズム：既に述べた通りO(n).
    - ユークリッドの互除法：
      - n=mq+r, q>=1より, n−r=mq>=m>rとなり, r<n/2. rはループ2回の繰り返しの後に新たなnとして使われることになるため, ループを2回繰り返すごとに, nの値は半分未満になる. したがって, 最多でもループを回る回数は $2\lceil \log_2 n \rceil - 1$回.
        - 以上により, ユークリッドの互除法のオーダーはO(log(n)).
  - 空間計算量
    - いずれもn, mが大きくなっても必要な記憶領域は変わらないのでO(1).

これまで示したアルゴリズムの計算量

- 多項式の値の計算
  - 時間計算量
    - 素朴なアルゴリズム：
      - 外側のループはn+1回実行される．
      - 内側のループは0+1+2+...+n=n(n+1)/2回実行される．
      - 後者が主要な項となりO(n^2)．
    - ホーナーの方法：
      - ループは１重で，n回実行される．
      - O(n)
  - 空間計算量
    - いずれもn個の要素を格納した配列を用いるのでO(n)．

これまで示したアルゴリズムの計算量

- 配列データの探索
  - 時間計算量
    - 線形探索：
      - dが配列内に存在しない場合，ループはn回実行される．
      - O(n)．
    - 2分探索：
      - ループを実行するごとに，探索範囲はn/2，n/4，...と半減していき，$\lceil \log_2 n \rceil$ 回の繰り返しにより，$n/2^{\lceil \log_2 n \rceil} \fallingdotseq 1$．
      - O(log(n))
  - 空間計算量
    - いずれもn個の要素を格納した配列を用いるのでO(n)．

これまで示したアルゴリズムの計算量

表 1.1　例題の計算量

最大公約数	素朴な方法（リスト 1.1） $O(n)$	ユークリッドの互除法（リスト 1.4） $O(\log n)$
多項式の値	素朴な方法（リスト 1.2） $O(n^2)$	ホーナーの方法（リスト 1.5） $O(n)$
配列の探索	線形探索（リスト 1.3） $O(n)$	2 分探索（リスト 1.6） $O(\log n)$

計算量と計算時間

表 1.2　入力サイズと計算時間（明示されていない単位は秒）

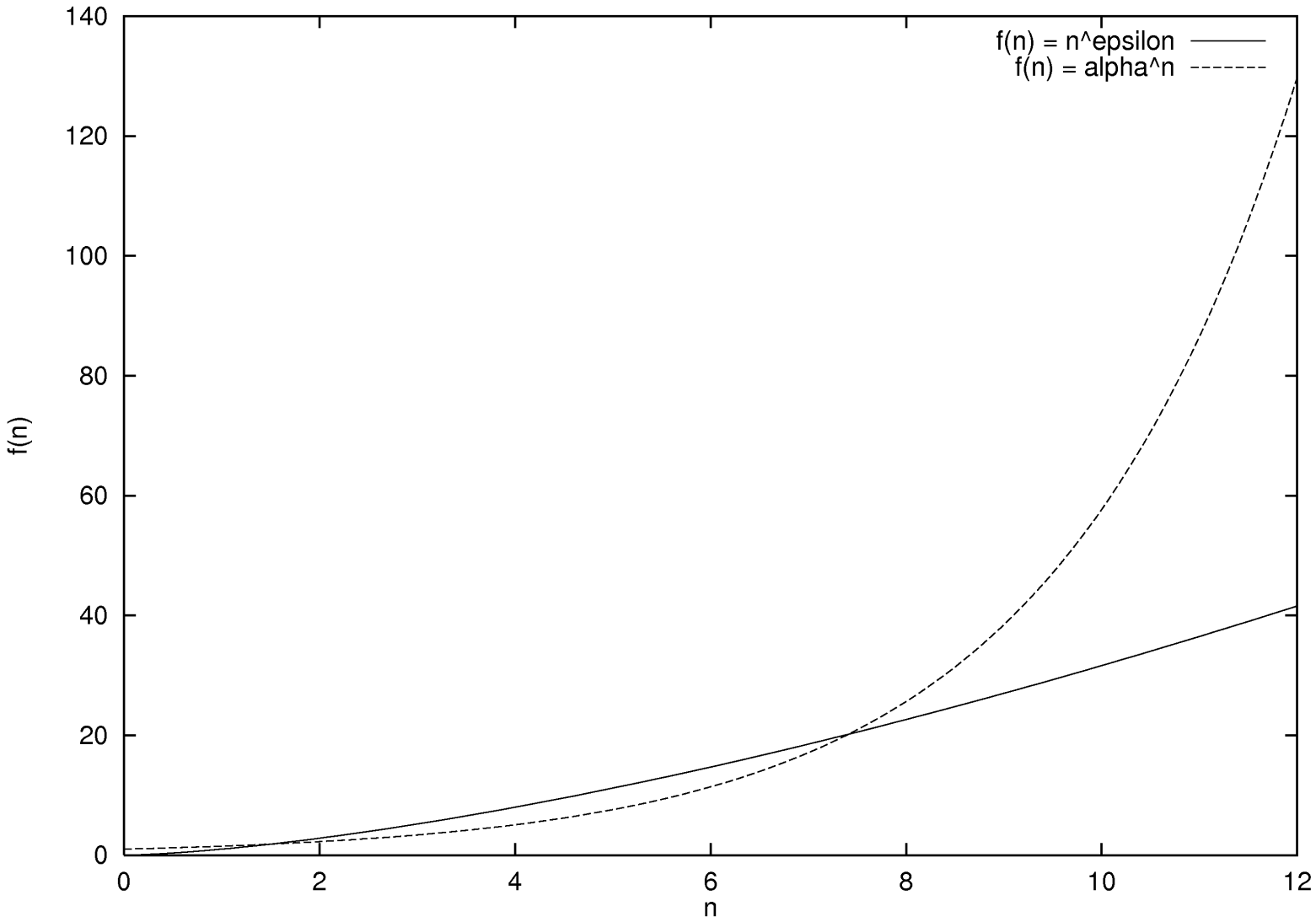
計算量	入力サイズ: n					
	10	20	100	1000	10000	100000
$\log n$	3.3×10^{-6}	4.3×10^{-6}	6.6×10^{-6}	1.0×10^{-5}	1.3×10^{-5}	1.7×10^{-5}
n	1.0×10^{-5}	2.0×10^{-5}	1.0×10^{-4}	0.001	0.01	0.1
$n \log n$	3.3×10^{-5}	8.6×10^{-5}	6.6×10^{-4}	0.01	0.13	1.7
n^2	1.0×10^{-4}	4.0×10^{-4}	0.01	1	100	2.8 時間
n^3	1.0×10^{-3}	0.008	1	17 分	280 時間	32 年
2^n	1.0×10^{-3}	1	4.0×10^{16} 年			
$n!$	3.6	77000 年	3.0×10^{144} 年			

表 1.3　コンピュータの速度と問題サイズ

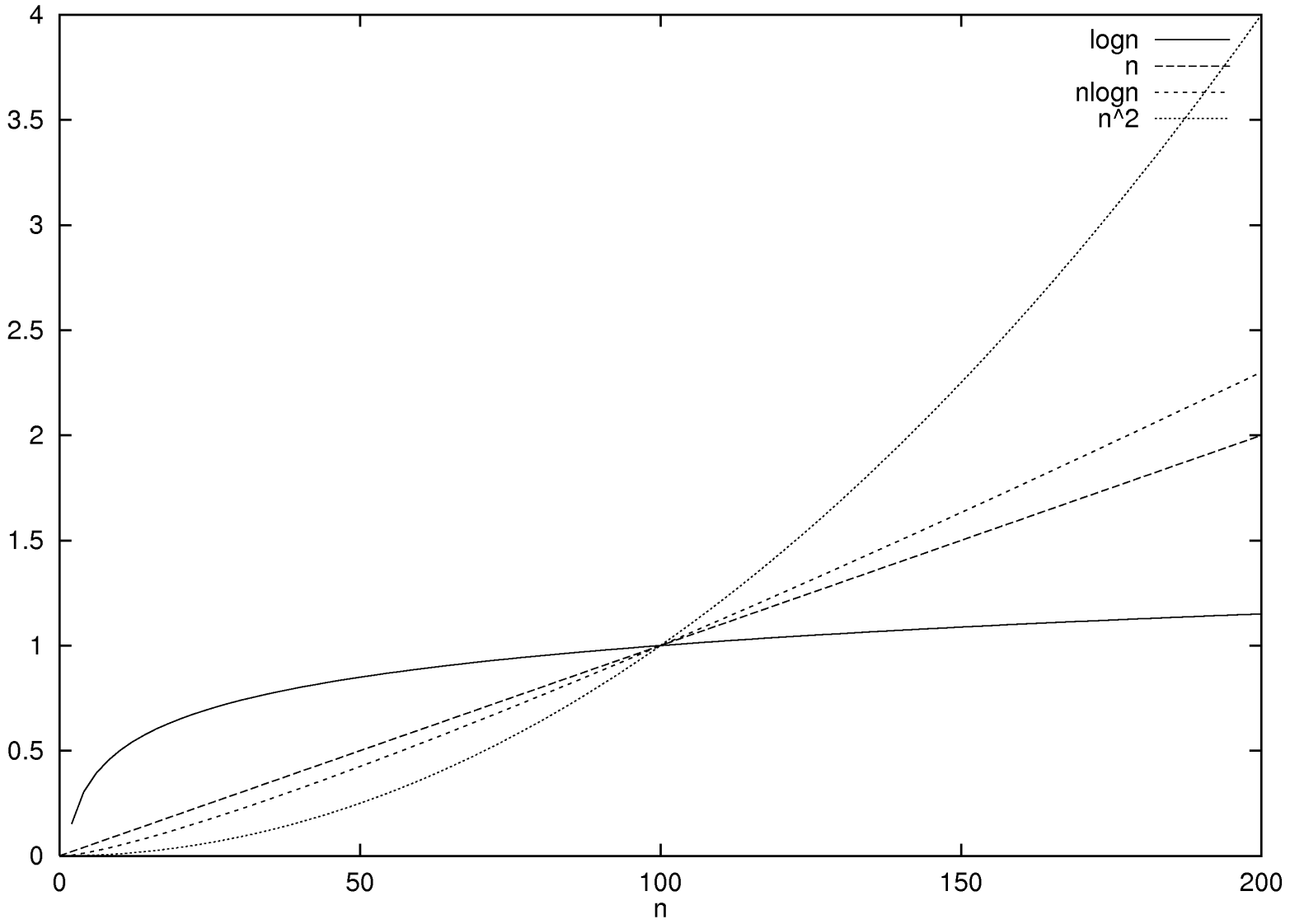
計算量	コンピュータの速度: a 倍				
	1	10	100	1000	10000
$\log n$	n	n^{10}	n^{100}	n^{1000}	n^{10000}
n	n	$10n$	$100n$	$1000n$	$10000n$
$n \log n$(注)	n	$7.7n$	$63n$	$520n$	$4500n$
n^2	n	$3.2n$	$10n$	$32n$	$100n$
n^3	n	$2.2n$	$4.6n$	$10n$	$22n$
2^n	n	$n+3$	$n+7$	$n+10$	$n+13$
$n!$(注)	n	n	n	$n+1$	$n+1$

(注) $n \log n$ と $n!$ については，$n = 1000$ を基準にして計算したものを例示

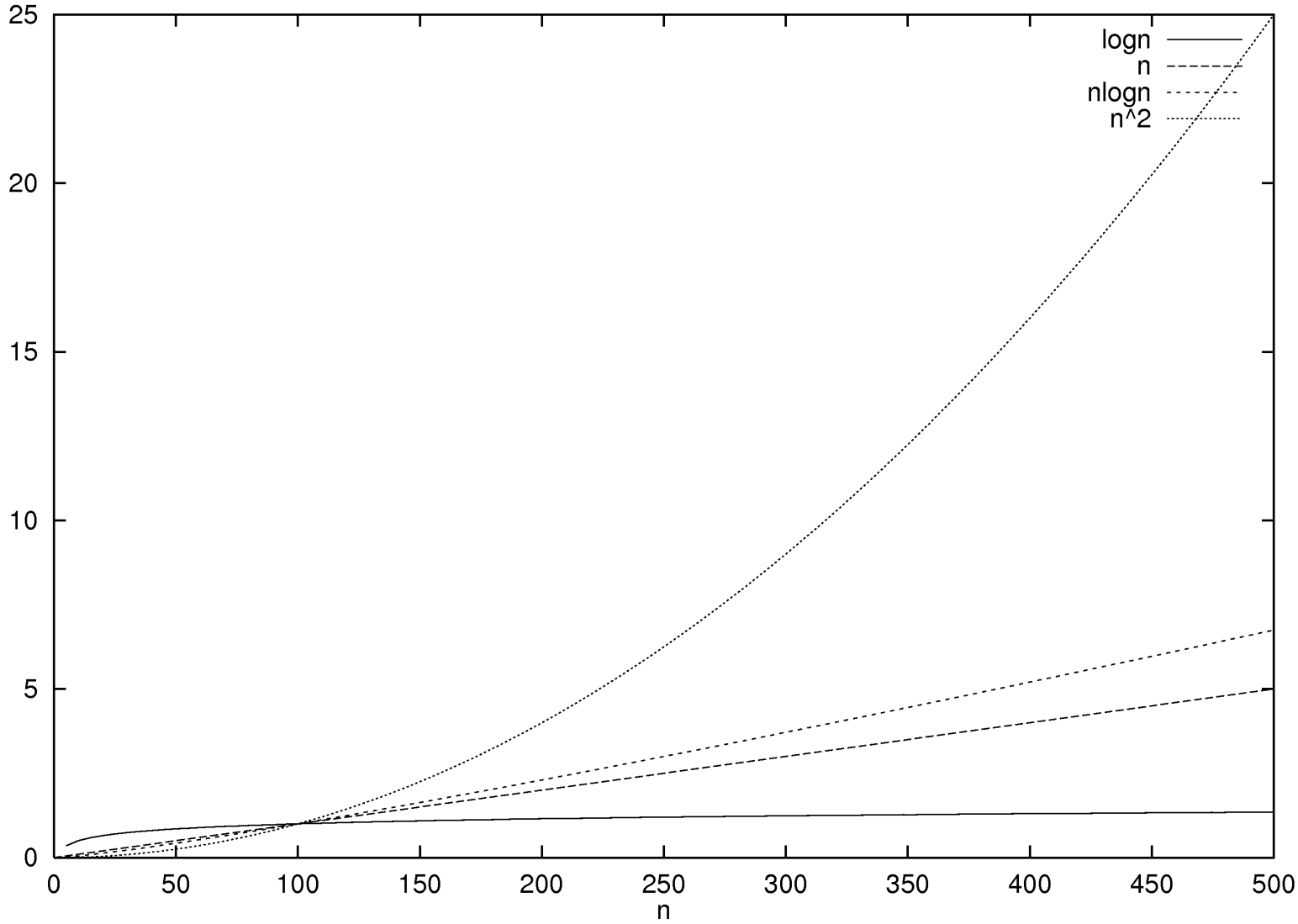
$$O(n^{\varepsilon})\,(0<\varepsilon)<O(\alpha^{n})\,(1<\alpha)$$



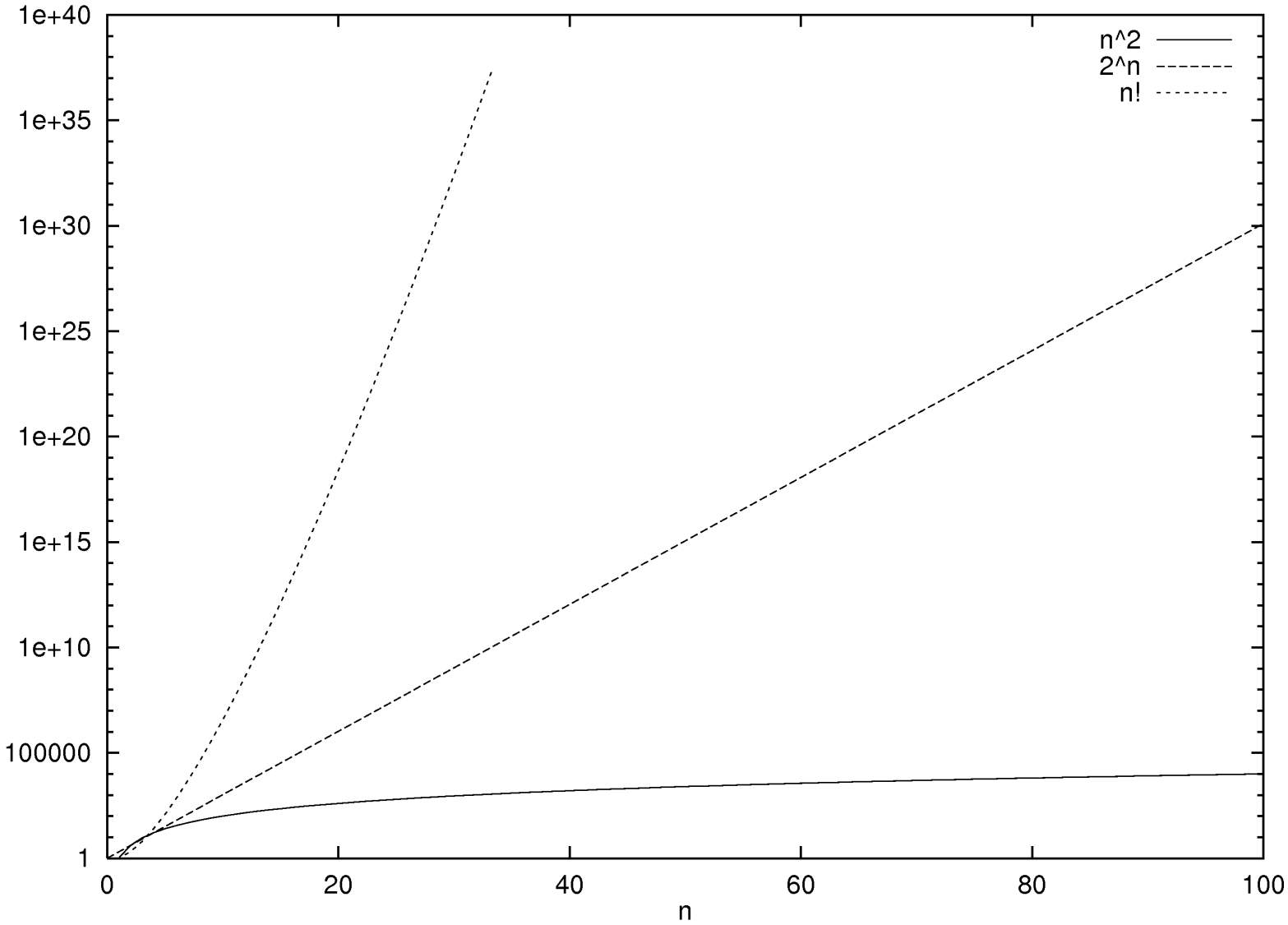
logn, n, nlogn, n^2の比較



logn, n, nlogn, n^2の比較



n^2, 2^n, $n!$の比較


1e+40
1e+35
1e+30
1e+25
1e+20
1e+15
1e+10
100000
1
0
20
40
60
80
100
n
n^2
2^n
n!

様々な計算量

- 計算量の大小関係

$$O(1) < O(\log n) < O(n^a)\ (0 < a \le 1) < O(n \log n)$$
$$< O(n^b)\ (1 < b) < O(\alpha^n)\ (1 < \alpha) < O(n!) < O(2^{2^n})$$

  - O(log n) < O(nᵃ) (a>0) とは，f(n)=O(log n)，g(n)=O(nᵃ)の時，十分大きなnに対して，f(n)<g(n)となるという意味．
    $\lim_{n\to\infty}|(\log n)/n^a| = 0$の証明は，ここでは省略．
    解析の教科書参照．
  - Stirlingの公式（解析の教科書参照）

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

    十分大きいnに対して成立．ただし，eは自然対数の底．

計算量と実行時間

- 多項式時間アルゴリズム
  - 計算量が多項式
- 指数時間アルゴリズム
- 最適アルゴリズム
  - それよりも計算量が少ないアルゴリズムが存在しないもの.
  - 最悪計算量が必要な操作の理論的な最小値に一致する.

再帰的アルゴリズム

- フィボナッチ数
  - $F_0=0$, $F_1=1$, $F_n=F_{n-1}+F_{n-2}$ ($n>1$)
- 再帰的アルゴリズム

```
f(n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return f(n-1)+f(n-2);
}
```

再帰的アルゴリズム

- 計算量
  - Fnを求めるのに必要な足し算の回数をa_nとすると，$a_0=a_1=0$，$a_n=a_{n-1}+a_{n-2}+1$.
  - $a_n/a_{n-1}=1+1/(a_{n-1}/a_{n-2})+1/a_{n-1}$.
  - nが十分大きい時，a_n/a_{n-1}がRに収束すると仮定すると，$R \fallingdotseq 1+1/R$．$R^2-R-1=0$.
  - R>0より，$R=\frac{1+\sqrt{5}}{2}$．よって，$a_n=\left(\frac{1+\sqrt{5}}{2}\right)^{n-2}$.
  - よって，$O(\alpha^n)$.

再帰的アルゴリズム

- フィボナッチ数を求める非再帰的アルゴリズム

```
f(n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    for (f0 = 0, f1 = 1, i = 2; i <= n; i = i+1) {
        f = f0+f1; f0 = f1; f1 = f;
    }
    return f;
}
```

- 計算量
  - ループはn-1回実行される.
  - O(n).

データ構造

- コンピュータ上でデータを表現する方法
- 論理構造
  - グラフ(graph)，リスト(list)，木(tree)等，現実世界のデータをモデル化して捉えるための枠組み．
- 物理構造
  - コンピュータの記憶領域の構造により近いレベルのデータ構造．
  - 配列：連続した記憶領域を用いてまとまったデータを表現．（順配置等と呼ばれることもある．）
  - リンク配置：セルを連結させながらデータの集まりを表現．
- 論理構造は，通常，何らかの物理構造を用いて表現される．
  - 木：リンク配置で表現されることが多いが，配列を用いる場合もある．

アルゴリズムとデータ構造の関係

- アルゴリズムとそれに用いるデータ構造は密接に関係.
- 例
  - 2分探索：整列済みの配列を前提.
  - データの格納に配列ではなく連結リストを用いた場合には，2分探索の効率的な実行はできない.
- データ構造によって新たなデータの登録処理の計算量が異なる.
  - 線形探索を前提とした配列：新たなデータは配列の末尾に追加すればよいので，登録処理の計算量はO(1).
  - 2分探索を前提とした配列：新たなデータの挿入位置を探すのにO(log(n))．さらに挿入位置以降のデータを全て後ろにずらす必要．後者に平均O(n).