データ構造とアルゴリズム

第２回 アルゴリズムと計算量

小池 英樹 (*koike@c.titech.ac.jp*)

アルゴリズムとは？

- アルゴリズム (algorithm, 算法)
  - 問題を解くための機械的に実行可能な手続き

- 正しいアルゴリズムの条件：
  - (1) どんな入力データに対しても間違った答を与えないこと。言い換えると、答を与える場合にはその答が正しいこと。
  - (2) どんな入力データに対しても有限の時間で答を与えること。すなわち、計算を完了すること。

アルゴリズムの例：最大公約数

- 最大公約数(greatest common divisor, GCD)
- Euclidの互除法
- 例：a=315, b=189
  - 315 = 189 * 1 + 126
  - 189 = 126 * 1 + 63
  - 126 = 63 * 1
  - よって63が315と189の最大公約数

アルゴリズムの例：最大公約数

- 自然言語による記述
  - 「２つの自然数a, b (a ≧ b)について，aのbによる剰余をrとすると，aとbの最大公約数はbとrの最大公約数に等しい．よって，bをrで割った剰余，除数rをその剰余で割った剰余，と剰余を求める計算を繰り返すと，剰余が0になった時の除数がaとbの最大公約数．」

アルゴリズムの例：最大公約数

- 手続き的記述
  - step 1: aをbで割って，余りをrとする．
  - step 2: r=0であれば，アルゴリズムは終了する．このとき，bが最大公約数である．
  - step 3: a←bとする(bの値をaに代入）．次にb←rとして，step 1に戻る．

アルゴリズムの例：最大公約数

- プログラミング言語Cによる記述

```
int gcd(int a0, int a1)
{
    int a, b, r;

    a = a0;
    b = a1;
    while (b != 0) {
        r = a % b;
        a = b;
        b = r;
    }
    return a;
}
```

アルゴリズムの例：最大公約数

- プログラミング言語Pascalによる記述

```
function gcd(a0, a1: integer): integer;
  var a, b, r: integer;
  begin
    a := a0;
    b := a1;
  while b <> 0 do
      begin
          r = a mod b;
          a := b;
          b := r
      end;
  gcd := a
end
```

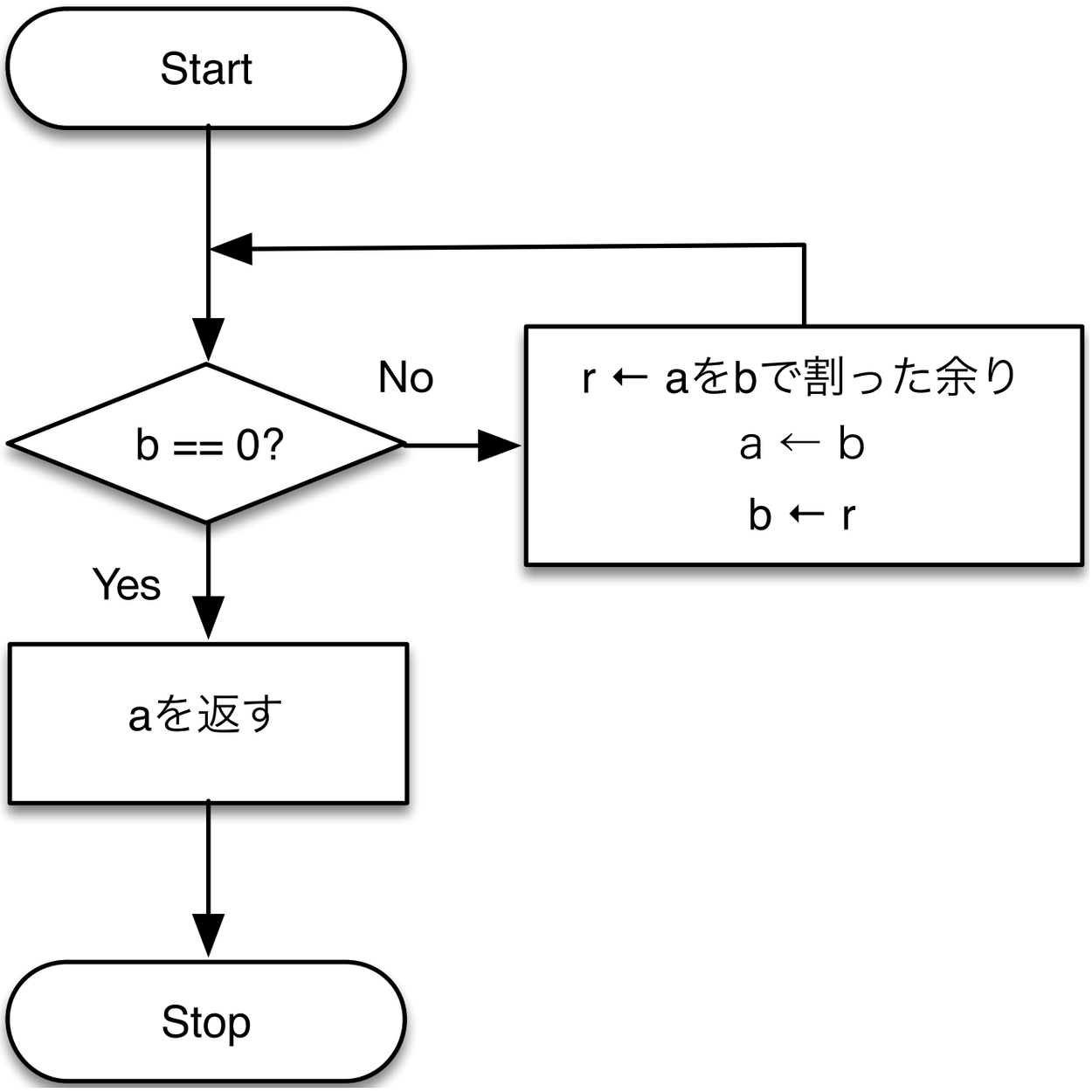
アルゴリズムの例：最大公約数

- 擬似コード
  - 繰り返しや条件分岐といった制御構造に着目.
  - 自然言語での記述を許す.
  - 変数宣言や文法等の詳細を省略.

```
gcd: aとbの最大公約数を返す
    while bがゼロになるまで以下を繰り返す
        r ← aをbで割った剰余
        a ← b
        b ← r
    aを返す
```

アルゴリズムの例：最大公約数

- 図式言語(フローチャート)による記述の例



アルゴリズムの記述法

	自然言語	擬似コード	プログラミング言語	図式言語
可読性	○(?)	○	×	○
制御構造	×	○	○	○
論理性	×	○	○	○
移植性	×	○	△	○
記述のコンパクトさ	○	○	○	×

アルゴリズムの例：最大公約数（再帰版）

- 自然言語による記述
  - 「２つの自然数a, b (a ≧ b)について，aのbによる剰余をrとすると，aとbの最大公約数はbとrの最大公約数に等しい．

    よって，bをrで割った剰余，除数rをその剰余で割った剰余，と剰余を求める計算を繰り返すと，剰余が0になった時の除数がaとbの最大公約数．」

アルゴリズムの例：最大公約数（再帰版）

- 手続き的記述
  - step 1: aをbで割った余りをrとする
  - step 2: もし，rが0ならばbが最大公約数である．
  - step 3: そうでなければ，bとrの最大公約数がaとbの最大公約数である．

アルゴリズムの例：最大公約数（再帰版）

- 再帰とは
  - 自分自身を定義するに自分自身を使用する
- 例：nの階乗 n!の計算
  - $n! = 1 \qquad (n = 0)$
    $n! = n^* (n-1)! \qquad (n \geqq 1)$
- 例：フィボナッチ数列fib(n)の計算
  - $fib(0) = 1$
    $fib(1) = 1$
    $fib(n) = fib(n-1) + fib(n-2) \qquad (n \geqq 2)$

アルゴリズムの例：最大公約数（再帰版）

- 擬似コードによる記述

```
gcd: aとbの最大公約数を返す
    r ← aをbで割った余り
    if rが0
        bを返す
    else
        gcd(b, r)を返す
```

アルゴリズムの例：最大公約数（再帰版）

- プログラミング言語Cによる記述

```
int gcd(int a, int b)
{
    int r;

    r = a % b;
    if (r == 0)
        return b;
    else
        return gcd(b, r);
}
```

アルゴリズムの例：最大公約数（再帰版）

- プログラミング言語Schemeによる記述

```
(define gcd
  (lambda (a, b)
    (let ((r (mod a b)))
      (if (= r 0)
        b
        (gcd b r)))))
```

アルゴリズムの例：最大公約数

- ループ型と再帰型の比較

```
int gcd(int a0, int a1)
{
    int a, b, r;

    a = a0;
    b = a1;
    while (b != 0) {
        r = a % b;
        a = b;
        b = r;
    }
    return a;
}
```

```
int gcd(int a, int b)
{
    int r;

    r = a % b;
    if (r == 0)
        return b;
    else
        return gcd(b, r);
}
```

良いアルゴリズムとは

- 性能：
  - **計算時間**
  - **使用するメモリの量**
  - 外部記憶への入出力
  - ネットワーク通信量
  - …
- 他の要素：
  - 読みやすさ(readability)
  - モジュール性(modularity)
  - 再利用可能性(reusability)
  - 移植性(portability)
  - …

計算時間

- プログラムの実行時間を決める要素
  - 入力
  - コンパイラが出す目的プログラムの質
  - 使用する機械命令の性質と速さ
  - プログラムのもとになったアルゴリズムの計算量

- 単純に議論することはできない.

アルゴリズムの計算量

- 計算量(computational complexity)
  - **時間計算量(time complexity)：** <- 計算の複雑さ，手間ともいう
    - アルゴリズムが答を出すまでにどの程度の計算時間を必要とするかの概算.
      - ステップ数
  - **領域計算量(space complexity)**
    - 計算の途中経過を保持するために必要な記憶領域の広さ
    - メモリ量

アルゴリズムの解析(ANALYSIS OF ALGORITHMS)

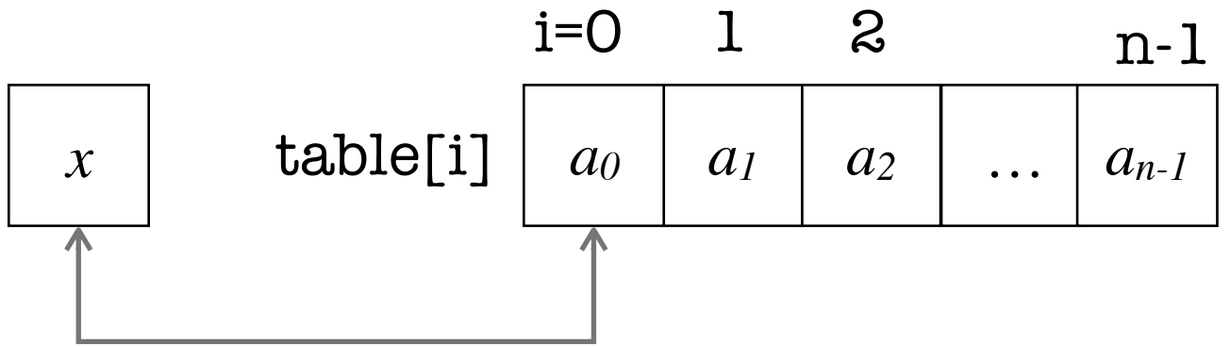
- 時間計算量をプログラム実行のステップ数ではかる.

- ただし，ステップ数は同じアルゴリズムでも，言語やコンパイラ等で変化する.

- 正確なステップ数は必要ない.
  - -> 漸近的計算量

例：線形探索

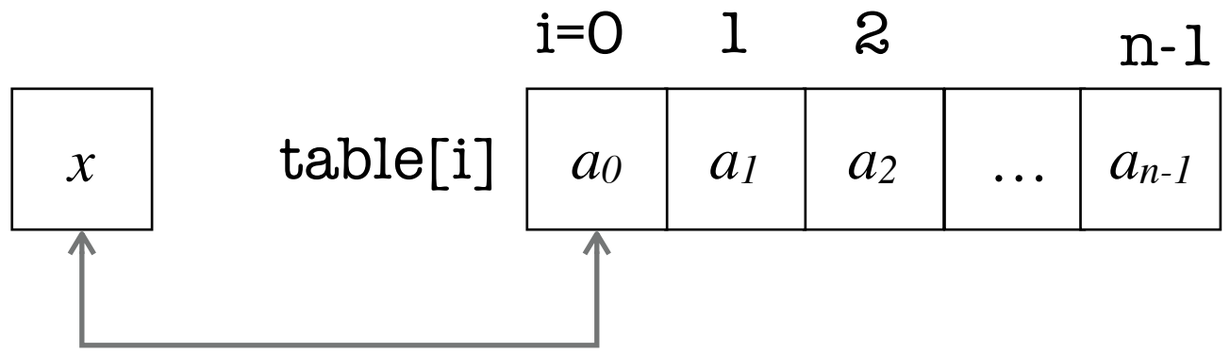
- n個のデータを持つ表(table)の中に，あるデータxがあるかどうかを先頭から順番に調べる（探索する）．




```
found = 0;
for (i=0; i<n; i++) {
    if (x == table[i]) {
        found = 1;
        return;
    }
}
```

例：線形探索

- 計算量
  - データが見つかる最善の場合：
    table[0]==x. if文は1回.
  - データが見つかる最悪の場合：
    table[n-1]==x. if文をn回.
    - 平均するとループを回る回数は
      約 n/2.
- データが見つからない場合：
  - 計算時間は cn+A (c, Aは定数).
- よって，計算量は cn




```
found = 0;
for (i=0; i<n; i++) {
    if (x == table[i]) {
        found = 1;
        return;
    }
}
```

例：２分探索

- 表(table)のデータをあらかじめ小さい順に並べておき，探索範囲の中央値を順番に調べる．

```
found = 0;
low = 0; high = n-1;
while (low <= high) {
    mid = (low + high) / 2;
    if (x < table[mid]) {
        high = mid-1;
    } else {
        low = mid+1;
    }
}
found = (high > 0) ∧ (x = table[high]);
```

例：２分探索

- 計算量
  - ループを 1 回回るたびに探索範囲を半分に絞り込む.
  - 1回でn/2，２回でn/4, ...
  - $\log_2 n$ 回ループを回れば，範囲の幅が1以下になって終了する.
- よってループ 1 回の計算時間をdとして，$d \log_2 n$ 程度.

O記法 (O-NOTATION)

- パラメータnについての関数T(n)に対して、ある一定の値n_0と正の定数cがあり、$n \geqq n_0$を満たすすべてのnに対して
  $T(n) \leqq c\, f(n)$となるとき、

  $$T(n) = O(f(n))$$ <- ビッグオー

  と表記する.

- 「T(n)はオーダーf(n)である」または「T(n)はビッグオーf(n)である」と読む

- 漸近的上界(asymptotic upper bound).

O記法 (O-NOTATION)

- 漸近的上界はできるだけ単純で精度の高いものが良い
  - $2n^2 + 5n + 1000 = O(n^2)$
  - $2n^2 + 5n + 1000 = O(n^2+n)$　<-より複雑
  - $2n^2 + 5n + 1000 = O(n^3)$　<-より精度が低い
- O(1)は定数オーダー(constant order)と呼ばれ，nに独立なある定数で抑えられることを意味する.
- $2.5n^2 = O(n^2)$の'='は厳密な意味での等号ではない.
  - $100n^2 = O(n^2)$は正しいが，$2.5n^2 = 100n^2$は言えない
  - $2.5n^2 = O(n^2)$は正しいが，$O(n^2) = 2.5n^2$ は正しくない.

O記法 (O-NOTATION)

- 探索の例：
  - 線形探索のオーダーは n.　... O(n)
  - ２分探索のオーダーは log n.　... O(log n)

注意：今後特にことわらない限り
対数の底は2.

O記法の演算

- $T_1(n) = O(f(n)), T_2(n) = O(g(n))$のとき
  - $T_1(n) + T_2(n) = O(\max(f(n), g(n)))$
  - $T_1(n)\ T_2(n) = O(f(n)\ g(n))$

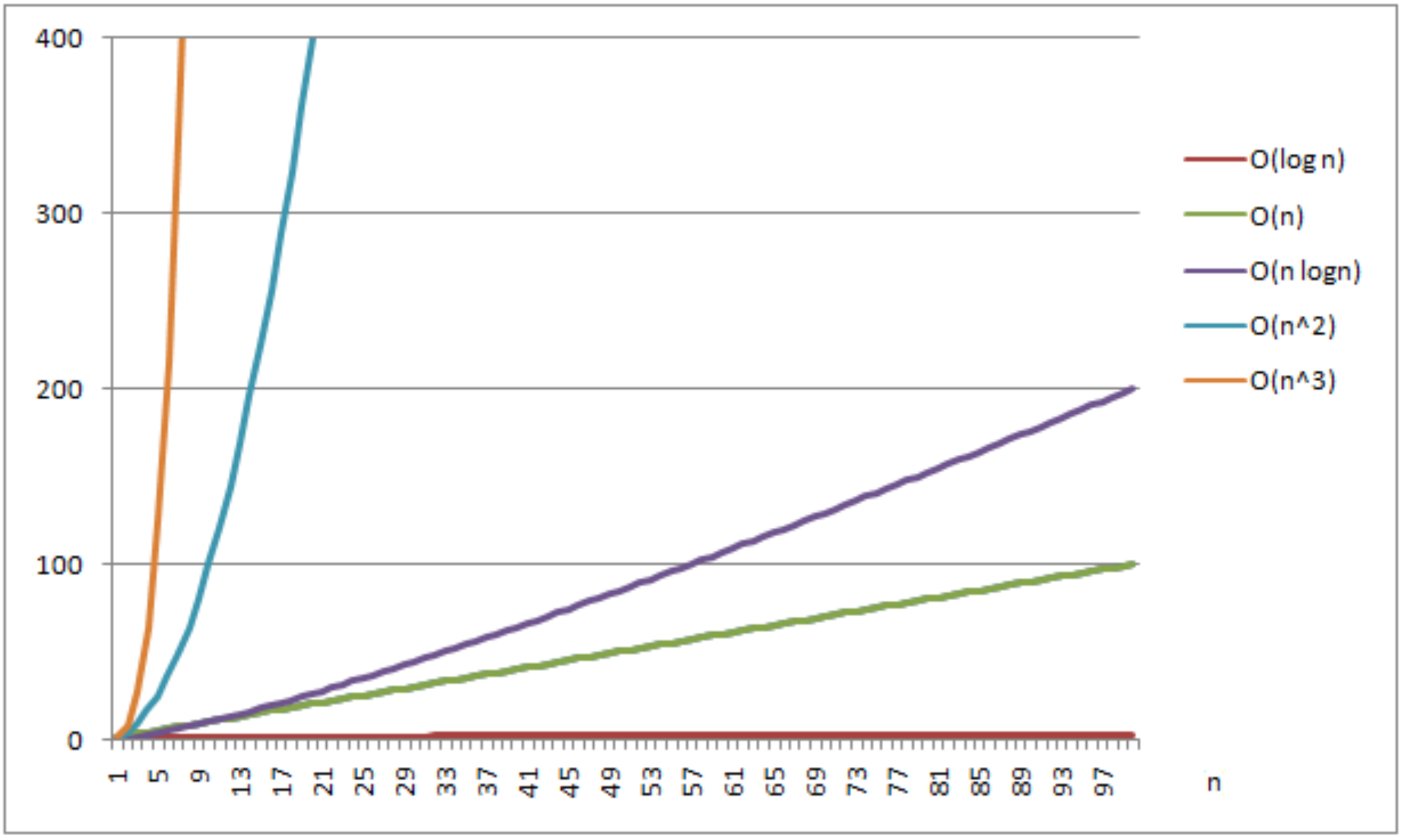
- 例：$T_1(n) = O(n^2), T_2(n) = O(n^3)$のとき
  - $T_1(n) + T_2(n) = O(n^3)$
  - $T_1(n)\ T_2(n) = O(n^5)$

計算量の漸近的評価

n=100のときに１秒かかると仮定した場合.

	10	20	50	100	200	500	1000	10000
O(1)	1	1	1	1	1	1	1	1
O(log n)	0.5	0.65	0.85	1	1.15	1.35	1.5	2
O(n)	0.1	0.2	0.5	1	2	5	10	100
O(n log n)	0.05	0.13	0.42	1	2.3	6.75	15	200
O(n^2)	0.01	0.04	0.25	1	4	25	100	2.78時間
O(n^3)	0.001	0.008	0.125	1	8	125	1000	11.6日
O(2^n)	0.001	1.02	34.8年	-	-	-	-	-

計算量の漸近的評価


400
300
200
100
0
1
5
9
13
17
21
25
29
33
37
41
45
49
53
57
61
65
69
73
77
81
85
89
93
97
n
O(log n)
O(n)
O(n logn)
O(n^2)
O(n^3)

例：反復の計算量

- kに関するfor文：反復回数がO(n)，(2)の文がO(1)なので，f(n)＝n, g(n)＝1．全体としてO(n * 1) ＝ O(n)
- jに関するfor文：反復回数がO(n)，反復の中の1回の計算量はO(1)＋O(n)＝O(n)．全体としてO(n * n) ＝ O(n^2)
- iに関するfor文：反復回数がO(n)，反復の中の1回の計算量はO(n^2)．全体としてO(n * n^2) ＝ O(n^3)

```
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        c[i][j] = 0;                          (1)
        for (k = 0; k < n; k++) {
            c[i][j] += a[i][k] * b[k][j];     (2)
        }
    }
}
```

例：再帰プログラムの計算量

- fact(n)の実行時間をT(n)とする．
- (1)(2)の実行時間はO(1), (3)はO(1)+T(n-1).
- $T(n) = \begin{cases} c + T(n-1) & n>1 \quad \ldots (a) \\ d & n \leqq 1 \end{cases}$
- n>2とすると

  $T(n) = 2c + T(n-2) \quad n>2$

```
int fact(int n)
{
  if (n <= 1)                  (1)
      return 1;                (2)
   else
      return n*fact(n-1); (3)
}
```

- (a)のnにn-1を代入してT(n-1)=c + T(n-2)とし，これを(a)に代入する．さらに(a)を使ってT(n-2)を展開すると

  $T(n) = 3c + T(n-3) \quad n>3$
- これを繰り返して，一般に

  $T(n) = ic + T(n-i) \quad n>i$
- i = n-1として

  $T(n) = c(n-1) + T(1) = c(n-1) + d$
- よって T(n) = O(n)

Ω記法(Ω-NOTATION)

- $n \geqq n_0$のとき，不等号を逆転した$T(n) \geqq c\ f(n)$が成立する場合，

$$T(n) = \Omega(f(n))$$ ←ビッグオメガ

と表す.

- 漸近的下界(asymptotic lower bound).

- 例：$T(n) = n^3 + 2n^2$は$\Omega(n^3)$である.

- 例：$T(n) = n^3 + 2n^2$は$\Omega(n^2)$である.

Θ記法(Θ-NOTATION)

- $T(n)=O(f(n))$　かつ　$T(n)=\Omega(f(n))$ であるとき

$$T(n) = \Theta(f(n))$$　　<- ビッグシータ

と表す.

- 漸近的にタイトな限界(asymptotic tight bound)

オーダーの計算法

- 規則１：T(n)がnの多項式ならば，最大次数の項のオーダーになる．
  - 例：$2n^2 + 5n + 100 = O(n^2)$
  - 例：$10n + 2\sqrt{n} + 5 = 10n^1 + 2n^{0.5} + 5 = O(n)$
- 規則２：次のオーダーの式が成立する．
  - $\log(n) = O(n)$
  - $n = O(2^n)$
  - 任意のc>0に対して，$\log n = O(n^c)$
- 規則３：T(n)がいくつかの項の和ならば，最大次数の項のオーダーになる
  - 例：$3n + 2\sqrt{n} + 100\, n \log n + 5 = O(n \log n)$

計算量の各種の定義

- 最大計算量(worst case complexity)
  - 最も具合の悪い入力データを与えた時の計算量
  - 必ずしも実勢を反映しない
- 平均計算量(average case complexity)
  - 可能な入力データすべてに対する計算量の平均
  - 入力データの範囲や，それぞれの入力データの出現確率の見積もりが難しい

領域計算量（SPACE COMPLEXITY)

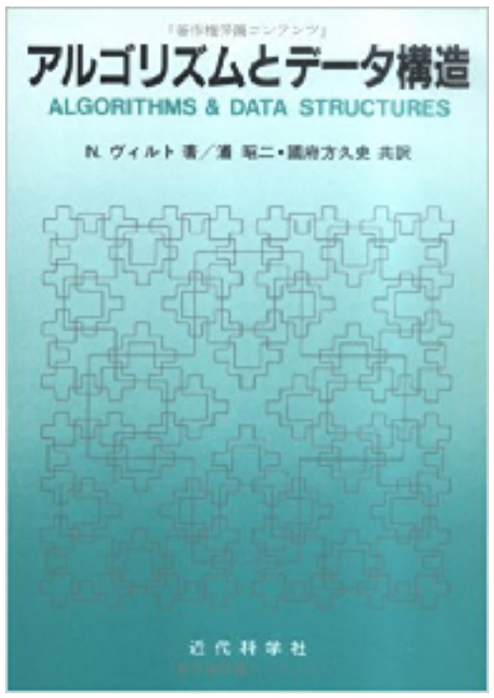
- アルゴリズムが使用する計算機の記憶領域（メモリー）の評価

- 領域計算量が $O(n^2)$ なのに時間計算量が$O(n)$ということはあまり考えられない.
- よって，時間計算量の方が第一義的.

データ構造

- 計算のために，記憶領域に効率よくデータを配置するための配置法
- 時間と記憶領域を効率化

- “Algorithms + Data structures = Programmings”
  - Niklaus Wirth
  - Pascalの開発者



抽象データ型(ABSTRACT DATA TYPE)

- データ構造とそれを操作する手続きをまとめてデータ型の定義とすることでデータ抽象(data abstraction)を行う手法.

- 例：線形リストとそれを操作する手続き

  データ構造：線形リスト

  操作：create()

  insert()
  delete()
  …

- オブジェクト指向言語へと展開.

レポート課題

- (1) Euclidの互除法のCプログラムを作り，3465と1323の最大公約数を求めなさい．プログラムソースコードと実行結果画面のスナップショットを送る．

- (2) 次の関数を増加率の順に並べよ．

  (a) n, (b) $\sqrt{n}$, (c) $\log n$, (d) $\log \log n$, (e) $\log^2 n$,
  (f) $n/\log n$, (g) $\sqrt{n} \log^2 n$, (h) $(1/3)^n$, (i) $(3/2)^n$, (j) 14.

- 締切：2017/12/7 17:00
- 提出方法：以上を 1 つのpdfファイルにまとめてメールに添付して送る．
- 送付先：koike@c.titech.ac.jp
- Subject名：アルゴリズム　レポート #1