アルゴリズム

* 与えられた問題を解くための明確に定義された有限個の規則の集合。
* 問題は一般的な用語を用いて臨む入出力関係を指定する。
* 具体的な問題は一般には値の集合の形式をとり、入力として与えられる。
* 解を出力として生成する。

アルゴリズムの評価

* 同じ結果を得るアルゴリズムは幾通りにも考えられる。
* どのアルゴリズムを使うのが良いのだろうか？
  + 高速か省メモリか
  + メンテナンスのしやすさ
* 外部から評価指標を持ち込む必要があり、その一つに計算量がある。

計算量

* アルゴリズムが要するリソース使用量に関係する量。
* 「そのアルゴリズムは対象が変化するとどのくらいの割合でリソース使用量が変化するか？」
* リソース
  + ステップ
  + メモリ
* 時間計算量
  + ステップ数に関係
* 空間計算量（領域計算量）
  + メモリ量に関係

計算モデル

* 計算量を求めるためにはどのような計算機で演算を行うかを想定する必要がある
* それには資源とコストのモデルを含む必要がある。
* Random Access Machine
  + 単一プロセッサ（命令は１命例ずつ逐次実行）
  + 命令の種類の制限
* 現実の計算機は並列プロセッサであったり複雑な処理をこなす命令を含んでいるがこれ以降はRAMを想定

時間計算量の求め方

* ステップ数を求める
* １命令１ステップ
  + （中身が不明な）関数の呼び出しも１ステップとしてカウント
  + 中身が明瞭な場合はそれも考慮
* 分岐に入るか否は最悪の場合を想定
  + 最悪時間計算量
  + 最良時間計算量、平均時間計算量といった考え方もある。
* 最終的にはオーダーに直すので細部は考慮しない

例（アルゴリズム１）

1: n = （配列bの長さ）;

2: for(i=0; i<n-1; ++i) {

3: mini = i;

4: for(j=i+1; j<n; ++j) {

5: if(b[j] < b[mini])

6: mini = j;

7: }

8: tmp = b[i];

9: b[i] = b[mini];

10: b[mini] = tmp;

11: }

• 1行目：ステップ数1

• 2行目："i=0"でステップ数1

• 2～11行目は、以下のステップ数の総和（0～n-2）

• 2行目："i<n-1", "++i"でステップ数2

• 3行目：ステップ数1

• 4行目："j=i+1"でステップ数1

• 4～7行目は、以下のステップ数の総和（i+1～n-1）

• 4行目："j<n", "++i"でステップ数2

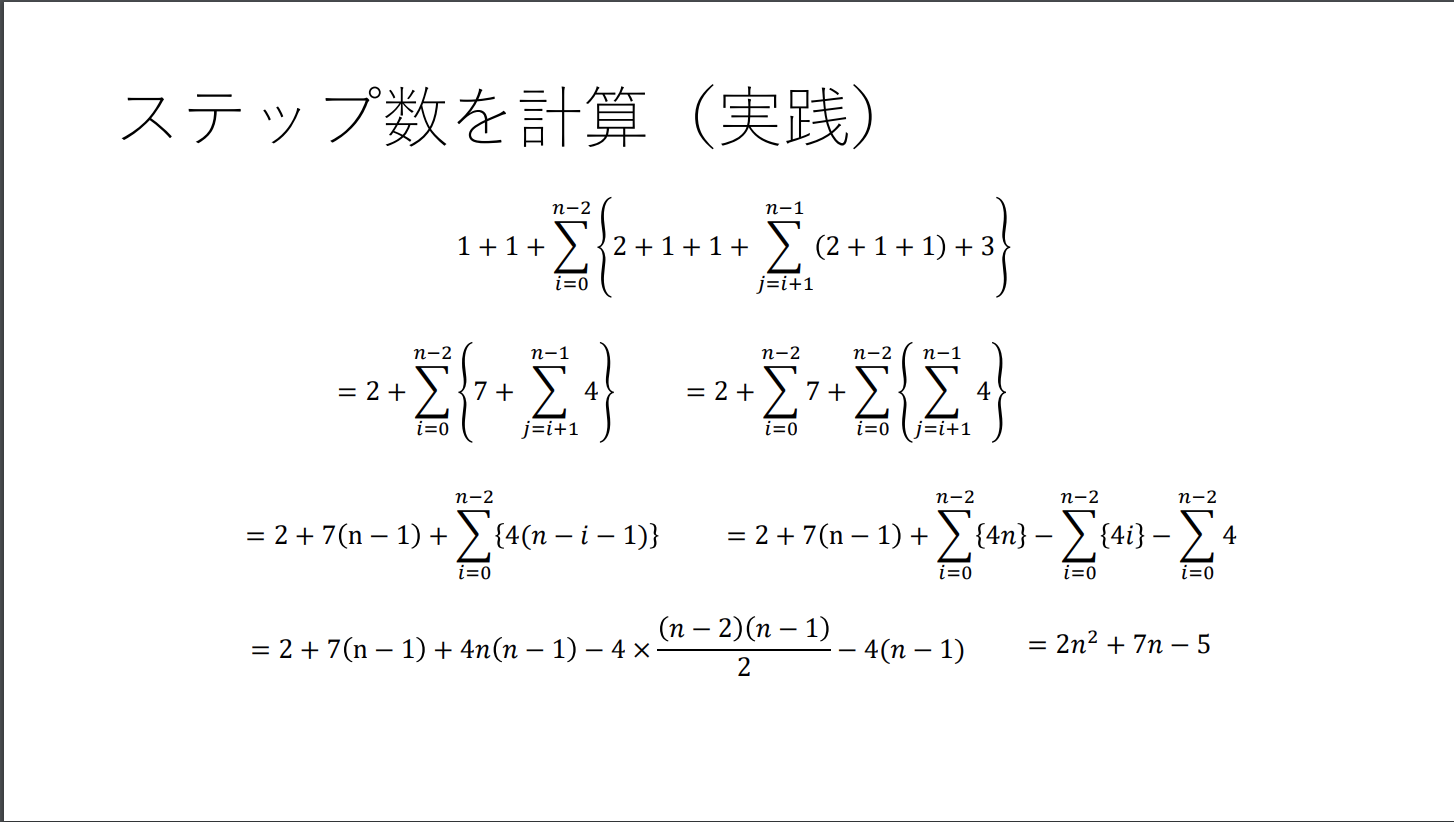
• 5行目：ステップ数1

• 6行目：最悪の場合、毎回実行される可能性があるので、ステップ数1

• 8～10行目は、ステップ数3

物体, アンテナ が含まれている画像

自動的に生成された説明



例（アルゴリズム２）

p1 = 0;

p2 = (配列hの長さ) -1;

while(true)

{

if(p1>p2)

{

return -1;

}

m=(p1+p2)/2;

if(h[m]==v)

{

return m;

}

else if(h[m] < v)

{

p1=m+1;

}

else

{

p2=m-1;

}

}

• 1,2行目：それぞれステップ数1

• 3～11行目は以下のステップの総和（1～x）

• 3,4行目：見つかるまで(xまで毎回)ステップ数1

• 5行目は見つからなかった場合最後（xのとき）にステップ数1

• 6,7行目は見つからなかった場合以外（x以外のとき）それぞれステッ

プ数1

• 8行目は見つかる場合最後（xのとき）にステップ数1

• 9,10,11行目は結果がまだわからない時点でループごとにステップ数の

合計が2

スクリーンショット が含まれている画像

自動的に生成された説明

例（アルゴリズム３）

関数　fact

{

if(n<=0)

{

return 1;

}

else

{

return n\*fact(n-1);

}

}

• 1行目：0ステップ

• 2行目：1ステップ

• 3行目：n<=0のとき1ステップ

• 5行目：n>0のとき1ステップと、関数呼び出しで1ステップと、

さらに引数がn-1のこの関数を呼び出した時のステップ数を足

したものスクリーンショット が含まれている画像

自動的に生成された説明

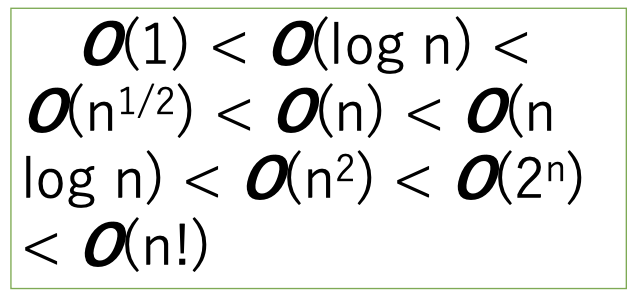
ο記法（ビッグオー記法）

* 本質的な計算量は対象の量が極限まで行った時のアルゴリズムの振る舞いを測るべきである
* そのとき最も大きい項以外は誤差になる
* 最も大きな影響を及ぼす項以外を無視しまた、残された項の係数も無視する。テキスト が含まれている画像

  自動的に生成された説明

計算量の比較

* 対象の量nが大きくなった時小さな割合で計算量が大きくなるアルゴリズムほど良いアルゴリズムと言える。
* いくつかの関数の例では次の順となるであろう



スクリーンショット が含まれている画像

自動的に生成された説明

空間計算量の求め方

* メモリ量を求める
  + １データ１メモリ（サイズがわかる場合はそのサイズ）
  + （中身が無名な）関数呼び出しも１メモリとしてカウント
    - 中身が明瞭な場合はそれも考慮
  + 分岐に入るか否かは最悪の場合を想定
    - 最悪空間計算量
    - 最良空間計算量、平均空間計算量もある