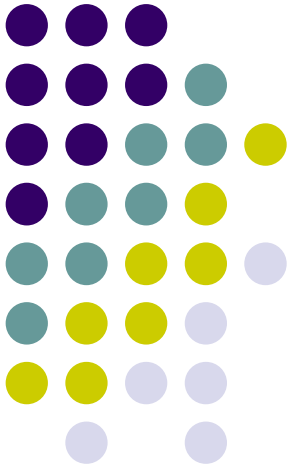
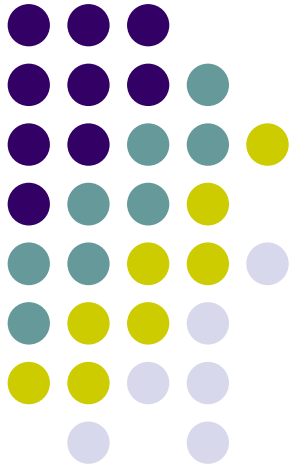
情報処理概論
後半4回目



今日の内容

- 再帰
  - 基本 (つづき)
  - ハノイの塔
  - ソート

再帰 (つづき)



再帰呼出しの基本

- (複雑そうな)問題の処理を
  - 1段簡単な問題の処理
  - それを用いた今の問題の処理

に分割する

- 一番簡単な問題に対する処理を考える

再帰呼出しの基本(つづく)

- 次のようなアルゴリズムによる関数を作る

- もしも，引数として一番簡単な問題が与えられたら
  - 一番簡単な問題に対する処理を行い解を返す
- そうでなければ
  - 一段簡単な問題を引数として自分自身を呼出し，その解を得る
  - 得た解を用いて問題を処理し解を得る

具体例
(順列よりちょっと簡単なものから)

- Nが与えられた時に，Nの階乗
  N!=1×2×3×．．．×(N-1)×N
  を求める
- 繰り返しだけでも簡単にできるけど．．．
  - F←1
  - Jを1からNまで繰り返す
    - F←F*J
- 練習ということで，再帰を使ったアルゴリズムを考えてみよう

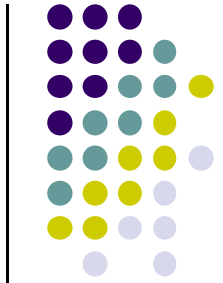
階乗を再帰を用いて計算するアルゴリズム

- 問題を分ける
  - 1段階簡単な問題: (n-1)!
  - その解を用いて問題を解く処理: n×(n-1)!
- 一番簡単な場合の処理: 0! = 1

階乗を求める関数fact(n)のアルゴリズム

  - もし引数が0なら
    - 1を返り値として帰る
  - そうでなければ
    - n-1を引数として自分自身を呼ぶ
    - 得られた返り値にnをかけた値を返り値として帰る

階乗を再帰を用いて計算するプログラム

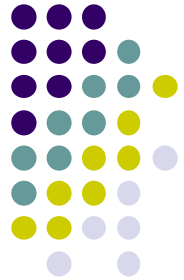


```
int fact(int n) {
  int r;
  if (n == 0) {
    r = 1;
  } else {
    r = fact(n-1)*n;
  }
  return r;
}
```

動作

- n=3の場合
- fact(3): fact(2)*3を計算しようとする
  - fact(2): fact(1)*2を計算しようとする
    - fact(1): fact(0)*1を計算しようとする
      - fact(0): 1を返り値として帰る
    - 1*1=1なので1を返り値として帰る
  - 1*2=2なので2を返り値として帰る
- 2*3=6なので6が答え

動作 (続き)



- n=4の場合
- fact(4): fact(3)*4を計算しようとする
  - fact(3): fact(2)*3を計算しようとする
    - fact(2): fact(1)*2を計算しようとする
      - fact(1): fact(0)*1を計算しようとする
        - fact(0): 1を返り値として帰る
      - 1*1=1なので1を返り値として帰る
    - 1*2=2なので2を返り値として帰る
  - 2*3=6なので6を返り値として帰る
- 6*4=24なので24が答え

動作 (続き)

- n=5の場合
- fact(5): fact(4)*5を計算しようとする
  - fact(4): fact(3)*4を計算しようとする
    - fact(3): fact(2)*3を計算しようとする
      - fact(2): fact(1)*2を計算しようとする
        - fact(1): fact(0)*1を計算しようとする
          - fact(0): 1を返り値として帰る
        - 1*1=1なので1を返り値として帰る
      - 1*2=2なので2を返り値として帰る
    - 2*3=6なので6を返り値として帰る
  - 6*4=24なので24を返り値として帰る
- 24*5=120なので120が答え

順列を再帰によって求めるアルゴリズム

- 問題を二つにわける
  - 一段簡単な問題: 要素数がひとつ少ない問題
  - 一段簡単な問題の解を用いた処理:
    - リストのどれかひとつに要素 + その要素を除いたリストにおける順列
    - リストの要素それぞれに対して上記を行う→繰り返し
- 一番簡単な問題に対する処理
  - 要素が1つしかない場合
  - その要素を解とする

順列を再帰によって求めるアルゴリズム (続き)

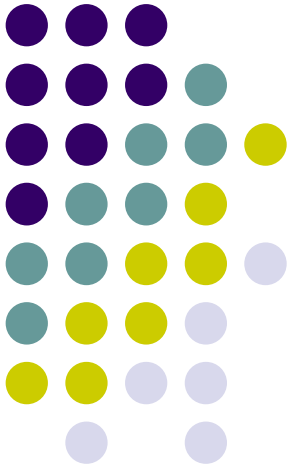
順列を求める関数permのアルゴリズム

- 引数1:順列生成対象のリスト S
- 引数2:その関数が呼ばれる外側ですでにできあがっている部分順列(出力用) T

順列を再帰によって求めるアルゴリズム (続き)

- もしSの長さが1ならば
  - Ｔと Ｓ を連結したものを出力
- そうでなければ
  - Sから要素をひとつずつ順次取出しながら(この要素を変数Aに入れる)以下を繰り返す
    - SからAを除いたリストをUとする
    - TにAを連結したリストをVとする
    - Uを引数1，Vを引数2としてpermを呼ぶ

確認問題11-1

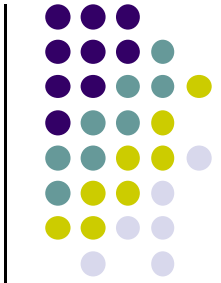


コンピュータの中では何が起っているのか

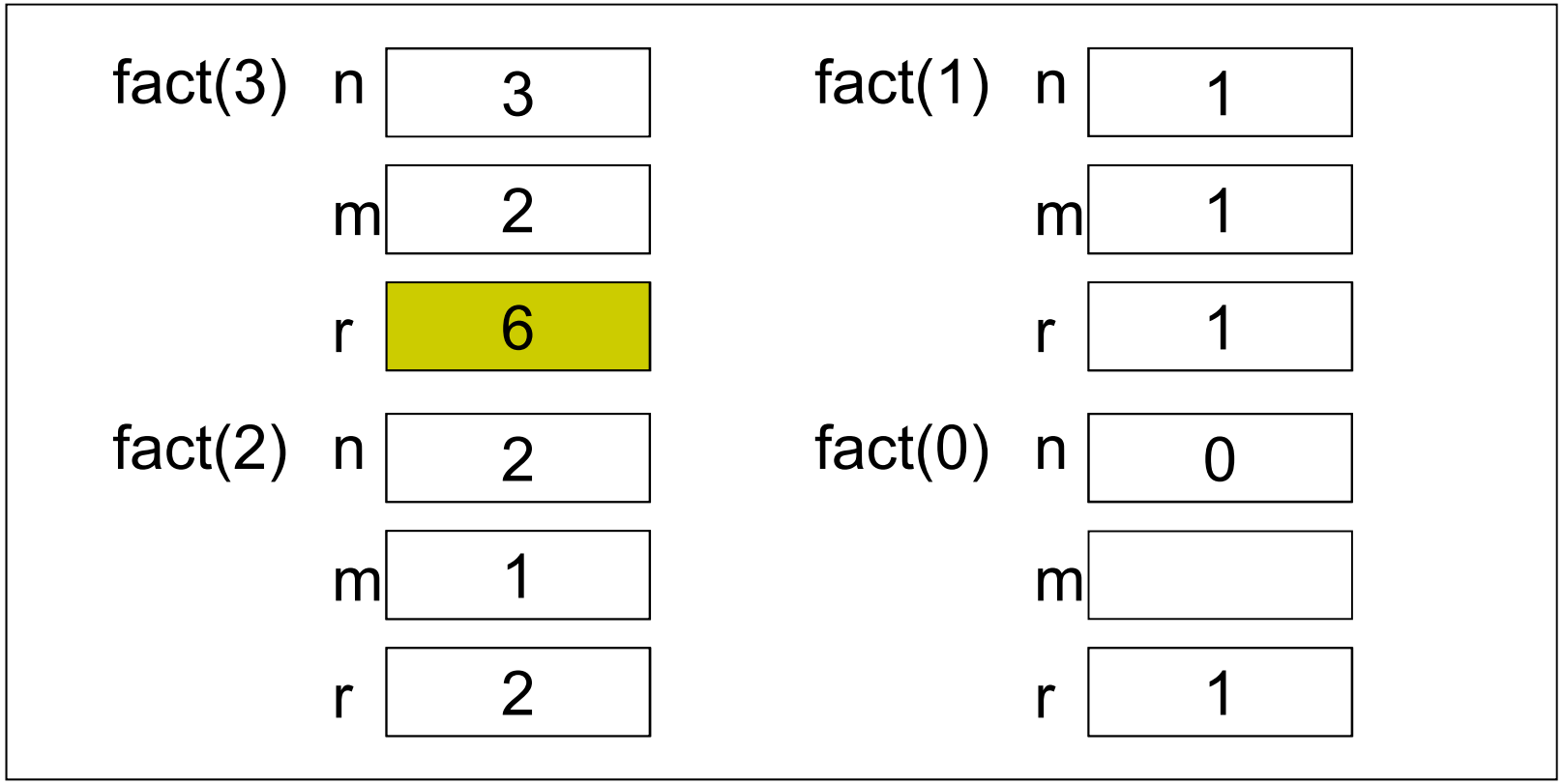
- 再び，階乗のプログラム
  - 動作をわかりやすくするため少し冗長に

```
int fact(int n) {
    int m,r;
    if (n == 0) {
      r = 1;
    } else {
      m = fact(n-1);
      r = n * m;
    }
    return r;
}
```

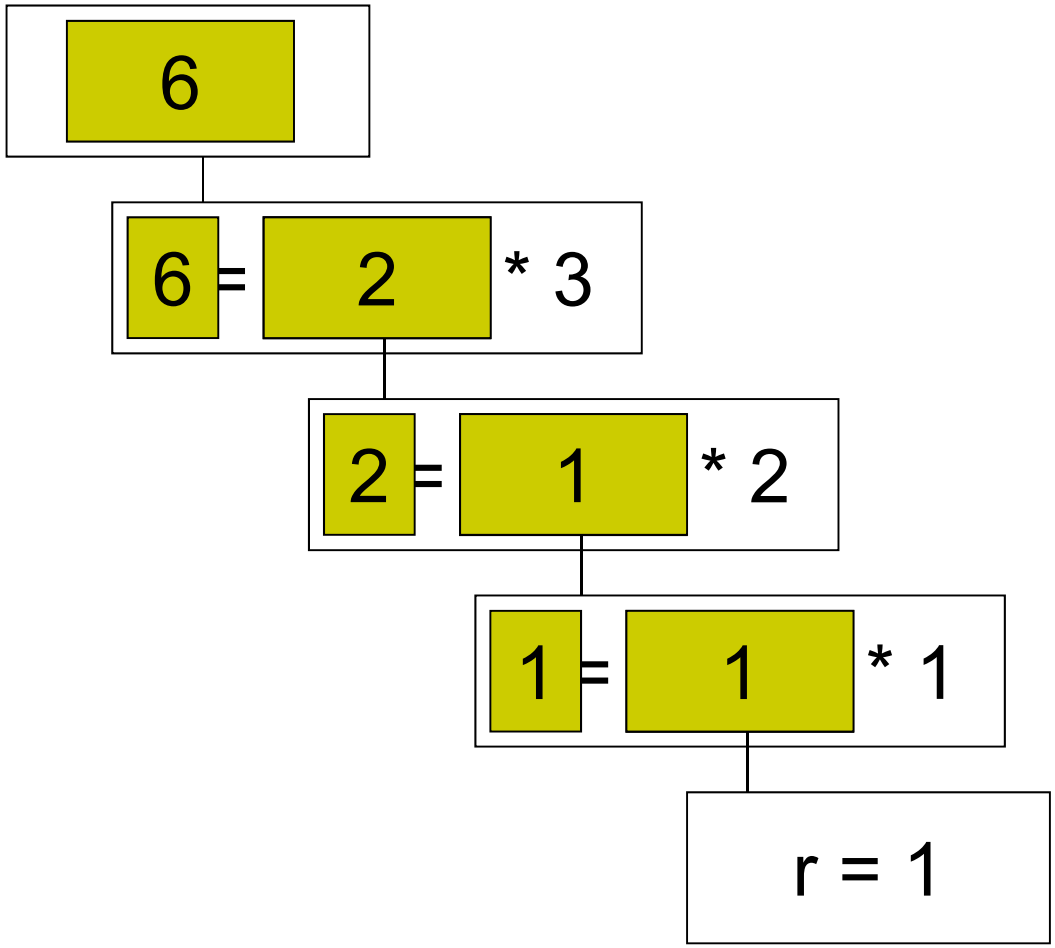
コンピュータの中では何が起っているのか（続き）



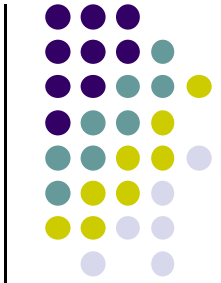
- n=3の場合



n=3の場合の動作



理解を助けるために．．．



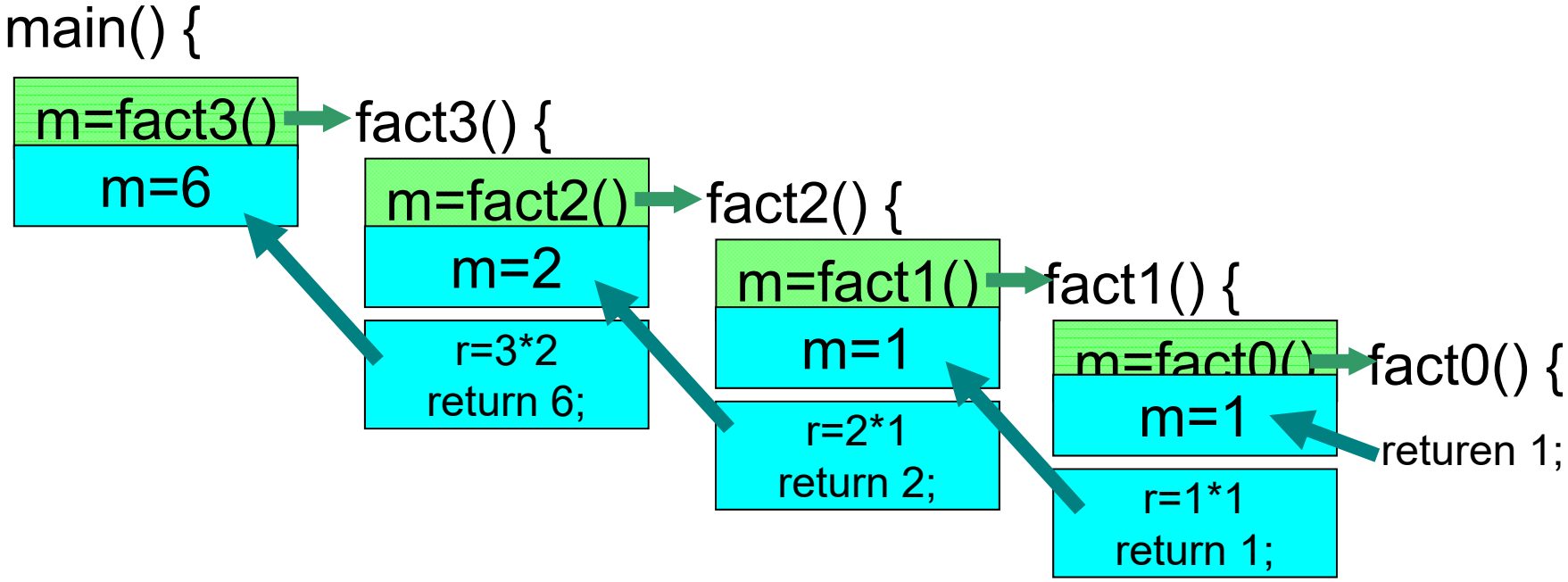
- 次のようなプログラムを考える

```
int main() {
  int m;
  m = fact3();
  printf(”3!=%d¥n”, m);
}

int fact3() {
  int m,r;
  m = fact2();
  r = 3 * m;
  return r;
}
```

```
int fact2() {
  int m,r;
  m = fact1();
  r = 2 * m;
  return r;
}

int fact1() {
  int m, r;
  m = fact0();
  r = 1 * m;
 return r;
}
```

```
int fact0() {
  return 1;
}
```

このプログラムの動作



ただし，これでは 3! しか計算できない
fact3(), fact2(), fact1()はやっていることは似ている

結局．．．

プログラム起動後に与えられる任意の数Nに対して，N!を計算するプログラム

```
int fact3() {
  int m,r;
  m = fact2();
  r = 3 * m;
  return r;
}
と
int fact0() {
  return 1;
}
```


```
int fact(int n) {
    int m,r;
    if (n == 0) {
      r = 1;
    } else {
      m = fact(n-1);
      r = n * m;
    }
    return r;
}
```

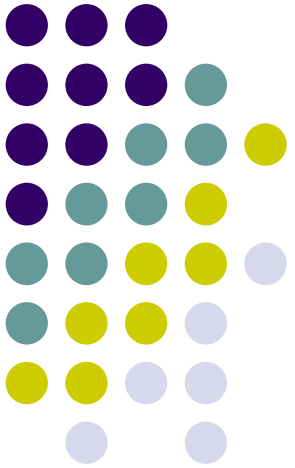
注目すべき点

- 変数について
  - 名前は同じでも，関数が呼び出されるごとに違う領域に確保されなければならない
    - fact(3)におけるnとfact(2)におけるnは同じ領域であってはならない

→ 局所変数(Cの場合は自動変数とも呼ぶ)

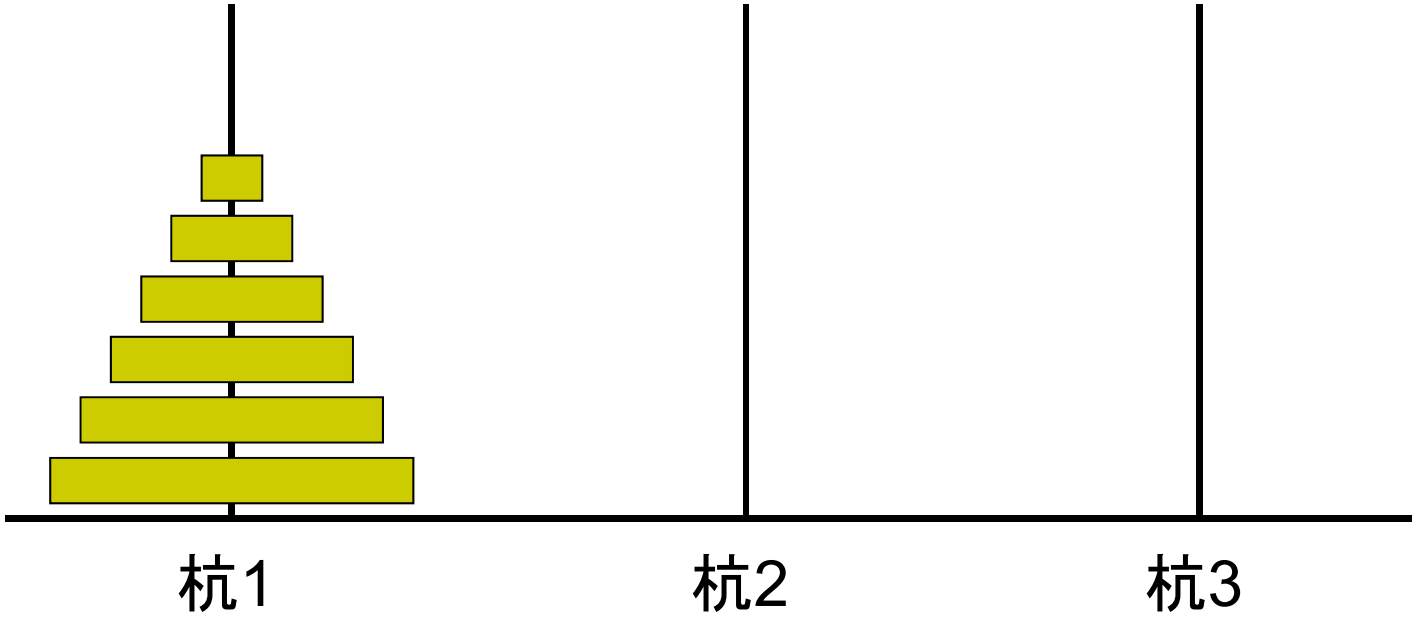
  - 関数が呼び出された時に変数領域が確保される
  - 関数が帰ると変数領域は開放される
- 引数について
  - 値渡しでなければならない

確認問題11-2



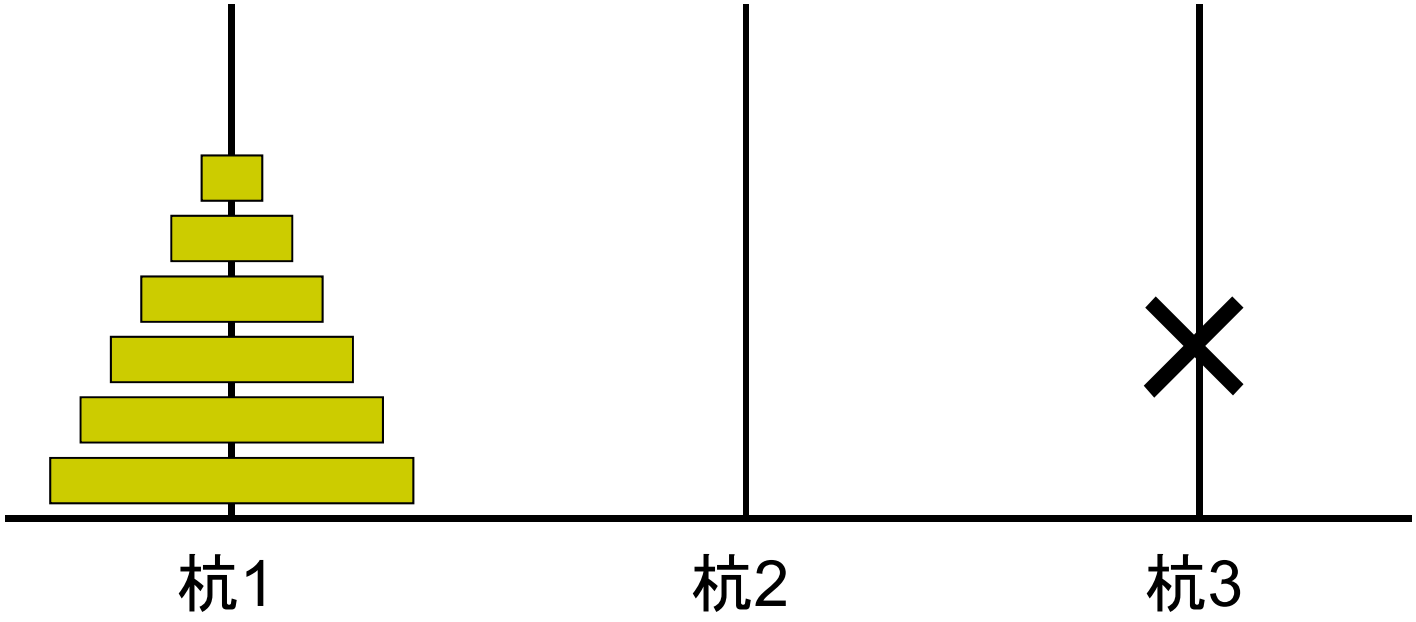
ハノイの塔

- 杭が3本
- 次第にサイズが小さくなる円盤がn個



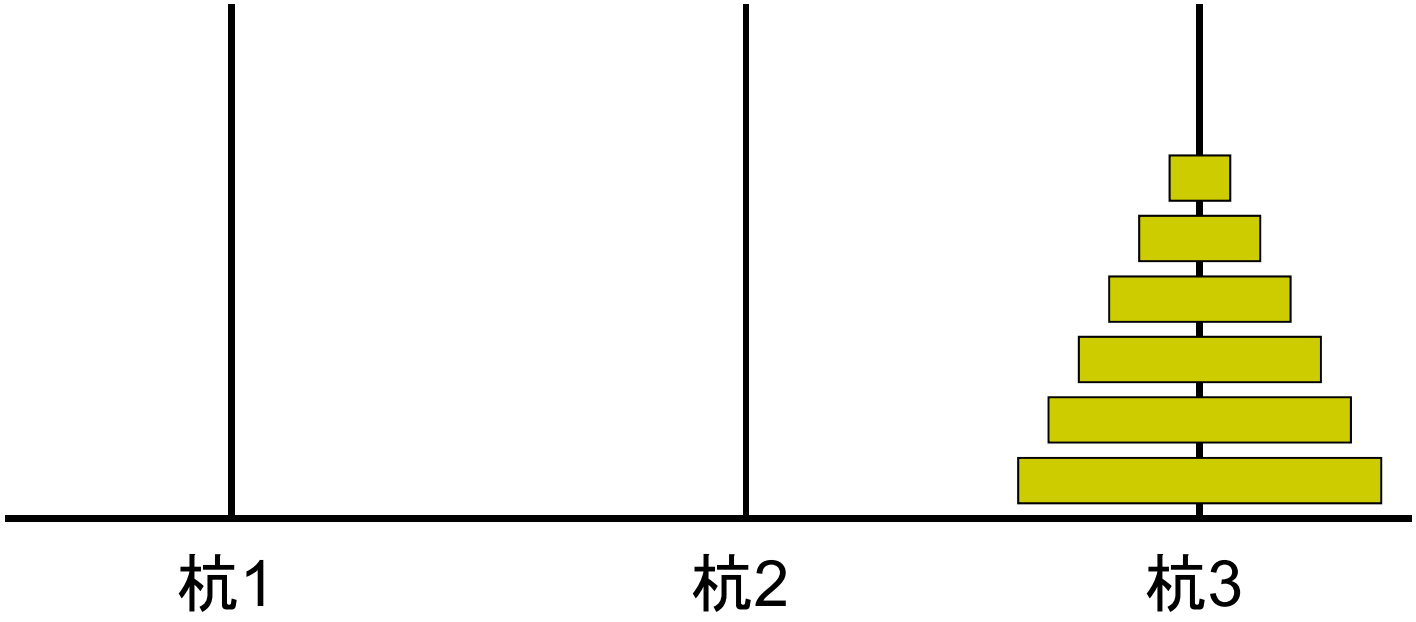
ハノイの塔 (続き)

- 杭から杭へと1枚ずつうごかす
- 決して下側の円盤が上の円盤より小さくならないように

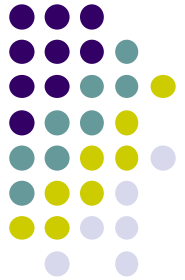


ハノイの塔 (続き)

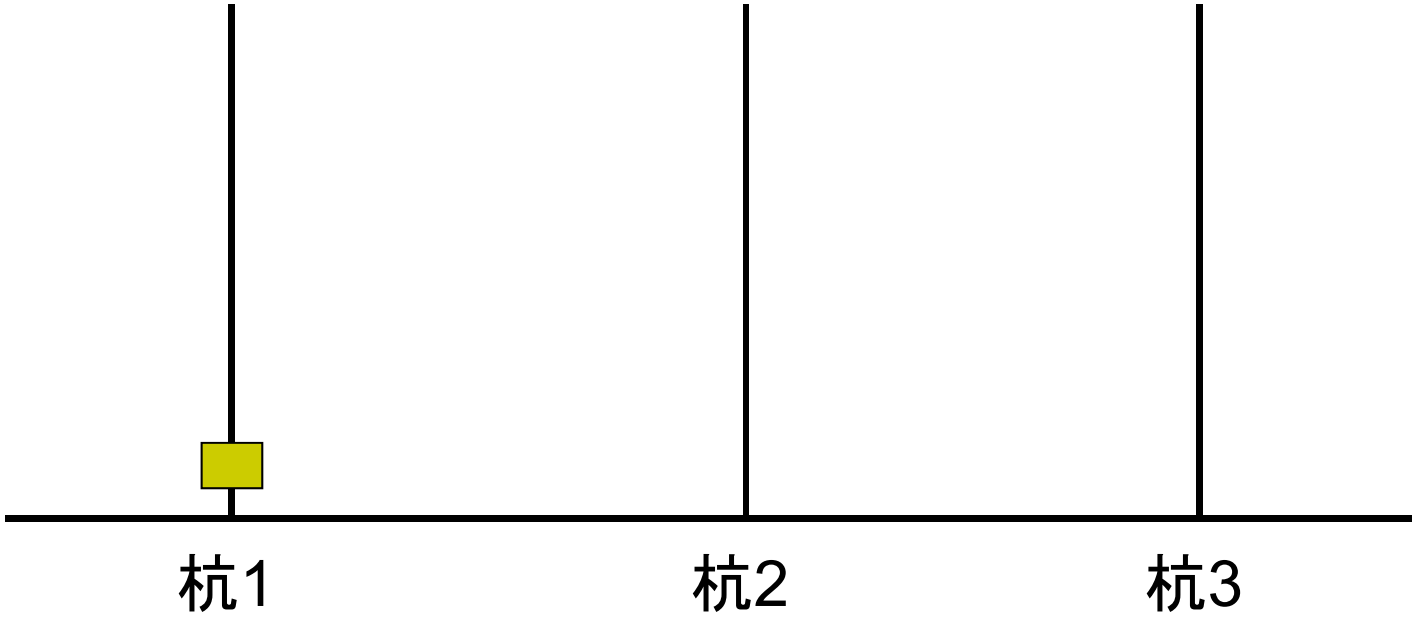
- すべての円盤を杭1から杭3に移す



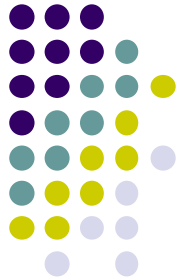
ハノイの塔
一番簡単な場合



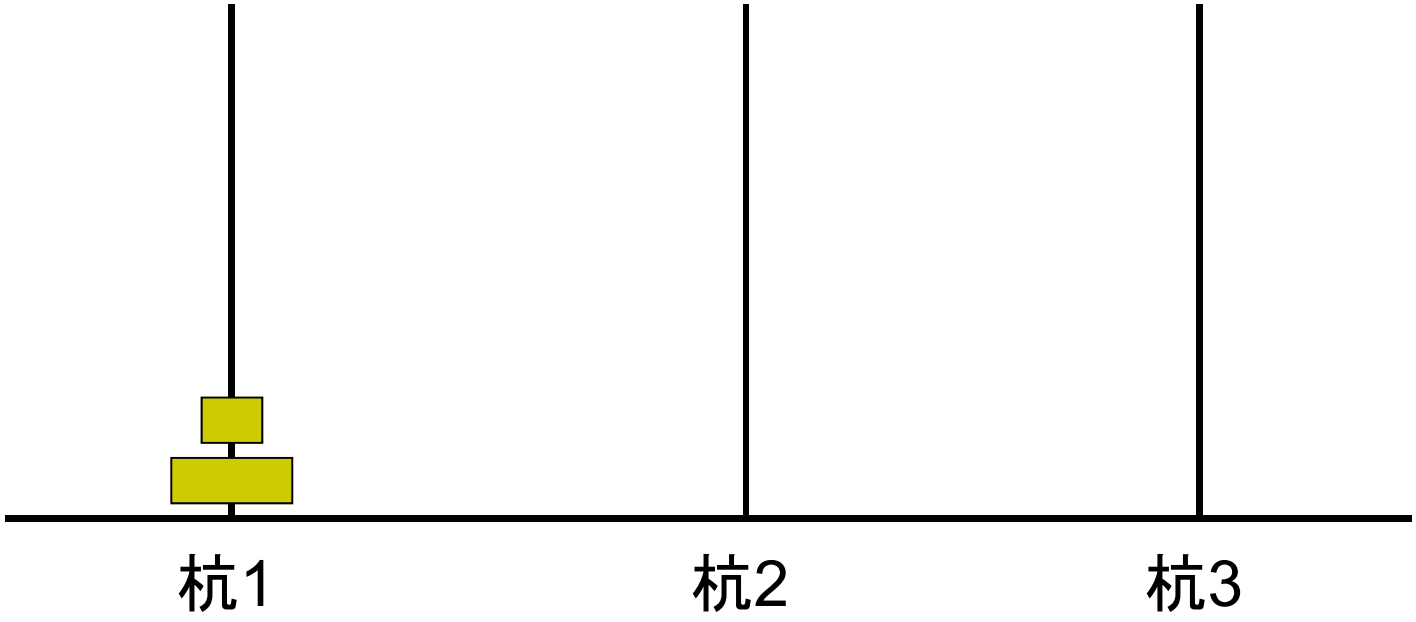
- n=1



ハノイの塔
もう少し考えてみる



- n=2

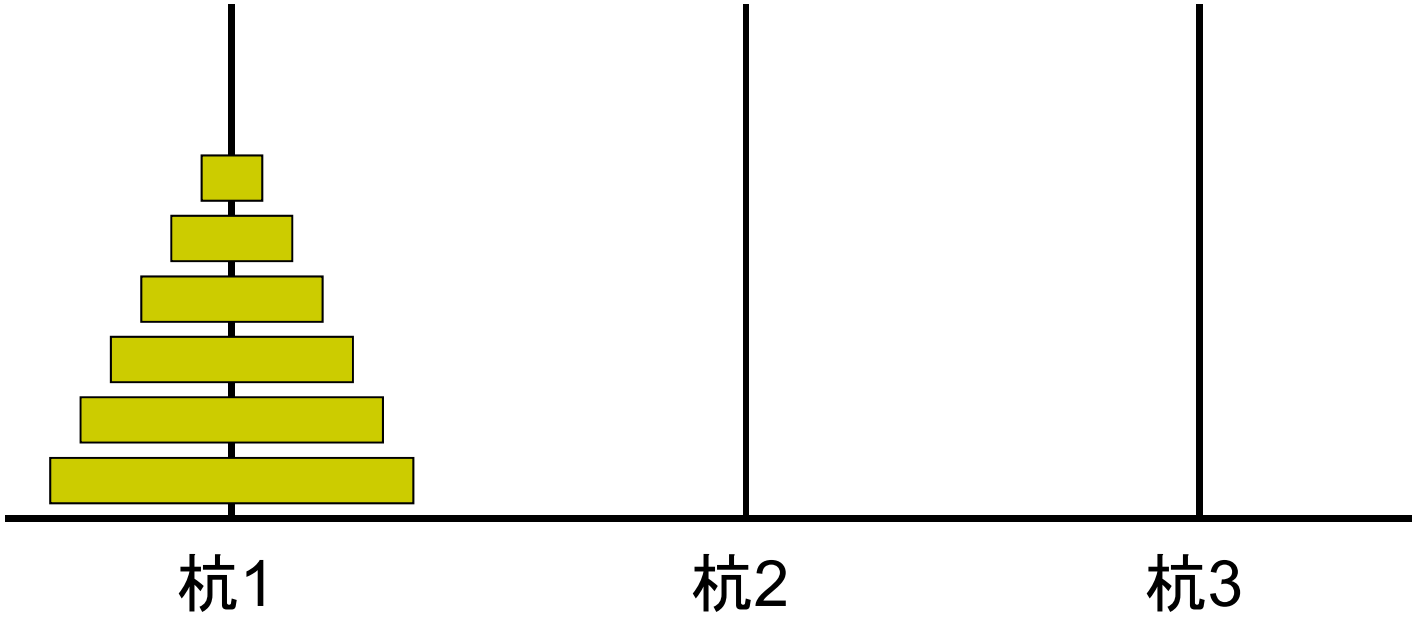


ハノイの塔
任意のnに対して

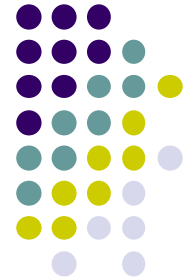
- n-1個の円盤を杭1から杭2に動かせたとする

→ 杭2から杭3へも動かせるはず

- 決して下側の円盤が上の円盤より小さくならないように



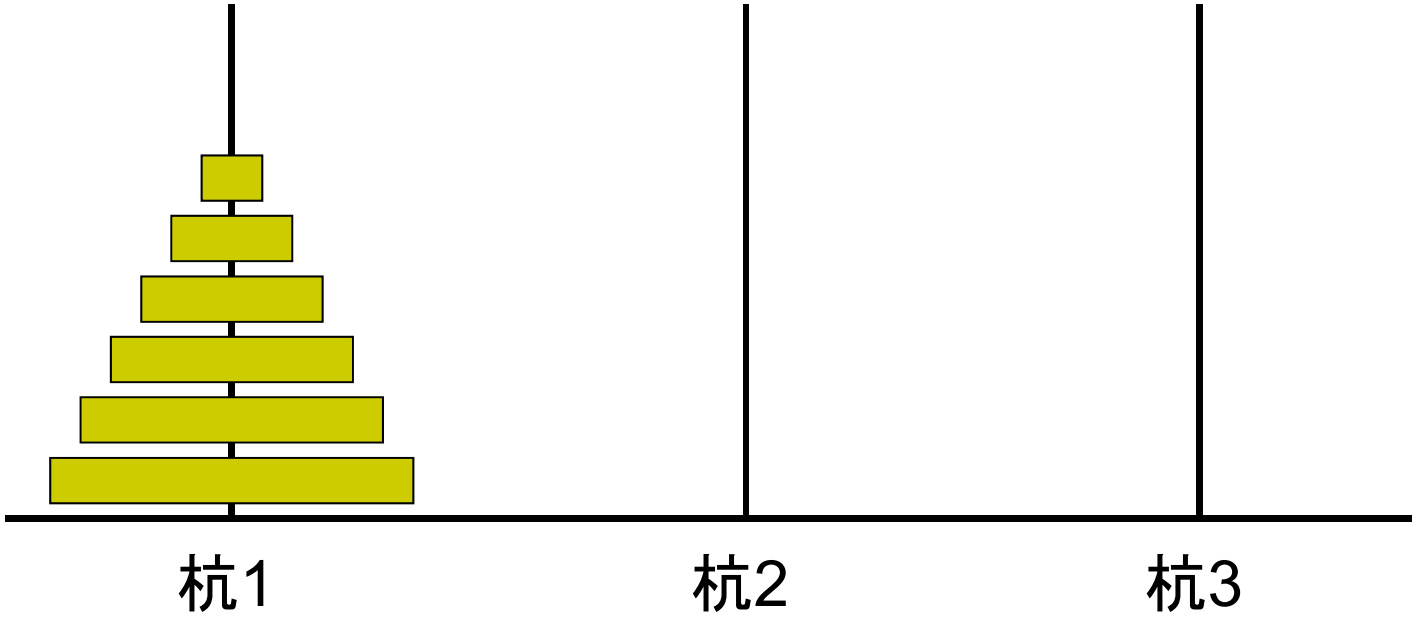
ハノイの塔
n-1個の円盤を杭1から杭2へ動かす



- n-2個の円盤を杭1から杭3に動かせたとする

→ 杭3から杭2へも動かせるはず

- 決して下側の円盤が上の円盤より小さくならないように



ハノイの塔
n個の円盤を杭xから杭yへ動かす

1. n-1個の円盤を杭xから杭zへ待避させる
2. n番の円盤を杭xから杭yへ動かす
3. n-1個の円盤を杭zから杭yへ動かす

- だたし n=1の時は

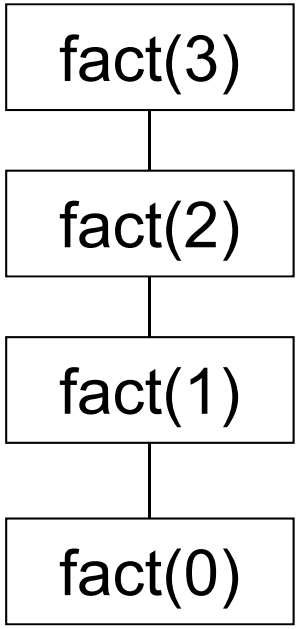
1. 1番の円盤を杭xから杭yへ動かす

ハノイの塔
アルゴリズム

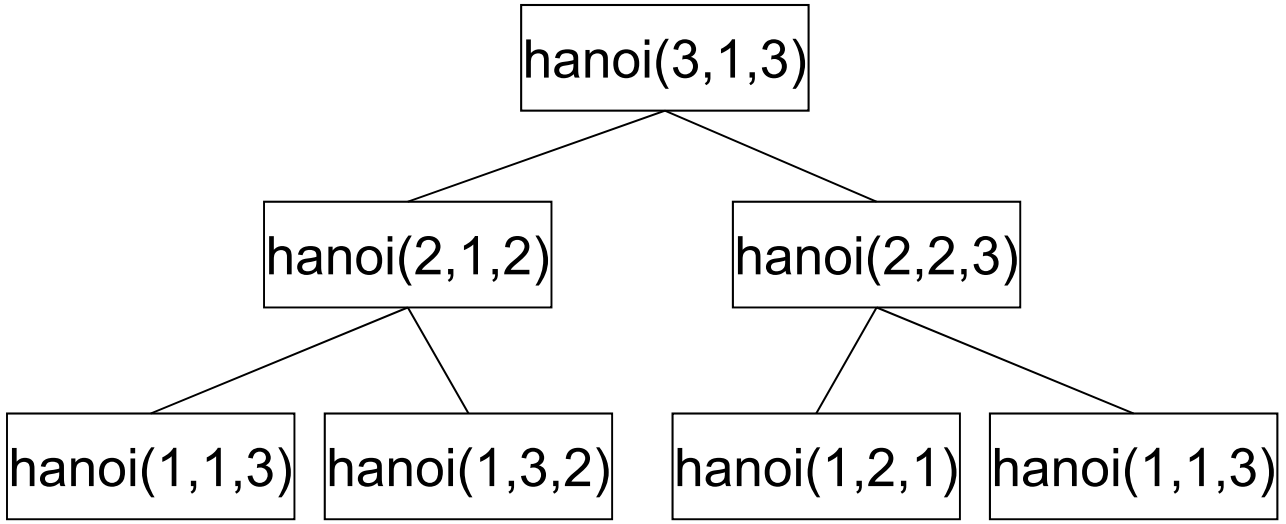
- 準備
  - 待避させる杭の番号を求める関数 tempPeg(x,y)
    - 1,2,3のうちxにもyにも指定されていない杭の番号を返す
- hanoi(n, x, y)
  1. もしも n が 1ならば 1番の円盤を x から y へ移して return
  2. そうでなければ,
     1. hanoi(n-1, x, tempPeg(x, y))
     2. n番の円盤をx から y へ移す
     3. hanoi(n-1, tempPeg(x, y), y)

階乗とハノイの塔の違い

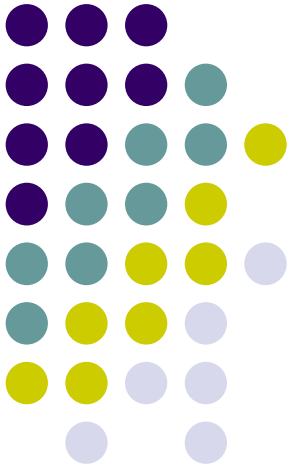
- 階乗での関数呼出し



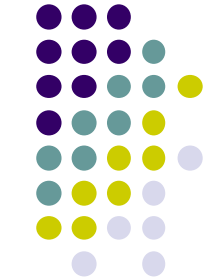
- ハノイの塔での関数呼出し



確認問題11-3

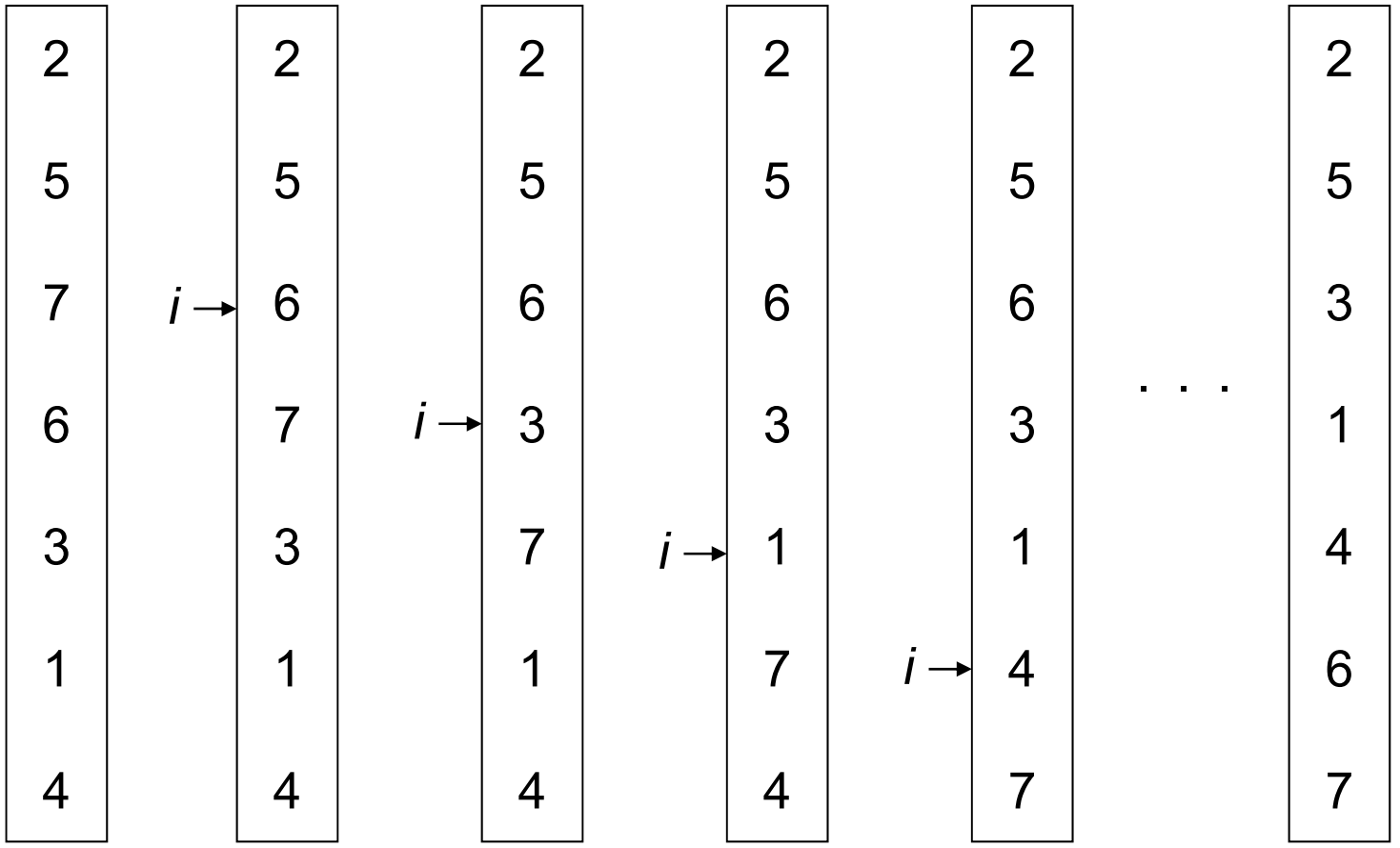


ソート

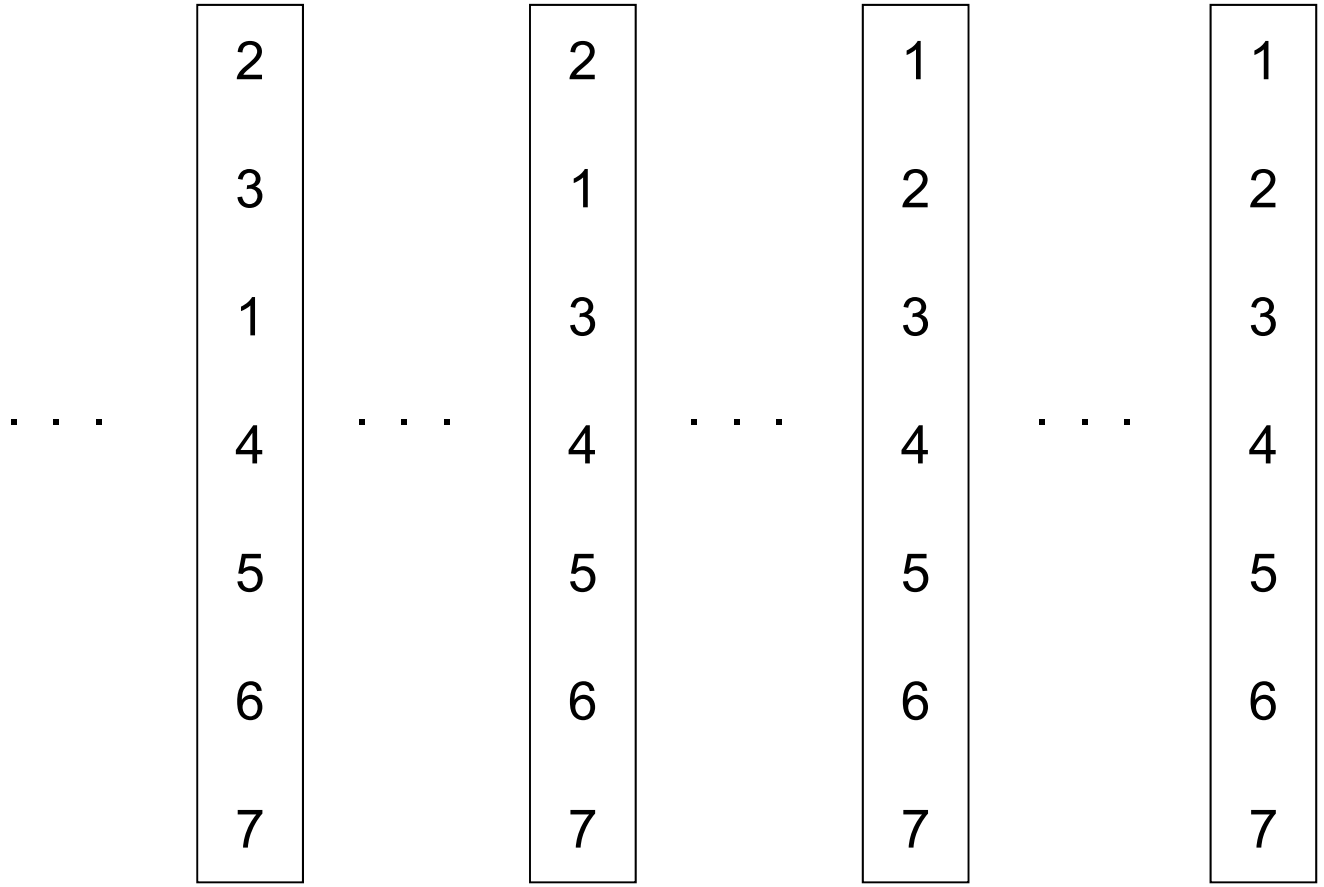


- ソートとは?
  - 与えられたリストLを値が小さい順(昇順)or大きい順(降順)に並べ替える
- 一番簡単なアルゴリズム: バブルソート
  - リストの要素数をNとする
  1. iを1からN-1まで変化させて
  2. (i+1)番目の要素がi番目の要素より小さければ入れ換える
  3. 1～2を入れ換えがなくなるまで繰り返す

バブルソートの動作例



バブルソートの動作例 (続き)



バブルソートの計算量

- 昇順になるようソートする
- 最良のケース
  - すべてが昇順に並んでいる場合: (N-1)回の比較
- 最悪のケース
  - すべてが降順に並んでいる場合:
    - 1巡目: (N-1)回の比較，(N-1)回の入換え
    - 2巡目: (N-1)回の比較，(N-2)回の入換え

      :
    - N巡目: (N-1)回の比較，0回の入換え
  - N×(N-1)の比較，N×(N-1)/2の入換え

バブルソートの計算量 (続き)

- 問題規模(今の例ではN)に対する計算量について，Nの変化に対する計算量の変化を表わしたもの
- すなわち，
  - 加算的な定数は無視
  - 固定値の乗数も無視

した，計算量とNとの関係を

計算量のオーダ

と呼び，*O(…)*で表わす

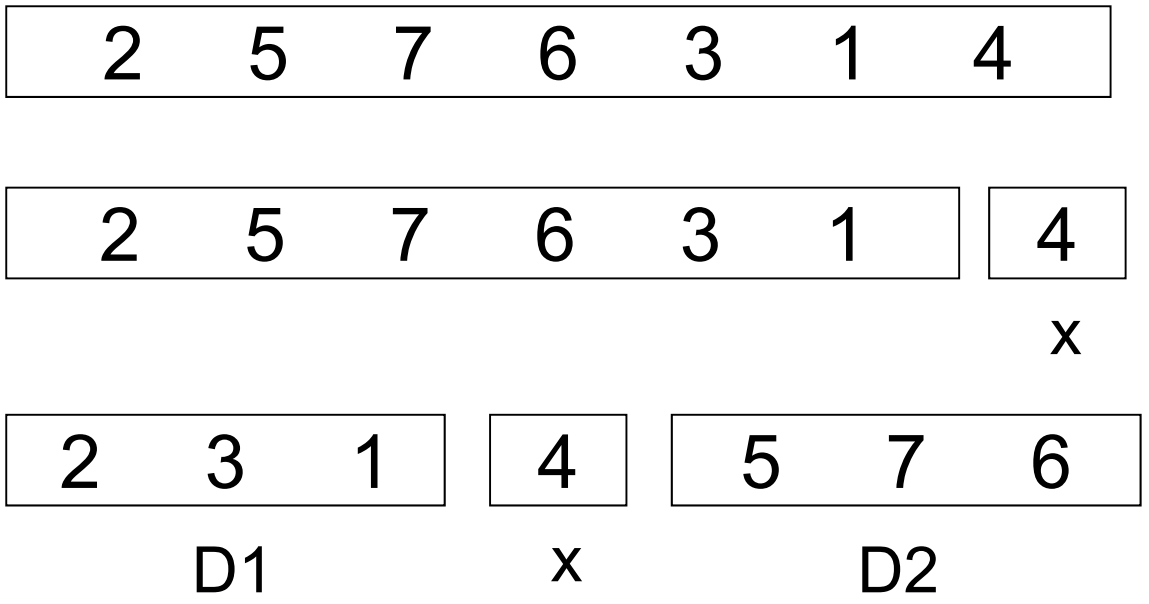
- バブルソートの計算量は $O(N^2)$

クイックソート

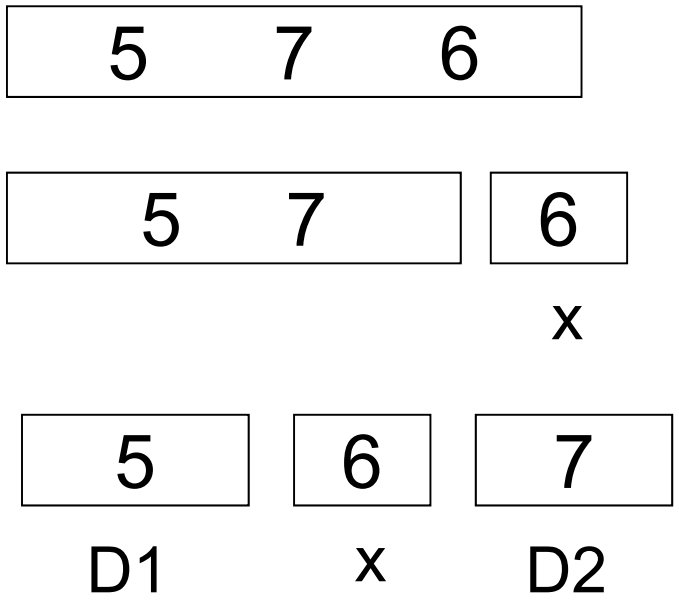
- アルゴリズム
  - Lの要素数が1ならば
    - 何もしない
  - そうでなければ
    - Lから一要素xを選択する
    - xより小さいLの要素をD1とする
    - x以上のLの要素をD2とする(ただしD2にはxは含まない)
    - D1をソートしてR1を得る
    - D2をソートしてR2を得る
    - R1, x, R2 の順にならべてソートされたリストとする

再帰呼出し

クイックソートの動作例



クイックソートの動作例 (続き)



クイックソートの計算量

- 1回の分割での比較回数: (N-1)
- 要素数Nのクイックソートの計算量

$$Q_N = N - 1 + Q_{N1} + Q_{N2}$$

  - ここでN1,N2は分割によって生じた左右の部分の長さ
  - N1+N2=N-1
- N1とN2は元のリストの並び方によって変るため計算量も異なるが，様々な場合が一様に発生するとして平均をとると

$$Q_N = N - 1 + 2(Q_0 + Q_1 + \cdots + Q_{N-1}) / N$$

$$Q_0 = 0, Q_1 = 1$$

- この漸化式を解くと

$$Q_N = 2N \log N \qquad \Rightarrow O(N \log N)$$

今日はここまで
次回予告

オブジェクト指向プログラミング

プログラムの実行時間