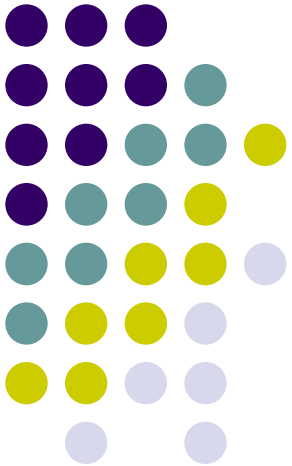
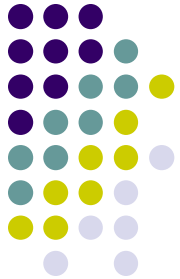
情報システム基盤学基礎1

Elements of Information Systems Fundamentals1

アルゴリズムとデータ構造　第2回



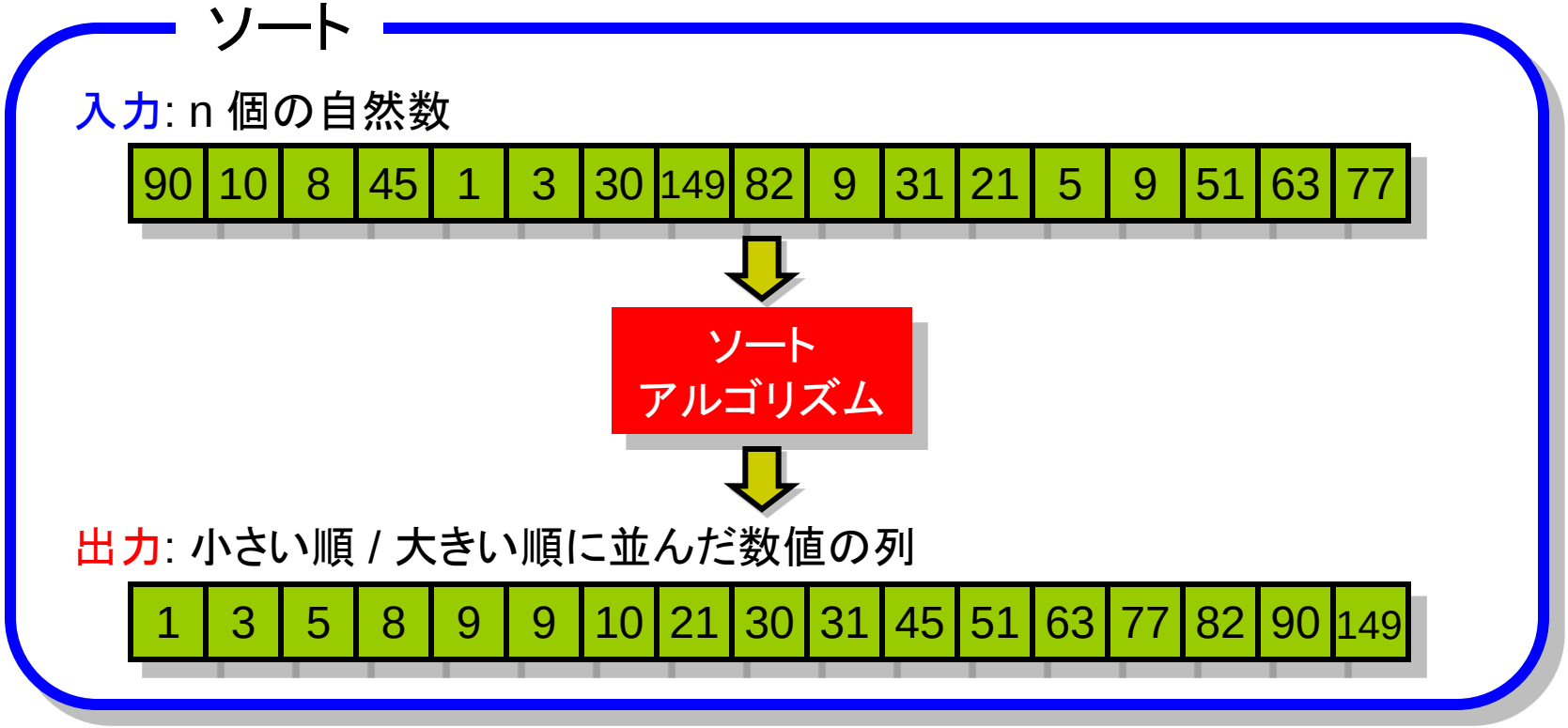
目次: sorting と分割統治法



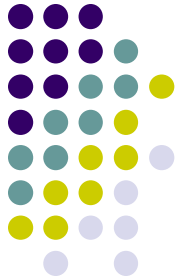
- ソートアルゴリズム
  - 挿入ソート
  - マージソート

Sorting

- ソート（整列）のアルゴリズム
- 計算量の見積り（big-o表記）



いろいろなソートのアルゴリズム



名前	計算量	手法
挿入ソート	$O(n^2)$	インクリメンタル
マージソート	$O(n \lg n)$	分割統治法
クイックソート	$O(n^2)$ 平均 $O(n \lg n)$	分割統治法

〔 n: 数の個数 〕

その他にも, バブルソート, ヒープソート等, たくさんある

挿入ソート（Insertion Sort）

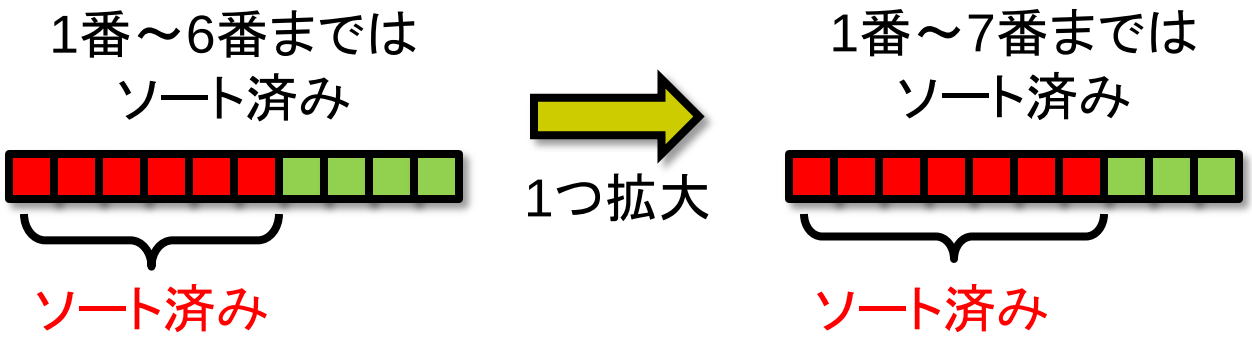
インクリメンタル(incremental)なアルゴリズム
（ソート済みの範囲を1個ずつ広げていく）

アイデア

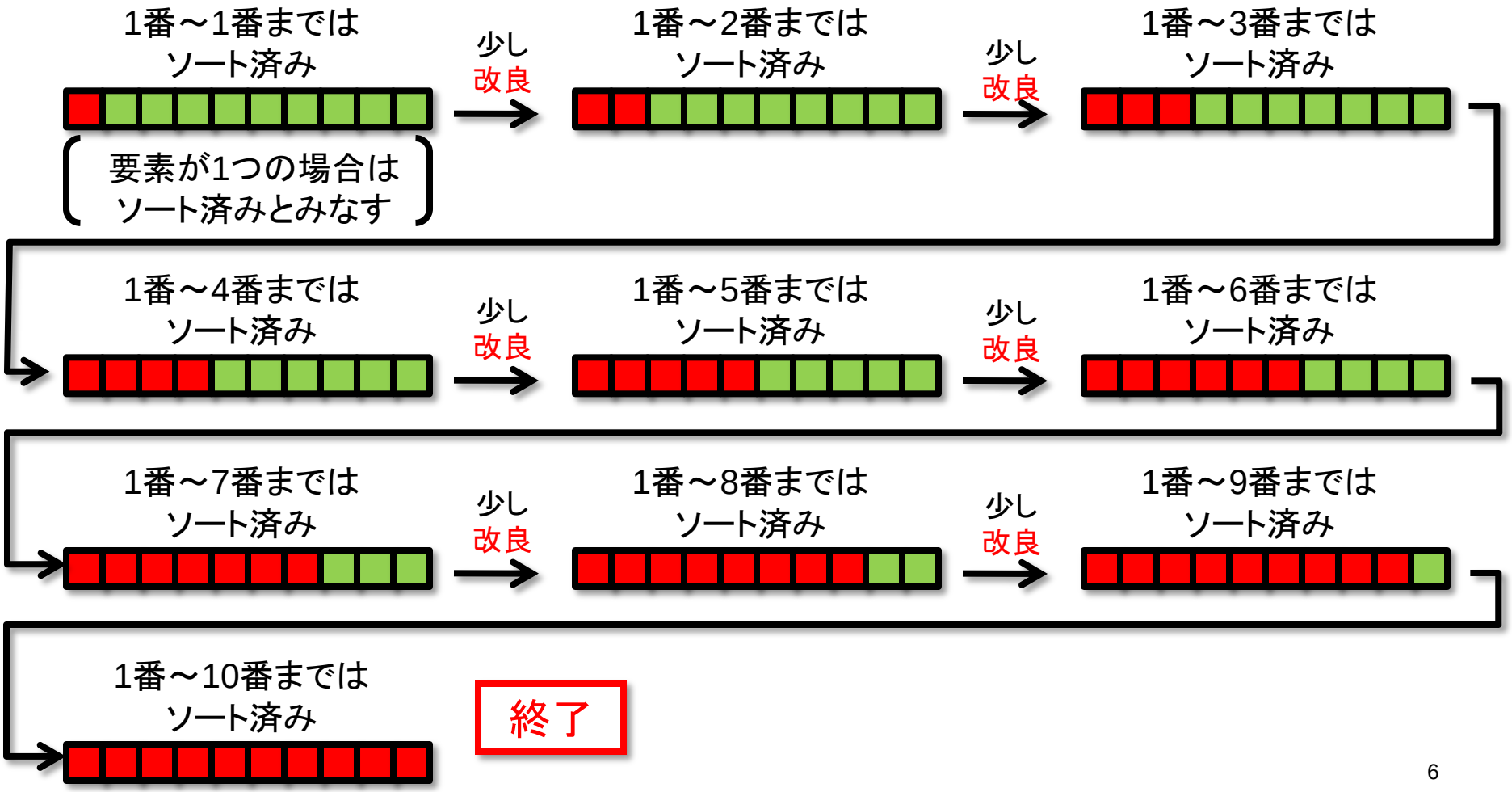
ソート済みの範囲を1個ずつ広げていく

10個の自然数があったとする

1番～6番まではソート済みだったとする



挿入ソートの全体像

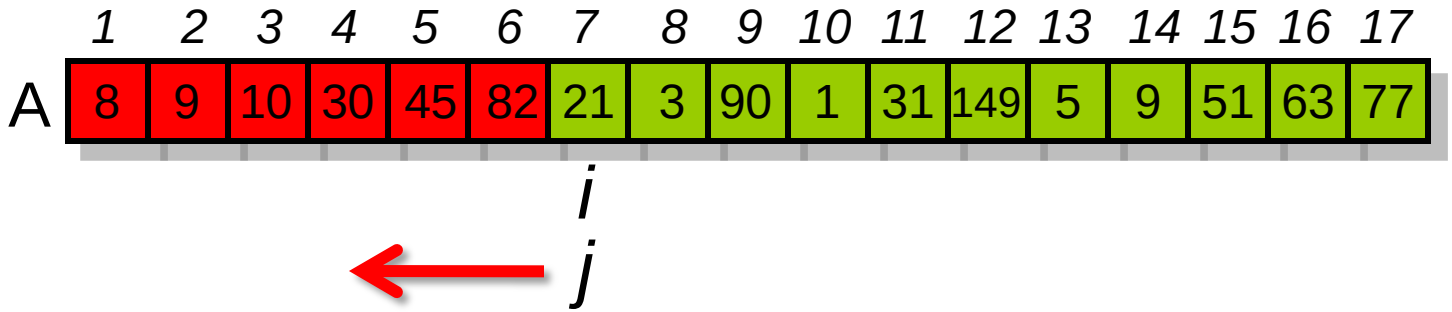

1番～1番まではソート済み
少し改良
1番～2番まではソート済み
少し改良
1番～3番まではソート済み
要素が1つの場合はソート済みとみなす
1番～4番まではソート済み
少し改良
1番～5番まではソート済み
少し改良
1番～6番まではソート済み
1番～7番まではソート済み
少し改良
1番～8番まではソート済み
少し改良
1番～9番まではソート済み
1番～10番まではソート済み
終了

挿入ソートの擬似コード

INSERTION-SORT(A)

1 **for** $i = 1$ **to** n
2 **do** $j \leftarrow i$
3 **while** ($A[j-1] \geqq A[j]$ かつ $j > 1$)
4 **do** $A[j-1]$ と $A[j]$ を交換
5 $j \leftarrow j-1$

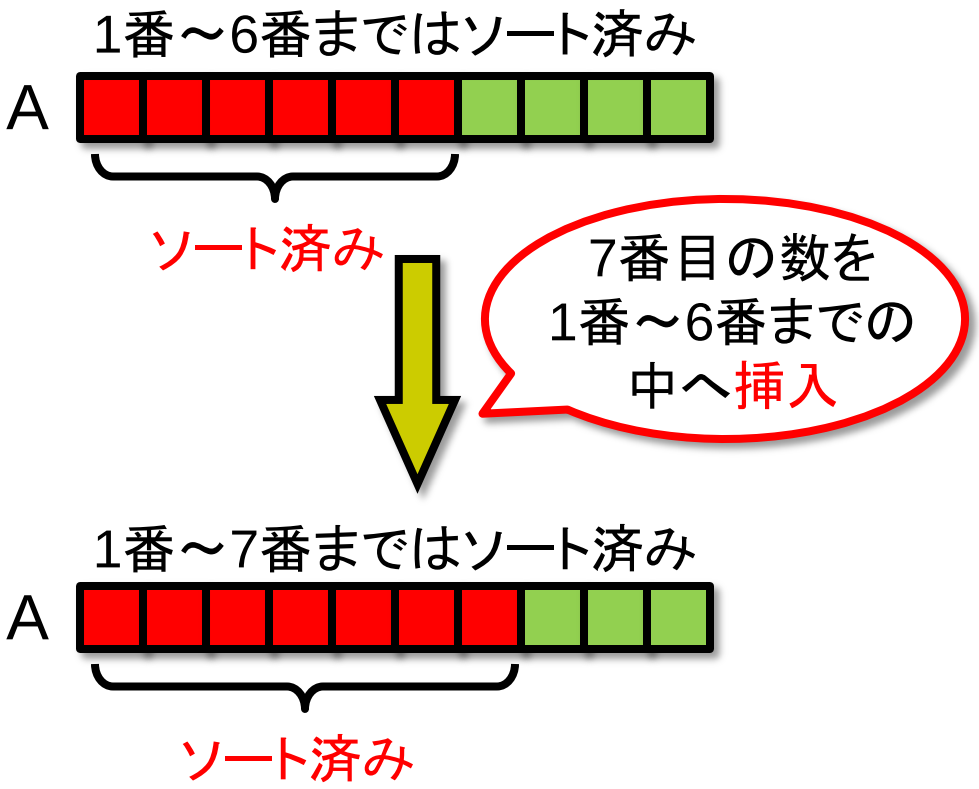
A[i]を追加する処理
＝ expand(i) 関数



1つ拡大する処理(expand)

10個の自然数があったとする

1番～6番まではソート済みだったとする



A[7] ≧ A[6] ?

YES → 終了

NO → A[7] と A[6] を交換

下へ続く。。。

A[6] ≧ A[5] ?

YES → 終了

NO → A[6] と A[5] を交換

下へ続く。。。

A[5] ≧ A[4] ?

YES → 終了

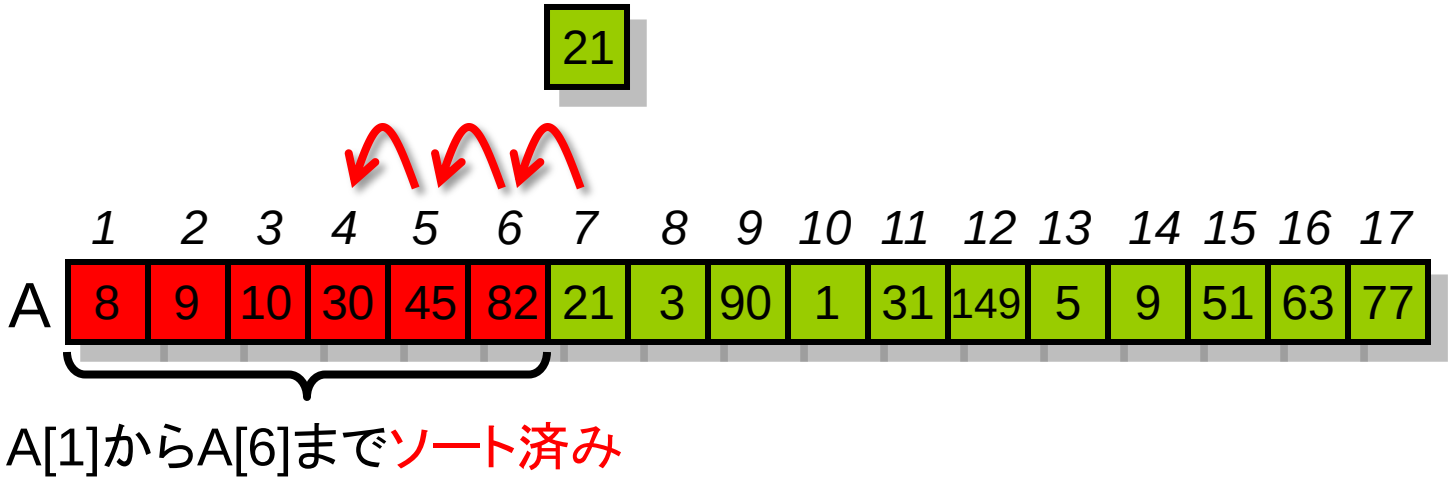
NO → A[5] と A[4] を交換

下へ続く。。。

■ ■ ■

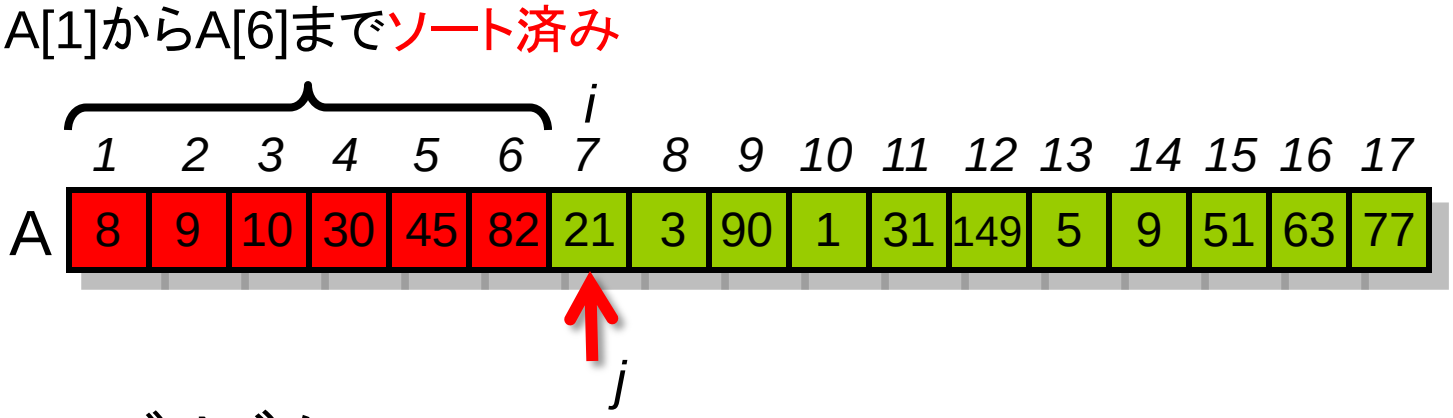
expandの処理: 例（直感）

A[7]を左へずらしていく
（自分よりも小さい要素が出現するまで）



A[1]からA[6]までソート済み

expandの処理: 例(具体的)



アルゴリズム

A[1] から A[*i*-1] までがソートされていると仮定

(*j* = *i* から始めて)

A[*j* -1] > A[*j*] の間は下記を繰り返す

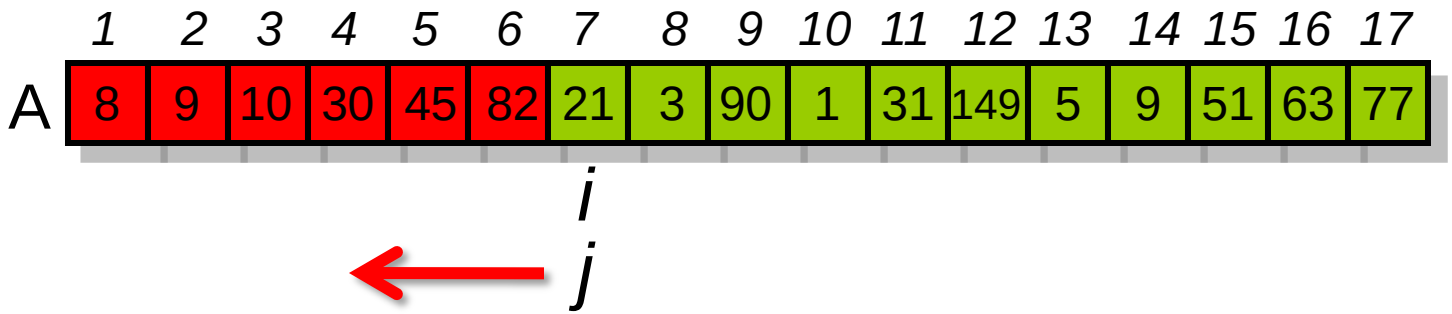
- A[*j* -1] と A[*j*] を交換
- *j* をデクリメント

挿入ソートの擬似コード

INSERTION-SORT(A)　　// Aは整数の配列

```
1    for i = 1 to n  do
          //   A[1] からA[i-1]まではソートずみの時に
          //   A[i]を追加してA[1..i]をソートする
2          j ← i
3          while  (A[ j - 1] ≧ A[ j ]  かつ  j > 1)  do
4                    A[ j – 1 ] と A[ j ] を交換
5                    j ← j – 1
```

expand(i)



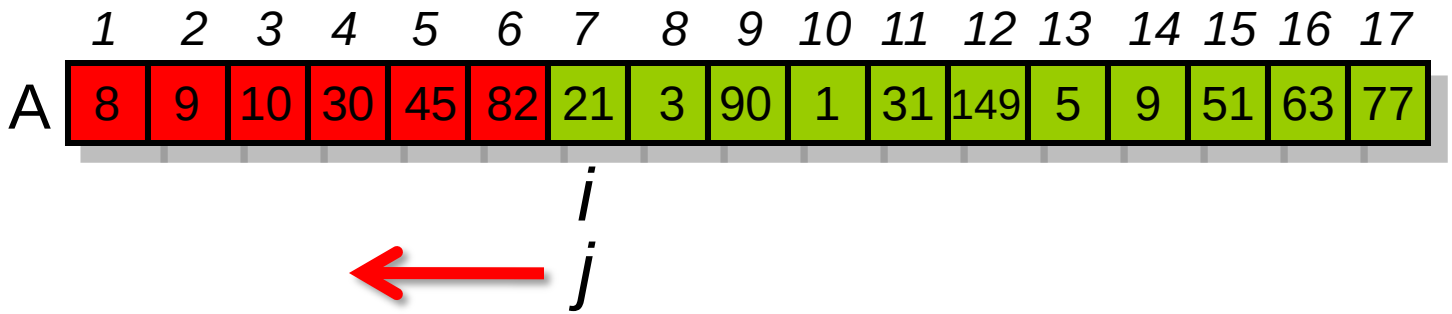
挿入ソートの擬似コード

INSERTION-SORT(A)

Loop invariant(ループ不変条件)を意識してアルゴリズムを組む

```
1    for i = 1 to n  do
          //  A[1] からA[i-1]まではソートずみの時に
          //  A[i]を追加してA[1..i]をソートする
2         j ← i
3         while  (A[ j - 1] ≧ A[ j ]  かつ  j > 1)  do
4              A[ j – 1 ] と A[ j ] を交換
5              j ← j – 1
```

expand(i)



計算量とは（復習）

アルゴリズムが終了するまでに実行する基本命令の回数
（※最悪の場合で考える）

↓

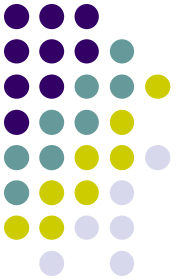
入力サイズの関数によって表現

- 比較
- 交換
- 代入 etc.

挿入ソートの場合

基本命令の回数を数えて，
n の式で表す

基本命令の回数を見積もる



INSERTION-SORT(A)

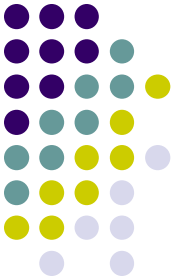
		各 i について
1	**for** $i = 1$ **to** n	1 回
2	**do** $j \leftarrow i$;	1 回
3	**while** $(A[j-1] \geqq A[j]$ かつ $j > 1)$	i 回
4	**do** $A[j-1]$ と $A[j]$ を交換	$i-1$ 回
5	$j \leftarrow j-1$;	$i-1$ 回

} 3 i 回

$$T(n) = \sum_{i=1}^{n} 3i = 3\sum_{i=1}^{n} i = \frac{3}{2} n(n+1) = \frac{3}{2}n^2 + \frac{3}{2}n$$

($T(n)$: n入力挿入ソートの計算時間)

Big-O表記で表現
（厳密ではないので注意!!）



Big-O表記の作り方

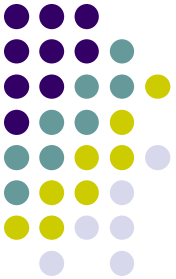
1. 式中で, 一番“効いてくる”項のみを考える

$$\frac{3}{2}n^2 + \frac{3}{2}n \Rightarrow \frac{3}{2}n^2$$

2. 係数を無視して、O( ) をつける

$$\frac{3}{2}n^2 \Rightarrow n^2 \Rightarrow O(n^2)$$

Big-O表記で見積もる



INSERTION-SORT(A)

		各 i について
1	**for** $i = 1$ **to** n	$O(1)$ 回
2	**do** $j \leftarrow i$;	$O(1)$ 回
3	**while** $(A[j-1] \geqq A[j]$ かつ $j > 1)$	$O(n)$ 回
4	**do** $A[j-1]$ と $A[j]$ を交換	$O(n)$ 回
5	$j \leftarrow j - 1$;	$O(n)$ 回

} $O(n)$ 回

$O(n)$ の命令を、各 i ($i = 1,2, \ldots n$)について行うので,

$$T(n) = O(n^2)$$

'=' は誤った書き方だが, 慣例的にこう書く

[$T(n)$: 入力サイズが n のとき, 挿入ソートの計算時間]

漸化式による計算量の導出

INSERTION-SORT(A)

		各 i について	
1	**for** $i = 1$ **to** n	$O(1)$ 回	$O(n)$ 回
2	**do** $j \leftarrow i$;	$O(1)$ 回	
3	**while** $(A[j-1] \geqq A[j]$ かつ $j > 1)$	$O(n)$ 回	
4	**do** $A[j-1]$ と $A[j]$ を交換	$O(n)$ 回	
5	$j \leftarrow j - 1$;	$O(n)$ 回	

T(n) = T(n-1) + c･(n-1). と書ける (c: 定数, n: データの大きさ)

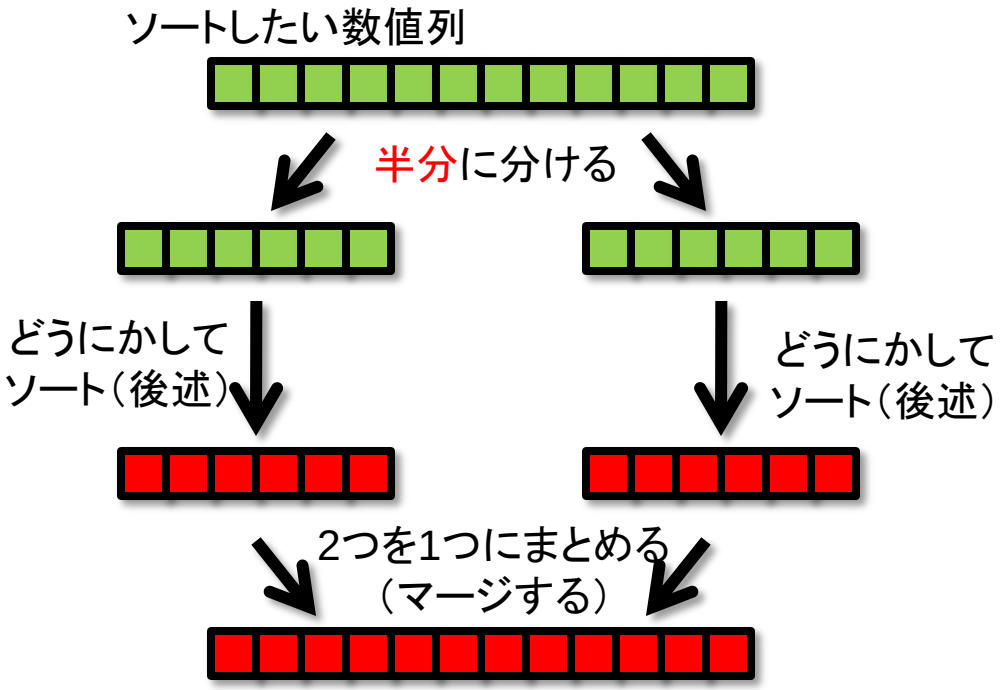
∴ T(n) = c ･((n-1) + (n-2) + …. + 2 + 1) = c ･(n･(n-1) /2)

= O(n^2)

($T(n)$: 入力サイズが n のとき, 挿入ソートの計算時間)

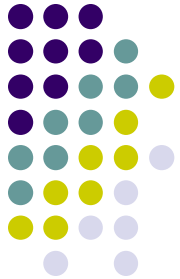
マージソート（Merge Sort）

*Merge*の意味：併合する，1つにまとめる



分割統治法に基づいたアルゴリズム

マージソート(MergeSort)の擬似コード



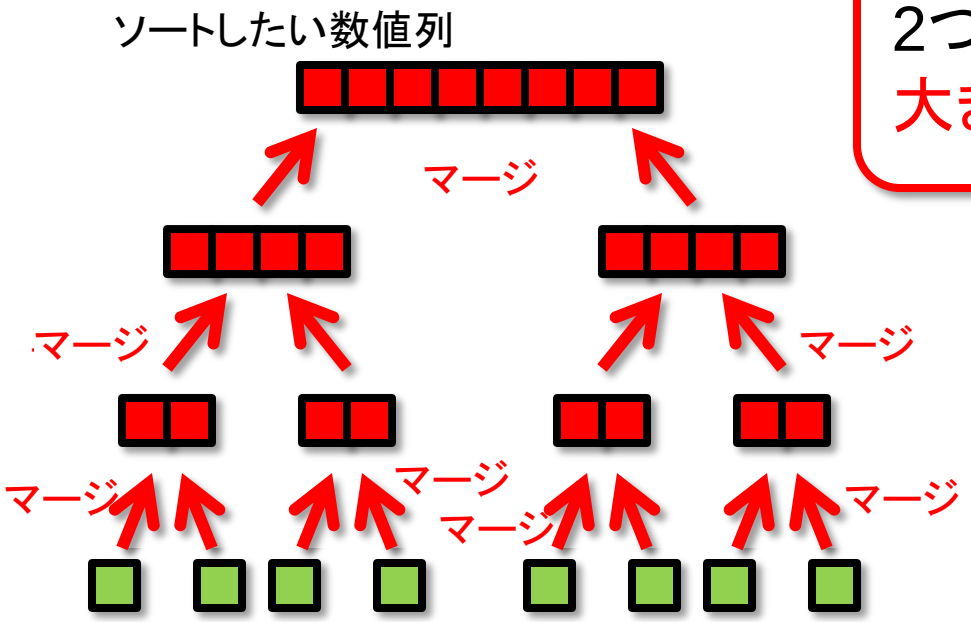
```
Sort(  A, i,   j   )      //  A[i] から A[j] を sorting

1.  If  ( i == j )   then  return A[i];
2.  else
3.              m  =  ( i + j -1 )/2
4.              return ( Merge( Sort( A, i, m ), Sort(A, m+1, j));
```

これだけ.　　Merge(Ax, Ay)　は
ソートずみの配列　Ax とAy を併合(merge)してソート済み配列を返す関数.

マージソートの全体イメージ

大まかな流れ: (1) できる限り2分割, (2) マージ処理



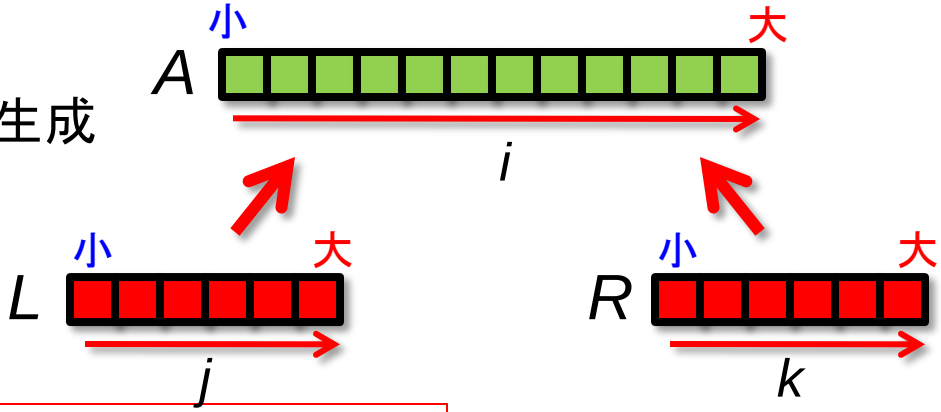
アルゴリズムの肝

2つのソート列から,
大きな1つのソート列を生成

2つのソート列から, 大きな1つのソート列を生成

ソートされた2つの配列 L, R から
ソートされた1つの大きな配列A を生成

- a : 配列A の長さ



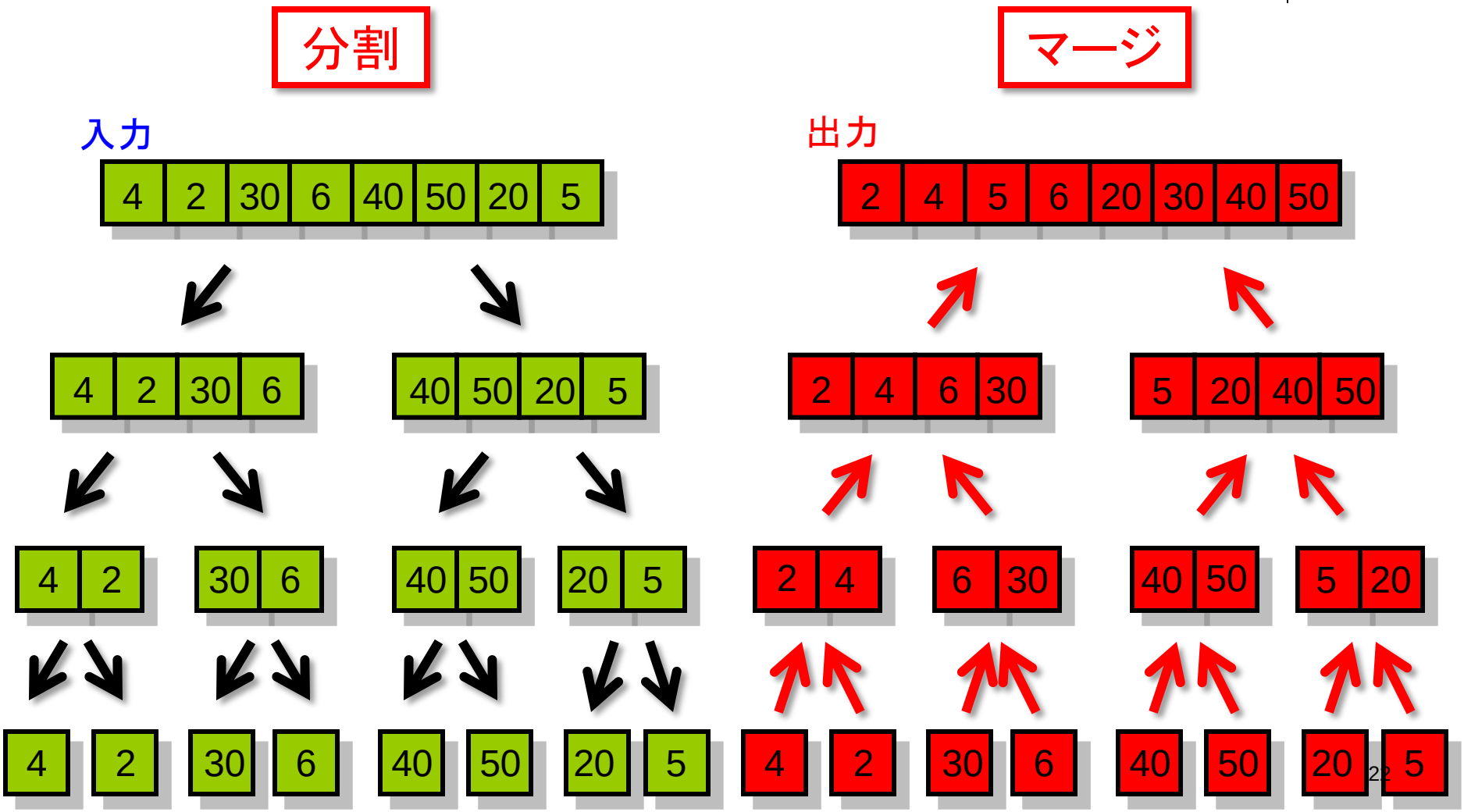
Merge(L, R) のアルゴリズム

初期化: $j \leftarrow 1$, $k \leftarrow 1$

各 $i = 1, 2, \ldots, a$ に対して

- **if** $L[j] < R[k]$
  { 気持ち: 現在の L の最小値と, 現在の R の最小値を比較 }
  then $A[i] \leftarrow L[j]$, $i \leftarrow i+1$, $j \leftarrow j+1$
  else $A[i] \leftarrow R[k]$, $i \leftarrow i+1$, $k \leftarrow k+1$

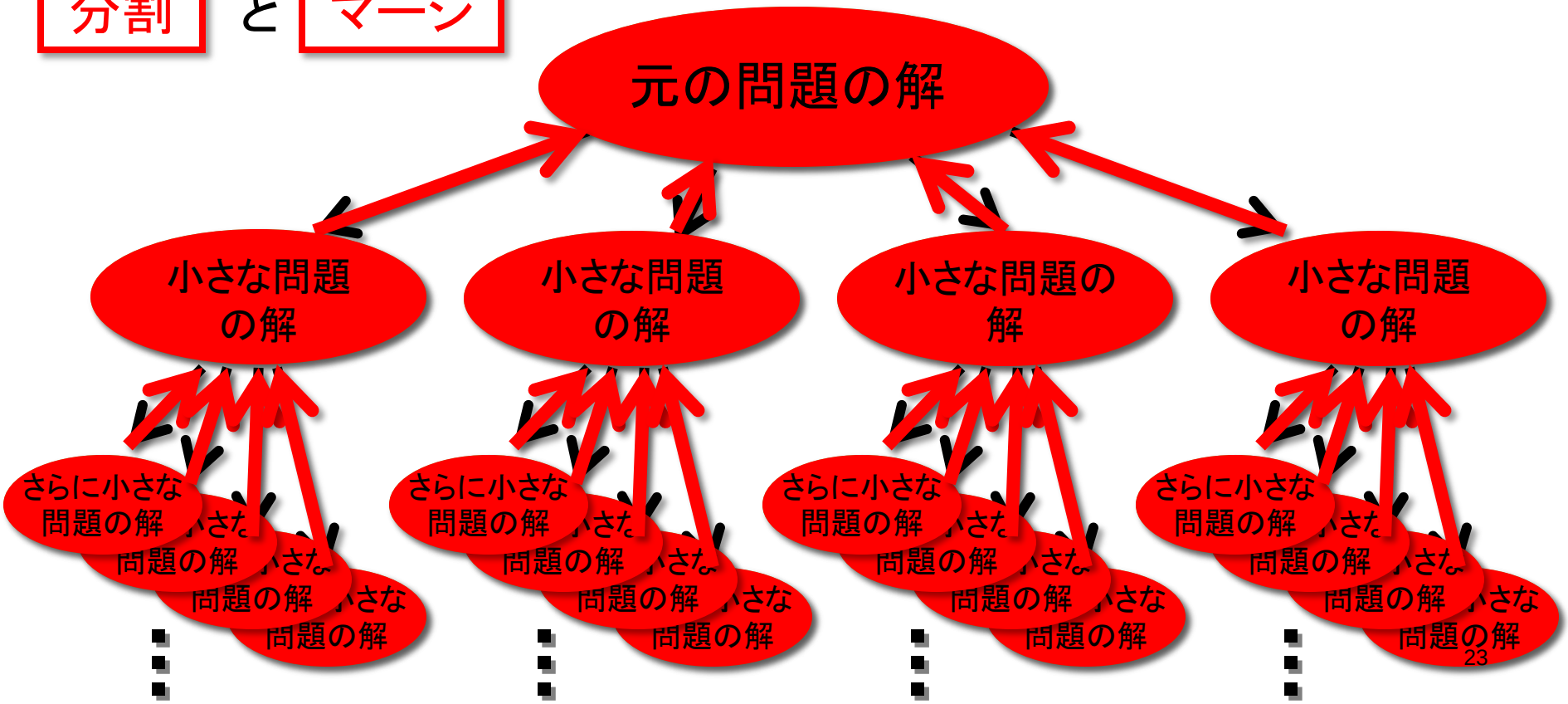
マージソートの実行例


分割
マージ
入力
出力
4 2 30 6 40 50 20 5
4 2 30 6
40 50 20 5
4 2
30 6
40 50
20 5
4
2
30
6
40
50
20
5
2 4 5 6 20 30 40 50
2 4 6 30
5 20 40 50
2 4
6 30
40 50
5 20
4
2
30
6
40
50
20
5

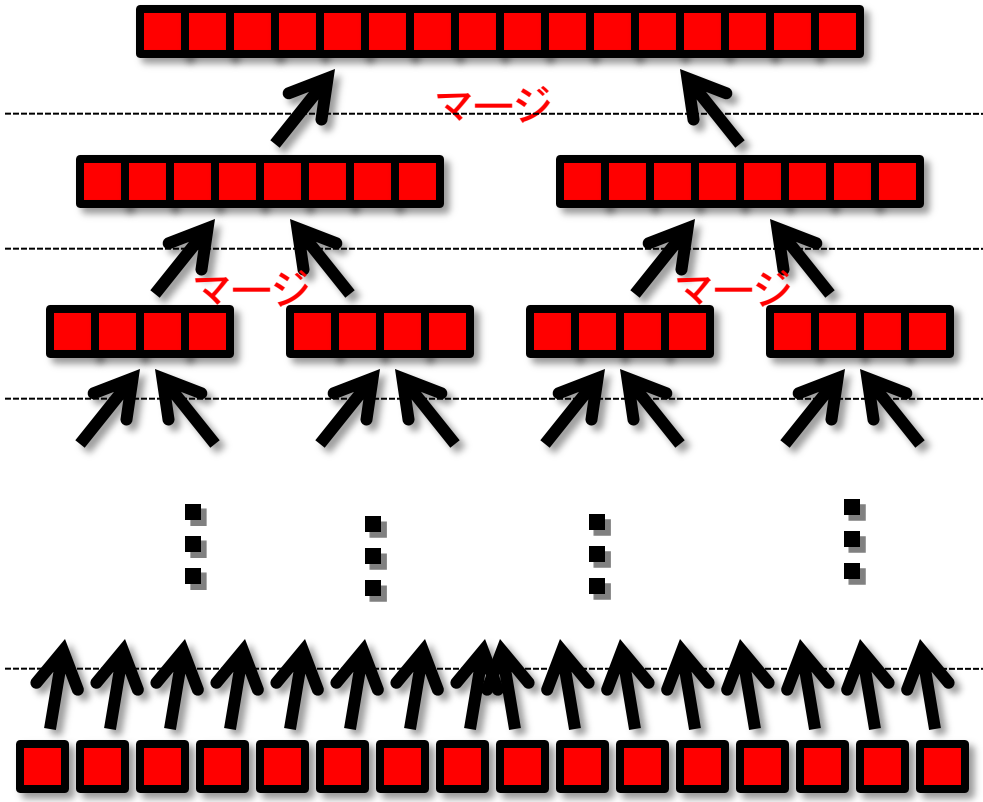
分割統治法（Divide-and-conquer）

マージソートのアルゴリズムをより抽象的に考えてみましょう

分割 と マージ



マージソートの計算量を見積もる
（注: 厳密ではないです）



各階層でマージのために必要な時間

左の合計

$O(n)$ ⇒ $O(n)$

$O(n)/2 + O(n)/2$ ⇒ $O(n)$

$O(n)/4 + O(n)/4 + O(n)/4 + O(n)/4$ ⇒ $O(n)$

2^i 個

$O(n)/2^i + \cdots + O(n)/2^i$ ⇒ $O(n)$

合計 $O(n \lg n)$ 時間

i : 木の深さ　$O(n)/2^i = 1$ より $i = \lg n$

漸化式による計算量の導出

$$T(n) = \begin{cases} 0 & (n=1) \\ 2T\left(\frac{n}{2}\right) + (n-1) & (n > 1) \end{cases}$$

T(n) = 2T(n/2) + (n-1) = 2^2T(n/4) + (n-1) + 2(n/2 -1) = ….
= 1(n/1) + 2(n/2) + 4(n/4) + … + 2^(log n) (1)
＝ O(n log n)

分割統治法（Divide-and-conquer）

大きさn の問題 P を解くとき，大きさ n/c の小さい問題　P(n/c) に分解して解き，その解を基にして P　の解を求める考え方.

$$T(n) = aT\left(\frac{n}{c}\right) + bn$$ と書ける.

$T(1) = b$

a, b, c は非負の定数.

分割統治法（Divide-and-conquer）

大きさn の問題 P を解くとき，大きさ n/c の小さい問題　P(n/c) に分解して解き，その解を基にして P　の解を求める考え方．

$$T(n) = aT\left(\frac{n}{c}\right) + bn$$

$T(1) = b$

a, b, c　は非負の定数．

$$T(n) = O(n) \ \ if \ \ a < c$$

$$T(n) = O(n \log n) \ \ if \ \ a = c$$

$$T(n) = O(n^k) \quad s.t. \quad k = log_c a \quad if \quad a > c$$

マスター定理
($n = 2^l$にすると
一般解を等比数列で
求められる）

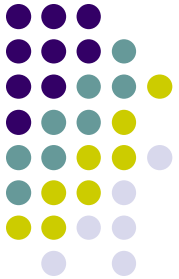
まとめ

- ソートのアルゴリズムを3つ紹介
  - 挿入ソート（計算時間：$O(n^2)$ 時間）
  - マージソート（計算時間：$O(n \lg n)$ 時間）

擬似コードを書くように「正当性がすぐに分かる」ようなプログラムを書く

＋　分割統治法 という問題の解き方：

$$T(n) = a \cdot T(n/c) + b \cdot n$$



おまけ

T(n) = 2 T(n/2) + n　なら　T(n) = O(n log n).

では:

T(n) = 3 T(n/2) + n なら？　　T(n) = n^log3

T(n) = T(n/5) + T(3n/4) + n なら？

→ T(n) = O(n).

// n個の要素集合からk番目に大きな要素を

// 見つける問題. Sortより速い.

付録： Quick Sort

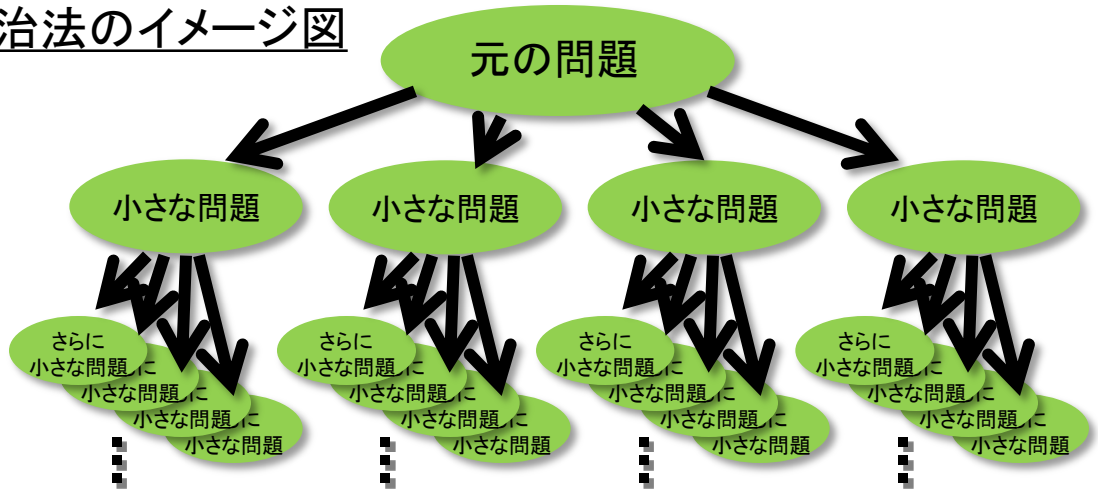
分割統治法にもとづいたアルゴリズム

マージソートと同じアイデア

⇩ 何が違う？

「どのように問題を分割するか?」が異なる

分割統治法のイメージ図



どうやって問題を分割するのか

アイデア

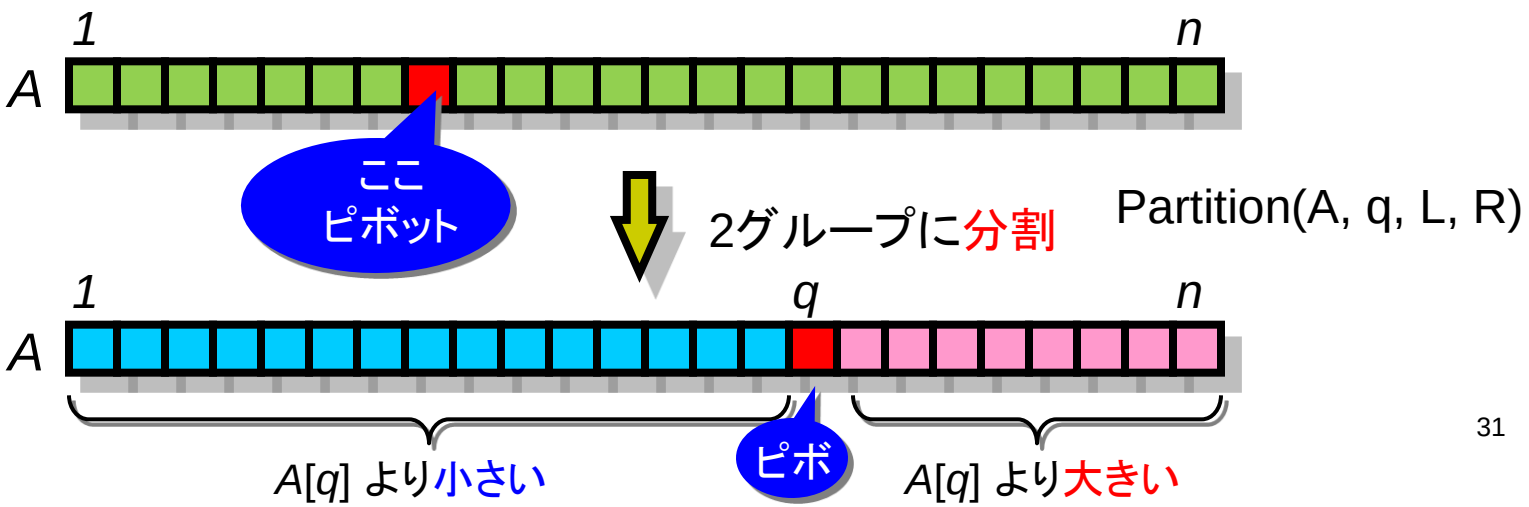
適当に選んだ自然数よりも

- 小さい自然数達と,
- 大きい自然数達

に分ける

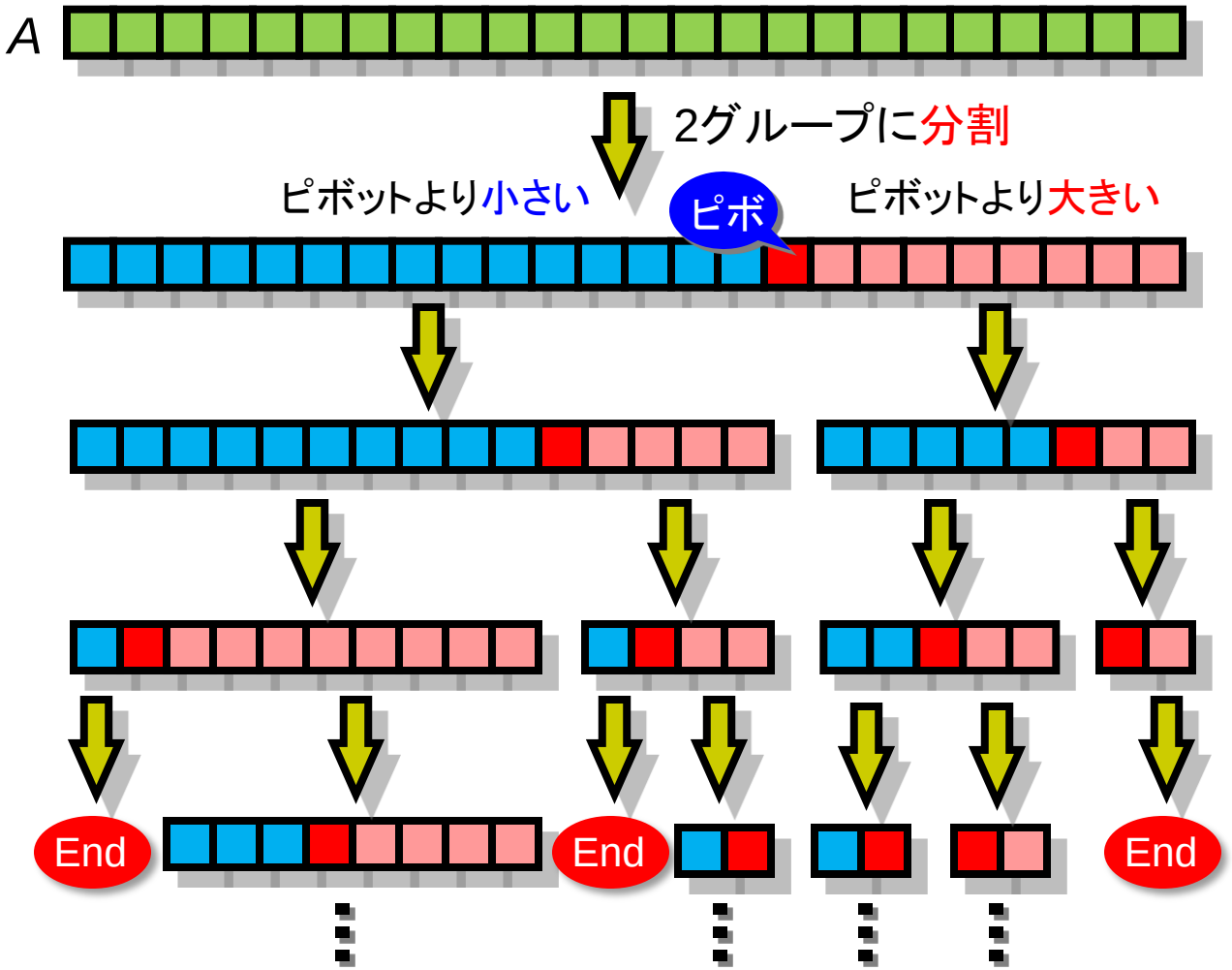
Pivot (番兵)

適当に選んだ自然数をピボットと呼ぶ



クイックソートの全体像

2グループへの分割を再帰的に実行

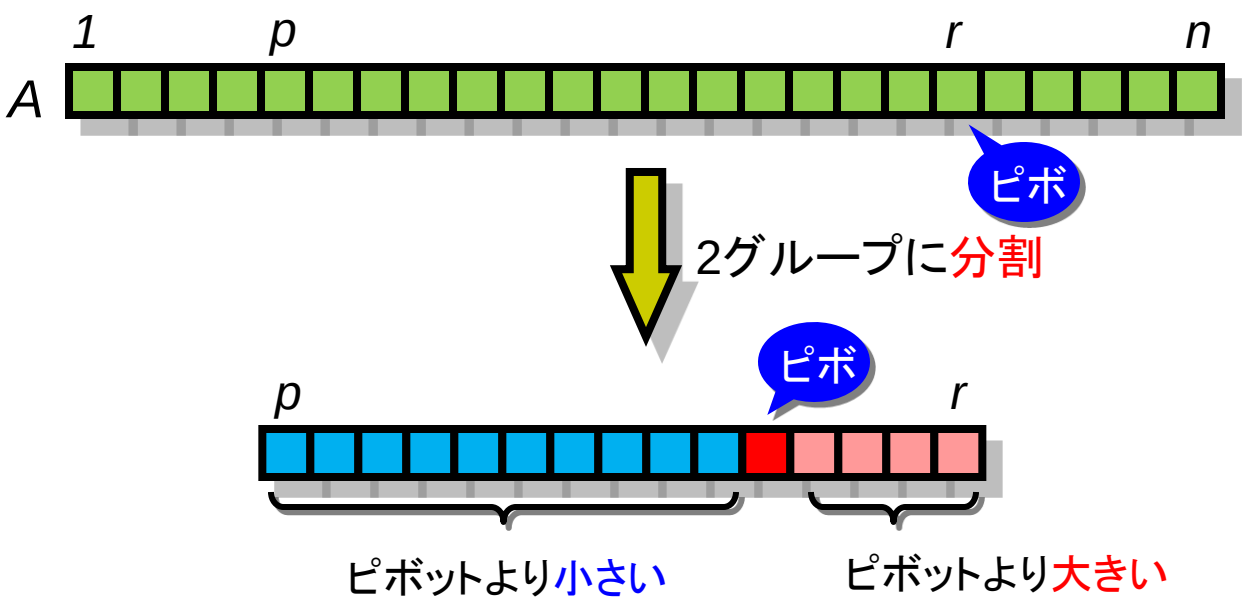


ピボットよりも 小 / 大 への分け方

配列 A 中の $A[p] \sim A[r]$ を分割することを考えましょう

$A[r]$ をピボットとして選ぶことにします

イメージ



ピボットよりも 小 / 大 への分け方

各 $j = p, p+1, \ldots, r-1$ について, $A[j]$ とピボットを比較

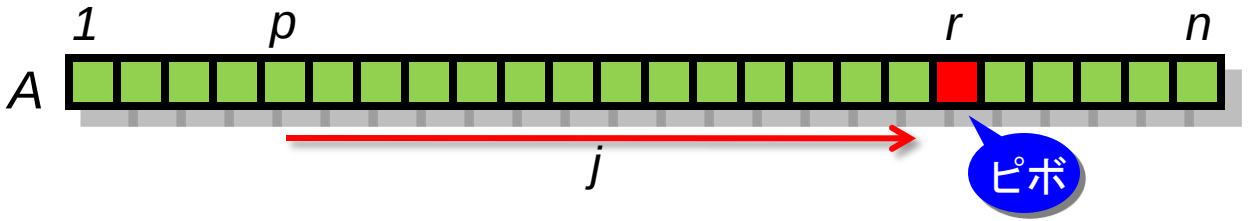
各 j について, $A[j]$ を適切な場所へ移動させる

具体的には

各繰り返しで($j \leftarrow p$ to $r-1$ の間で)

- $A[p] \sim A[i]$ はピボットより小さい自然数を含み,
- $A[i+1] \sim A[j-1]$ はピボットより大きい自然数を含む

ように保つ



ピボットよりも 小 / 大 への分け方

具体的には

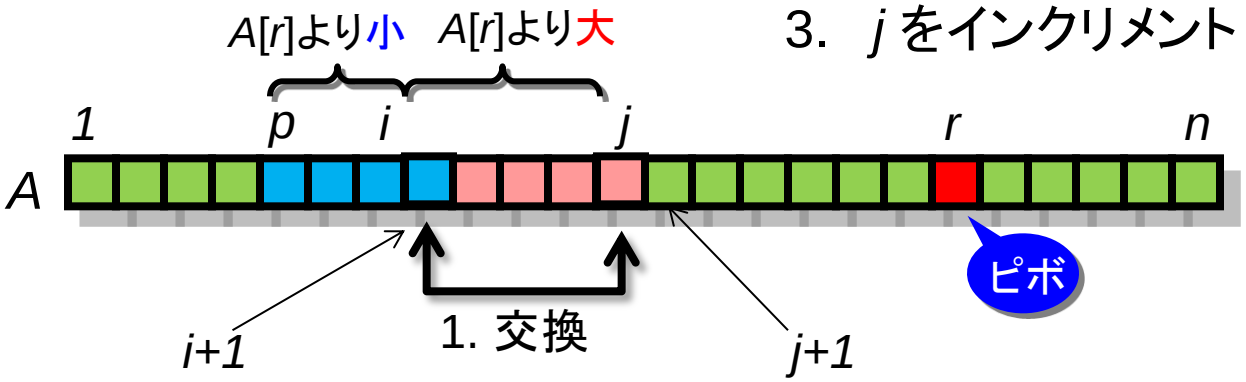
各繰り返しで($j \leftarrow p$ to $r-1$ の間で)

- $A[p] \sim A[i]$ はピボットより小さい自然数を含み,
- $A[i+1] \sim A[j-1]$ はピボットより大きい自然数を含む

ように保つ

Case1: $A[j] \leqq A[r]$ のとき

1. $A[i+1]$ と $A[j]$ を交換
2. i をインクリメント
3. j をインクリメント



ピボットよりも 小 / 大 への分け方

具体的には

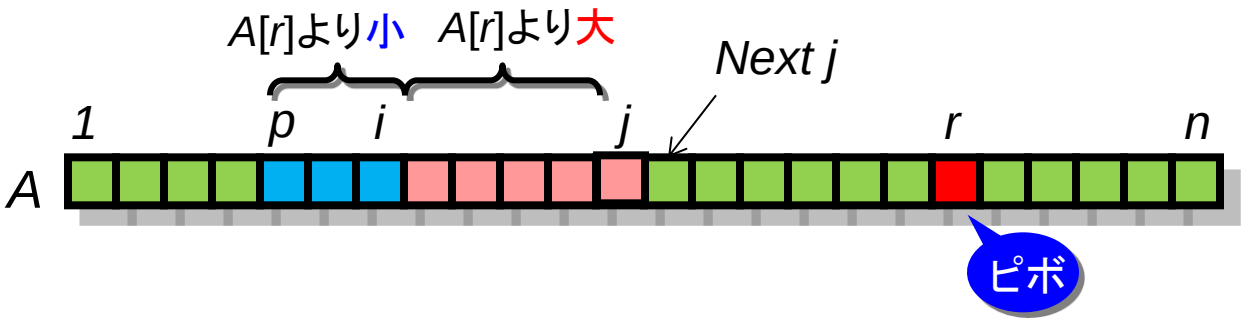
各繰り返しで($j \leftarrow p$ to $r-1$ の間で)

- $A[p] \sim A[i]$ はピボットより小さい自然数を含み,
- $A[i+1] \sim A[j-1]$ はピボットより大きい自然数を含む

ように保つ

Case2: $A[j] > A[r]$ のとき

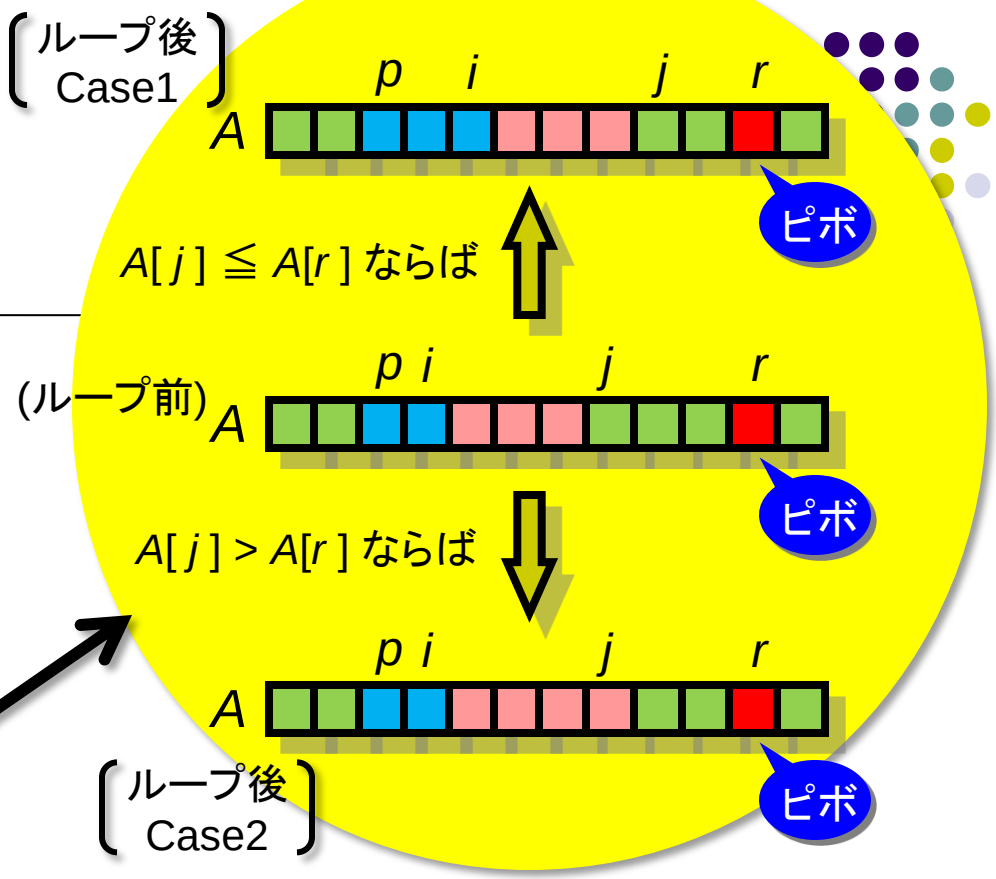
1. j をインクリメント

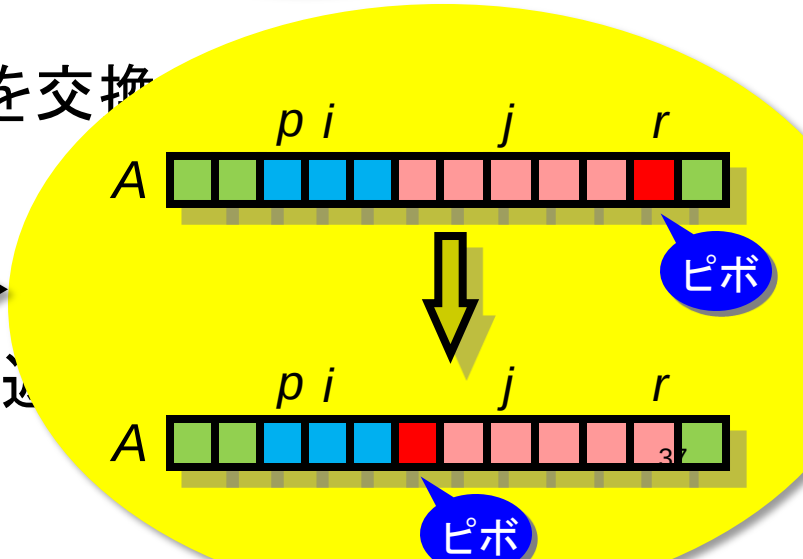


分割方法を擬似コードで表す

PARTITION(*A*, *p*, *r*)

1 *pivot* ← *A*[*r*]
2 *i* ← *p* – 1
3 **for** *j* ← *p* **to** *r* – 1
4 **do if** *A*[*j*] ≦ *pivot*
5 **then** *A*[*i* +1] と *A*[*j*] を交換
6 *i* ← *i* + 1
7 *A*[*i* + *1*] と *A*[*r*] を交換
8 return *i* + 1 { ピボットの位置を返す





クイックソート全体の擬似コード

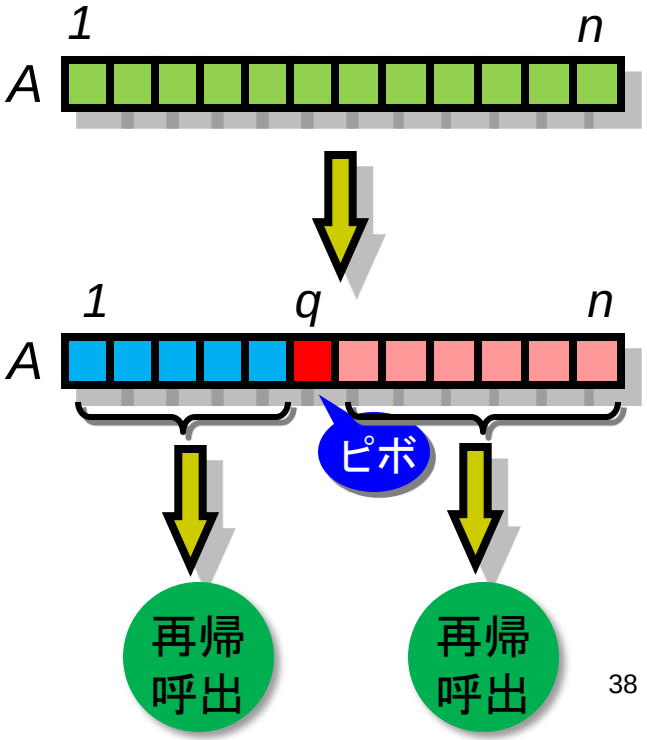
QUICKSORT(*A*, *p*, *r*)

```
1   if  p < r
2       then  q ← PARTITION(A, p, r)
3             QUICKSORT(A, p, q-1)
4             QUICKSORT(A, q+1, r)
```

PARTITION(*A*, *p*, *r*)

```
1   pivot ← A[ r ]
2   i ← p – 1
3   for  j  ←  p  to  r – 1
4     do if  A[ j ] ≦ pivot
5           then  A[ i +1 ] と A[ j ]  を交換
6                 i  ←  i + 1
7   A[ i + 1 ] と A[ r ] を交換
8   return i + 1    { ピボットの位置を返す}
```

1番最初の関数呼出は,
QUICKSORT(*A*, 1, *n*)



クイックソートの計算時間

平均の計算時間: $O(n \lg n)$ 時間

詳細は省略

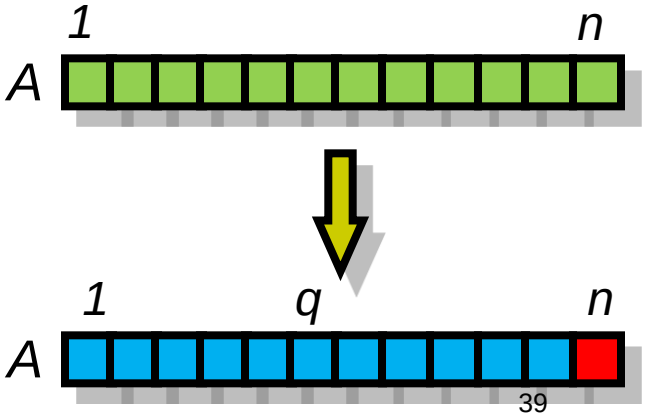
（マージソートのときと同様に，
再帰木を作ってみると直感的解釈が得られる）

最悪時の計算時間: $O(n^2)$ 時間

範囲中(p～r)の最大値／最小値を
ピボットとして選んだ時

↓

“悪い”分割になってしまう



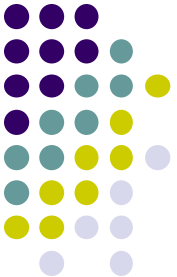
まとめ

- ソートのアルゴリズムを3つ紹介
  - 挿入ソート（計算時間: $O(n^2)$ 時間）
  - マージソート（計算時間: $O(n \lg n)$ 時間）
  - クイックソート（平均計算時間: $O(n \lg n)$ 時間,
    最悪時計算時間: $O(n^2)$ 時間）

（一点追加法,
incremental method）

分割統治法. 最悪時の計算量で n log n.

Pivotの選び方によるが
ランダムにpivotを選ぶと
平均計算量で n log n



諸注意

　　このスライドは　2010年4月に　山中克久　博士
（FS専攻DB講座助教・当時）
がMIT教科書を参考にして
作成したオリジナル版を基にして　今回　大森が改訂したものです．優れた例題解説はオリジナル版の著者の貢献であり，もし　分かりにくいところがあれば全て改訂者の責任です．
この事実を明記して，スタッフの努力に謝意申し上げます．

2013年4月5日記．　　FS専攻DB講座　大森匡