アルゴリズムとデータ構造 第2回

整列のアルゴリズム(SORTING ALGORITHMS)

塩浦昭義
情報科学研究科　准教授
shioura@dais.is.tohoku.ac.jp
http://www.dais.is.tohoku.ac.jp/~shioura/teaching

先週の復習

時間計算量と入力サイズ

- アルゴリズムの時間計算量(time complexity)の数え方
  - 問題を解くまでのステップ数を計算
  - アルゴリズムの1ステップ：加減乗除，余りなどの計算，代入など
- 時間計算量は，問題の入力サイズ(入力長)の関数として表現
  - 問題の入力サイズ(input size)：入力データをコンピュータ上で表現したときのサイズ．入力される数値の数，数値のビット長など
  - 例1：a_0 と a_1 の最大公約数
    - 入力サイズは $\log_2 a_0$ と $\log_2 a_1$
  - 例2：n 個の数 $a_1, a_2, \ldots, a_n$ を大きい順に並べる：
    - 入力サイズは n もしくは $\log_2 a_1 + \cdots + \log_2 a_n$

関数のオーダー記法

- 時間計算量の評価を簡単にするためにオーダー記法(order notation)を使う

ある関数 $T(n)$ と 関数 $f(n)$ に対して $T(n) = O(f(n))$
(「$T(n)$ はオーダー $f(n)$」と読む)
$\iff$ ある定数 c, n_0 が存在して，$T(n) \leq cf(n)$ $(\forall n \geq n_0)$

- 例1 : $2n^2 + 5n + 1000 = O(n^2)$

  ($n \geq 1000$ のとき，$2n^2 + 5n + 1000 \leq 10n^2$)

  n_0　　c

- 例2 : $\max\{500, n\} = O(n)$

  ($n \geq 500$ のとき，$\max\{500, n\} \leq n$)

※直感的な意味 : n が増えるときの増加率最大の項を取り出す
重要でない情報は無視

時間計算量の評価：最悪と平均

- 時間計算量の（理論的な）評価方法
  - 最悪時間計算量(worst case time complexity)：
    同じ入力サイズの問題例の中で最大の時間計算量を求める
  - 平均時間計算量(average time complexity：
    同じ入力サイズの問題例に対し，それらの時間計算量の平均を求める
  - これ以外の方法もある
- 時間計算量のオーダーが多項式(polynomial) (log n, n^2, n^5,…)
  ➔（理論的には）速いアルゴリズム
  　その問題は解きやすい
- 時間計算量のオーダーが指数(exponential) (2^n, n!, n^n,…)
  ➔（理論的には）遅いアルゴリズム
  　その問題は解くのが難しい

今日の内容

ソーティングのアルゴリズム

ソート（Sorting）

- 全順序つきの集合の要素を順番に並べること
- 例1：数字の整列（小さい数から大きい数へ）

  ５１、２３、４６、９、３０

  ⇒　９、２３，３０，４６、５１
- 例2：名前の五十音順による整列

  しおうら、たなか、とくやま、すずき

  ⇒　しおうら、すずき、たなか、とくやま
- この講義では数字の整列を扱う
- 入力データは n 個の実数，配列(array)に入っていると仮定
- 出力データは順番に並べられた n 個の実数，配列に入れる

ソートのアルゴリズム
Algorithms for Sorting

様々なアルゴリズムが存在

- バブルソート
- 挿入ソート

最悪計算時間　$O(n^2)$

- ヒープソート
- マージソート

最悪計算時間　$O(n \log n)$

- クイックソート
- などなど

平均計算時間　$O(n \log n)$

最悪$O(n^2)$, 実用的には高速

今日の講義:「配列」を使って実行できる配列のアルゴリズムを紹介

バブルソートの動き（その1）
Behavior of Bubble Sort

A[i] = 配列の i 番目の要素 (i = 1, 2, …, n)

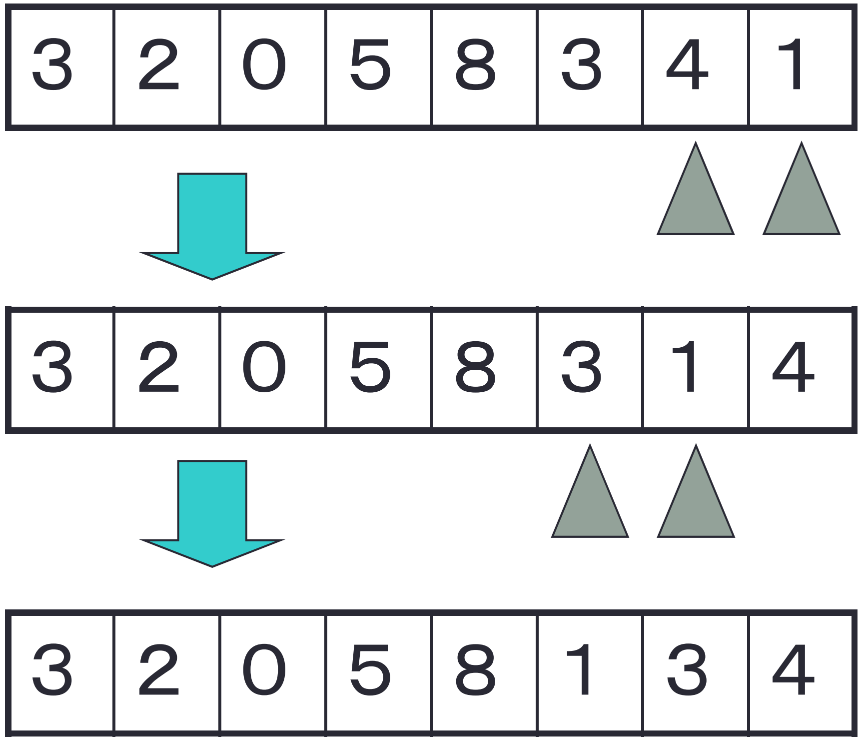
1回目の反復

- A[7]とA[8]の大小を比較

A[7] > A[8]
　⇒ 2つの要素を入れ替え

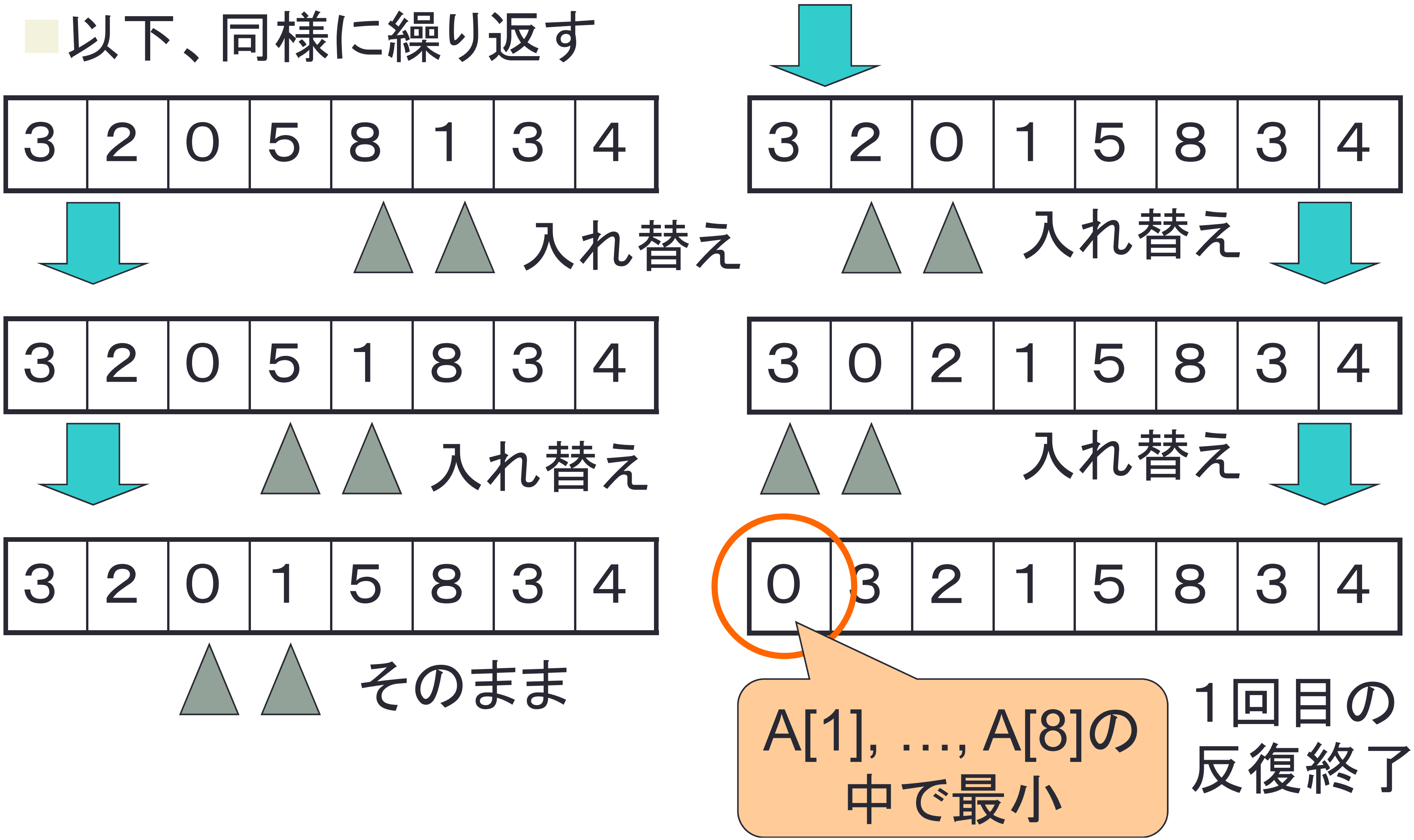
- A[6]とA[7]の大小を比較

A[6] > A[7]
　⇒ 2つの要素を入れ替え



バブルソートの動き(その2)

- 以下、同様に繰り返す



バブルソートの動き(その3)

2回目の反復

A[2], …,A[8] に対して1回目の反復と同じ作業を行う

0	3	2	1	5	8	3	4

全体で2番目に小さい

A[2], …, A[8]の中で最小

0	1	3	2	3	5	8	4

3回目の反復

A[3], …,A[8] に対して1回目の反復と同じ作業を行う

0	1	3	2	3	5	8	4

全体で3番目に小さい

A[3], …, A[8]の中で最小

0	1	2	3	3	4	5	8

バブルソートの動き（その4）

4回目の反復

0	1	2	3	3	4	5	8

5回目の反復

0	1	2	3	3	4	5	8

6回目の反復

0	1	2	3	3	4	5	8

7回目の反復

0	1	2	3	3	4	5	8

ソート完了！

0	1	2	3	3	4	5	8

バブルソートの計算時間(その1)
Time Complexity of Bubble Sort

- 一回目の反復
  A[n-1]とA[n]の比較、入れ替え
  A[n-2]とA[n-1]の比較、入れ替え
  ⋮
  A[1]とA[2]の比較、入れ替え
  ⇒ c (n-1) 時間 （c は定数）
- 2回目の反復： c（n – 2) 時間
- 3回目の反復： c（n – 3) 時間

バブルソートの計算時間(その2)

- 一般に、k 回目の反復 : c(n − k) 時間

∴ バブルソートの計算時間は(最悪の場合でも)

$c \times \{(n-1) + (n-2) + \cdots + 2 + 1 + 0\} = c(n-1)(n-2)/2 = O(n^2)$

※途中でソートが終わっていたら

以降の反復を省略できる

(例:4回目の反復以降)

➔ 運が良ければ $O(n^2)$時間より早く終了

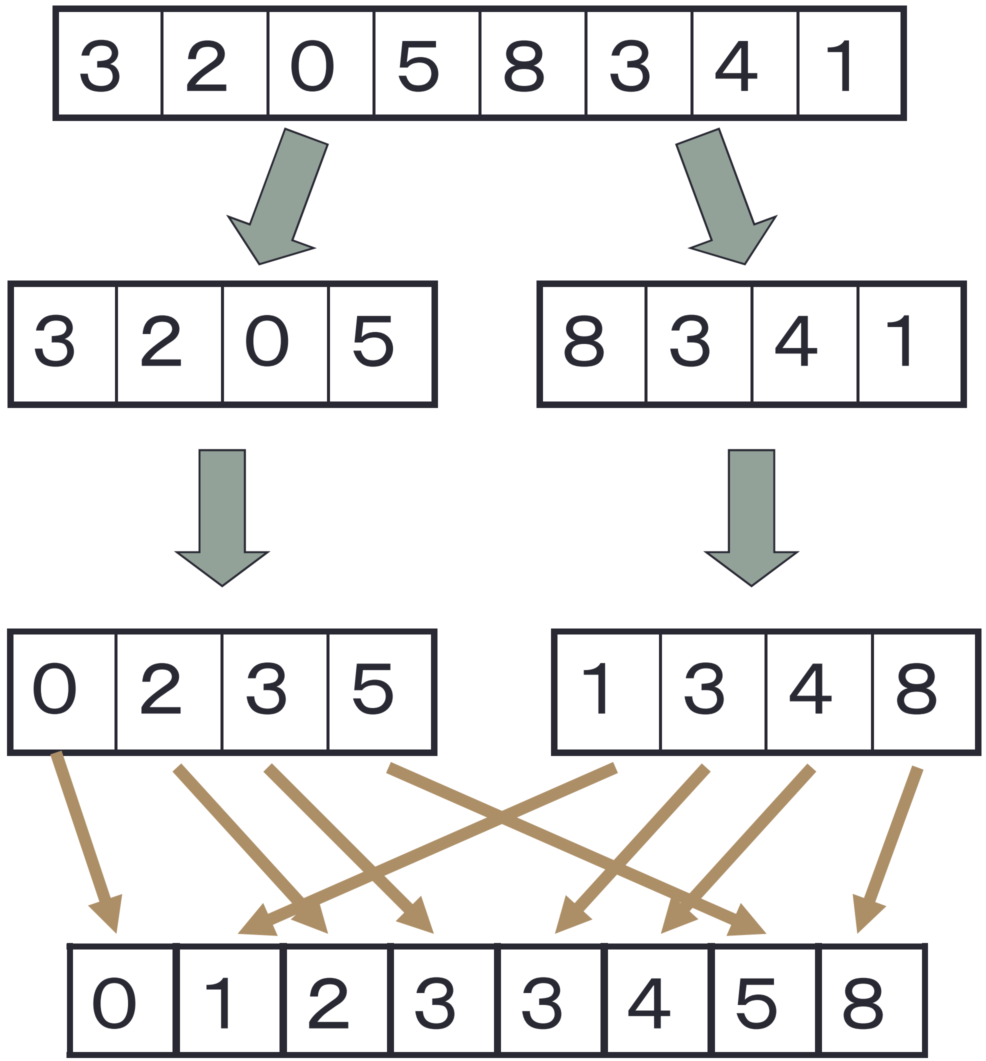
マージソートのアイディア
Idea of Merge Sort

- アイディア: 分割統治法 (divide-and-conquer method)

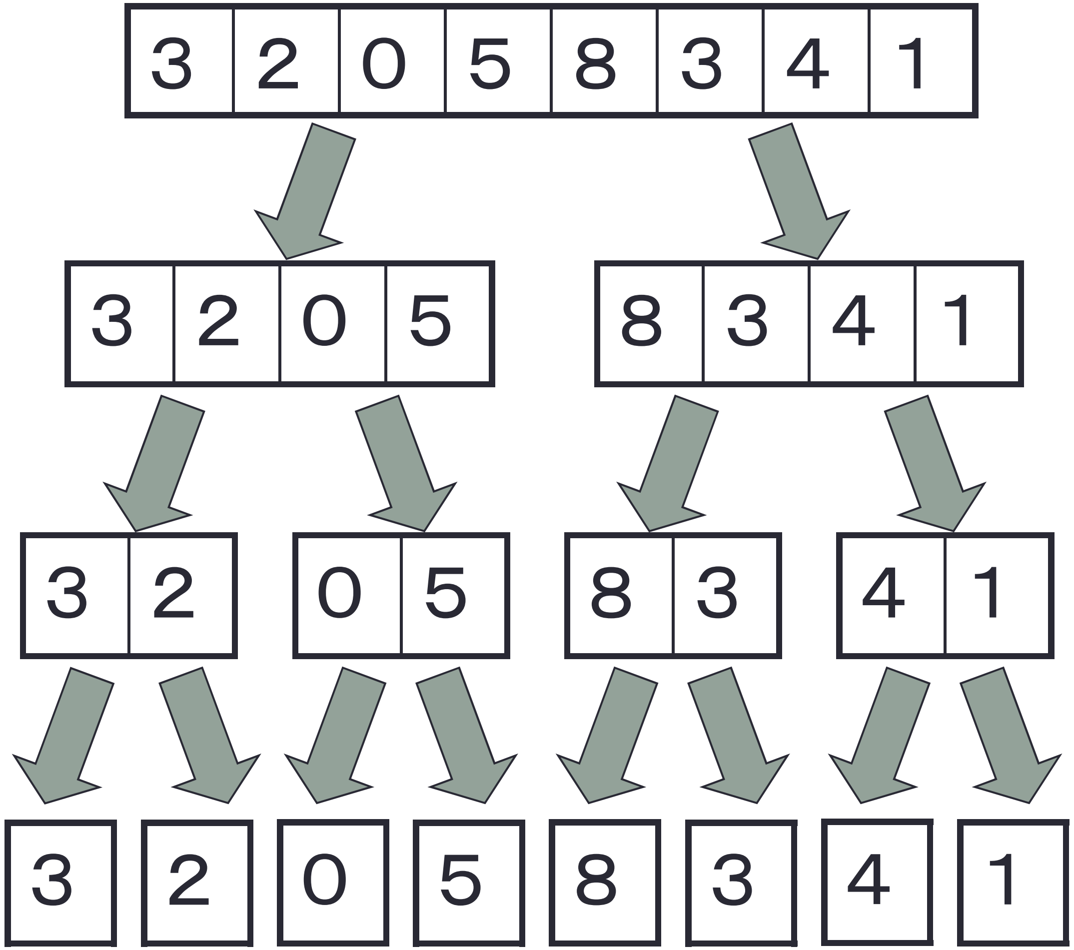
①与えられた配列を2分割

②2分割された配列をそれぞれ再帰的にソート

③ソートされた2つの配列をマージ（統治）



マージソートの動き（前半）
Behavior of Merge Sort

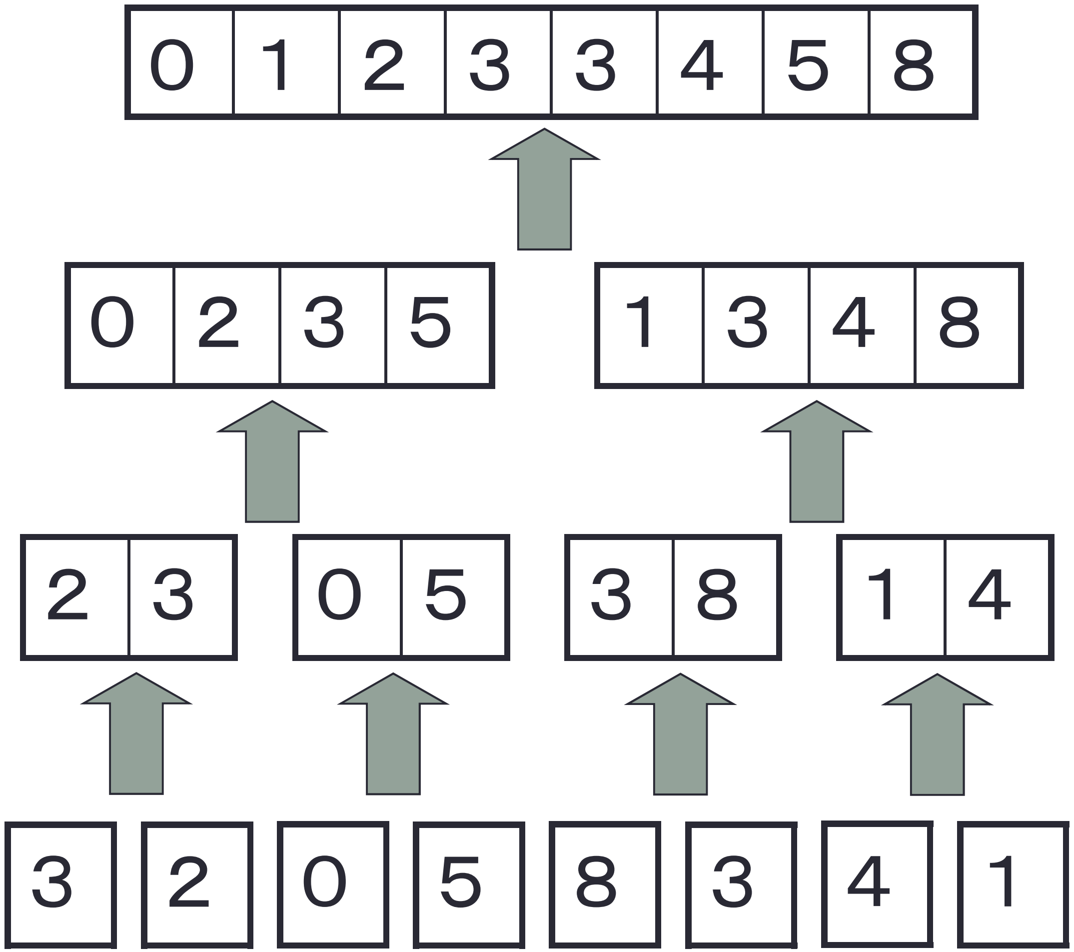


配列を2分割
（大きさ：8→4）

配列を2分割
（大きさ：4→2）

配列を2分割
（大きさ：2→1）

マージソートの動き(後半)



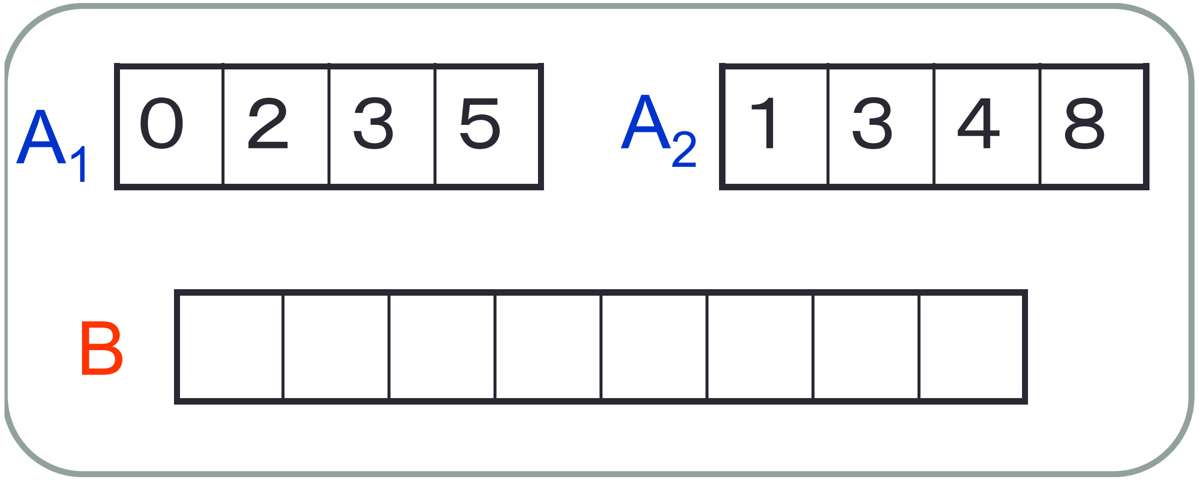
ソート列をマージ
(大きさ:4→8)

ソート列をマージ
(大きさ:2→4)

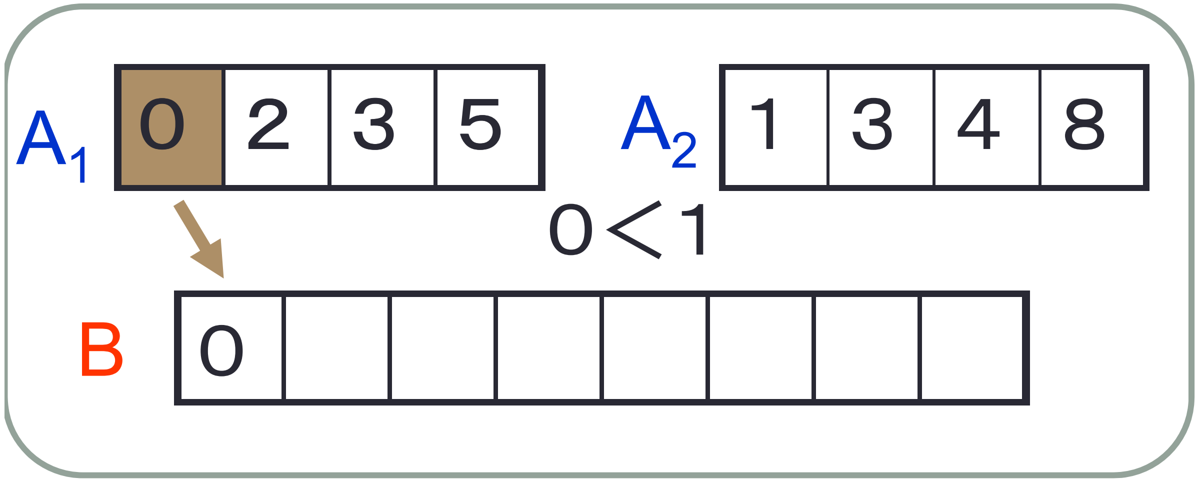
ソート列をマージ
(大きさ:1→2)

ソート列のマージ(その1)
Merge of Sorted Sequences

ソート列の併合(マージ)は線形時間で出来る



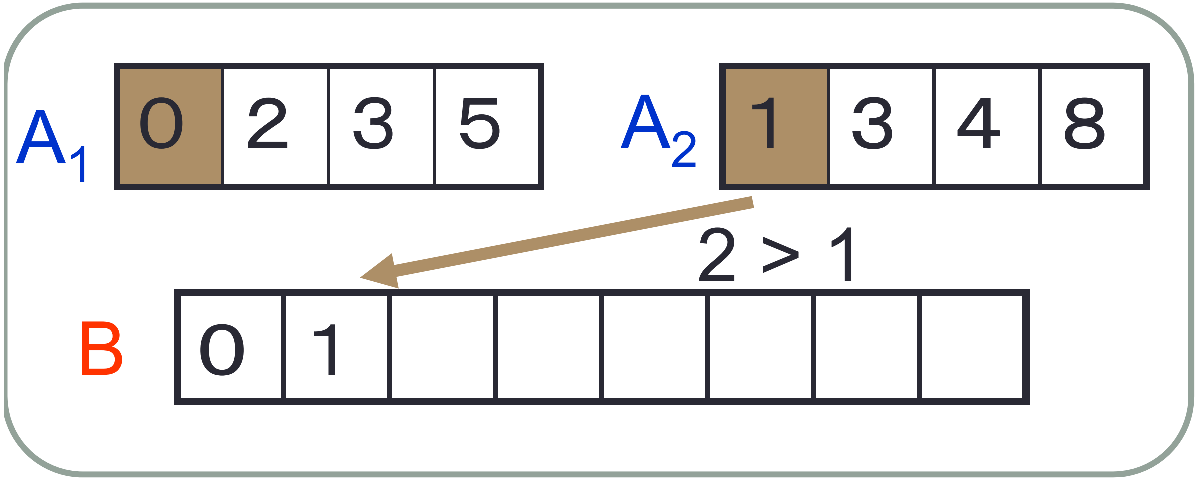
マージした結果を格納する配列Bを用意



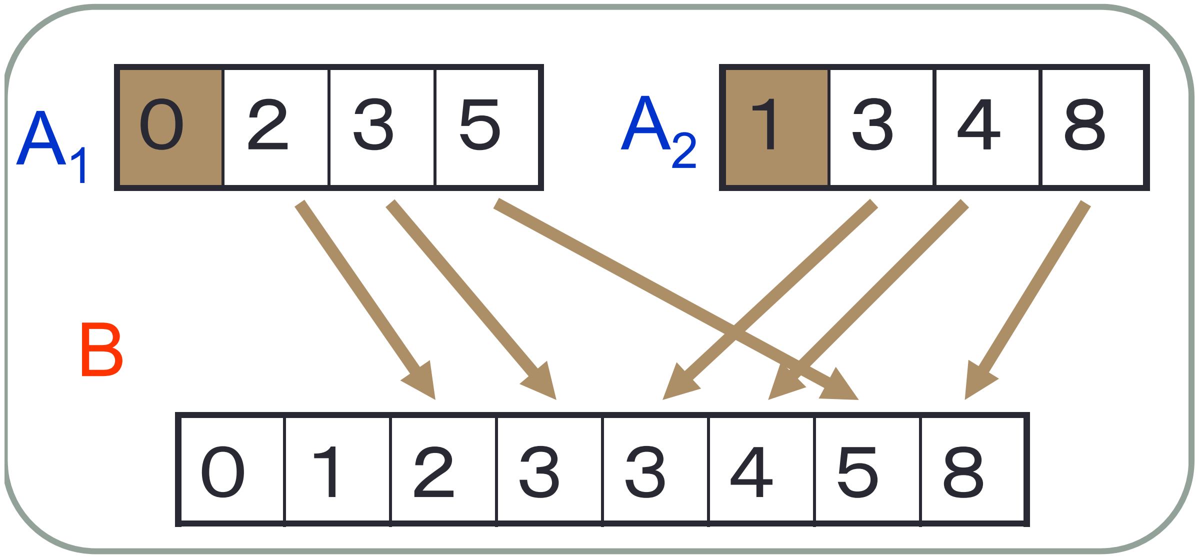
A_1 と A_2 の先頭の数字を比較

小さいほうを B の空欄の先頭へ移動

ソート列のマージ(その2)



A_1 と A_2 の先頭の
数字を比較
小さいほうを B の
空欄の先頭へ移動



この作業を
繰り返す

ソート列のマージ(その3)

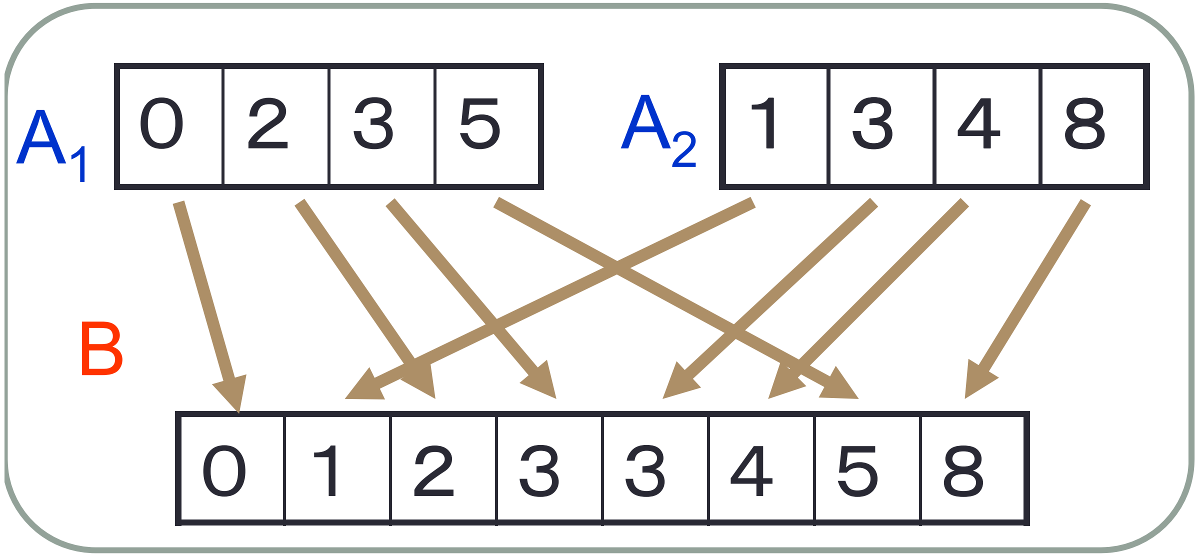
計算時間の解析

$n_1 = A_1$ の要素数、$n_2 = A_2$ の要素数

A_1 と A_2 の要素ひとつひとつに対して、

他の要素との比較
B に要素を書き込む

} 定数時間 c



$c\,(n_1+n_2)$ 時間で実行可能

マージソートの計算時間(その1)
Time Complexity of Merge Sort

T(n) = n 個の要素のソートの時間(n は2のべき乗と仮定)

①与えられた配列を2分割

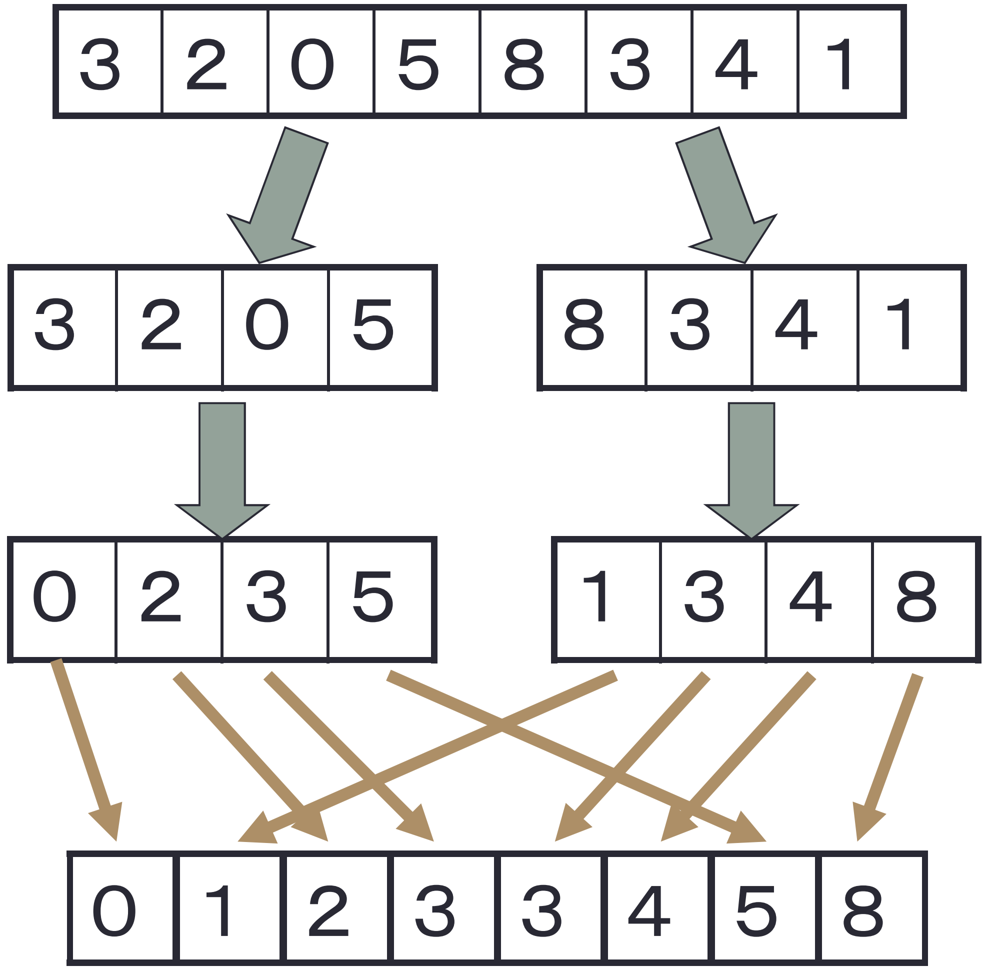
c' n 時間

②2分割された配列をそれぞれ再帰的ソート

T(n/2) × 2　時間

③ソートされた2つの配列をマージ（統治）

c n 時間



マージソートの計算時間（その2）

$T(n)$ = n 個の要素のソートの時間

$T(n) = 2\,T(n/2) + (c+c')\,n$　が成り立つ

　⇒ 解は　$T(n) = (c+c')\,n \log_2 n = O(n \log n)$

※ n が2のべき乗でないときも，解析を少し修正すれば O(n log n) の時間計算量が証明できる

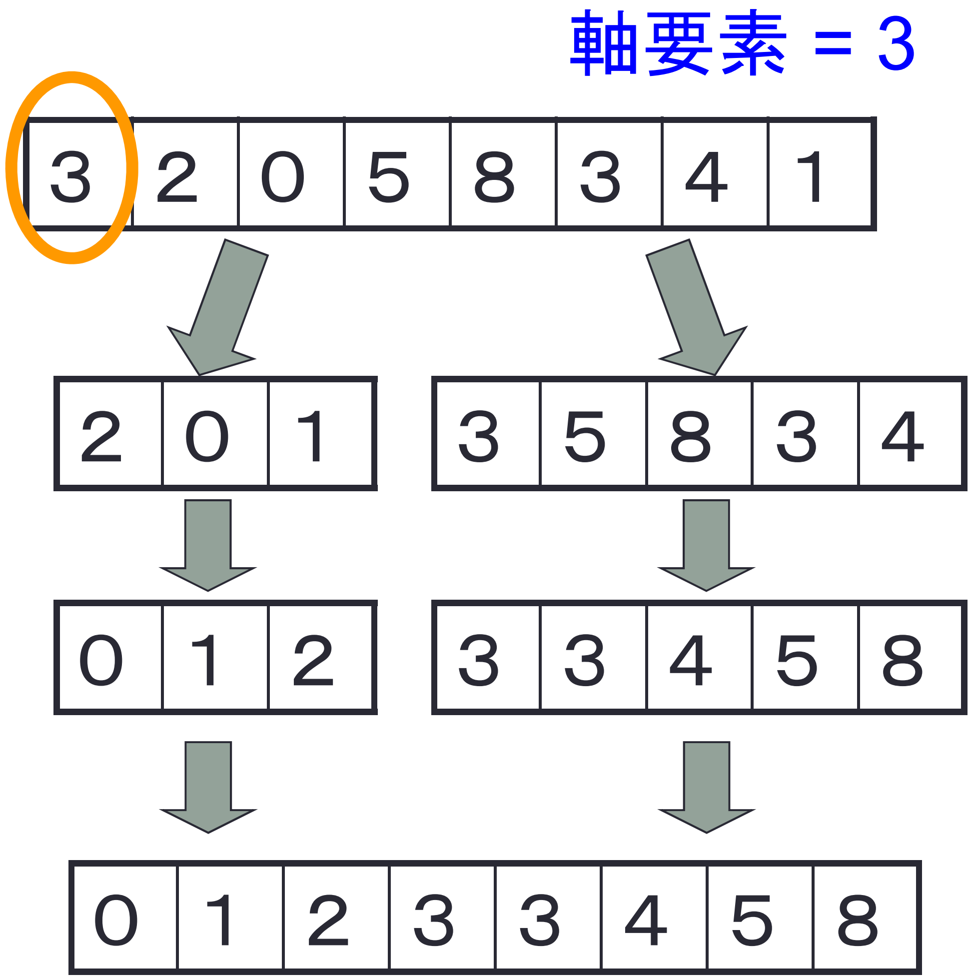
クイックソートのアイディア
Idea of Quick Sort

①A[1], …, A[n]から
ひとつの値（軸要素, pivot）
を選ぶ

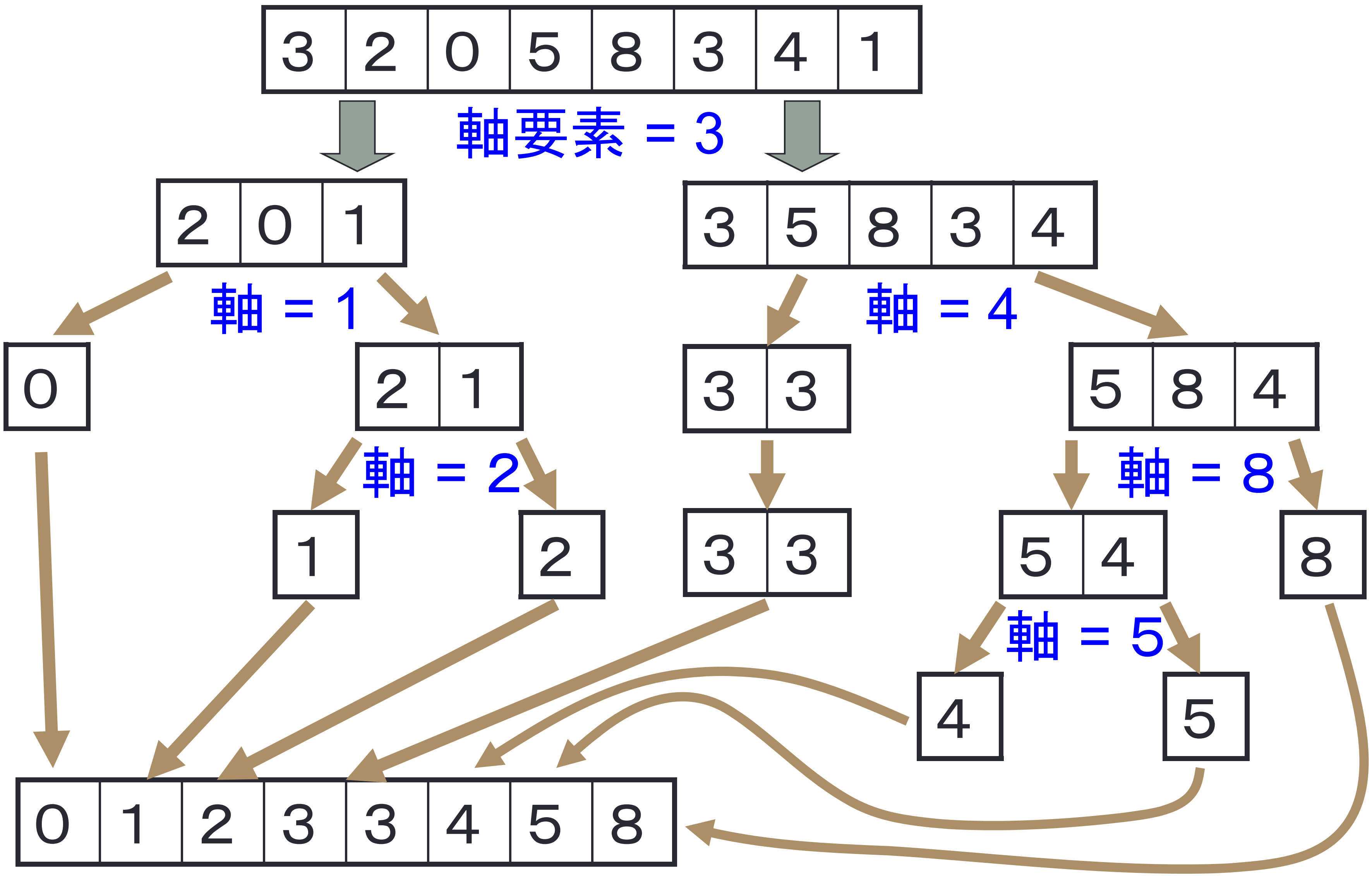
② 軸要素未満の要素と
それ以外に分割

③2分割された配列を
それぞれ再帰的にソート

④ソートされた2つの配列
をつなげる



クイックソートの動き Behavior of Quick Sort



軸要素の選び方(その1)
How to Choose Pivot

良い選び方:

配列をほぼ二等分する

軸要素の選択に時間をかけない

3	2	0	5	8	3	4	1

軸要素 = 3

2	0	1

3	5	8	3	4

悪い選び方:

２分された配列の大きさがアンバランス

3	2	0	5	8	3	4	1

軸要素 = 1

0

3	2	5	8	3	4	1

3	2	0	5	8	3	4	1

軸要素 = 0

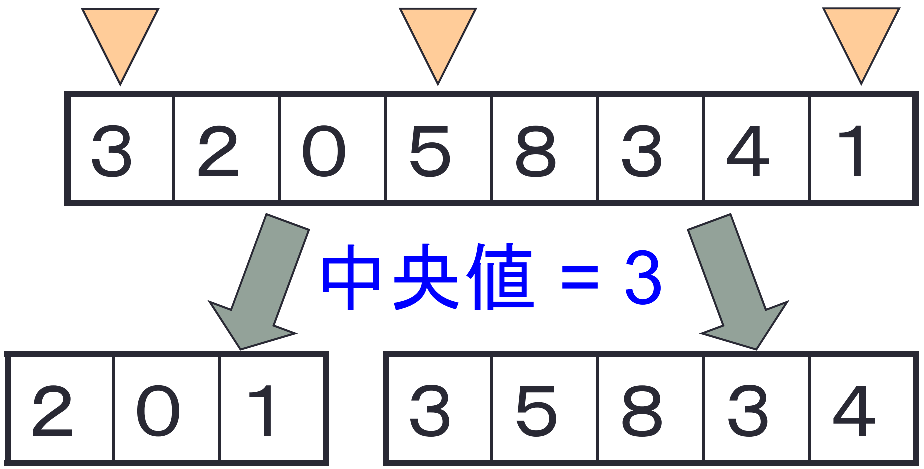
3	2	0	5	8	3	4	1

軸要素の選び方(その2)

よく使われる選び方:

(a) ランダムにひとつ選ぶ

(b) 左端、右端、真中の3要素の中央値



軸要素の選び方(その3)

よく使われる選び方:

(c) A[1], A[2]のうち、大きい値
　同じ場合は次の異なる値と比較

A[1]=A[2]
⇒A[3]と比較

3	2	0	5	8	3	4	1

大きい値 = 3

2	0	1

3	5	8	3	4

3	3	2	5	8	0	4	1

大きい値 = 3

2	0	1

3	3	5	8	4

クイックソートの計算時間

$T(n)$ = n 個の要素のソートの計算時間

一般の場合: k 個と n – k 個に分割される ($1 < k < n$)

⇒ $T(n) \leqq T(k) + T(n-k) + c\,n$ (c: 定数)

⇒ 帰納法により、解は $T(n) \leqq 2c\,n^2$

∴最悪計算時間は $O(n^2)$

実際には, 数列がほぼ半分に分割される

⇒ 実用上の計算時間は$O(n \log n)$ に近い

平均時間も $O(n \log n)$ (解析はちょっと難しい)

演習問題(〆切:次回の授業開始5分後まで)

- 次の数値に対して,

 バブルソート, マージソート, クイックソート

を適用した場合の挙動を(講義で行なった程度に)

詳しく説明せよ.

27, 17, 3, 16, 13, 10, 1, 5