アルゴリズムとデータ構造

コンピュータサイエンスコース
知能コンピューティングコース

第6回

ソートアルゴリズムの計算量の下界
第p要素選択アルゴリズム

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

中間試験について

- 日時：6月17日（木）8:50～10:20
- 過去にレポートを一度も出していない場合，受験は不可
- 教科書，ノート等の持ち込みは一切不可
- 座席はこちらで指定
- 試験内容：今日（第6回目）までの講義で教えたところ
  - アルゴリズムやデータ構造の挙動
  - 時間計算量の解析，および関連する証明問題
  - 用語の定義，など
- 50点満点，24点以下は追試レポートもしくは単位不可

ソートアルゴリズムの計算量の下界

- ソートアルゴリズムの時間計算量
  - バブルソート，クイックソートは $O(n^2)$
  - マージソート，ヒープソートは $O(n \log n)$

2要素の比較のみを使ったソートアルゴリズム

- 数値情報を利用
- 数値の範囲が限定されている

  - バケットソート，基数ソートは $O(n)$
    （バケツの数，桁数などを定数と見なしたとき）
- ソートのために最低限必要な計算時間は？

定理：2要素の大小比較に基づくソートアルゴリズムは $\Omega(n \log n)$ の時間計算量を必要とする.

ソートアルゴリズムの決定木（その1）

- ２要素の比較に基づくソートアルゴリズムの進行状況を決定木により表現
  - 比較を1回行なう度に状況がどのように変化するかを表す
- 例：A[0] ＝ a, A[1] ＝ b, A[2] ＝ c に対するバブルソート

a, b, c の値および大小関係は未知

簡単のため，全ての値は異なると仮定

➔ソート結果の候補は

abc, acb, bac, bca, cab, cba の6通り

比較を行なう度に，ソート結果の候補数が減少

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

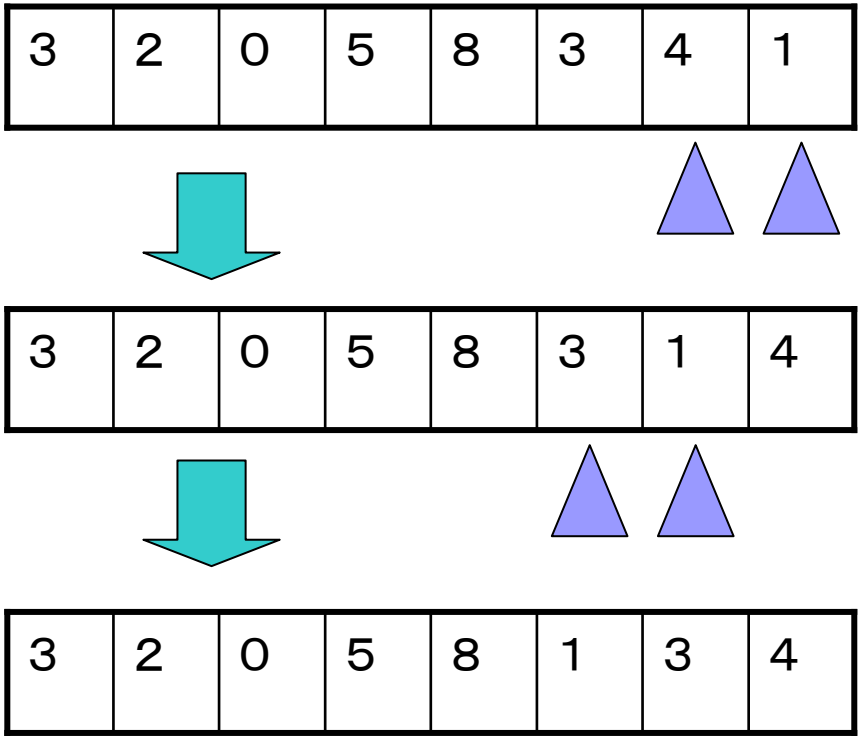
A[7] > A[8]

⇒ 2つの要素を入れ替え

■A[6]とA[7]の大小を比較

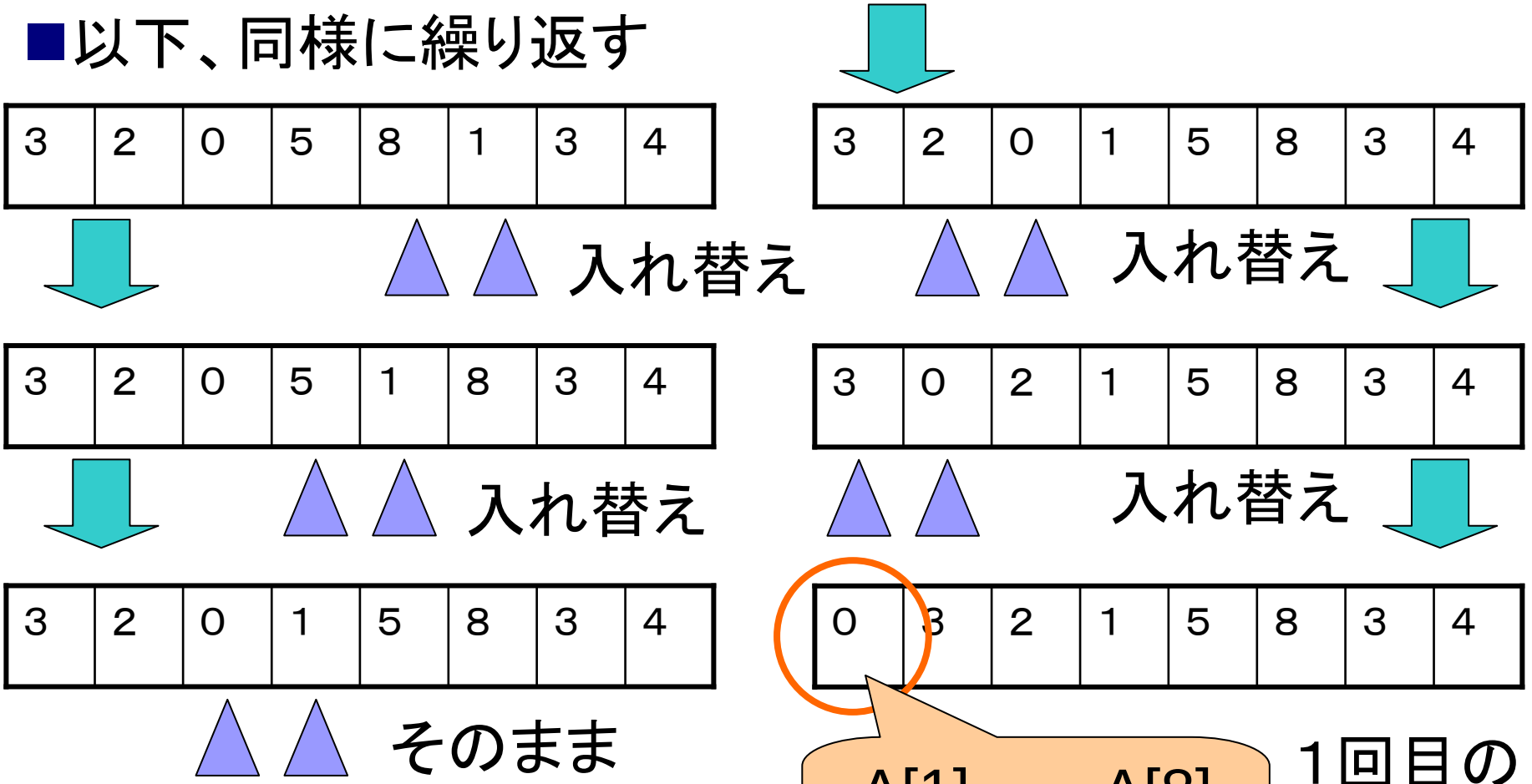
A[6] > A[7]

⇒ 2つの要素を入れ替え



バブルソートの動き（その2）

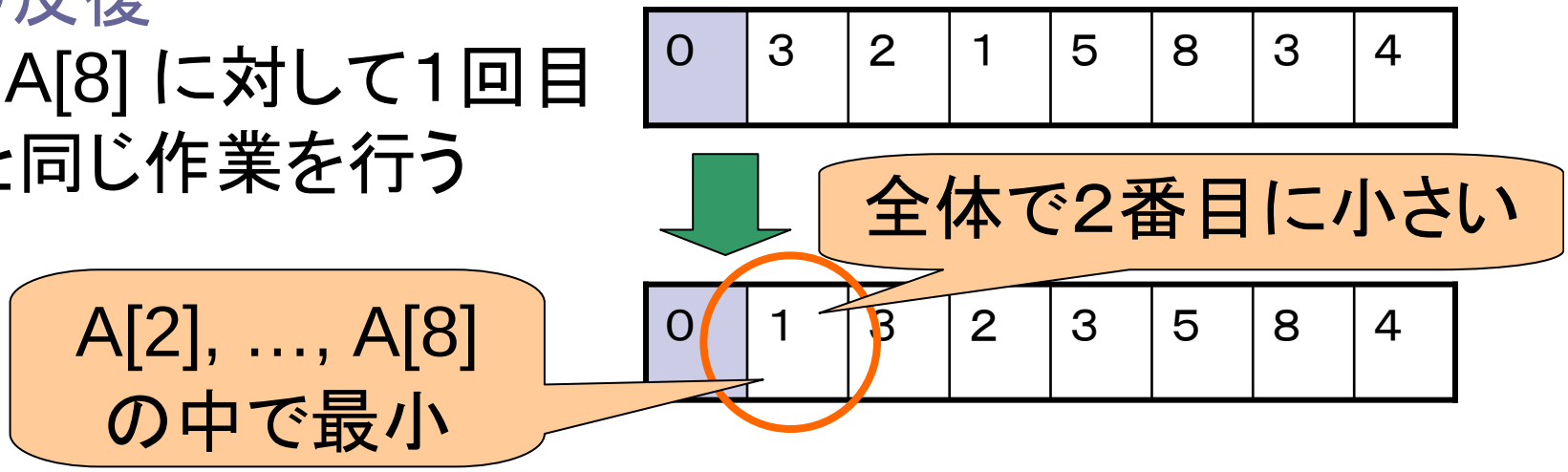
■以下、同様に繰り返す



バブルソートの動き（その3）

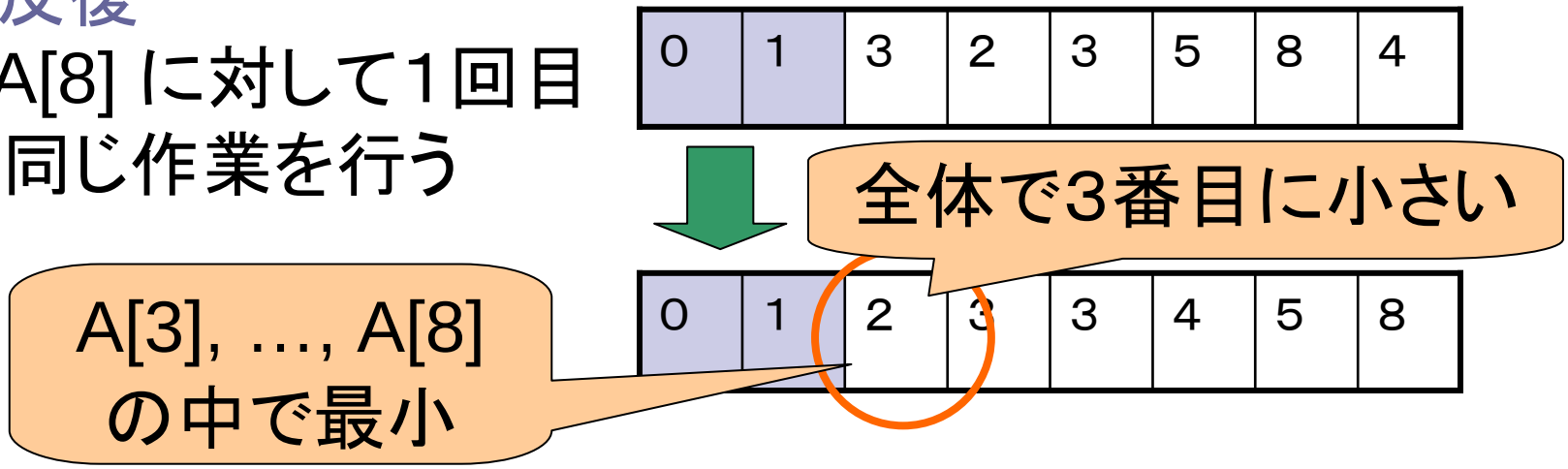
2回目の反復

A[2], …,A[8] に対して1回目の反復と同じ作業を行う



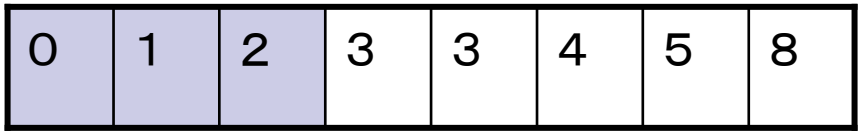
3回目の反復

A[3], …,A[8] に対して1回目の反復と同じ作業を行う

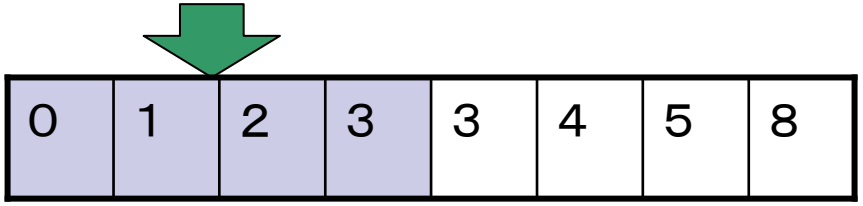


バブルソートの動き（その４）

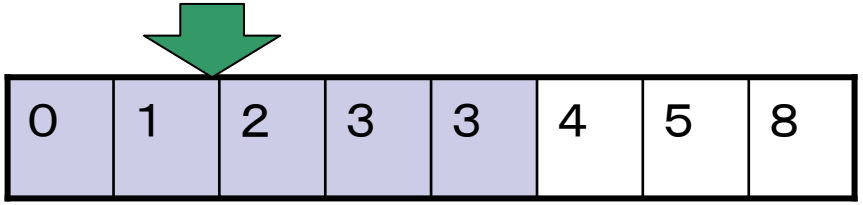
４回目の反復



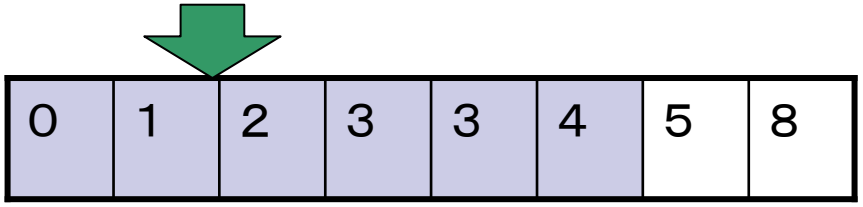
５回目の反復



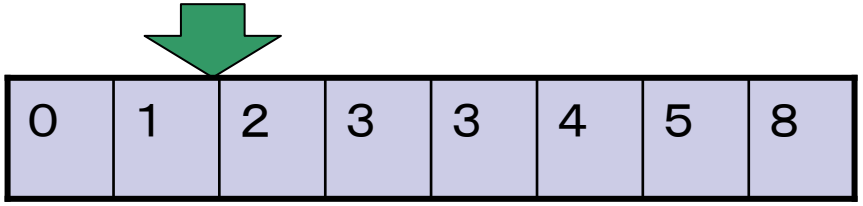
６回目の反復



７回目の反復

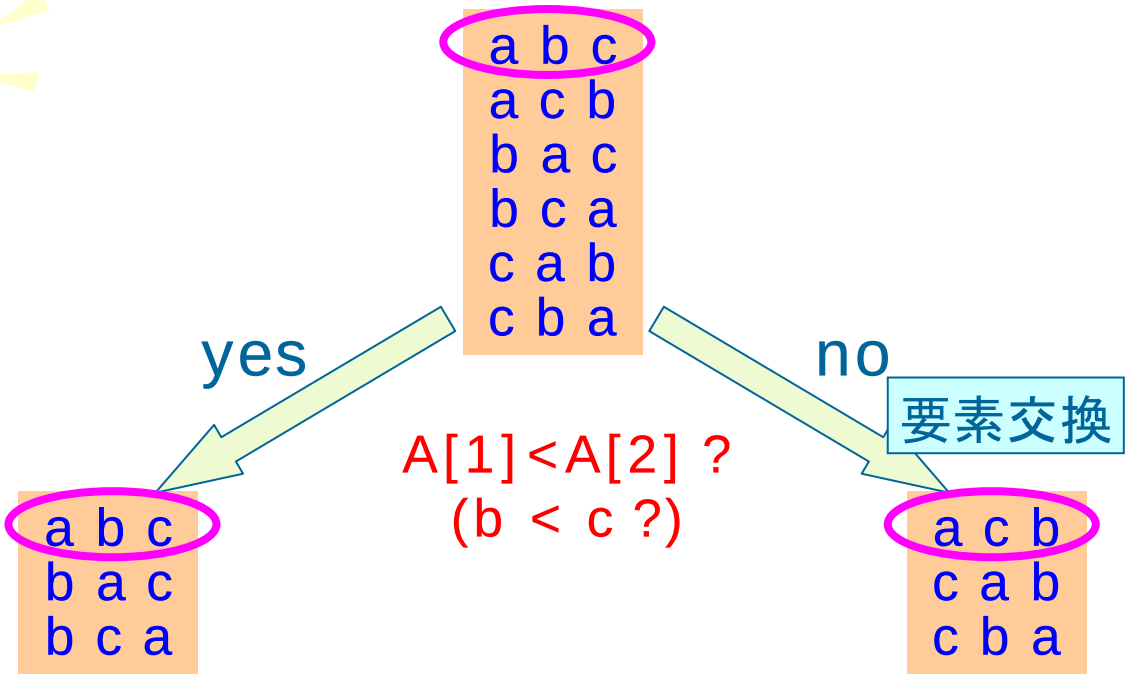


ソート完了！



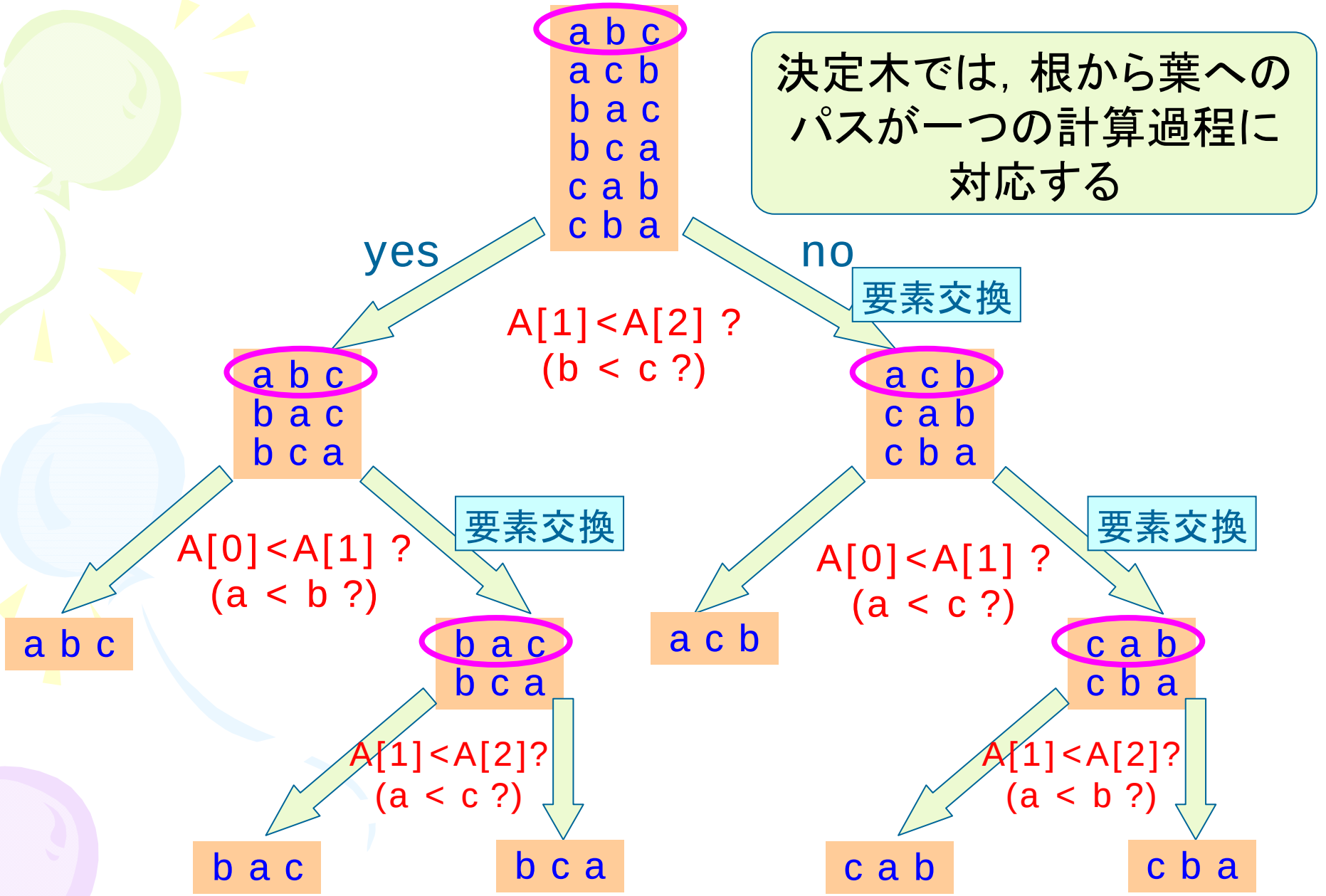
ソートアルゴリズムの決定木（その2）

- バブルソートでは，最初に A[1]＝b と A[2]＝c を比較
  - b<c の場合，
    - ソート結果の候補は abc, bac, bca の3通り
    - バブルソートでは要素の入れ替えをしない➔ A: abc
  - b>c の場合，
    - ソート結果の候補は acb, cab, cba の3通り
    - バブルソートではA[1]とA[2]を入れ替え➔ A:acb
    - 以下，この場合を考える


a b c
a c b
b a c
b c a
c a b
c b a
yes
no
A[1]<A[2] ?
(b < c ?)
要素交換
a b c
b a c
b c a
a c b
c a b
c b a

ソートアルゴリズムの決定木（その3）

- バブルソートでは，次に A[0]＝a と A[1]＝c を比較
  - a<c の場合，
    - ソート結果の候補は acb の1通り
    - バブルソートでは要素の入れ替えをしない➔ A：acb
    - バブルソートでは，この後（不要な）A[1]とA[2]の比較をして終了➔A：acb をソート結果として出力
  - a>c の場合，
    - ソート結果の候補は cab，cba の2通り
    - バブルソートではA[0]とA[1]を入れ替え➔ A:cab
    - バブルソートでは，この後A[1]とA[2]の比較を行なう.


a b c
a c b
b a c
b c a
c a b
c b a
決定木では，根から葉へのパスが一つの計算過程に対応する
yes
no
A[1]<A[2] ?
(b < c ?)
要素交換
a b c
b a c
b c a
a c b
c a b
c b a
A[0]<A[1] ?
(a < b ?)
A[0]<A[1] ?
(a < c ?)
a b c
a c b
b a c
b c a
c a b
c b a
A[1]<A[2]?
(a < c ?)
A[1]<A[2]?
(a < b ?)
b a c
b c a
c a b
c b a
アルゴリズムは，ソート結果の候補が1つに決まるまで比較を繰り返す

比較回数の下界値

- 決定木を使って，アルゴリズムでの比較回数の下界値を解析

決定木における，根から葉へのパスの長さ
≦ アルゴリズムでの比較の回数

この値を知りたい！

∴　決定木の高さ h
＝根から葉へのパスの長さの最大値
≦アルゴリズムでの比較の回数の最大値

比較回数の下界値

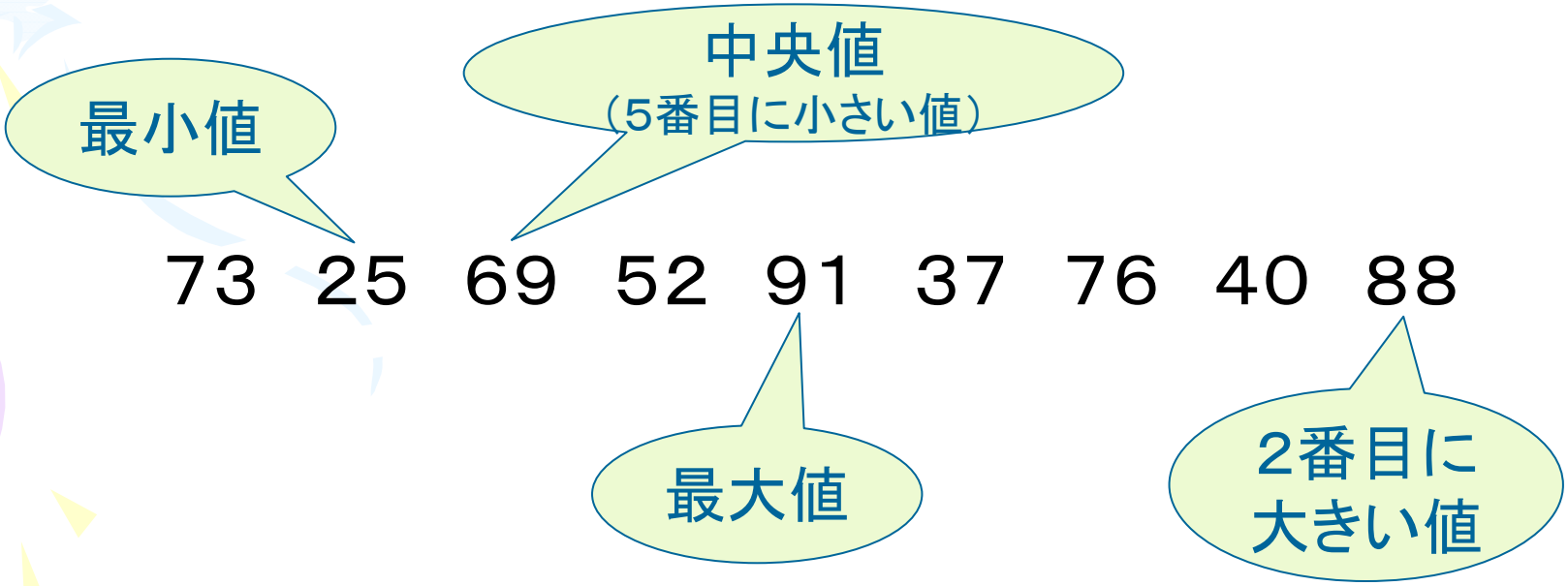
- 決定木の葉は n 要素の順列(ソート列)に対応
  - ➔ n! 個の葉が存在する
  - 高さ h の二分木の葉の数 ≦ 2^h
    - ➔ n! ≦ 2^h ➔ h ≧ log n!
  - スターリングの近似公式より
    - log n! ≒ n log n – n （n が十分に大きいとき）

∴アルゴリズムでの比較の回数の最大値
≧ 決定木の高さ h
≧ n log n – n = Ω(n log n)

定理:2要素の大小比較に基づくソートアルゴリズムは Ω(n log n) の時間計算量を必要とする.

第p要素の選択

- n 個の実数の中から p 番目に小さい(大きい)ものを選ぶ
  - p=1➔n個の実数の中の最小値
  - p=n➔n個の実数の中の最大値
  - p= $\lceil n/2 \rceil$ または $\lfloor n/2 \rfloor$ ➔n個の実数の中の中央値(median)



第p要素の選択

- n 個の実数の中から p 番目に小さい(大きい)ものを選ぶ
  - p=1➔n個の実数の中の最小値
  - p=n➔n個の実数の中の最大値
  - p= $\lceil n/2 \rceil$ または $\lfloor n/2 \rfloor$ ➔n個の実数の中の中央値(median)
- 最小値(最大値)は簡単に O(n) 時間で求めることが可能
  - 最小値の暫定値を b =A[0] とする
  - 各 i = 1, 2, …, n-1 に対し A[i] と b を比較,
    A[i] の方が小さければ b = A[i] とする
  - 最後に b を最小値として出力
- 1 < p < n のときの時間計算量は？

第p要素を求めるための時間計算量

- n 個の実数の中から p 番目に小さいものを選ぶ
  - 簡単なやり方：n 個の実数を昇順にソートしたあとで，p番目の要素を選ぶ ➔ O(n log n)時間

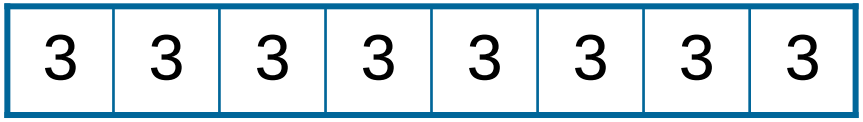
- 今回紹介する方法---ソートを使わないアルゴリズム
  - QUICKSELECT：平均時間計算量 O(n)，実用上高速
  - SELECT：最悪時間計算量 O(n)，実用上は少し劣る
  - (LAZYSELECT：平均時間計算量 O(n)，実用上高速)

QUICKSELECTの手順（その１）

- クイックソートと同様のアイディアを用いたアルゴリズム

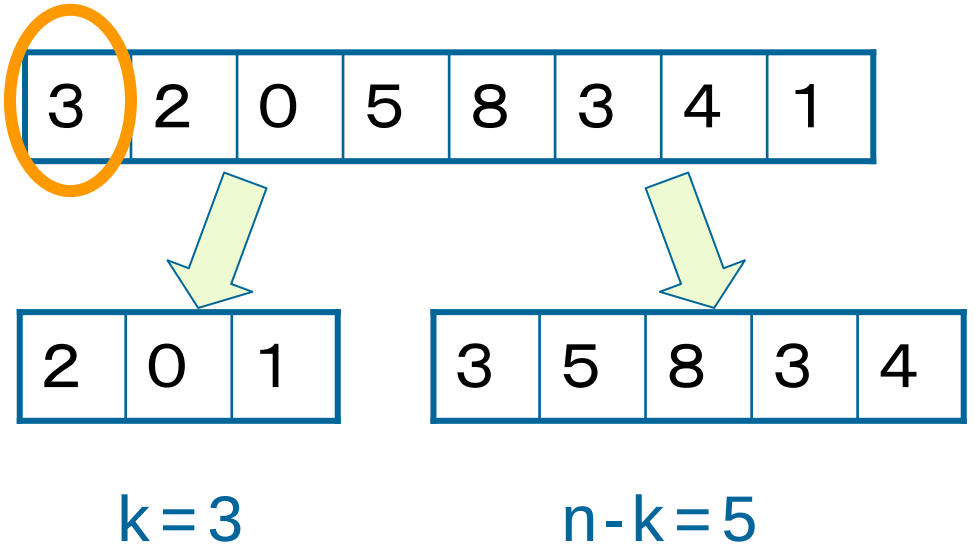
(0) A[0], …, A[n-1]が全て同じ要素
➔いずれかの要素を出力，終了

p＝5 の場合



(1) A[0], …, A[n-1]から
ひとつの値（軸要素）を選ぶ

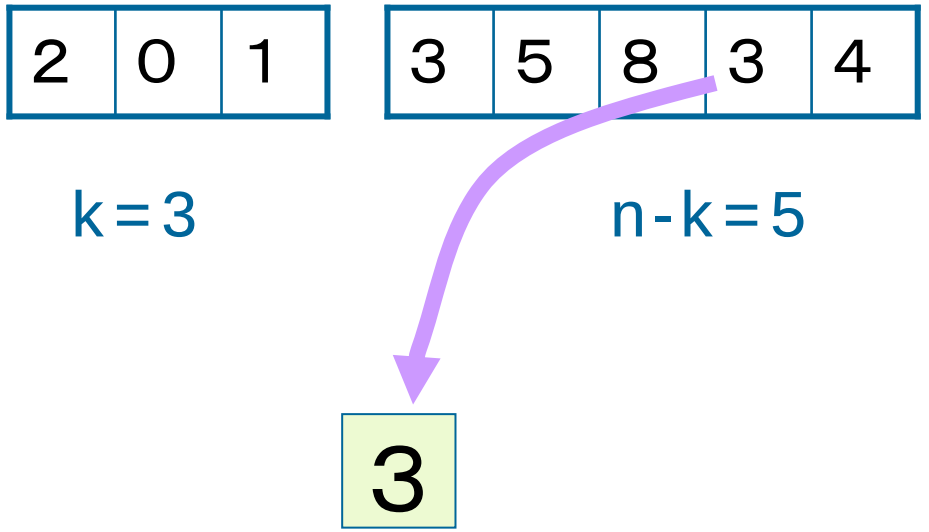
(2) 軸要素未満の要素と
それ以外に分割
k＝軸要素未満の要素数



QUICKSELECTの手順(その2)

(3) p≦k のとき,
　　軸要素未満の配列から第 p 要素を再帰的に選択
　p > k のとき,
　　軸要素以上の配列から第 p-k 要素を再帰的に選択

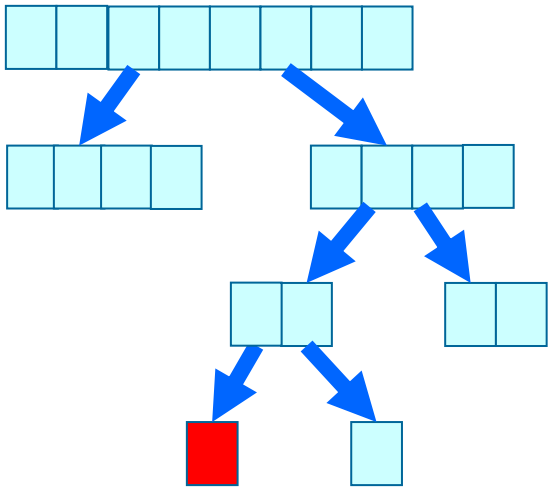
p=5 の場合



p=5>3=k なので
右の配列から
第 5-3=2 要素を
再帰的に選択

QUICKSELECTの時間計算量（その1）

- 大きさ n の配列の分割は cn 時間で可能
- 分割後の配列の大きさは
  軸要素の選び方に依存



- 運がよい場合：毎回の分割で
  配列の大きさが半分になる

➔ $cn + c(n/2) + c(n/4) + \ldots + c\cdot 2 + c\cdot 1$
$\leqq cn \times 2 = O(n)$ 線形時間

- 普通の場合：毎回の分割で配列の大きさが
  $(1-\varepsilon)$倍以下になる（ε は定数，$0 < \varepsilon < 1$）

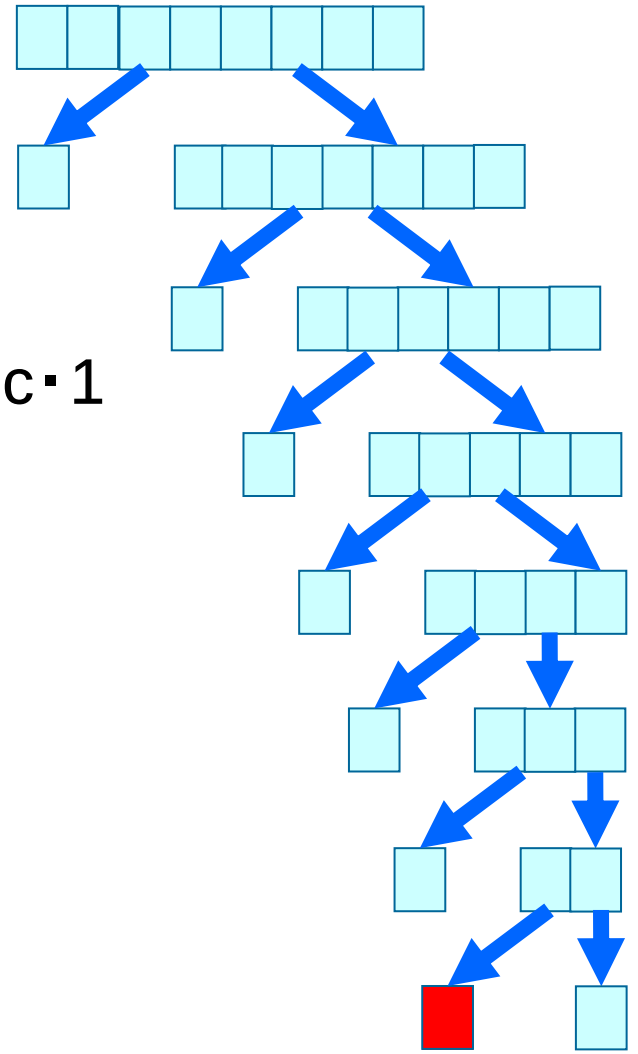
➔ $cn + c(1-\varepsilon)n + c(1-\varepsilon)^2 n + c(1-\varepsilon)^3 n$
$+ \ldots + c\, n/(1-\varepsilon) + c\cdot 1$
$\leqq cn \times 1/(1-\varepsilon) = O(n)$ 線形時間

QUICKSELECTの時間計算量（その2）

- 大きさ n の配列の分割は cn 時間で可能

- 最悪の場合（運が悪い場合）：毎回の分割で
  配列の大きさが1ずつ減少
  ➔ $cn + c(n-1) + c(n-2) + \ldots + c \cdot 2 + c \cdot 1$
  $\leqq cn(n-1)/2 = O(n^2)$

- 軸の選び方をランダムにする
  ➔ 平均的な時間計算量は O(n)
  解析はかなり面倒なので省略



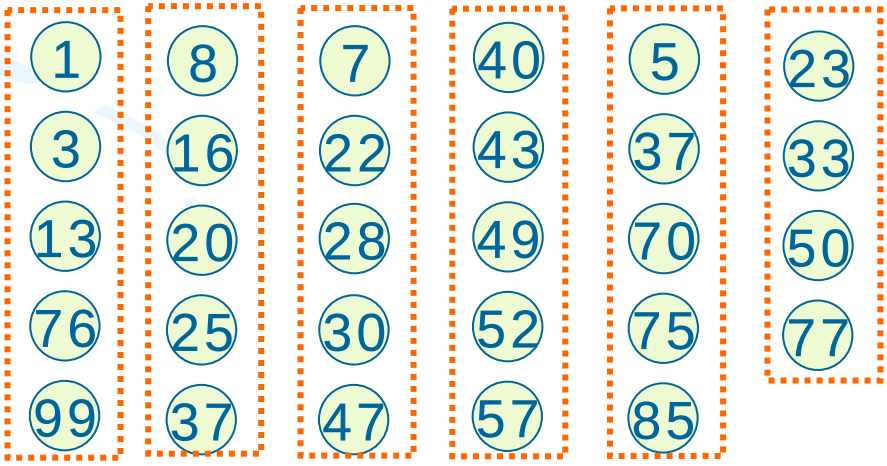
SELECTの概要

- QUICKSELECTで時間計算量がO(n)になるケース
  - 毎回の分割で配列の大きさが(1-ε)倍以下になる
    （εは定数，0<ε<1)
- どんな入力に対しても，このケースが起こるように
  軸要素の選び方を工夫する
    ➔アルゴリズムSELECT　(ε = 1/4)

SELECTでの軸要素の選び方

- 説明を簡単にするため，全ての要素が異なると仮定
  - A[i] と A[k] が等しい場合は，
    配列の番号i, k によって大小関係を決めればよい
    （i<k ならば A[i]が「小さい」, i > k ならばA[k]が「小さい」）

手順1： n 個の要素を5個ずつのグループに分ける．
5個未満のグループが一つ出来たら，それは別にしておく．
➔ ちょうど5個のグループの数は $\lfloor n/5 \rfloor$



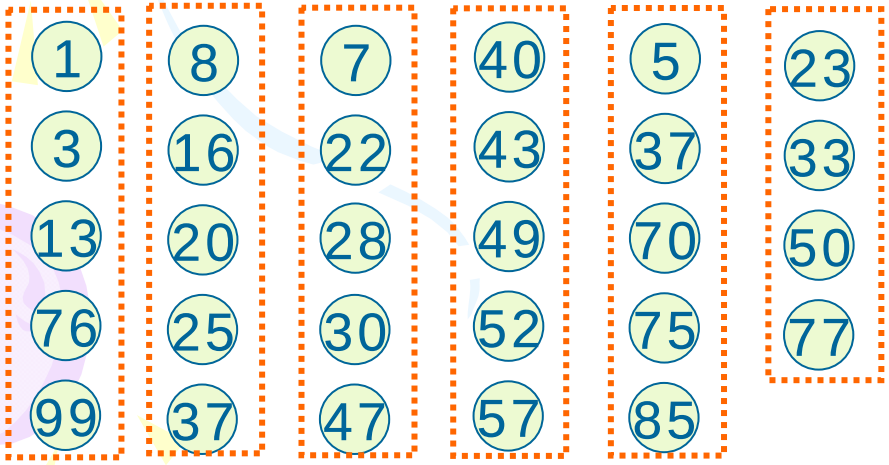
$n = 29$

$\lfloor n/5 \rfloor = 5$

SELECTでの軸要素の選び方

手順2：$\lfloor n/5 \rfloor$個の各グループにおいて，中央値（3番目に小さい値）を計算し，取り出す．

手順3：手順2で取り出した$\lfloor n/5 \rfloor$個の要素に対し，中央値（$\lceil \lfloor n/5 \rfloor /2 \rceil = \lfloor (n+5)/10 \rfloor$番目に小さい値）を計算し，軸要素 a とする



$n = 29$

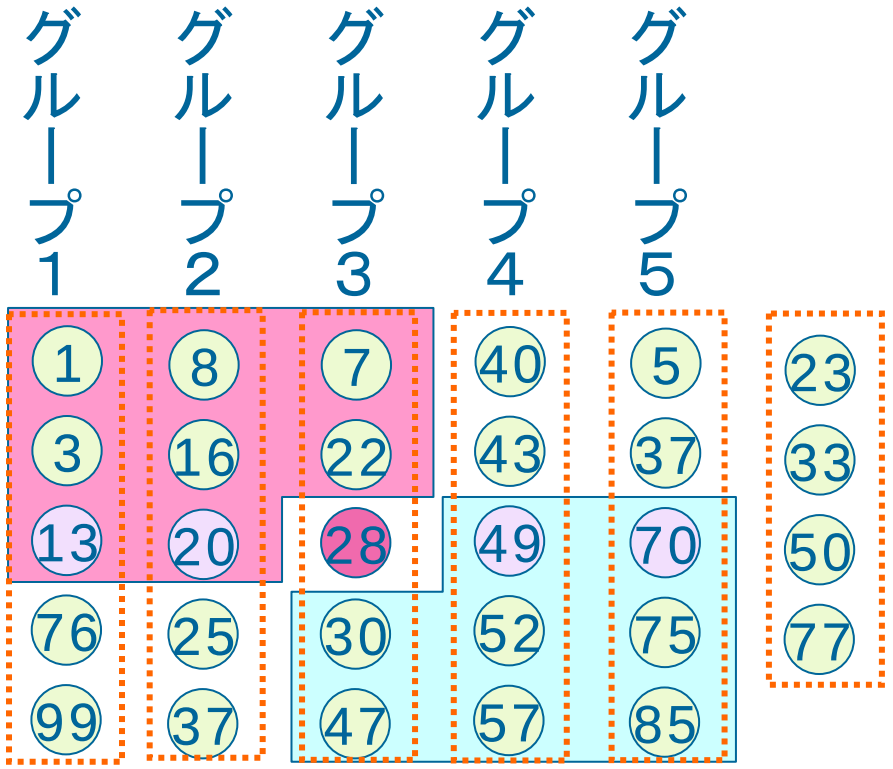
$\lfloor n/5 \rfloor = 5$

中央値の集合
13, 49, 70, 28, 20
この中の中央値は
28← 軸要素 a

各反復での要素数の減少率

グループ1，2のそれぞれの中央値は軸要素 a 未満
➔グループ1，2には a未満の要素が3つ以上存在
➔a 未満の要素数
≧3×3グループ-1＝8

グループ4，5のそれぞれの中央値は軸要素aより大きい
➔グループ4，5には aより大きい要素が3つ以上存在
➔a より大きい要素の数
≧3×3グループ-1＝8



軸要素 a より大きい要素の割合は？
軸要素 a より小さい要素の割合は？

各反復での要素数の減少率

軸要素 a は，各グループの中央値の中の中央値，$\lceil \lfloor n/5 \rfloor /2 \rceil = \lfloor (n+5)/10 \rfloor$ 番目に小さい要素

「グループの中央値＜a」となるグループ数は $\lfloor (n+5)/10 \rfloor - 1$

➔各グループには a未満の要素が3つ以上存在

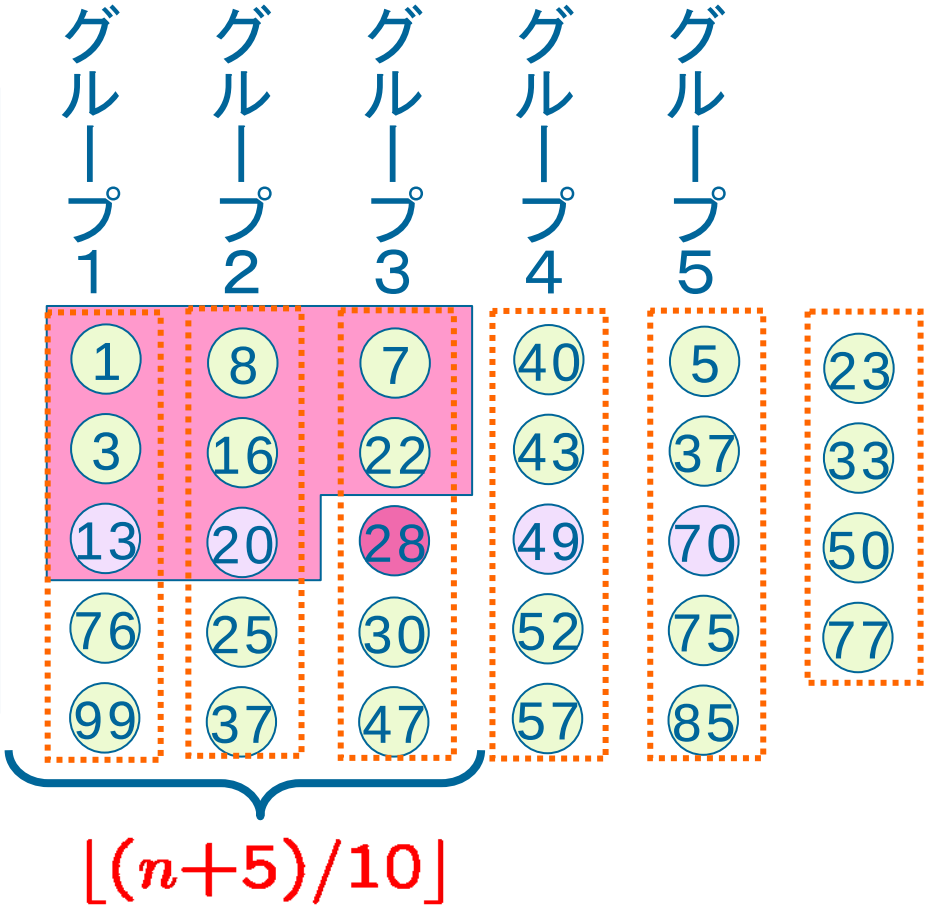
➔a 未満の要素数

$\geqq 3 \times \lfloor (n+5)/10 \rfloor - 1$

n≧50のとき，この値はn/4以上

n≧50,かつ見つけたい要素が軸要素以上➔

次の反復での要素数≦3/4n



各反復での時間計算量の解析

軸要素 a は，各グループの中央値の中の中央値，$\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$ 番目に大きい要素

「グループの中央値＞a」となるグループ数は $\lfloor n/10 \rfloor - 1$

➔各グループには aより大きい要素が3つ以上存在

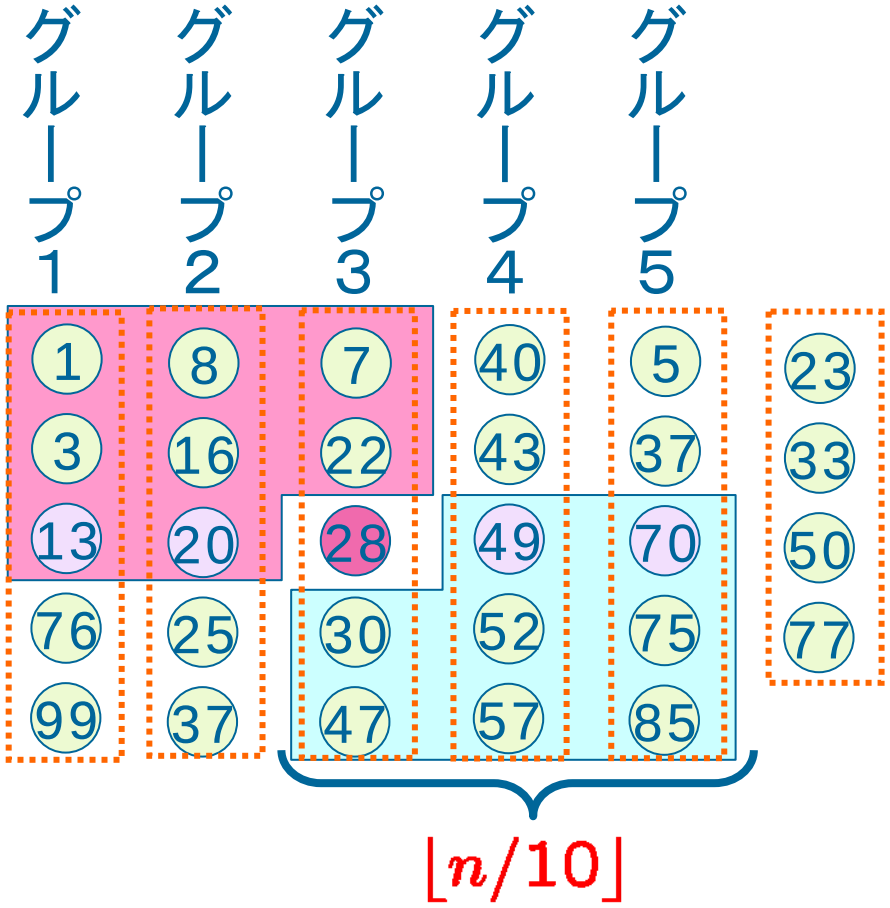
➔a より大きい要素の数

$\geqq 3 \times \lfloor n/10 \rfloor - 1$

n≧80のとき，この値はn/4以上

n≧80,かつ見つけたい要素が軸要素未満➔

次の反復での要素数≦3/4n



SELECTの時間計算量の解析

- アルゴリズムSELECTでは，
  - n≧80のとき，一回の反復で要素数が n から 3/4n 以下に減少する
  - n<80のとき，定数時間で第p要素を選択することが可能（要素数nが定数以下なので）

➔整数 k が $(3/4)^k \geqq 80$, $(3/4)^{k+1} < 80$を満たすとすると，合計の時間計算量は

$c\{n + 3/4n + (3/4)^2 n + \ldots + (3/4)kn + 80\}$

$\leqq c \times 4n + 80$ $c = O(n)$ となる

今日のレポート（締切：6/3）

- 次の要素集合に対して第9要素(9番目に小さい要素，p＝9)を求めたい．QUICKSELECTおよびSELECTを適用したときのアルゴリズムの挙動を詳しく説明せよ．なお，QUICKSELECTでの軸の選び方は，クイックソートの(c)を使うこと．

50, 23, 67, 43, 91, 29, 45, 77, 69, 89,
31, 33, 73, 22, 60, 48, 41, 97, 84, 30