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3回

クイックソート，バケットソート，基数ソート

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

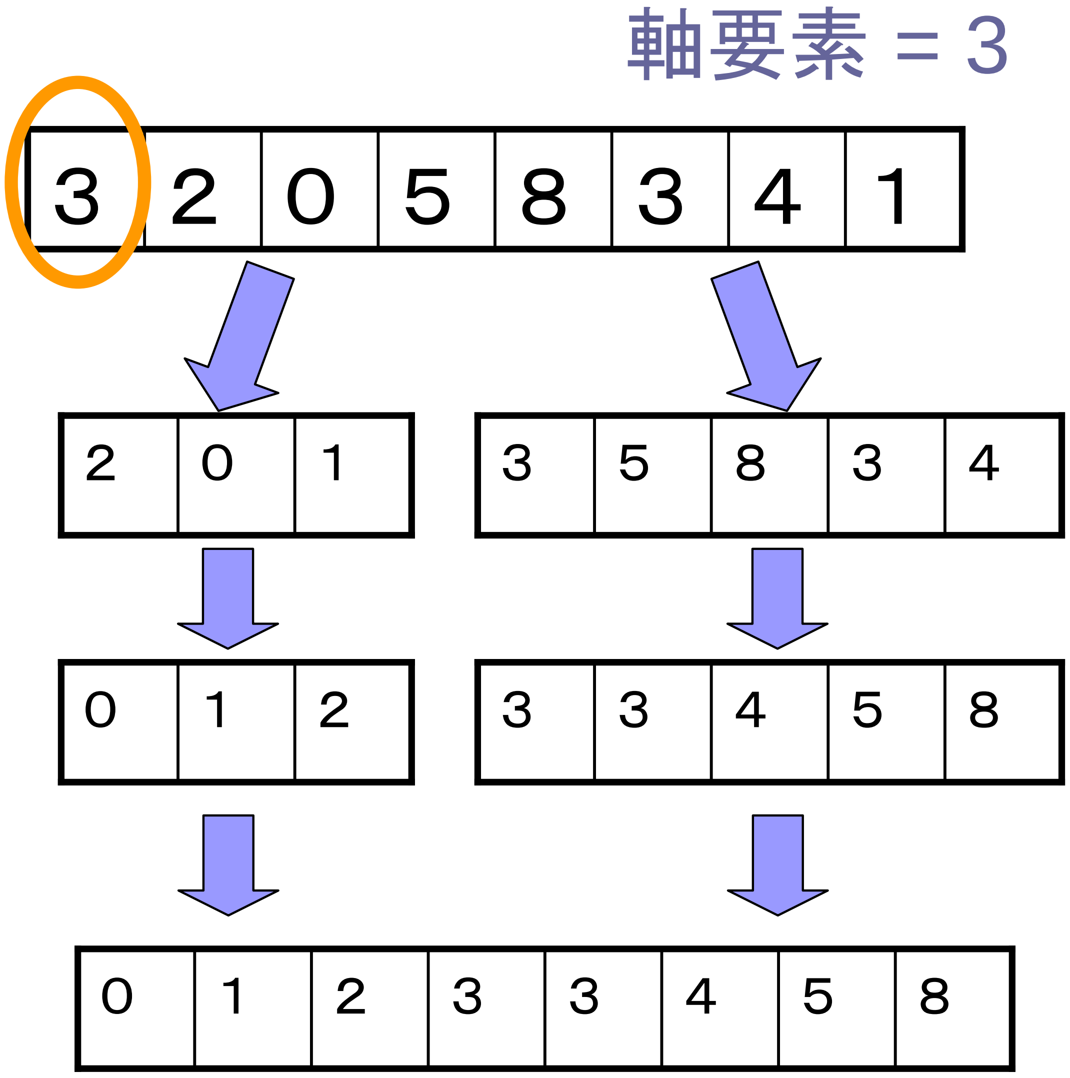
クイックソートのアイディア(p96-101)

①A[1], …, A[n]から
ひとつの値（軸要素）を選ぶ

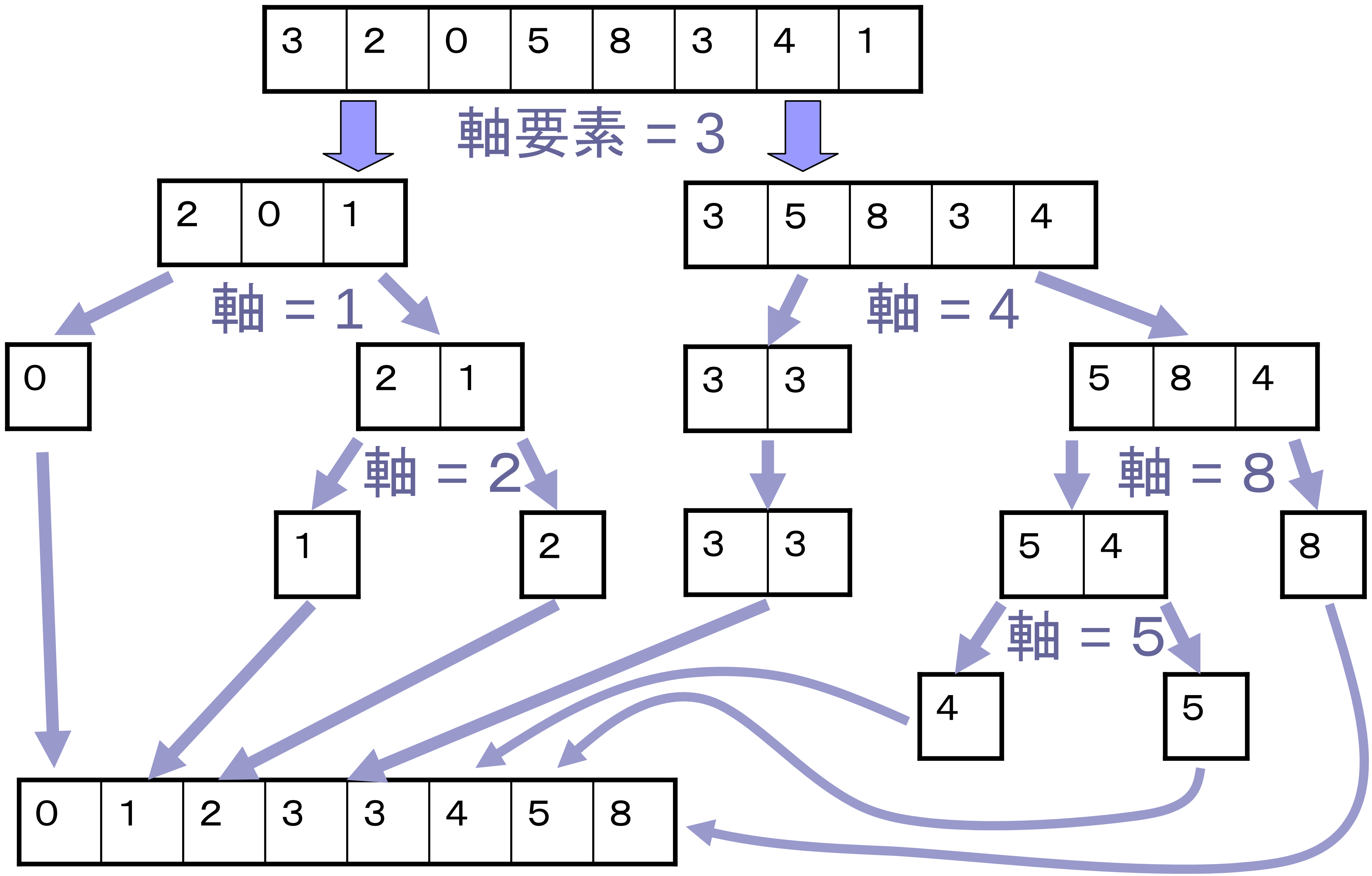
② 軸要素未満の要素と
それ以外に分割

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

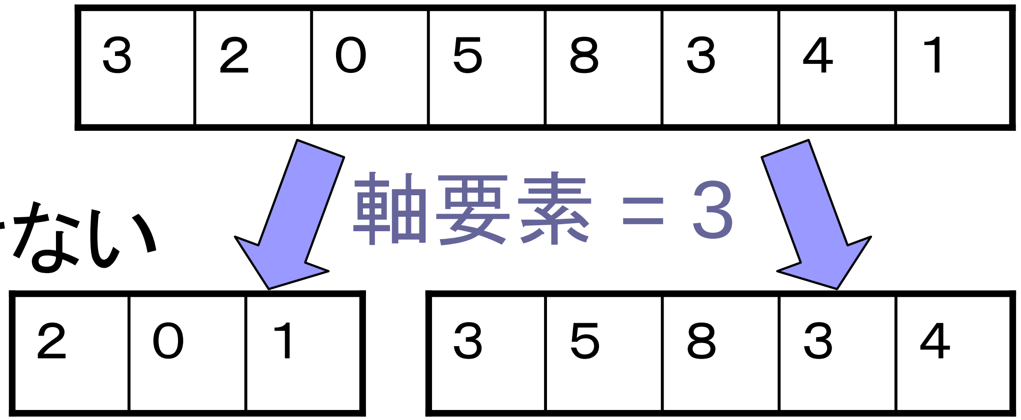

3 2 0 5 8 3 4 1
軸要素 = 3
2 0 1
3 5 8 3 4
軸 = 1
軸 = 4
0
2 1
3 3
5 8 4
軸 = 2
軸 = 8
1
2
3 3
5 4
8
軸 = 5
4
5
0 1 2 3 3 4 5 8

軸要素の選び方(その1)

良い選び方:

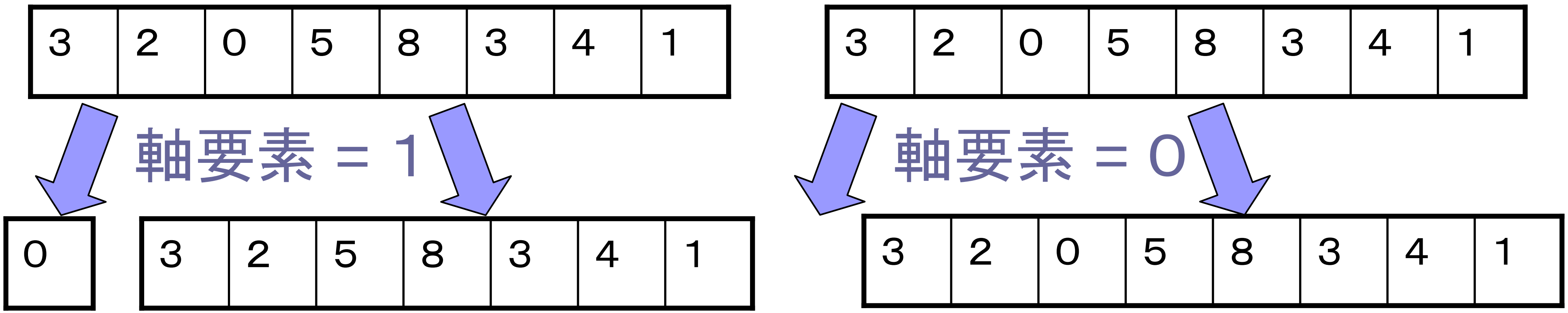
配列をほぼ二等分する

軸要素の選択に時間をかけない



悪い選び方:

2分された配列の大きさがアンバランス

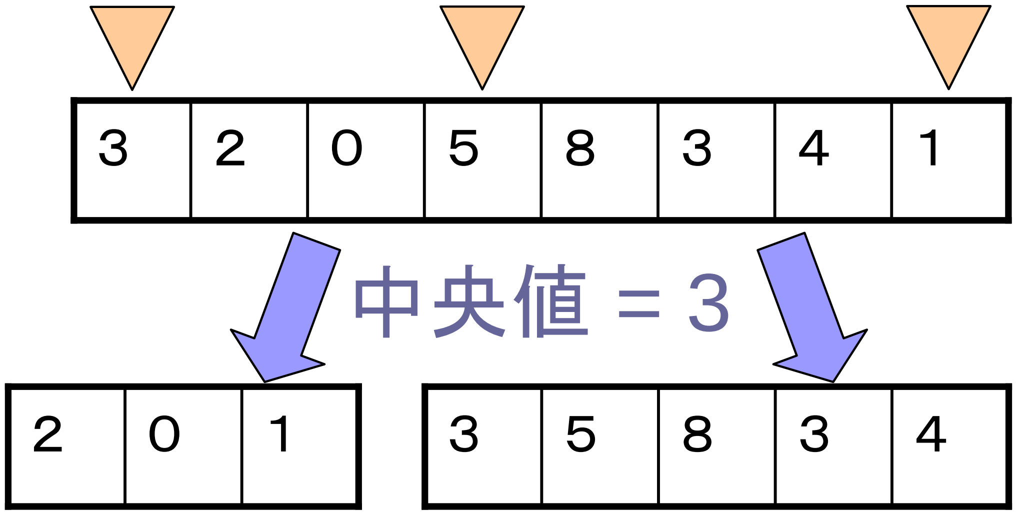


軸要素の選び方（その2）

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

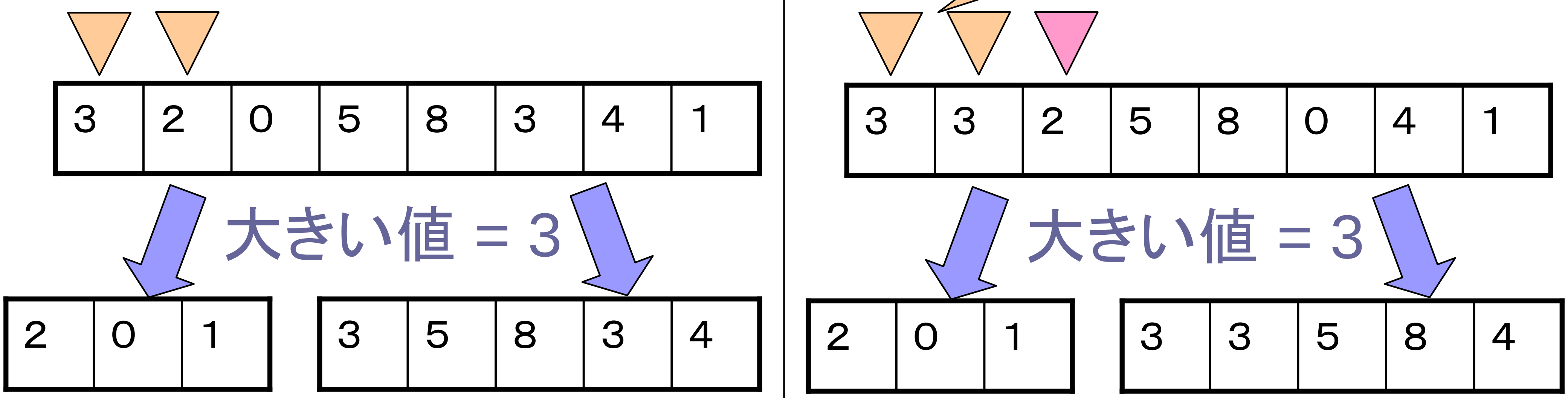


軸要素の選び方（その3）

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

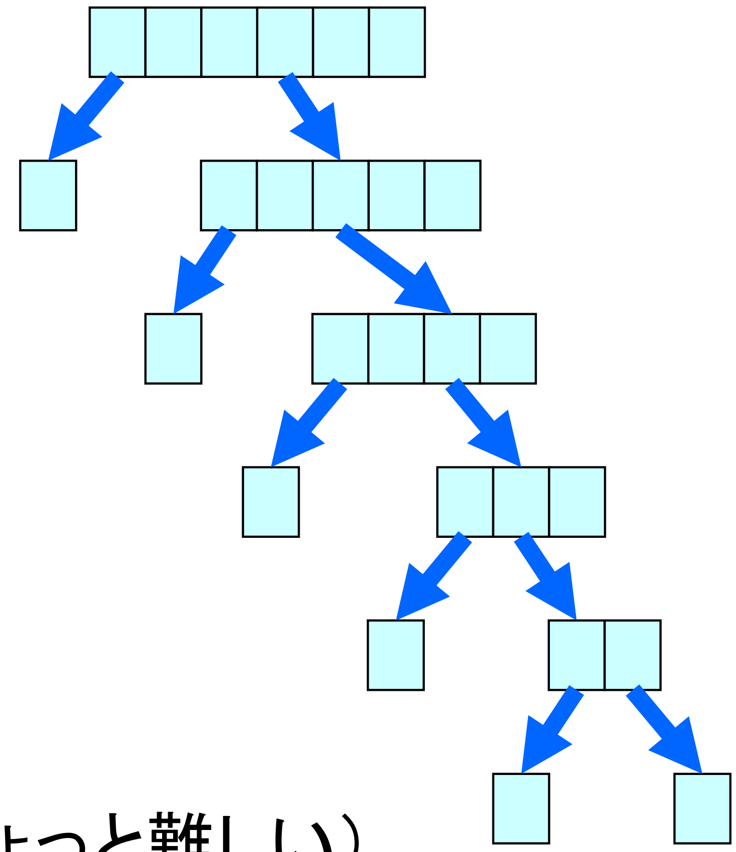
A[1]=A[2]
⇒A[3]と比較



クイックソートの計算時間

- 各レベルでの分割に必要な計算時間の合計＝ cn
- 分割の深さ:最悪の場合 n

∴最悪計算時間＝cn^2 = $O(n^2)$



実用的には，数列がほぼ半分に
分割されることが多い

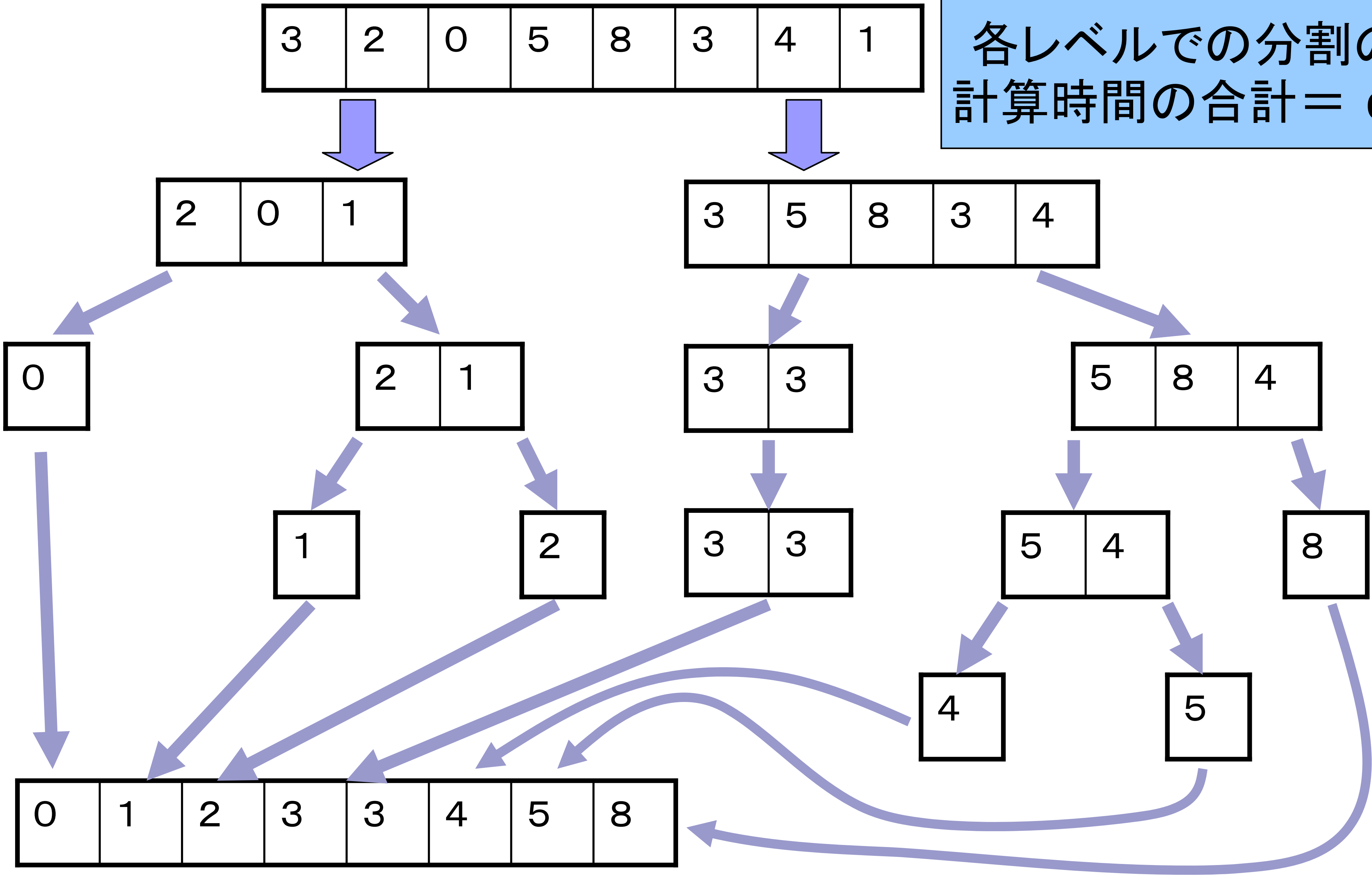
➔ 分割の深さは O(log n) に近い

➔ 実用上の計算時間はO(n log n) に近い

平均時間計算量は O(n log n) （解析はちょっと難しい）

クイックソートの各レベルでの計算時間

各レベルでの分割の
計算時間の合計＝ cn



データ構造とは？

- アルゴリズムの中で，与えられた問題に関連するデータ集合を管理するための道具
- 良いデータ構造とは？
  - データ管理に必要な時間が短い
  - シンプル
  - 必要な領域計算量（記憶容量，領域量）が小さい

集合を管理する

- 整数の集合が与えられている

  4, 5, 8, 2, 9, 1, 3

- ときどき，新しい整数が追加される

  7を追加 ➔ 4, 5, 8, 2, 9, 1, 3, 7

- ときどき，ある整数が削除される

  9 を削除 ➔ 4, 5, 8, 2, 1, 3, 7

- アルゴリズム（プログラム）の中でどのように表現するか？

集合を管理する: 配列の利用(その1)

配列A[0], A[1], ..., A[N]を使って表現(N: 十分大きな数)
集合の中に整数 k が　ある➔ A[k] = 1, ない➔ A[k] = 0

4, 5, 8, 2, 9, 1, 3 ⇒

0	1	2	3	4	5	6	7	8	9
0	1	1	1	1	1	0	0	1	1

集合の中に整数 k が複数存在する場合も対応可能
A[k] = 集合の中に存在する k の個数

4, 1, 8, 2, 8, 1, 3, 1 ⇒

0	1	2	3	4	5	6	7	8	9
0	3	1	1	1	0	0	0	2	0

整数 k を追加 ➔ A[k] を1増やす --- O(1) 時間で可能
整数 k を削除 ➔ A[k] を1減らす --- O(1) 時間で可能

欠点: 整数 N より大きい整数が追加されるとダメ
実際の集合のサイズより大きい配列が必要

無駄な領域計算量

集合を管理する:
配列の利用(その2)

集合に含まれる整数を配列に代入(非負の整数を仮定)

空のところには-1を入れる

4, 5, 8, 2, 9, 1, 3 →

0	1	2	3	4	5	6	7	8	9
4	5	8	2	9	1	3	-1	-1	-1

7 を追加 →

0	1	2	3	4	5	6	7	8	9
4	5	8	2	9	1	3	7	-1	-1

O(1)時間で可能

5 を削除 →

0	1	2	3	4	5	6	7	8	9
4	8	2	9	1	3	7	-1	-1	-1

“5”より後ろの整数を移動させる

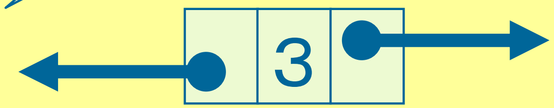
→ O(n)時間 (n: 集合のサイズ)

集合を管理する: 双方向リストの利用(p26-32)

- 双方向リストを利用して集合を表現する
  - 必要な領域計算量は集合のサイズに等しい
  - 整数の追加，削除はO(1)時間で可能

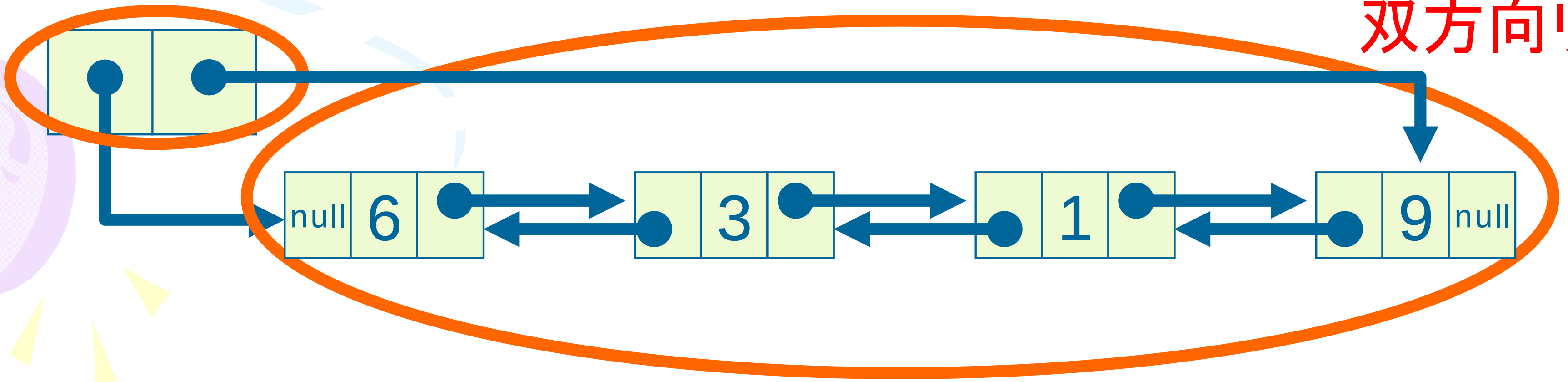
C言語では，構造体により実現可能

セルの構成:
要素(整数)
前のセルへのポインタ
次のセルへのポインタ



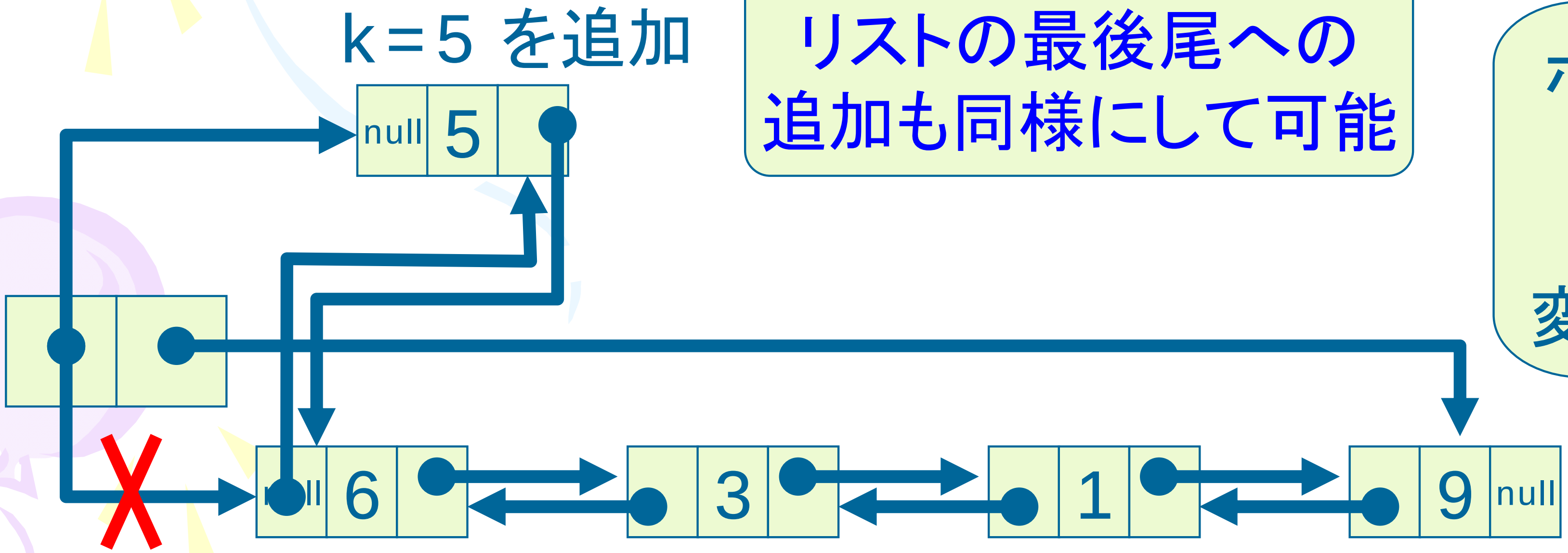
双方向リストの最初と最後のセルへのポインタ

双方向リストの本体



双方向リストの利用：要素の追加

- リストの先頭への整数kの追加---O(1) 時間で可能
  1. 新しいセル C を準備，セルに整数 k と書く
  2. Cの前のセルへのポインタを，null とする
  3. Cの次のセルへのポインタを，現在の最初のセルとする
  4. 現在の最初のセルの前のセルへのポインタを，Cに変更
  5. 連結リストの最初のセルへのポインタを，Cに変更



リストの最後尾への追加も同様にして可能

ポインタ変更の順番を間違えると変なリストになる

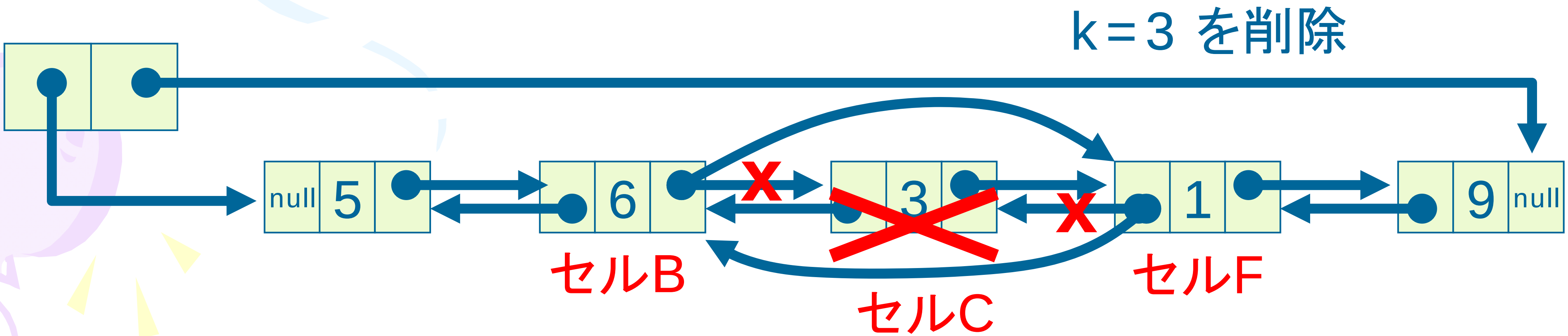
双方向リストの利用：要素の削除

- 整数 k のセル C の削除（Cへのポインタが既知）

  --- O(1) 時間で可能

1. C の前のセルを B, 次のセルを F とする
2. セル B の前のセルを F に変更
3. セル F の次のセルを B に変更
4. セル C を消去

セル C の位置がリストの最初または最後の場合は修正が必要

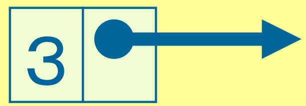


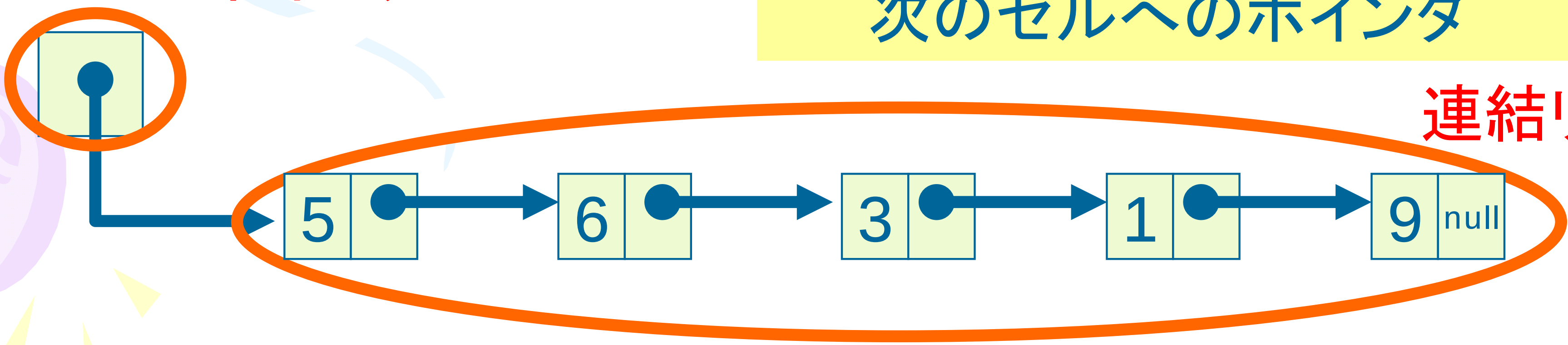
集合を管理する: 連結リストの利用

- 連結リスト: 双方向リストをシンプルにしたもの
  - 必要な領域計算量は集合のサイズに等しい
  - 要素の追加はO(1)時間で可能(双方向リストのときと同じ)
  - 先頭の要素の削除はO(1)時間で可能
  - 先頭以外の要素の削除はO(1)時間では不可能

連結リストの
最初の
セルへのポインタ

セルの構成:
要素(整数)
次のセルへのポインタ

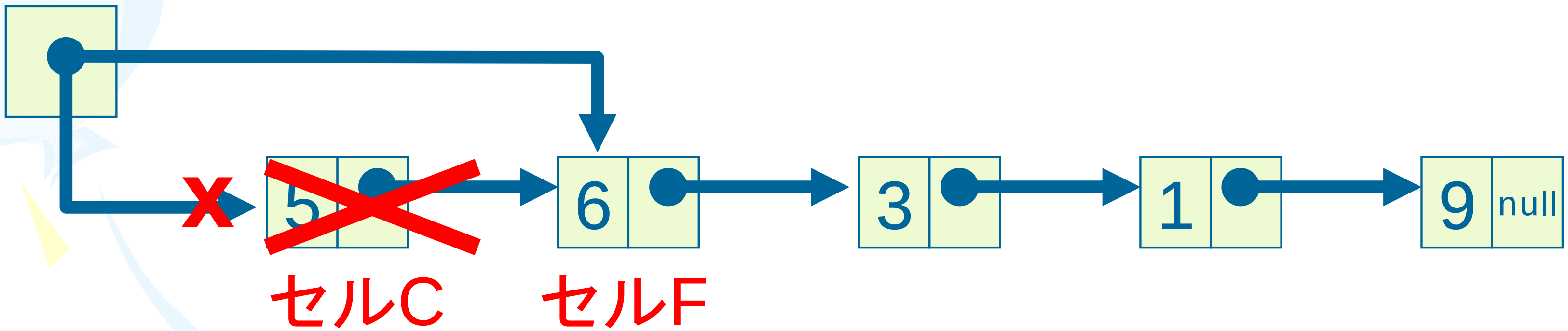




連結リストの
本体

連結リストの利用：先頭の要素の削除

- 先頭にあるセル C の削除 --- O(1) 時間で可能
  1. C の次のセルを F とする
  2. 先頭のセルへのポインタを，Fに変更
  3. セルCを削除



- 先頭以外のセルの削除はなぜ難しい？
  - 削除したいセルの前にあるセルがどれか，わからないから
  - 前のセルを知るには，リストを先頭から調べる必要有り

    ➔O(1)時間では不可能

バケットソート(p89-91)

- 整列の対象となっている整数の範囲が事前にわかっている(例:各入力値a_iが$0 \leqq a_i \leqq m-1$を満たす)
- 整数の範囲(mの大きさ)があまり大きくない

→ バケットソートにより高速に整列が可能
時間計算量 $O(m+n)$

バケット(bucket)

= バケツ

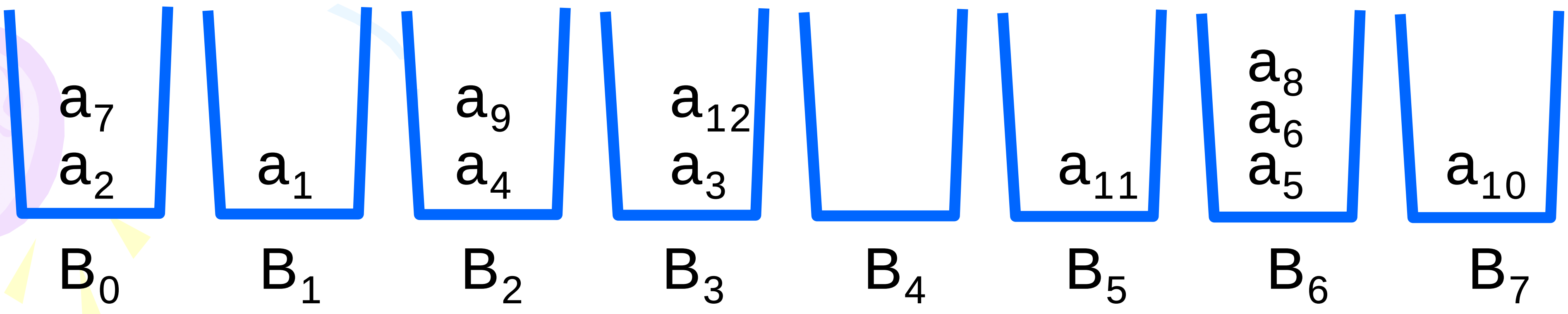
バケットソートのアイディア

- $0 \leqq a_i \leqq m-1$を満たす整数$a_1, a_2, \ldots, a_n$ を整列
- 各 j $(0 \leqq j \leqq m-1)$に対し，空のバケツB_jを用意
- 各 a_i に対し，$a_i = j$ ならば a_i をバケツB_jに入れる
- バケツ $B_0, B_1, \ldots, B_{m-1}$ の中身を（入れた順に）並べる

➔ソートが完了

0以上7以下の整数

1	2	3	4	5	6	7	8	9	10	11	12
1	0	3	2	6	6	0	6	2	7	5	3



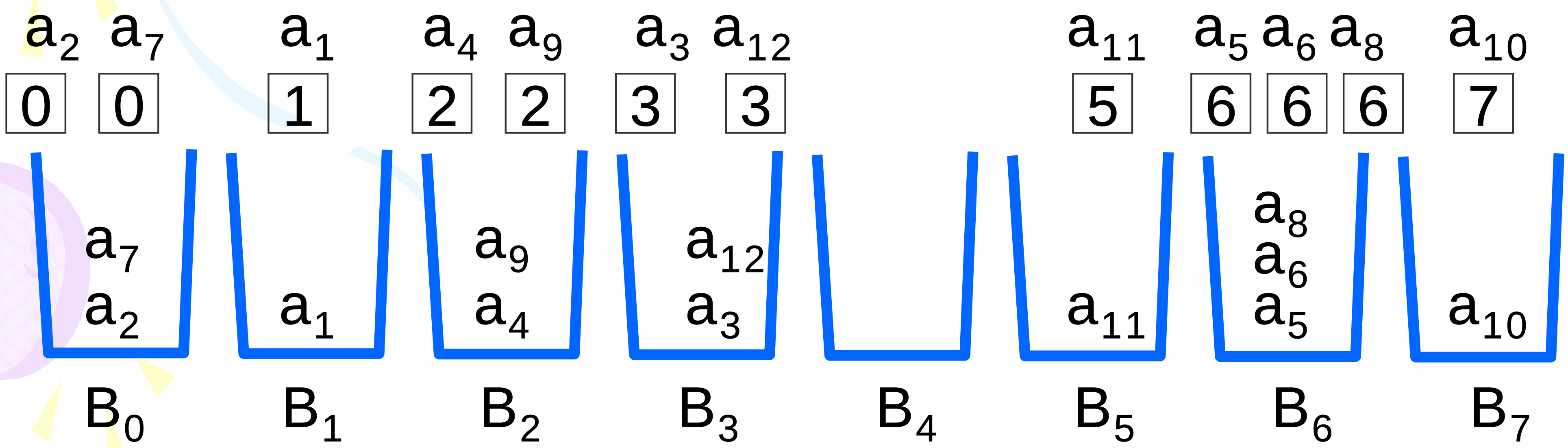
バケットソートのアイディア

- バケツ B_0, B_1, …, B_{m-1} の中身を（入れた順に）並べる

➔ソートが完了

同じバケツの中の要素は
元の順番通りに並べられている

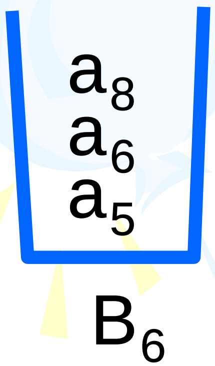
1	2	3	4	5	6	7	8	9	10	11	12
1	0	3	2	6	6	0	6	2	7	5	3

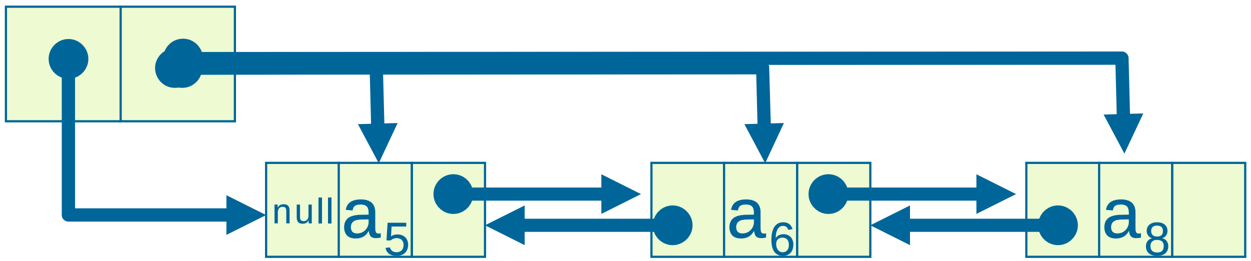


バケットソートの実現

- 各バケツは双方向リストで実現
- 挿入するときは最後尾に
- 削除するときは先頭から

同じバケツの要素は元の順番通りに並ぶ





バケットソートの計算時間

- m個のバケツを準備➔O(m)時間
- n個の要素をバケツに挿入➔O(1)xn=O(n)
- バケツに入れた要素を取り出す➔O(m+n)時間

∴ 全体でO(m + n) 時間

空間計算量もO(m + n)

※ 整数の範囲が狭いときには有効

m が大きいときは時間計算量も空間計算量も膨大

基数ソート:カードを並べる方法 (p91-94)

- 例:3桁の数字からなる学籍番号の書かれたカードを番号順に(机の上で)並べたい

815	256	974	370	056	532	●●●●

- よく使われる方法:
  - まず百の位の値によってグループ分け,ソート
  - 次に各グループを十の位によってグループ分け,ソート
  - 最後に各グループを一の位によってソート
  - ソートされたカードをまとめる

机の上で実現可能か?

カードを並べる

815　256　974　370　056　532　・・・・

百の位＝0のグループ　百の位＝1のグループ　百の位＝2のグループ　・・・・

十の位＝0　十の位＝1　・・　十の位＝0　十の位＝1　・・　十の位＝0　十の位＝1　・・

0　1　・・

グループの数が多くなりすぎて，
机の上に収まらない．．．

基数ソートのアイディア

- よく使われる方法:
  - まず百の位の値によってグループ分け，ソート
  - 次に各グループを十の位によってグループ分け，ソート
  - 最後に各グループを一の位によってソート
  - ソートされたカードをまとめる
- 基数ソートの手順:
  - まず一の位の値によってバケットソート
  - 次に十の位によってバケットソート
  - 最後に百の位によってバケットソート

なぜこの方法で
ソートが
できるのか？

基数ソートの例

- 簡単のため，各桁の数字は0,1,2,3のみとする

123	013	322	102	021	311	222	110	200

まず一の位の値によってバケットソート

110	200	021	311	322	102	222	123	013

次に各グループを十の位によってバケットソート

200	102	110	311	013	021	322	222	123

バケットソートを利用
➔十の位が同じ数字ならば，元の順番（一の位に関する昇順）通りに並ぶ
➔下２桁に関して昇順に並んでいる

基数ソートの例

次に各グループを十の位によってバケットソート

200	102	110	311	013	021	322	222	123

下２桁に関して昇順に並んでいる

最後に各グループを百の位によってバケットソート

013	021	102	110	123	200	222	311	322

バケットソートを利用

➔百の位が同じ数字ならば，元の順番（下２桁に関する昇順）通りに並ぶ

➔下３桁に関して昇順に並んでいる

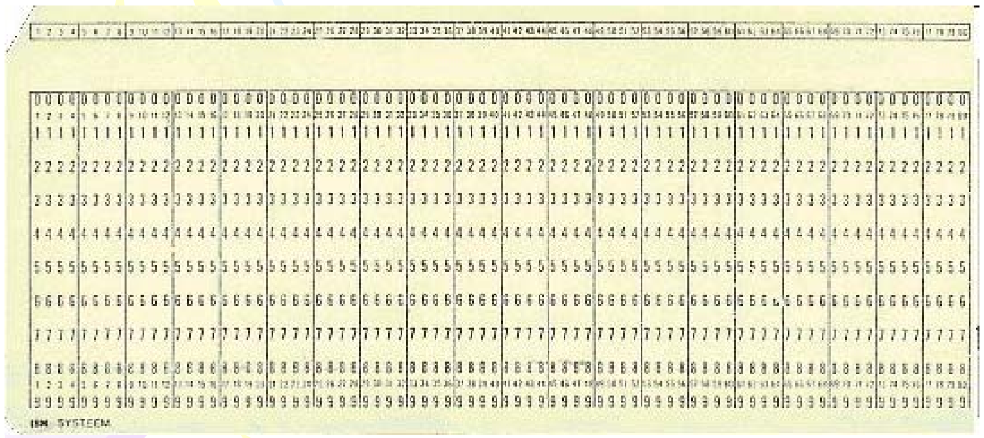
基数ソートの計算時間

- 桁数が K 以下の n 個の（非負）整数をソートする場合
  - バケットソートを K 回実行
  - 一回のバケットソートでは10個(0,1,2,..,9)個のバケツを使用➔ O(10 + n) =O(n)時間
  - 全体では O(n) x K = O(Kn)時間
- K 桁以下の m 進数をソートする場合は O(K(m + n)) 時間

- K文字以下のアルファベットで書かれた名前・単語のソートも可能
  - 文字の種類が m, 名前・単語の数が n ならば O(K(m+n)) 時間

実際に使われていた基数ソート

- 基数ソートは実際にカードを機械を使ってソートするときに使われていた
- カードには，各桁ごとに対応する数字のところに穴を開ける
- 基数ソートをするときには，各桁ごとに0,1,2,...,9に対応する穴を参照する



http://www.museumwaalsdorp.nl/computer/en/punchcards.html

http://smoto.mii.kurume-u.ac.jp/~smoto/edu/ala/history/p0.html

演習問題（締切：5／7）

(1) 数列{27, 17, 3, 16, 13, 10, 1, 5}に対して，クイックソートを適用せよ．なお，軸の選び方は(c) をつかうこと．

(2) 以下の英単語にして基数ソートを適用せよ．

COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX

(3) 双方向リストの最後尾に要素を追加することはO(1)時間で出来る．一方，連結リストの最後尾に要素を追加するにはどうすればよいか？その方法と計算時間を説明せよ．

(1), (2) の問題に対しては，ソートの途中の計算過程についても（授業で行なったように）説明すること．