データ構造とアルゴリズム 第9週

掛下　哲郎

kake@is.saga-u.ac.jp

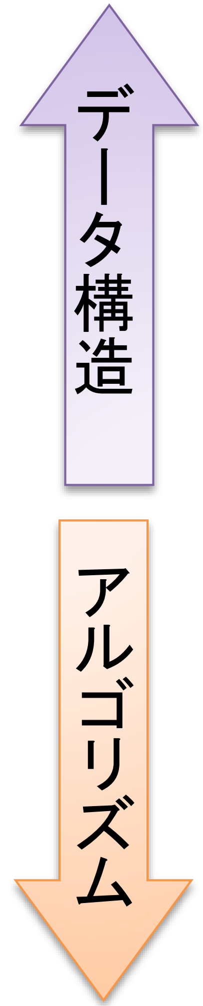
前回のまとめ

- データの整列
  - データを順番に並べるアルゴリズム
- トピック

1. バブルソート(Bubble Sort)
2. 選択ソート（Selection Sort）
3. 挿入ソート（Insertion Sort）
4. シェルソート（Shell Sort）
5. ヒープソート（Heap Sort）

講義スケジュール

週	講義計画
1－2	導入
3	探索問題
4－5	基本的なデータ構造
6	動的探索問題とデータ構造
7	アルゴリズム演習（第1回）
8－9	データの整列
10－11	グラフアルゴリズム
12	文字列照合のアルゴリズム
13	アルゴリズム演習（第2回）
14	アルゴリズムの設計手法
15	計算困難な問題への対応

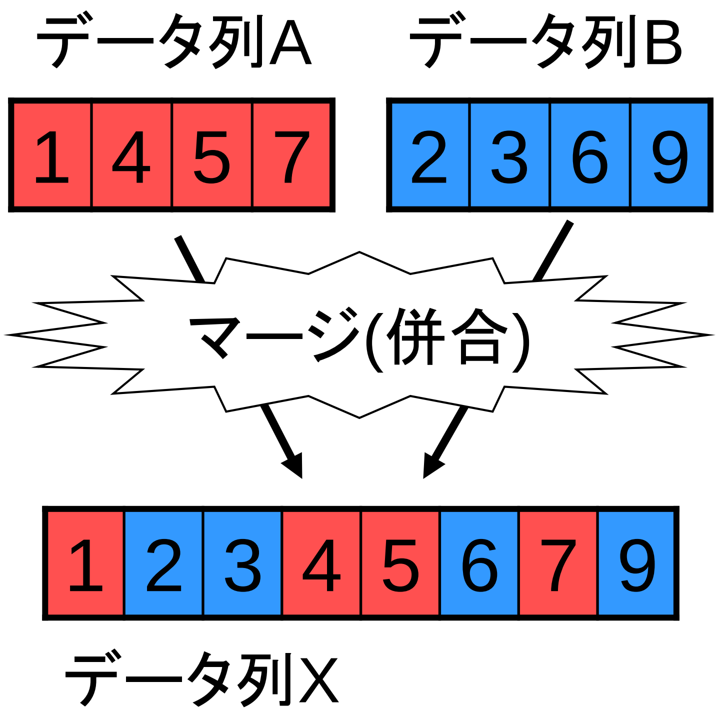


今日の内容

- データの整列（sorting; ソーティング）の続き
- トピック
    - 高度なソート方法
        - マージソート(Merge Sort)
        - クイックソート(Quick Sort)
    - 整列に必要な最小限の比較回数
    - 比較を用いないソート方法
        - バケットソート(Bucket Sort)

マージソート (merge sort)

- 「併合ソート」ともよぶ
- 基本手順
  - データ列をA, Bに分割
  - それぞれのデータ列をソート
  - 2つのソート済みデータ列A, Bを1つのソート済みデータ列Xにまとめる（マージ）
- アイデア
  - マージ（併合）処理が高速にできることを利用

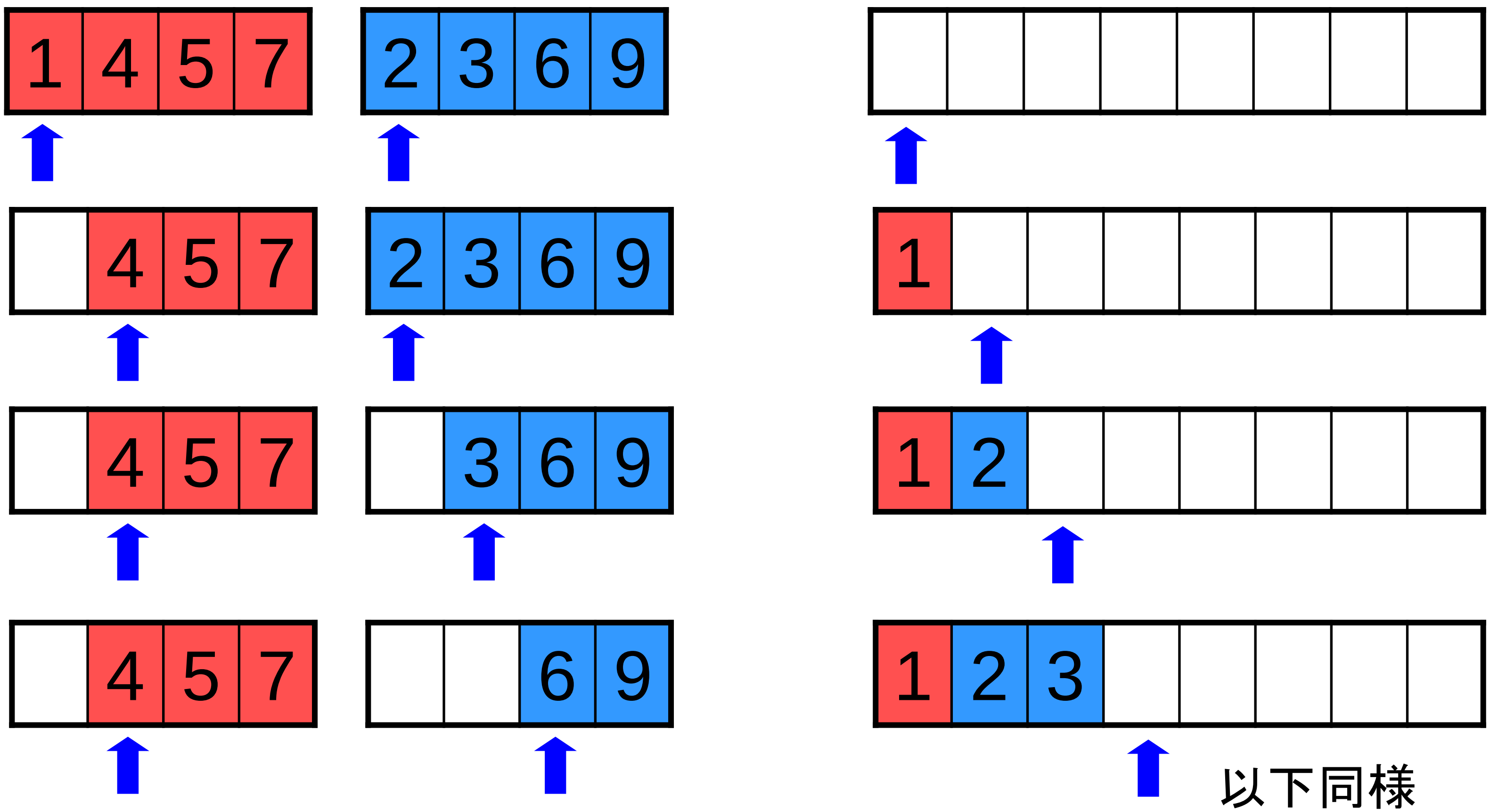


要書き込み

併合(マージ)の計算量

- 併合は，データ数に比例した手間で済む
- 問題：マージしたいデータと同じサイズの領域が必要

O(|A|+|B|)



マージアルゴリズム

- 問題：2つのソート済み列AとBを，1つのソート済み列Xにまとめる.
- アルゴリズム

1. A，B，Xの現在位置を，それぞれの先頭とする.
2. AかBの現在位置に要素がある限り，以下の処理を繰り返す.

   2.1 AとBの現在位置の要素のうち，前者が小さいならば以下の処理を行う.

      2.1.1 Aの現在位置の要素をXの現在位置に移動する.

      2.1.2 Aの現在位置を1つ進める.

   2.2 そうでない場合，以下の処理を行う.

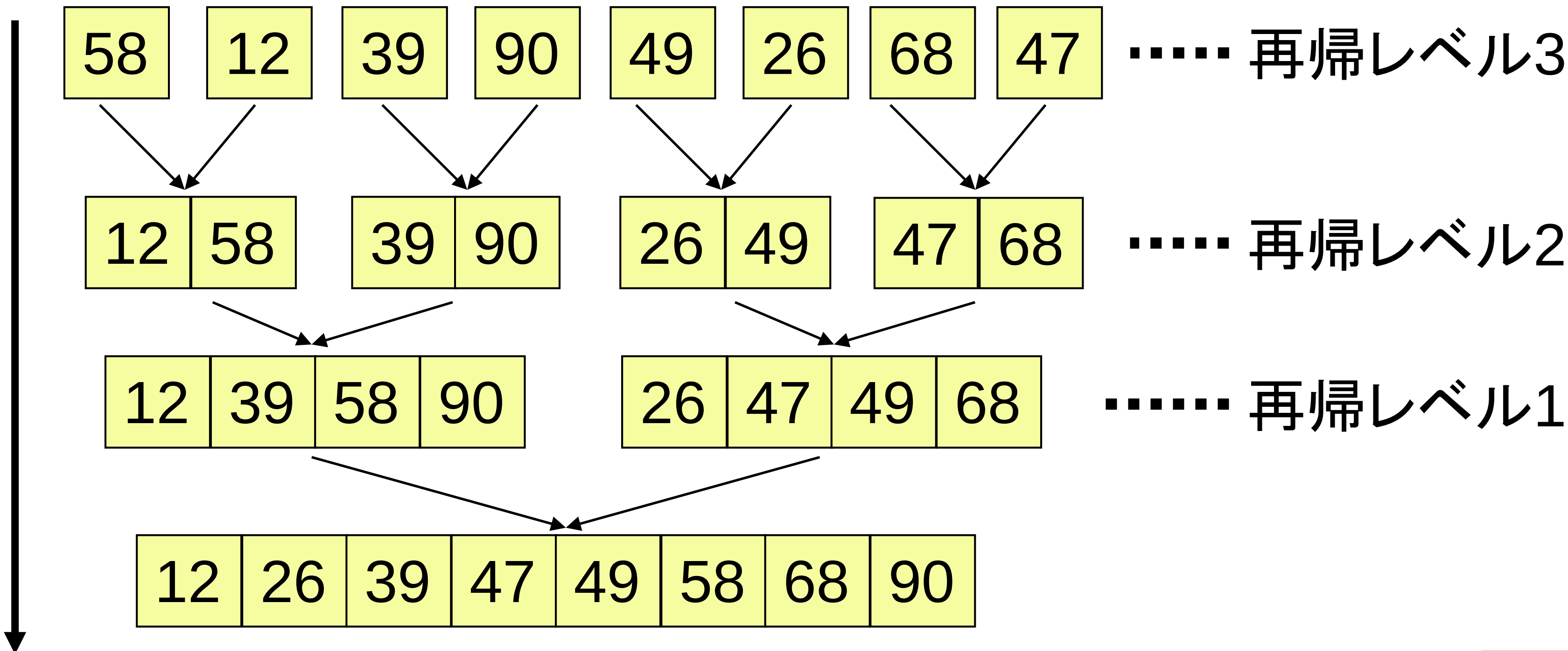
      2.2.1 Bの現在位置の要素をXの現在位置に移動する.

      2.2.2 Bの現在位置を1つ進める.

   2.3 Xの現在位置を1つ進める.

マージソートの計算手順

- ボトムアップ方式のマージソート
  - 長さ1のデータ列からスタートし，隣同士のデータ列の併合を繰り返す．



要書き込み

マージソートのアルゴリズム

1. データ列をAとBに分割する.
2. それぞれのデータ列を，マージソートアルゴリズムを用いて再帰的にソートする.
3. 2つのソート済みデータ列A, Bをマージして，1つのソート済みデータ列Xにまとめる.

再帰呼び出し：ある関数から，その関数自身を呼び出す

マージソートの効率

- データ数nの場合のマージソートの効率をS(n)とする.
- ステップ1 : O(n)
- ステップ2 : 2× S(n/2)
- ステップ3 : O(n)

$$S(n) = 2 \times S(n/2)+n$$

$n = 2^k$と置く. ⇒ $S(2^k) = 2 \times S(2^{k-1})+2^k$

両辺を2^kで割る. ⇒ $S(2^k)/2^k = S(2^{k-1})/2^{k-1}+1$

$S(2^k)/2^k$を$T(k)$と置く. ⇒ $T(k) = T(k-1)+1$

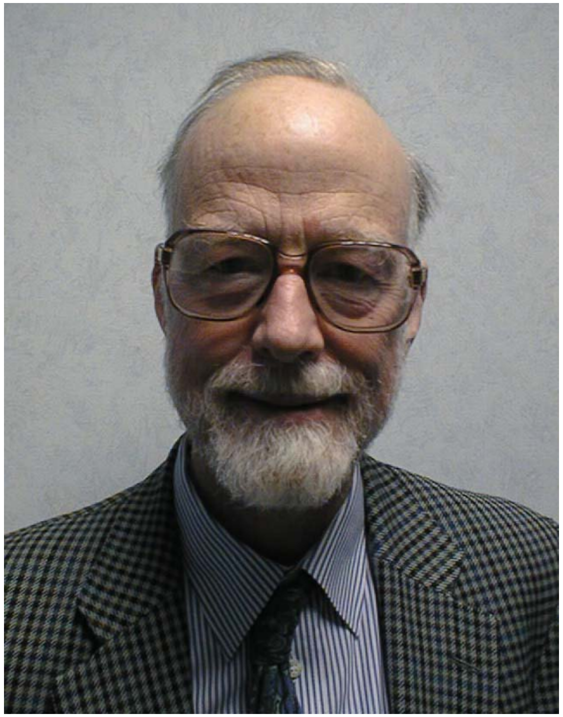
$T(k) = O(k)$ ⇒ $S(2^k) = O(k \times 2^k)$

$n = 2^k$より $k = \log_2 n$ ⇒ $\boxed{S(n) = O(n \log n)}$

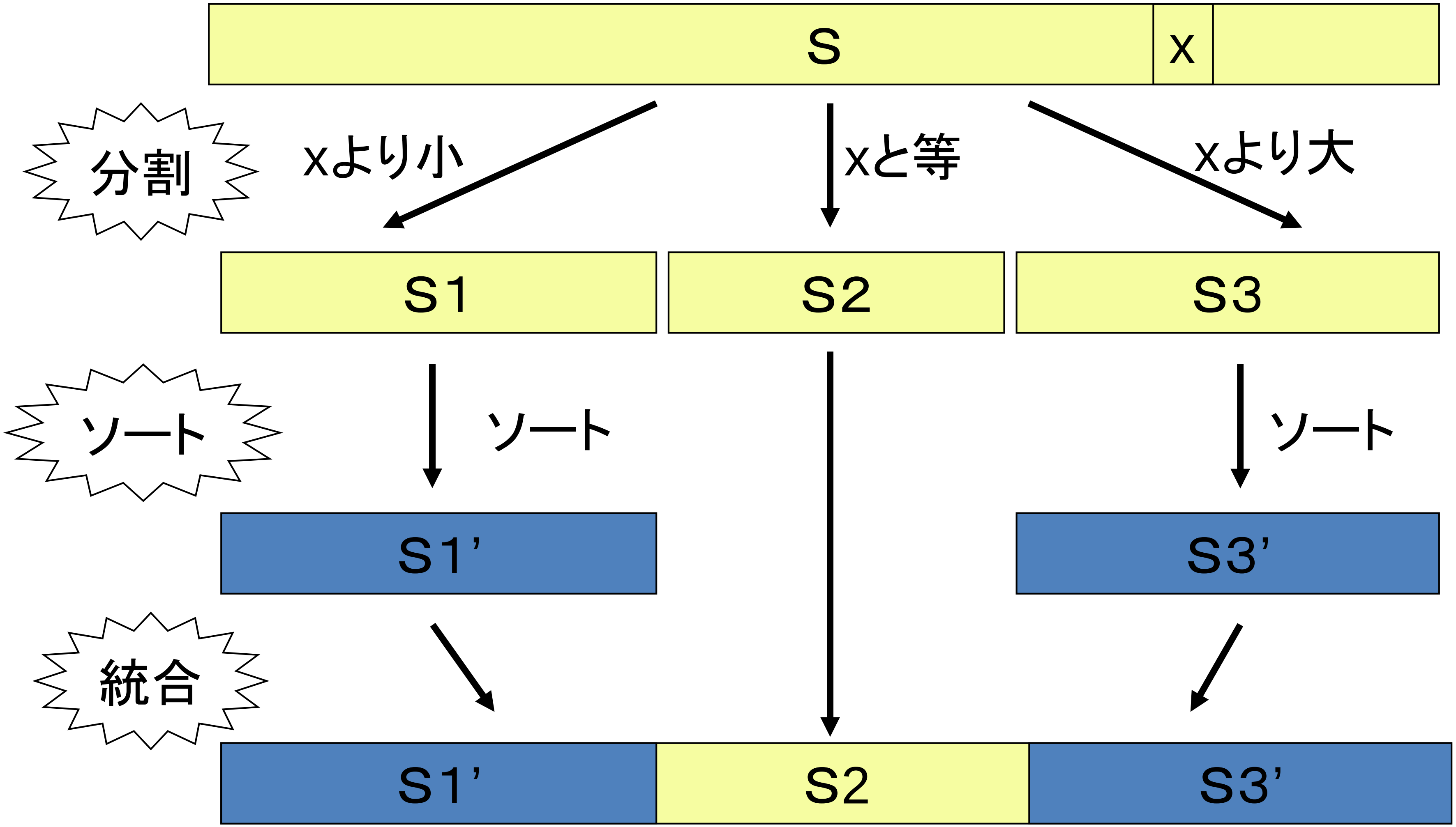
要書き込み

クイックソート(Quick Sort)

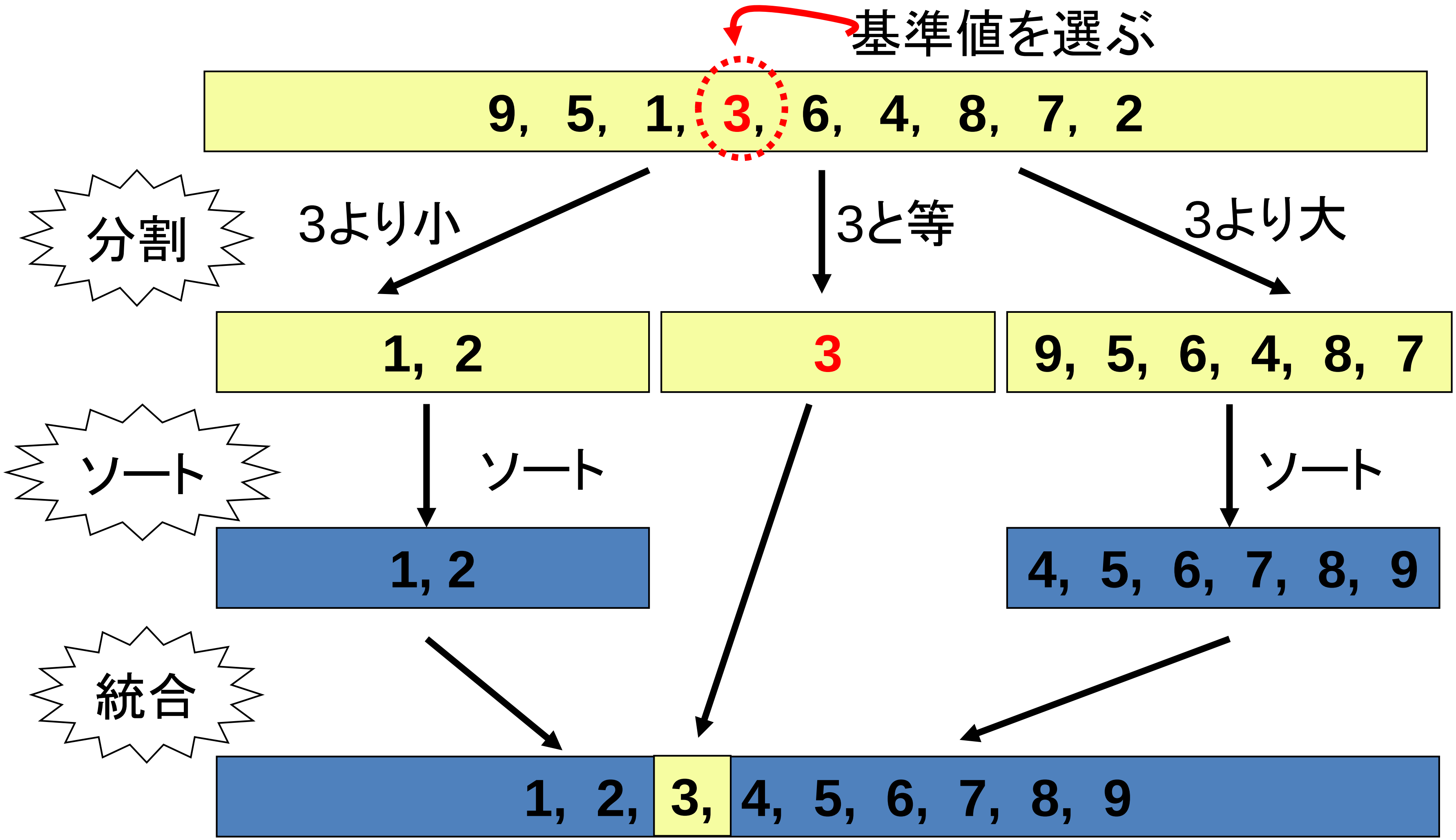
- とても有名．実際にも広く使用されている．
- 分割統治（divide and conquer）法の一つ
- 1960年，Hoare (ホーア)
  - オックスフォード大名誉教授
  - チューリング賞受賞者
  - Hoare理論
  - CSP（並行処理の理論）
  - その他多数の業績



クイックソートの手順(イメージ)



クイックソートの手順（具体例）



クイックソートの手順

(※ 並べたいデータ列をSとする)

1. |S| <= 1 → Sを返して終了
2. Sから要素を1つ選んで x とする.
   xを基準に, Sを3つのデータ列に***分割***
   S1 ・・・・ x より小さいデータが入る
   S2 ・・・・ x と等しいデータが入る
   S3 ・・・・ x より大きいデータが入る
3. S1とS3を, クイックソートで再帰的にソート
4. S1, S2, S3 の順に連結して返す.

要書き込み

再帰呼び出しの停止性

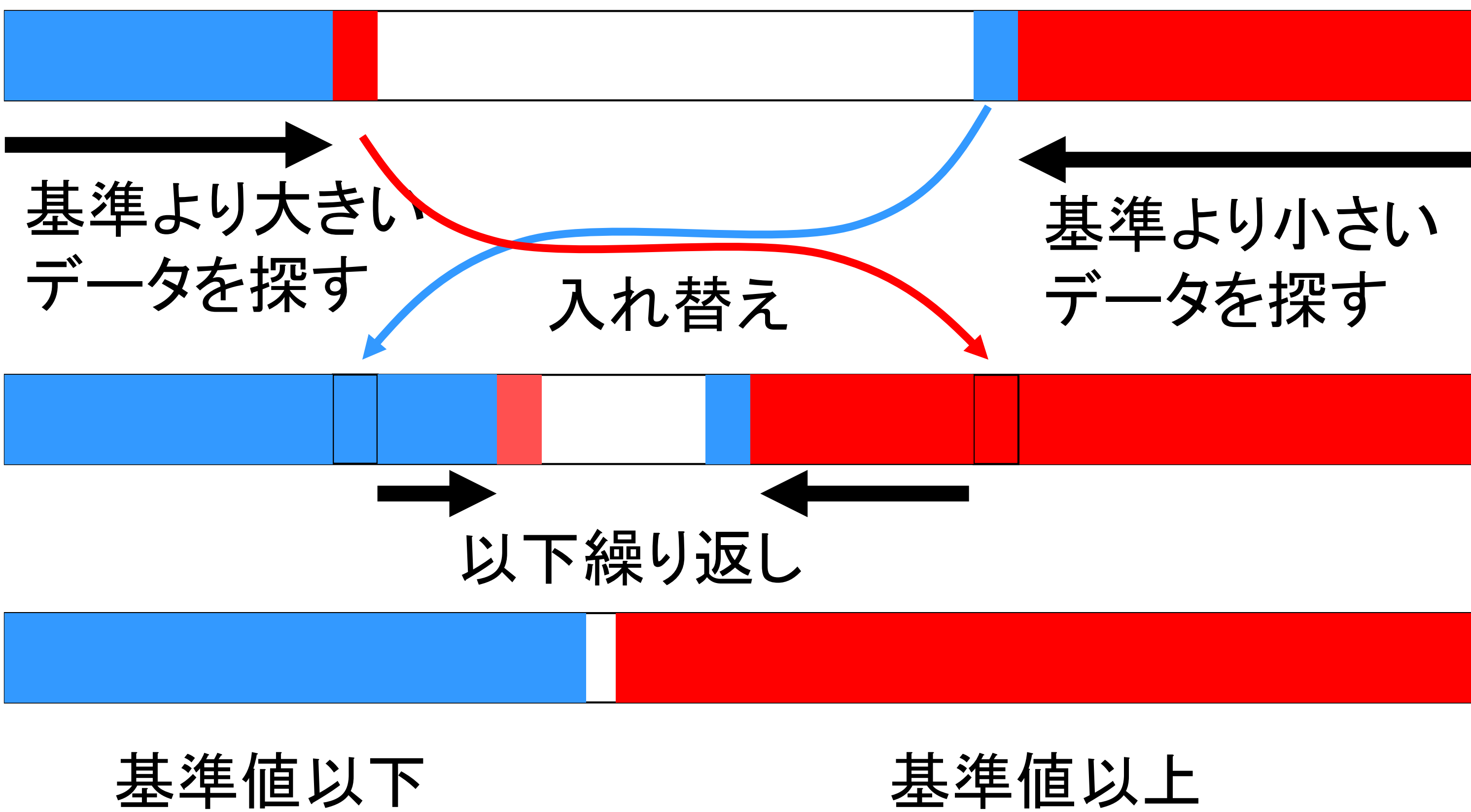
- 分割後のS1とS3を，クイックソートを使ってそれぞれソート（再帰呼び出し）する.
  - 再帰呼び出し：ある関数から，その関数自身を呼び出す

- |S2| >= 1より，S1やS3は，Sよりはサイズが小さくなる

  → 分割を続けていけば，最終的にはサイズが1以下となり，それ以上再帰呼出が続かない.
  → 停止

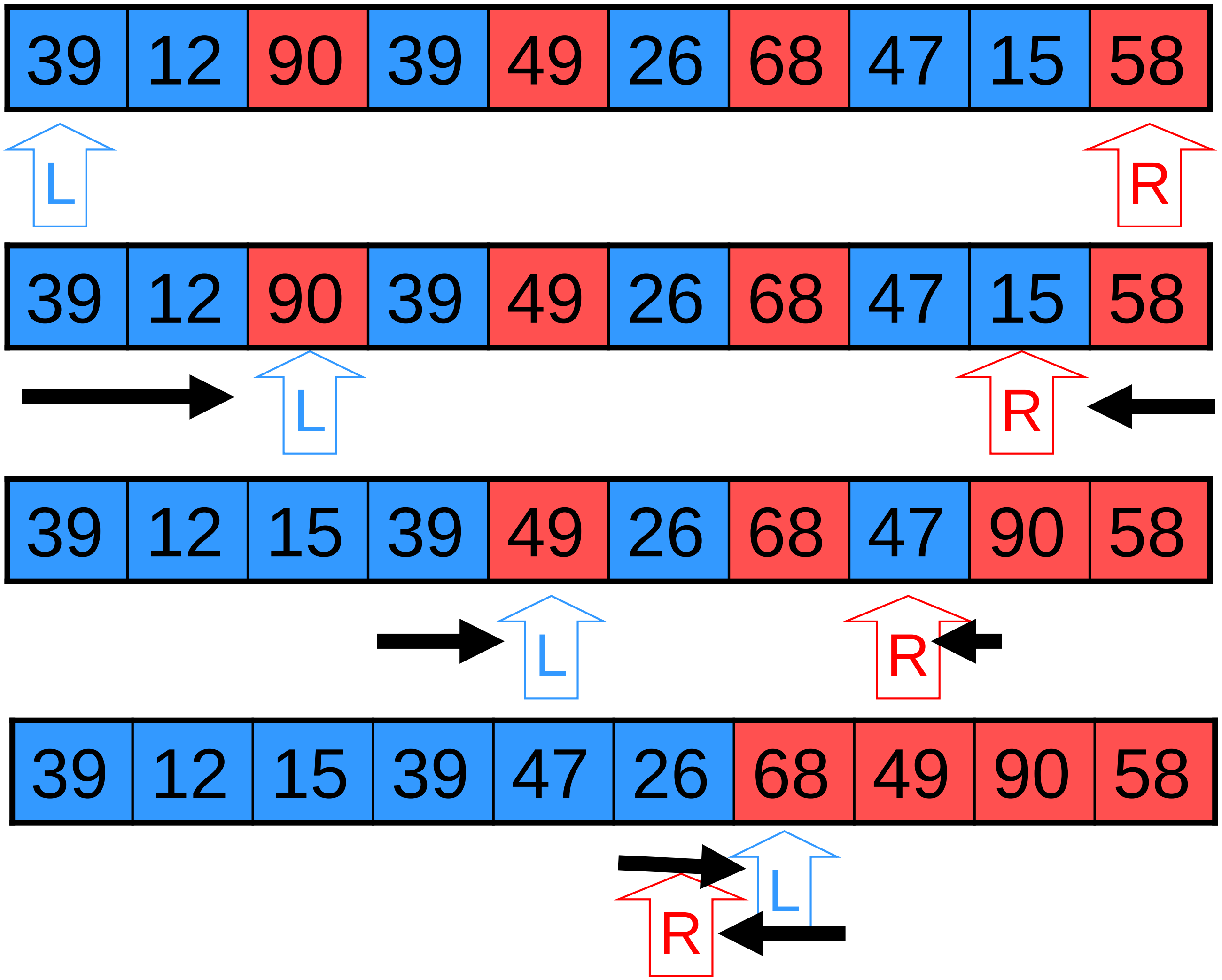
基準値による分割 → O(n)

- アイデア: (※2つに分割)
  - 両端から，真ん中へ向かって処理を進める



要書き
込み

分割の例



（基準値49
で分割）

分割のアルゴリズム

[問題]　配列に格納されたn個の整数と分割値xを与える．配列の先頭部分にはxより小さい整数だけを，その後にはxより大きい整数だけを格納せよ．ただし，xより小さい整数とxより大きい整数が少なくとも1つずつは存在すると仮定する．

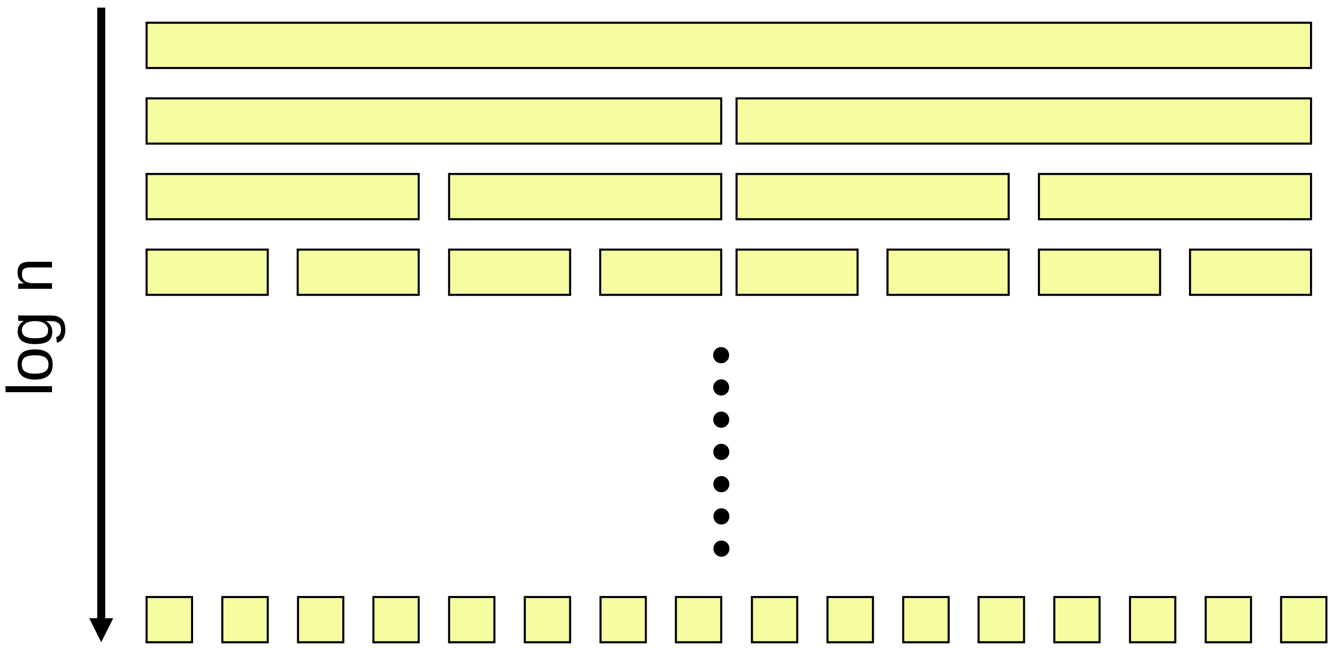
1. 場所1を配列の先頭，場所2を配列の末尾とする．
2. 場所1が場所2より前にある限り，以下の処理を繰り返す．
   2.1 場所1の要素がxよりも小さい限り，場所1を1つ進める．
   2.2 場所2の要素がxよりも大きい限り，場所2を1つ戻す．
   2.3 場所1の要素と場所2の要素を交換する．
   2.4 場所1を1つ進め，場所2を1つ戻す．
3. 場所1を境界として返す．

クイックソートのアルゴリズム

1. ソート範囲の要素数が2以下の場合には，必要に応じて要素を入れ替え，終了する.
2. そうでなければ以下の処理を実行する.
   2.1 分割値xを，ソート範囲の左端，中央（小数点以下は切り捨て），右端の位置にある要素の中央値とする.
   2.2 xに基づいて下限と上限の間の要素に分割アルゴリズムを適用し，境界を求める.
   2.3 下限と境界の間にある要素を再帰的にソートする.
   2.4 境界＋1と上限の間にある要素を再帰的にソートする.

クイックソートの計算量

- バランス良く分割していけるとき

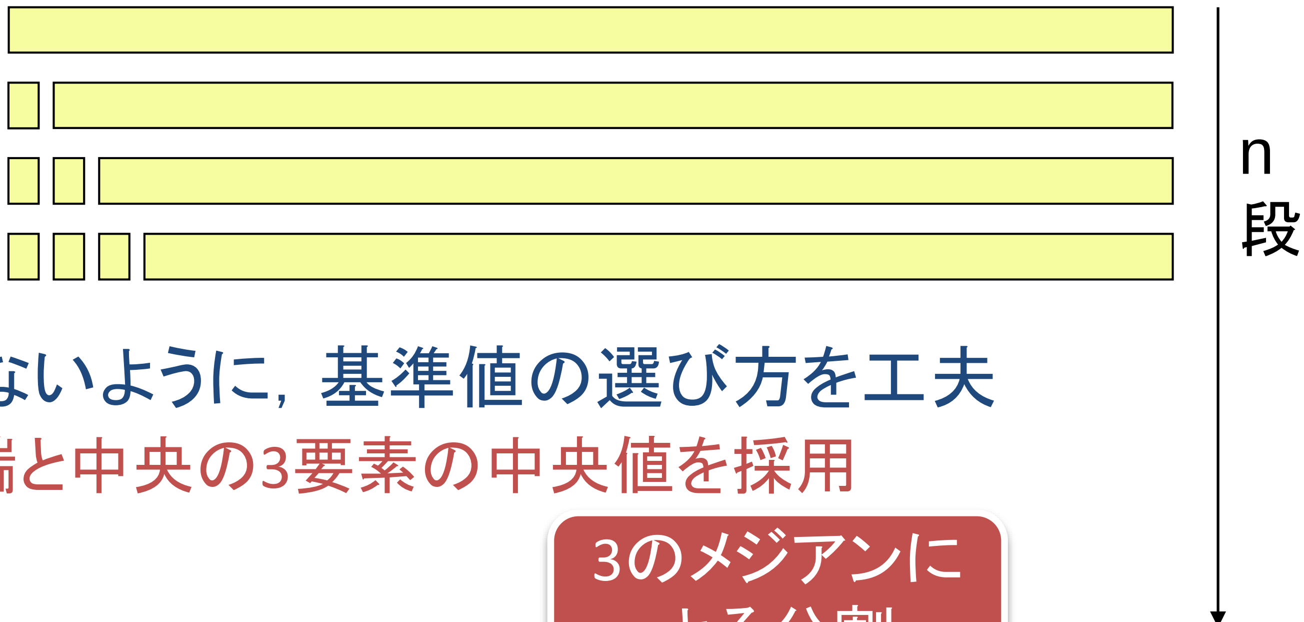


長さ	個数
n	1
n/2	2
n/4	4
n/8	8
⋮	⋮
1	n

分割に必要な手間の合計 O(n log n) ステップ

クイックソートの計算量

- 平均的にも，O(n log n) ステップ
- ただし，最悪時は $O(n^2)$ ステップ
  - 各分割で，基準値が偶然，最大値/最小値だった場合



  - そうならないように，基準値の選び方を工夫
    - 例：両端と中央の3要素の中央値を採用

3のメジアンによる分割

整理

- マージソート(Merge Sort)
  - マージ(併合)を利用
  - 常に O(n log n) の手間
  - マージのために、余分な記憶領域が必要
- クイックソート(Quick Sort)
  - 基準値で分割 → 分割した部分列を再帰的にソート
  - 平均 O(n log n), 最悪 $O(n^2)$

要書き込み

整列に必要な手間

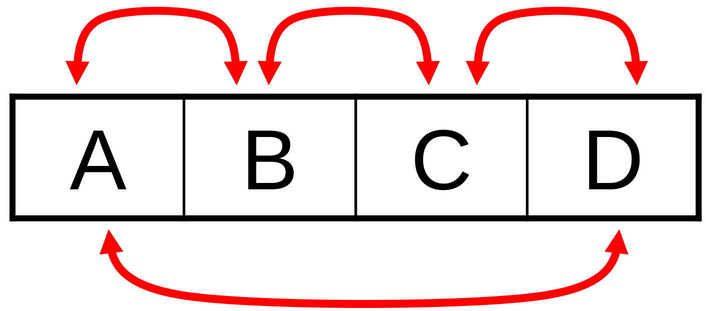
- n個のデータを並べ替えるとき、何回の比較が必要か？
- 正しく並べ替えるには「必要な情報」を得なければならない.
- 必要な情報を得るのに，最悪どれだけの手間がかかるのか？

比較を用いたソーティングアルゴリズムのオーダーの最小値は？

要書き込み

n回で充分？

- n=4で考察
- A,B,C,Dの大小関係を比較によって特定



- 右の例なら
  A＜B、 B>C、 C<D、 A<D
- 同じ結果を持つ別のデータ系列が存在
- n回の比較では一意に特定できないことがある
  → n回の比較だけでは不十分

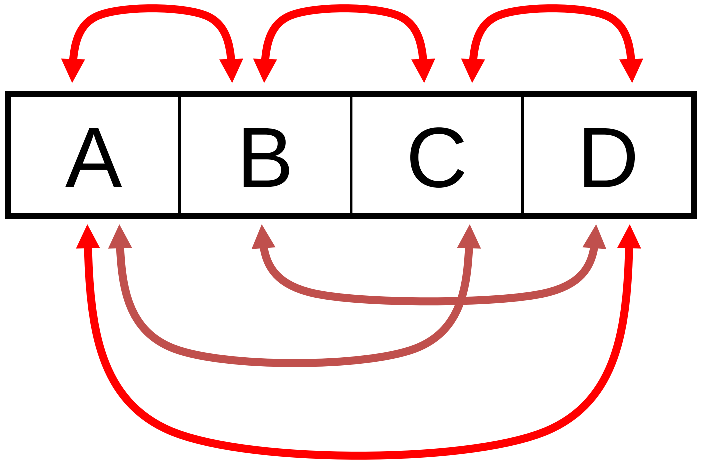
2	3	1	4

2	4	1	3

n^2回比較すれば充分

- nデータの、全てのペアを比較すれば完全
- nデータ中のペアの数

$$_nC_2 = n(n-1)/2$$

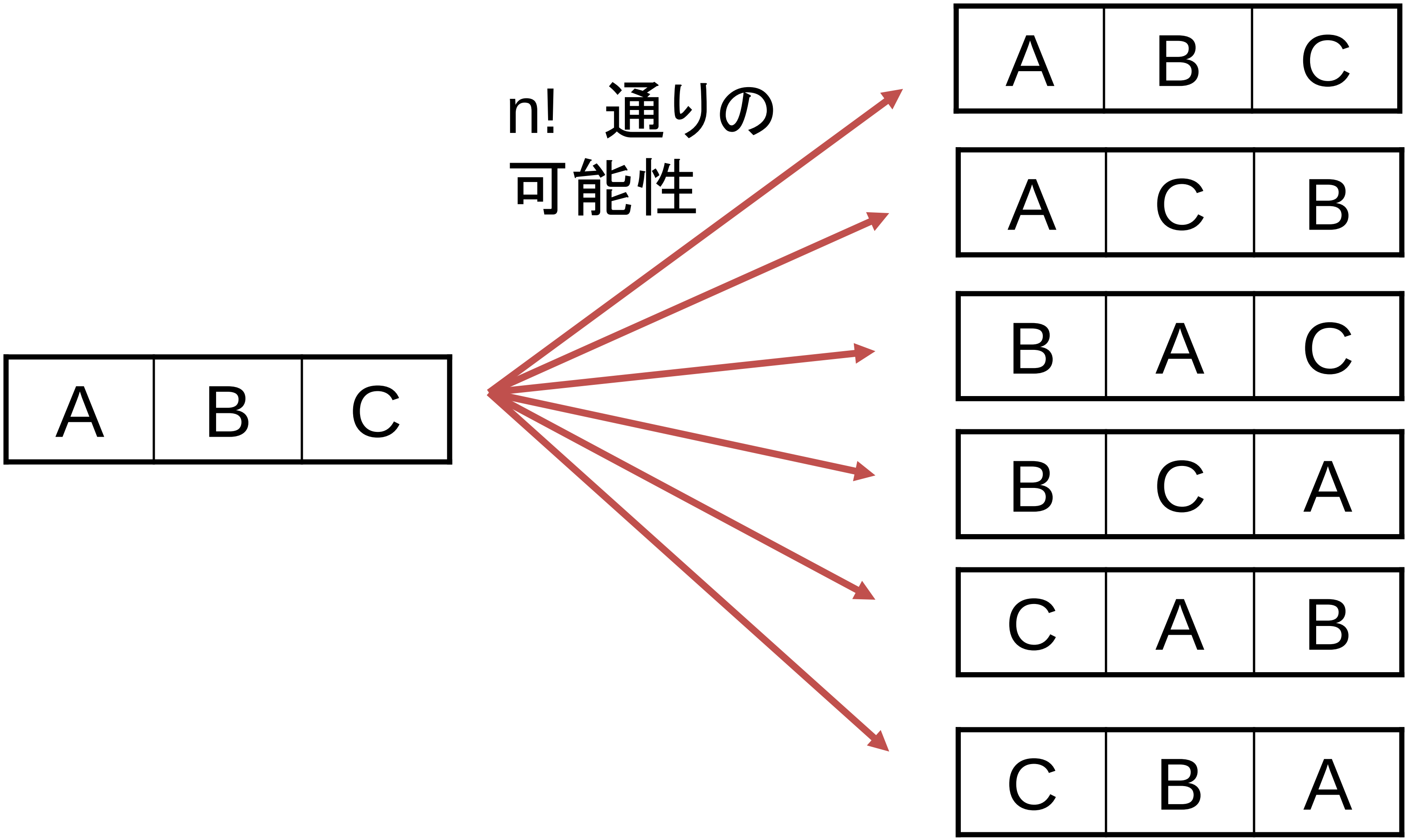


整列に必要な手間

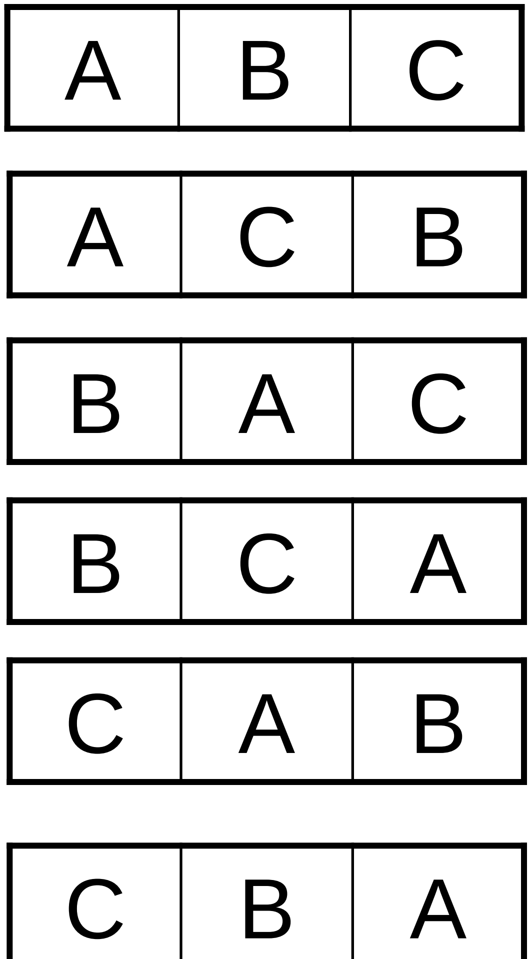
- 整列に必要な情報
  - n 回の比較では取得しきれない
  - n(n-1)/2 回の比較だと取得できる
- 比較に基づいた整列アルゴリズム
  - O(n)のアルゴリズムは理論的に不可能
  - O(n^2)のアルゴリズムは可能。実在。
- 疑問
  - O(n^2)のアルゴリズムは最適？
  - これ以上オーダーを改善できない？

比較回数の下界 → O(n log n)

n!　通りの
可能性



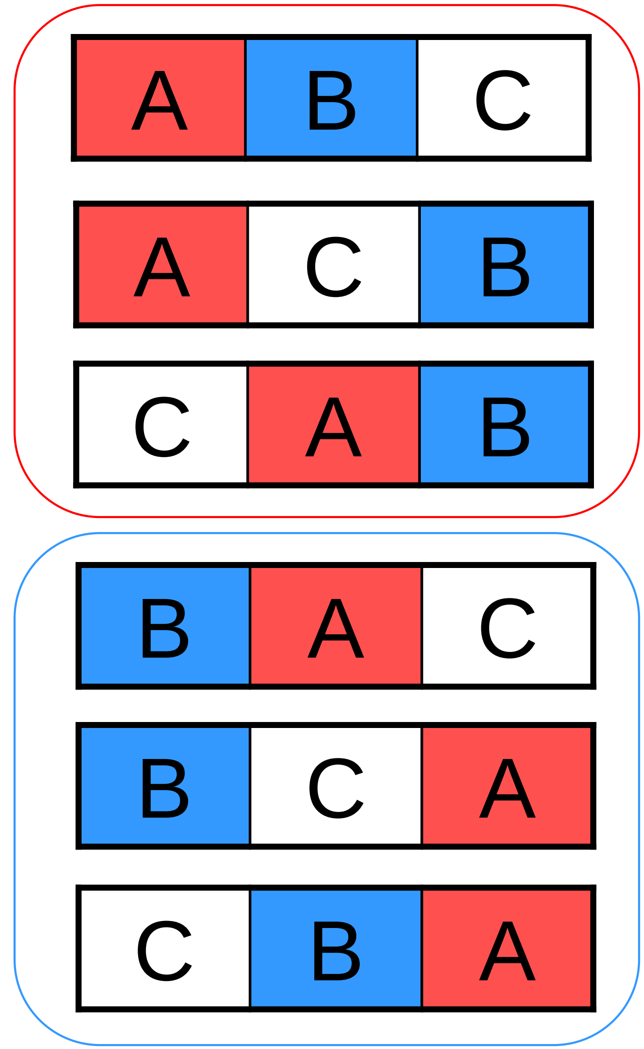
比較回数の下界 → O(n log n)



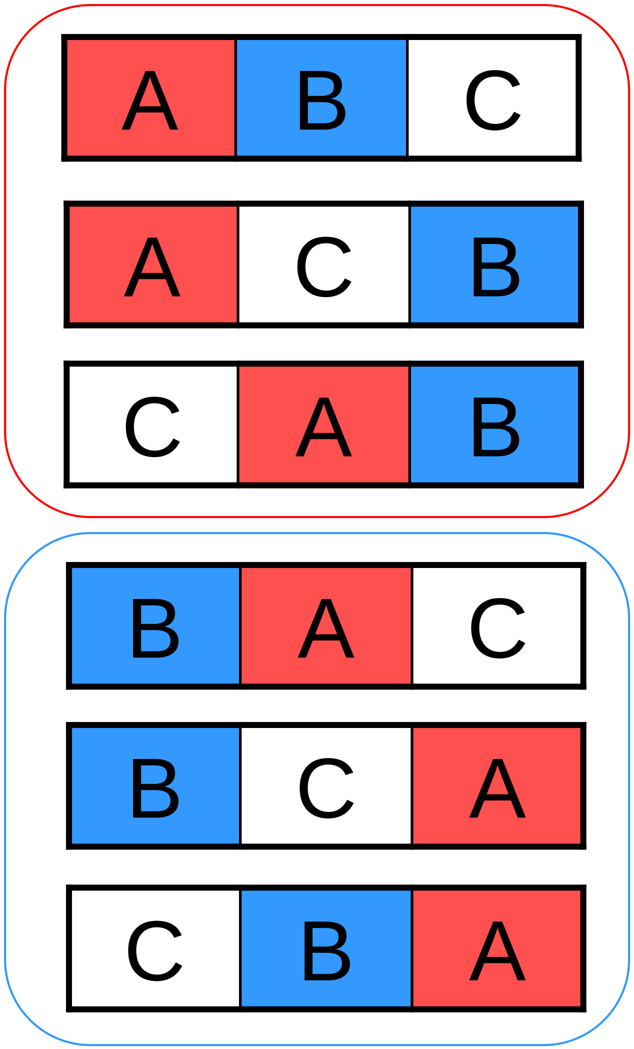
A<B

AとBの大小関係が分かると、大体半分に絞り込める

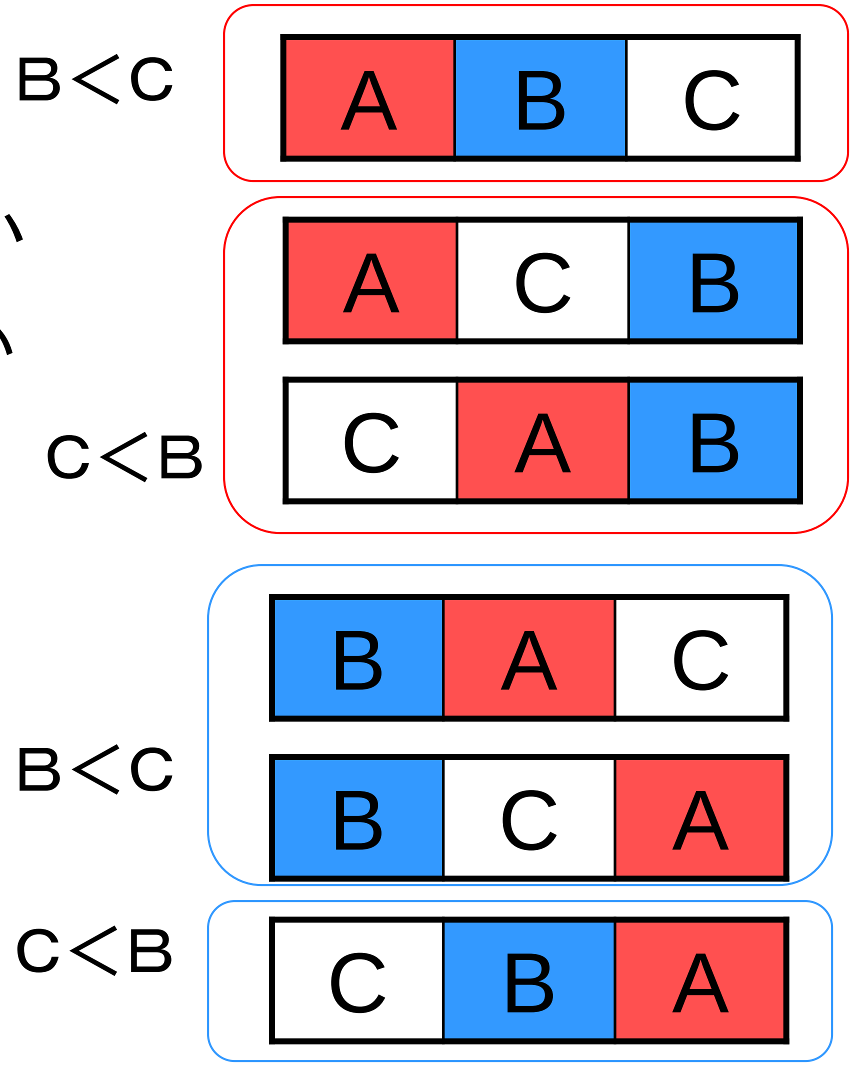
B<A



比較回数の下界 → O(n log n)



BとCの大小関係が分かると、更に絞り込める



比較回数の下界 → O(n log n)

- 最初は n! パターンの可能性がある
  - n個の相異なる数字を並べる並べ方
- 1回の比較で、可能性を大体半分に絞り込める
- 最終的に1パターンに絞り込むには$\log_2 (n!)$ 回比較しないとダメな場合がある
  - つまり、最悪時には$\log_2 (n!)$ 回の比較が必要
- $\log_2 (n!) \fallingdotseq n \log_2 n$ (スターリングの公式)
  - $\log_2(n!) = \log_2 1+\log_2 2 + \log_2 3 + \cdots + \log_2 n$

要書き込み

バケットソート

比較を行わなければ，より高効率にソートできる

- 適用条件
  - データ範囲が1～Nに限定されている
- 手順
  1. 1～Nの番号をつけた箱（バケット）を用意
  2. 各データを対応する箱に入れる
  3. 1番目の箱から順番にデータを取り出し，並べる
- 時間計算量：O(n+N) ⇒ 比較を行わないので効率が良い
- 空間計算量：O（N）⇒ 大容量のメモリが必要

eg.「32ビットデータ」→ $N=2^{32}=4[G]$

要書き込み

バケットソートのアルゴリズム

1. 最初のデータを読む.
2. データxがある限り，以下の処理を繰り返す.
   2.1 データxをx番目のバケット(一次元リスト)の先頭に挿入する.
   2.2 次のデータを読む.
3. バケットの各要素について，当該バケットの要素を表示する.

本日のまとめ

- マージソート(Merge Sort)
  - マージ(併合)を利用
  - 常に O(n log n) の手間
  - マージのために、余分な記憶領域が必要
- クイックソート(Quick Sort)
  - 基準値で分割 → 分割した部分列を再帰的にソート
  - 平均 O(n log n)，最悪 $O(n^2)$
- 比較回数の下界 → O(n log n)
- バケットソート(Bucket Sort)
  - データ範囲が1～Nに限定
  - 効率は良いが，大容量メモリが必要.

要書き込み

ソートアルゴリズムの特徴(その1)

ソートアルゴリズム	特徴
バブルソート	• アルゴリズムが最も単純で実現が容易 • 比較回数，コピー回数が$O(n^2)$必要.
選択ソート	• アルゴリズムが比較的単純 • コピー回数が$O(n)$で済むため，各要素のサイズが大きい場合に効率が良い.
挿入ソート	• データの交換(代入3回)が不要. 移動(代入1回)で済む. • 平均比較回数がバブルソート等の半分で済む. • データがほぼソートされている場合には，ほぼ$O(n)$時間で実行できる.
シェルソート	• 高々$O(n^{3/2})$時間でソートできる. • 多くの場合，挿入ソートより高速. ヒープソートより単純.
ヒープソート	• 最悪の場合でも$O(n \log n)$時間でソートできる. • ヒープを保存するために，$O(n)$の余分な作業領域が必要

ソートアルゴリズムの特徴（その2）

ソートアルゴリズム	特徴
マージソート	• 最悪の場合でもO(n log n)時間でソートできる. • マージ処理を実行するために，O(n)の余分な作業領域が必要
クイックソート	• 平均O(n log n)時間だが，最悪$O(n^2)$時間が必要. • 高速ソートアルゴリズムとして有名. • 実際にも良く使われている. • 余分な作業領域は，最悪の場合でもO(log n)で済む.
バケットソート	• データ範囲が1〜Nに限定されている. • バケットを保持するために，O(N)の余分な作業領域が必要. • データの相互比較を行わないため，O(n+N)時間で実行できる.

確認テスト（第9回）

- 以下のデータ列を、各種ソーティングアルゴリズムでソートした場合の変更過程を示せ。

7	3	6	1	2	5	4	8

  - バケットソート
  - ヒープソート
  - マージソート
  - クイックソート