

# アルゴリズムとデータ構造④

## ～ 分割統治法 ～

鹿島久嗣

# 分割統治法：

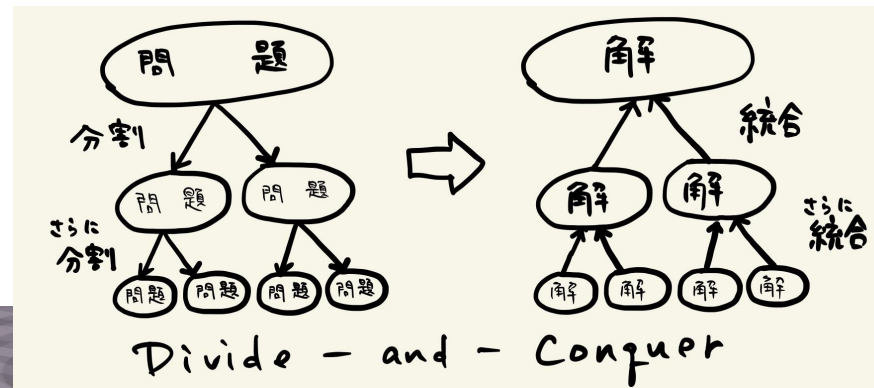
アルゴリズム設計指針の1つで、問題を小問題に分割して解く

- 特定の問題に対するアドホックな個別の解法ではなく、多くの問題に適用可能なアルゴリズムの一般的な設計指針

— 分割統治法、動的計画法、...

- 分割統治法：

- 元の問題を、同じ構造をもった小さな問題に分割
- 小さな問題の解を統合して元の問題の解を得る



## アルゴリズム設計指針の1つで、問題を小問題に分割して解く

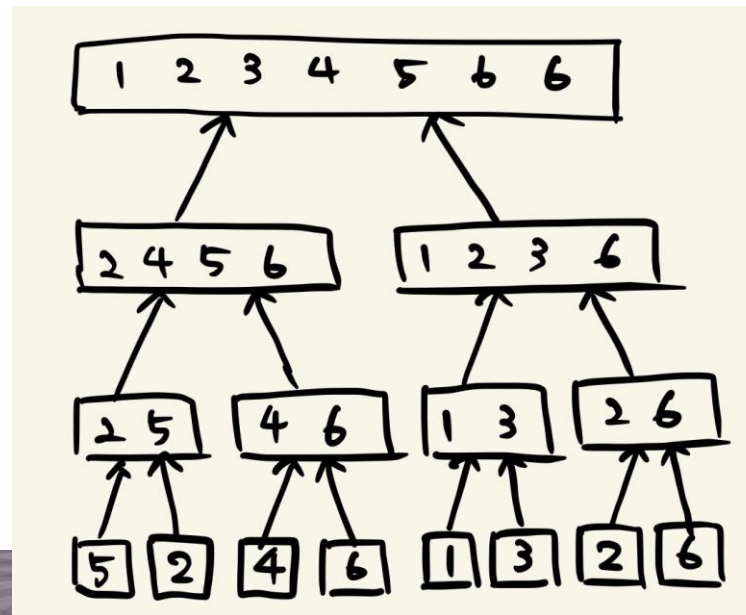
一分割：元の問題を、同じ構造をもった小さな問題に分割

—統合：小さな問題の解を統合して元の問題の解を得る



# 分割統治法の例： マージソート

- 入力された配列を前後に分割し、それぞれに対してマージソートを適用する
  - 再帰的に行うことで、サイズ1の配列まで到達する
  - 逆向きに統合して解を構成する
- 例：配列  $(5, 2, 4, 6, 1, 3, 2, 6) \rightarrow (5, 2, 4, 6)$  と  $(1, 3, 2, 6)$



マージソート：

マージソートの計算量は $O(n \log n)$

---

- $n = 2^k$ として $O(n \log n)$ 
  - 実用的には次に紹介するクイックソートが速い
- 計算量評価の再帰式：

$$T(n) = \begin{cases} O(1) & (n = 1) \\ 2T(n/2) + O(n) & (n \geq 2) \end{cases} = O(n \log n)$$

再帰                  統合

マージソート：

マージソートの計算量は $O(n \log n)$

---

■ 計算量評価の再帰式：

$$T(n) = \begin{cases} O(1) & (n = 1) \\ 2T(n/2) + O(n) & (n \geq 2) \end{cases}$$

■  $T(n) = 2T(n/2) + cn = 2 \left( T\left(\frac{n}{2^2}\right) + c \frac{n}{2} \right) + cn$

$$= 2 \left( 2 \left( \dots \left( 2 \left( \underbrace{T\left(\frac{n}{2^k}\right)}_c + c \frac{n}{2^{k-1}} \right) + c \frac{n}{2^{k-2}} \right) \dots \right) + c \frac{n}{2} \right) + cn$$

$$= c2^k + \underbrace{cn + \dots + cn}_k < n \log n.$$

# 分類定理（簡易版）：

## 計算量の再帰式から計算量を導く定理

- $T(n)$ の漸化式から $T(n)$ のオーダーを導く
- 定理： 大きさ $n$ の問題を大きさ $\frac{n}{b}$ の問題  $a$  個に分割した

$$\text{—つまり、 } T(n) = \begin{cases} c & (n = 1) \\ aT\left(\frac{n}{b}\right) + cn & (n \geq 2) \end{cases}$$

$$\text{—このとき： } T(n) = \begin{cases} O(n) & (a < b) \\ O(n \log n) & (a = b) \\ O(n^{\log_b a}) & (a > b) \end{cases}$$

# クイックソート： 分割統治法にもとづく高速なアルゴリズム

- 最もよく用いられる、分割統治に基づくソートアルゴリズム
- 平均計算量  $O(n \log n)$  だが、最悪では  $O(n^2)$  かかる
  - ただし、実用的には速い
  - その場でのソートが可能
- アルゴリズム  $\text{QuickSort}(A, p, r)$

$p$  : 配列中でソートする部分の先頭  
 $r$  : 配列中でソートする部分の末尾

1.  $q \leftarrow \text{Partition}(A, p, r)$  : 分割点  $q$  をみつけて分割
  2.  $\text{QuickSort}(A, p, q)$
  3.  $\text{QuickSort}(A, q + 1, r)$
- } 分割したそれぞれについて  
クイックソートを適用



クイックソートの分割関数  $\text{Partition}(A, p, r)$  :  
基準となる要素（枢軸）との大小比較で2グループに分割

- クイックソートではある数との大小で要素を2群に分割する
  - 比較対象の要素  $A[p]$  : 枢軸(pivot)とよぶ
- $A[p:r]$ を $A[p]$ 以下の要素と、 $A[p]$ 以上の要素に分割
  - $A[p]$ 以下の要素が新たに $A[p:q]$ となる
  - $A[p]$ 以上の要素が新たに $A[q + 1:r]$ となる
  - 2つのインデックス  $i, j$  を使って配列  $A[p:r]$  を操作 :
    1.  $j = r$  から左に走査して枢軸以上の要素を発見
    2.  $i = p$  から右に走査して枢軸以下の要素を発見
    3. 両者を入れ替える
    4. これを両者が出会うまで繰り返す ( $O(r - p)$ )

# クイックソートの計算量： 「平均で」 $O(n \log n)$ を実現できる

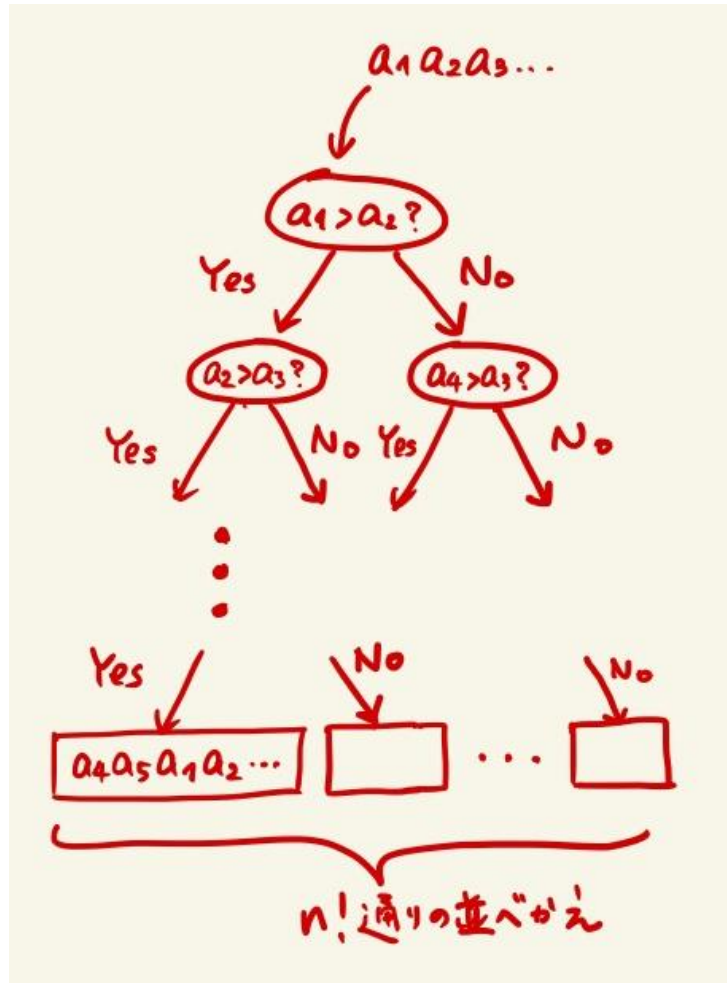
- 最悪の場合：  
 $n$ 個の要素が $n - 1$ 個と1個に分割されたとすると $O(n^2)$ 
  - 1回の分割でサイズが定数個しか減らない場合
- 最良の場合：  
 $n$ 個の要素が $\frac{n}{2}$ 個2セットに分割されたとすると $O(n \log n)$ 
  - 分割定理で $a = b$ の場合
  - 定数分の1のサイズに分割される場合
- 最悪の場合を避けるために：ランダムに枢軸を選択
  - 問題例には依存しない平均計算量を達成できる

ソートの計算量の下界：

$O(n \log n)$ より小さい計算量は達成できない

- ソートのアルゴリズムは最悪計算量 $O(n \log n)$ が必要
- $n$ 個の要素はすべて異なるとすると、ソート後に得られる列の可能性は $n!$ 通り
- ソートは2つの数の比較を繰り返すことで動く
- ソートの流れを2分木で書くことにする：
  - 各頂点で2つの数を比較して分岐
  - 葉は、ある特定の並べ替えに対応
  - 全ての並べ替えが可能であるために葉が $n!$ 個は必要
  - これを実現するためには少なくとも木の高さが $O(n \log n)$

# ソートの計算量の下界： $O(n \log n)$ より小さい計算量は達成できない



この高さがどう頑張っても  
 $O(n \log n)$ になることを示す

一番下では、とある並び替えが得られる

# ソートの計算量の下界：

$O(n \log n)$ より小さい計算量は達成できない

- 全ての可能な並び替えが得られるためには、最下段の要素が少なくとも  $n!$  は必要
- 図の高さがちょうど  $h$ （完全2分木）とすると、最下段の要素（葉）の数は  $2^h$ 
  - 逆に  $2^h$  個の葉をもつ木で最も低いのが完全2分木
- よって、 $2^h \geq n!$  でないといけない
- 対数をとると
$$h \geq \log n! \geq \log(n/e)^n = n \log n - n \log e = O(n \log n)$$
  - なお、Stirlingの公式  $n! \geq \sqrt{2\pi n}(n/e)^n \geq (n/e)^n$

木の高さが比較回数  
(= 計算量) に対応

順序統計量：

小さい方から $k$ 番目の要素は線形時間で発見可能

---

- 順序統計量：小さい方から $k$ 番目の要素
  - 自明なやり方：ソートを使えば $O(n \log n)$
  - 工夫すれば $O(n)$ で可能：
    - **平均的に $O(n)$ で見つける方法**
    - 最悪ケースで $O(n)$ で見つける方法
- の二つのやり方を紹介する

# 平均 $O(n)$ の順序統計量アルゴリズム： クイックソートと同じ考え方で可能

- $q \leftarrow \text{Partition}(A, p, r)$ を実行した結果：
  1.  $k \leq q$ であれば、求める要素は $A[p:q]$ にある
  2.  $k > q$ であれば、求める要素は $A[q+1:r]$ にある—再帰的にPartitionを呼ぶことで範囲を限定していく
- 平均的には問題サイズは半々になっていく：

$$T(n) = T\left(\frac{n}{2}\right) + O(n) = O(n)$$

注：クイックソートでは $2T\left(\frac{n}{2}\right)$ だった

分割のコスト