アルゴリズムの設計と解析

教授：　黄　潤和　（W4022）
rhuang@hosei.ac.jp
SA：　広野　史明（A4/A8）
fumiaki.hirono.5k@stu.hosei.ac.jp

Two main approaches

1. from typical problem types
2. from algorithm design techniques

The following two lectures are

- 高度なアルゴリズム設計・解析手法
  - L2 分割統治法 Divide and Conquer
  - L3 動的計画法 Dynamic Programming

クイックソートを通して，分割統治法の概念を理解する

L2. 分割統治法

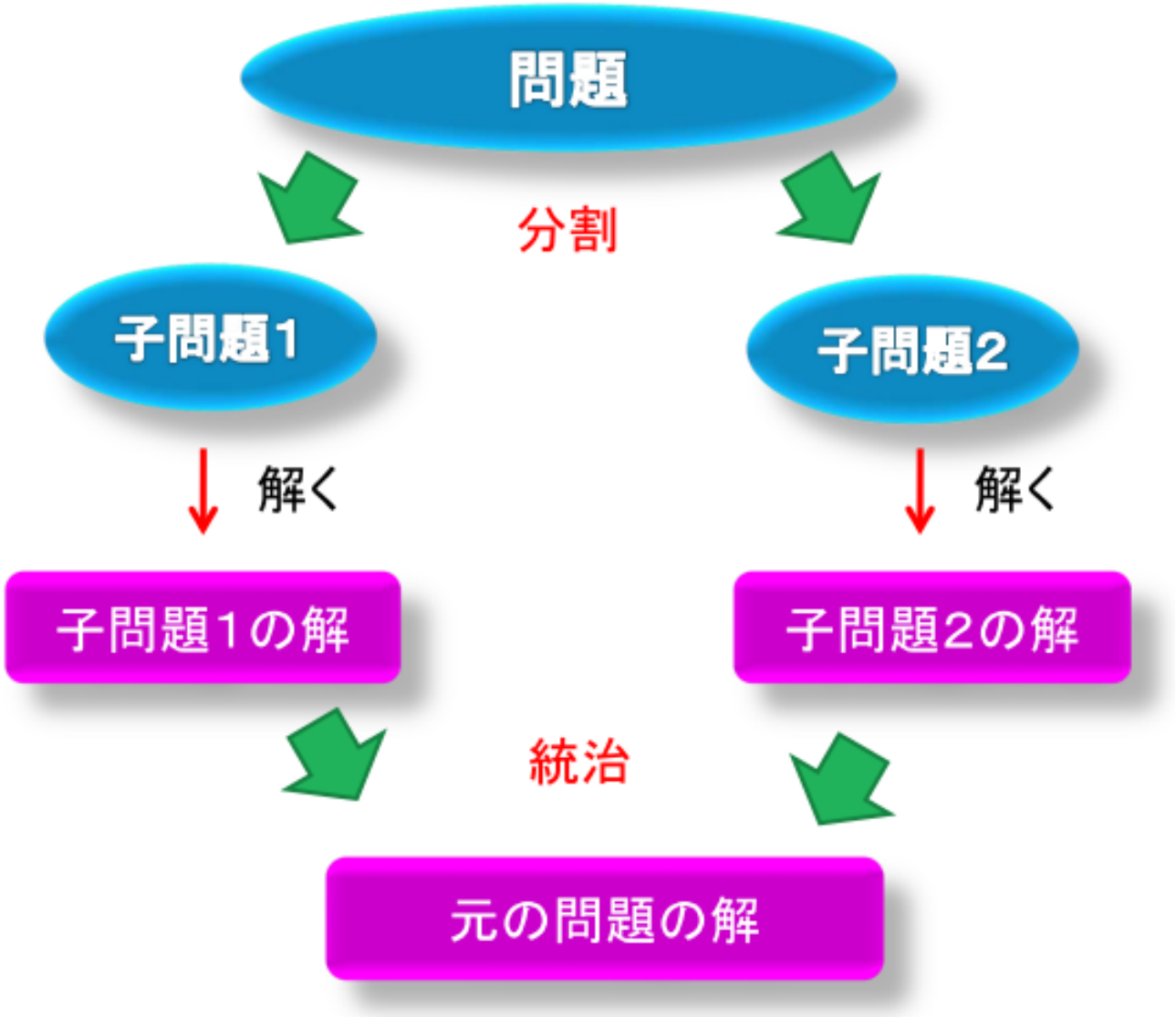
Divide and conquer

ソートのまとめ

- 本講義では，これまで4種類のソートアルゴリズムを学んだ．それぞれの特徴を表にまとめると次のようになる(それぞれの特徴をしっかり把握しておくこと)

	最悪実行時間	最良実行時間	平均実行時間	その場ソートか否か	その他の特徴
挿入ソート	$O(n^2)$	$O(n)$	$O(n^2)$	○	
マージソート	$O(n \lg n)$	$O(n \lg n)$	$O(n \lg n)$	×	分割統治法
ヒープソート	$O(n \lg n)$	$O(n \lg n)$	$O(n \lg n)$	○	ヒープはデータ構造としても有用
クイックソート	$O(n^2)$	$O(n \lg n)$	$O(n \lg n)$	○	• 分割統治法 • 実用上は最も高速

分割統治法とは



http://www.lab2.kuis.kyoto-u.ac.jp/~shuichi/algintro/alg-3s.pdf

ソーティング

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

ソーティング（並べ替え）
入力：n個の正の整数
出力：小さい順に並べ替えたもの

入力　10, 9, 2, 6, 4, 1, 8, 3
↓
出力　1, 2, 3, 4, 6, 8, 9, 10

アルゴリズム1
n個の数を並べる全ての組み合わせに対して、
それが小さい順に並んでいたら出力。

動作例：
入力　5, 1, 4, 3, 2, 6

5, 1, 4, 3, 2, 6 … ×
5, 1, 4, 3, 6, 2 … ×
5, 1, 4, 2, 3, 6 … ×
5, 1, 4, 2, 6, 3 … ×
5, 1, 4, 6, 2, 3 … ×
⋮
1, 2, 3, 4, 5, 6 … ○

入力がn個のとき、
チェックすべき組み合わせは
$n!$通りある。

It is *n* factorial

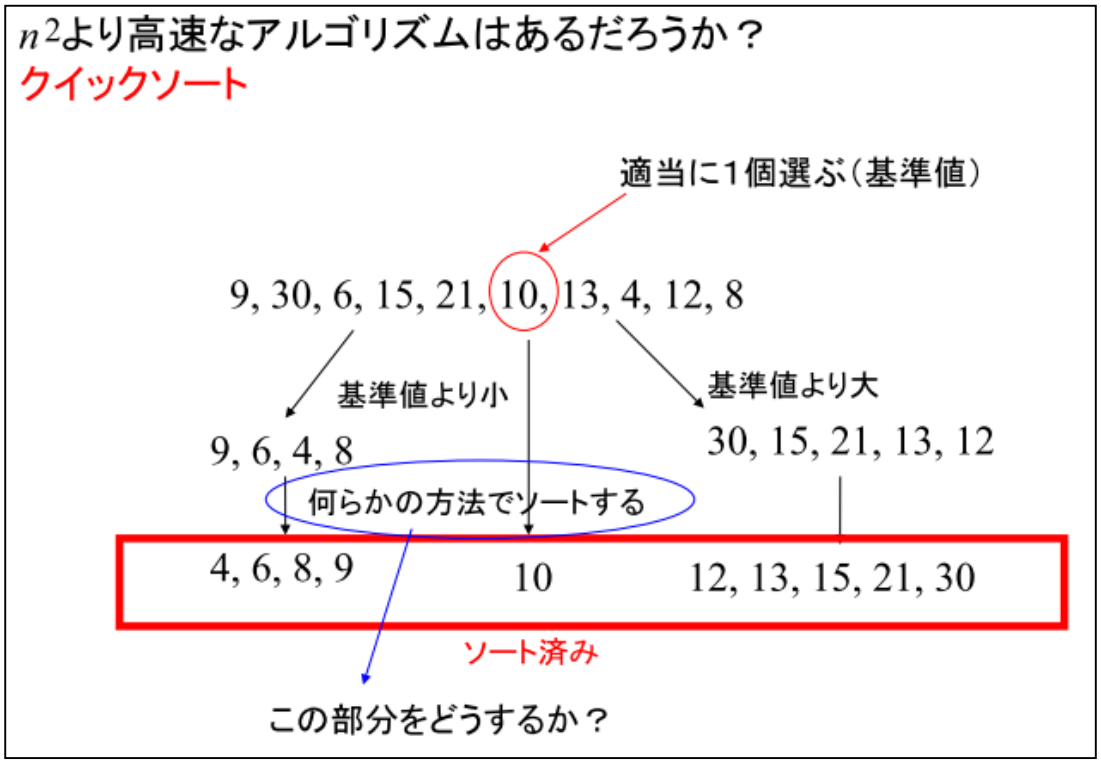
アルゴリズム2
n個の中から一番小さい数を選び、先頭へ。
n-1個の中から一番小さい数を選び、先頭へ。
n-2個の中から一番小さい数を選び、先頭へ。
⋮
2個の中から一番小さい数を選び、先頭へ。

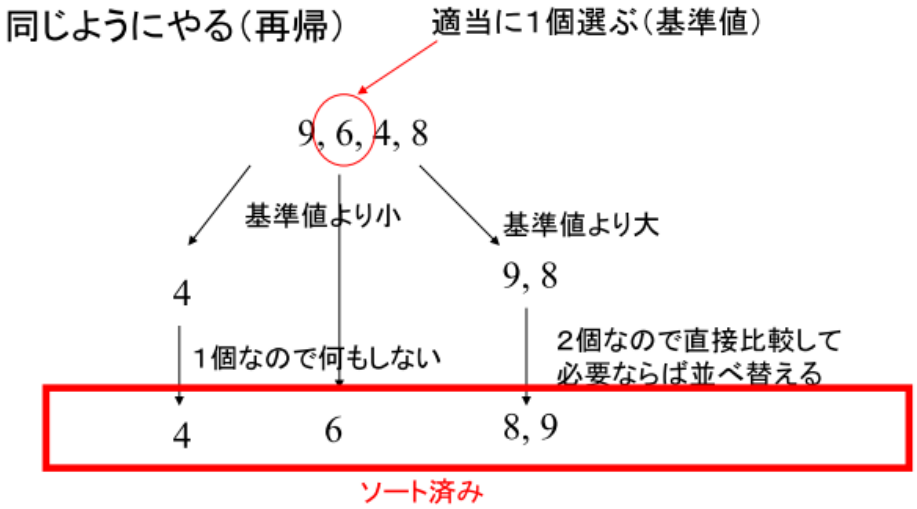
動作例：
入力　5, 1, 4, 3, 2, 6
→
1, 5, 4, 3, 2, 6
→
1, 2, 5, 4, 3, 6
→
1, 2, 3, 5, 4, 6
→

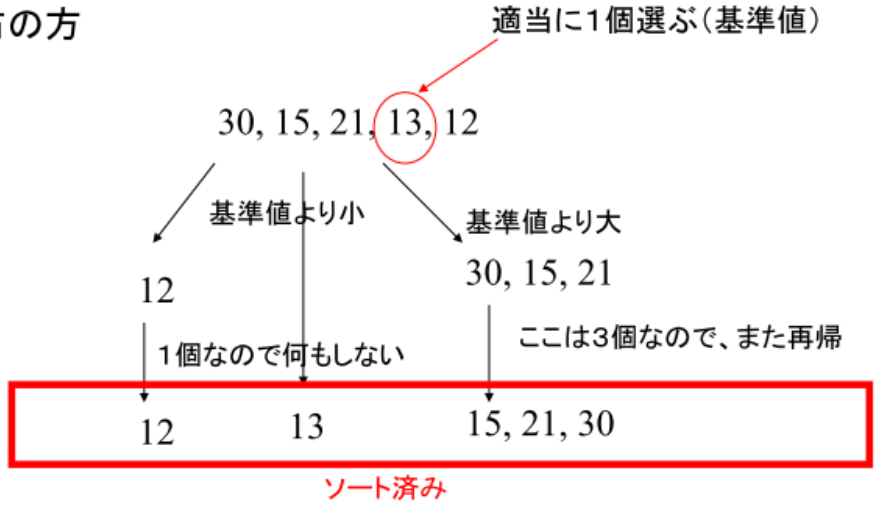
入力がn個のとき、
1回の走査でn個以下チェック
走査はn回やれば十分
↓
全体でn^2回以下

It is *n* square

http://www.lab2.kuis.kyoto-u.ac.jp/~shuichi/algintro/alg-3s.pdf

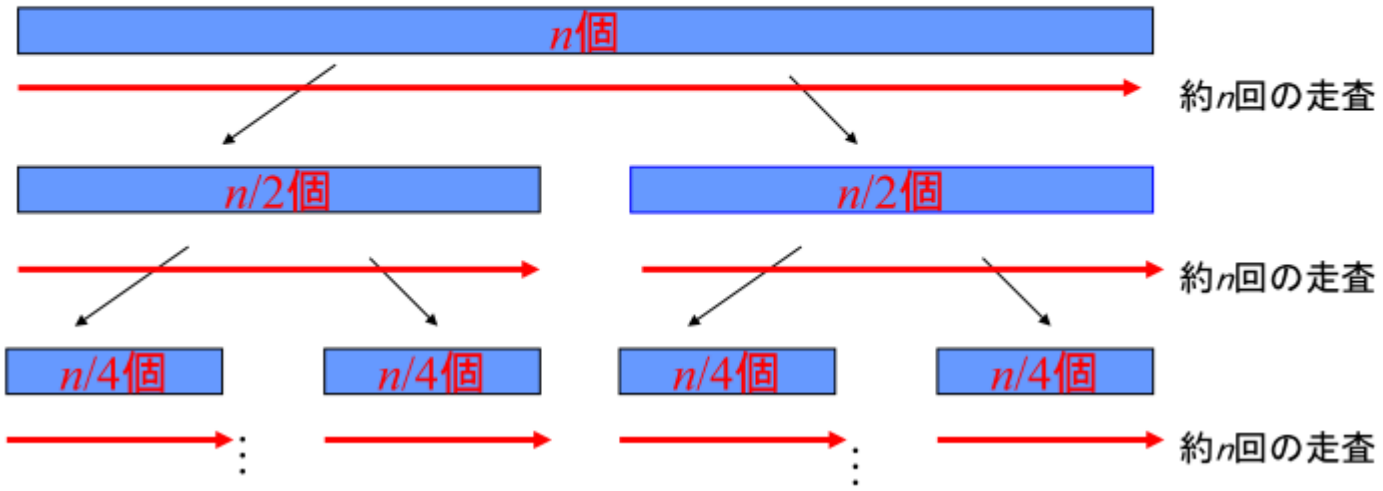




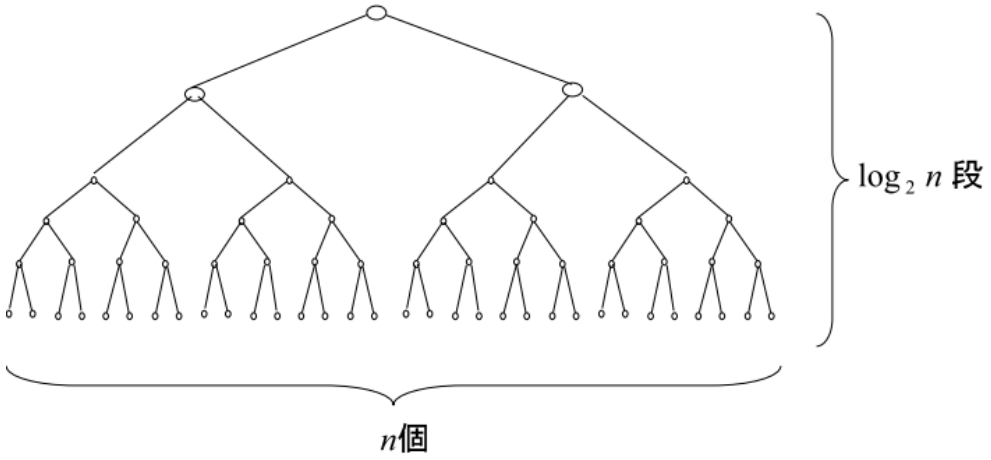


http://www.lab2.kuis.kyoto-u.ac.jp/~shuichi/algintro/alg-3s.pdf

計算時間
（毎回半分ずつにデータが分かれたと仮定して）



1個や2個になったら、これ以上分解しない　何段あるか？



http://www.lab2.kuis.kyoto-u.ac.jp/~shuichi/algintro/alg-3s.pdf

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

計算時間
- 1段で、約n回のスキャン
- 段数は$\log n$ 段

全体で$O(n \log n)$ 時間　　アルゴリズム2の n^2 から改善

	n^2	4	256	~10^6	~10^{12}
n		2	16	1024	1048576
$\log_2 n$		1	4	10	20

ところが。。。ある（都合の良い）仮定を置いていた。
→「毎回データが半分になる。」

問題：そうならなかった場合、最悪の場合は、計算時間はどうなるだろうか？

クイックソートにおける分割統治法

クイックソートは，マージソートと同様，**分割統治法**に基づいたアルゴリズムである．配列$A[p..r]$のソートを行うための分割統治の３段階（分割，統治，結合）を以下に示す．

分割： 配列$A[p..r]$を以下の条件を満たす二つの（空の可能性もある）部分配列$A[p..q-1]$と$A[q+1..r]$に分割する（添字qの値は，この分割手続きの中で計算）

- $A[p..q-1]$のどの要素も$A[q]$
- $A[q+1..r]$のどの要素も$A[q]$より大きい

統治： 二つの部分配列$A[p..q-1]$と$A[q+1..r]$をクイックソートでソートする

結合： なにもしない

Design idea and principle

クイックソートの擬似コード

QUICKSORT(A, p, r)

1. if $p < r$
2. $q =$ PARTITION(A, p, r)
3. QUICKSORT$(A, p, q-1)$
4. QUICKSORT$(A, q+1, r)$

クイックソートの動きのポイントとなるのは，上記擬似コードの2行目のPARTITION ()

PARTITIONの擬似コード

PARTITION(A, p, r)

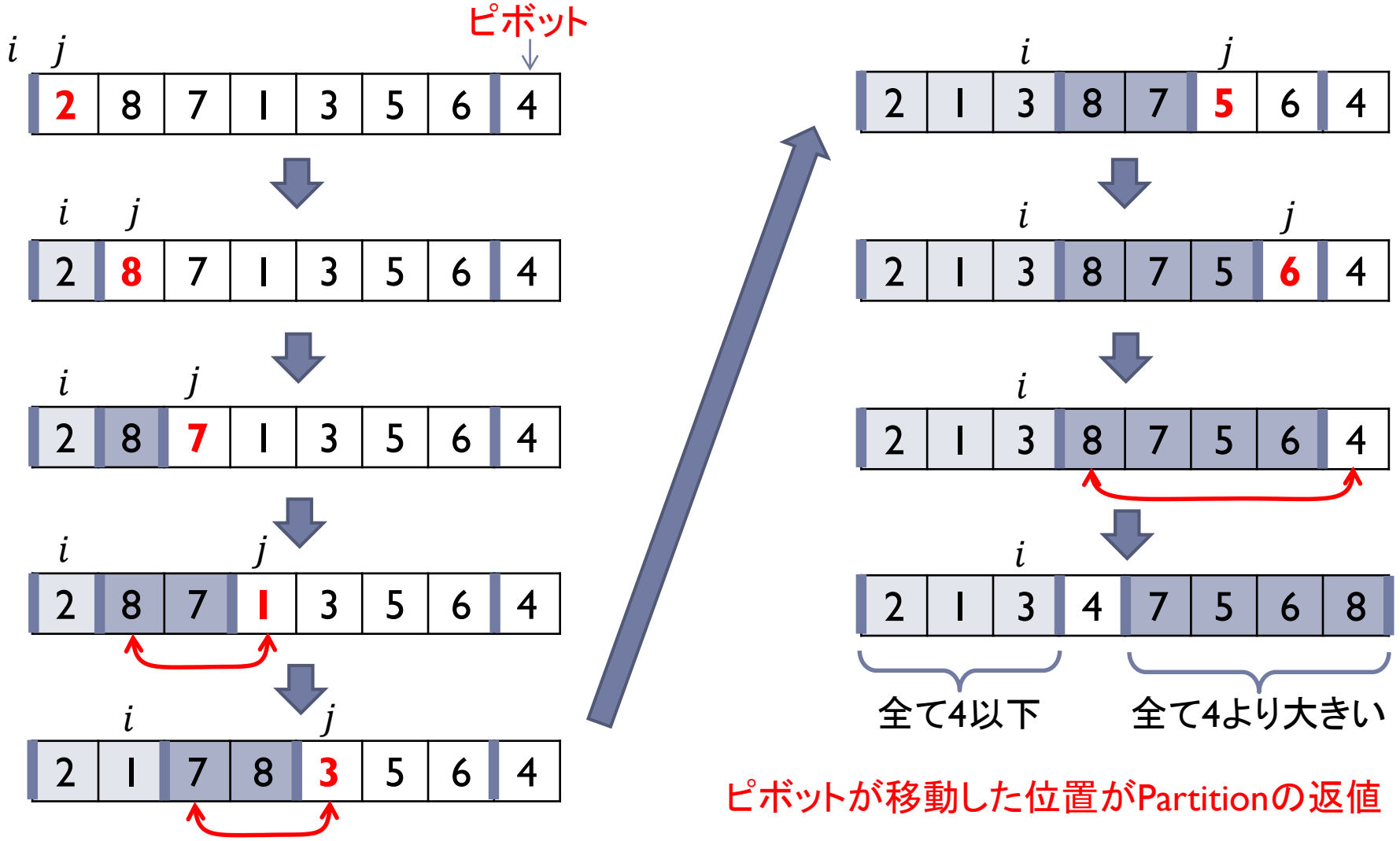
1. $x = A[r]$
2. $i = p - 1$
3. **for** $j = p$ **to** $r - 1$
4. **if** $A[j] \leq x$
5. $i = i + 1$
6. $A[i]$と$A[j]$を交換
7. $A[i + 1]$と$A[r]$を交換
8. **return** $i + 1$

$A[p..r]$の最後の要素である$A[r]$を基準(ピボット)とし，値が$A[r]$以下となる要素を配列の前半に，$A[r]$より大きい要素を配列の後半に再配置する

iは，$A[r]$以下となる値と，$A[r]$より大きい値の境界位置を示す変数（$A[p] \sim A[i]$はすべて$A[r]$以下の値となる）

$A[p..r]$を$A[p, q - 1]$と$A[q + 1, r]$の二つに分ける際のqの値は，ピボットの値以下となる要素が配列中にいくつあるかによって定まる

PARTITION(A, p, r)の動きの例



ピボットが移動した位置がPartitionの返値

クイックソートの性能①：最悪実行時間

- クイックソートの実行時間は，分割手続きPARTITIONが配列を均等に分割できるかどうかに依存する
- クイックソートが最悪の実行時間となるのは，分割手続きがn個の配列のソートを$n-1$個の配列のソートと0個の配列のソートに分ける場合（例：ソート済みの配列）
- PARTITIONの実行時間が$\Theta(n)$となることを考えると，この場合の実行時間$T(n)$の漸近式は以下のようになる:

$$T(n) = T(n-1) + T(0) + \Theta(n) = T(n-1) + \Theta(n)$$

アルゴリズム2
n個の中から一番小さい数を選び、先頭へ。
n-1個の中から一番小さい数を選び、先頭へ。
n-2個の中から一番小さい数を選び、先頭へ。
⋮
2個の中から一番小さい数を選び、先頭へ。

動作例：
入力 5, 1, 4, 3, 2, 6
1, 5, 4, 3, 2, 6
1, 2, 5, 4, 3, 6
1, 2, 3, 5, 4, 6

入力がn個のとき、
1回の走査でn個以下チェック
走査はn回やれば十分
全体でn^2回以下

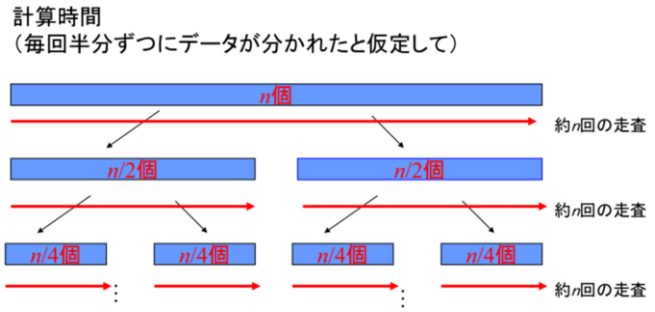
It is n square

- 上記の漸化式を解くと，$T(n) = \Theta(n^2)$が得られる。

クイックソートの性能②：最良実行時間

- 一方，クイックソートが最良の実行時間となるのは，分割手続きがn個の配列のソートを$\lfloor n/2 \rfloor$個の配列のソートと$\lceil n/2 \rceil - 1$個の配列のソートに分ける場合
- 最悪の場合と同様，PARTITIONの実行時間が$\Theta(n)$となることを考えると，この場合の実行時間$T(n)$の漸化式は以下のようになる：

$$T(n) \leq 2T(n/2) + \Theta(n)$$



- 分類定理より上記の漸化式の解は，$T(n) = \Theta(n \lg n)$となる

乱択版クイックソート

- クイックソートの実行時間は，ピボットとして選択された値がその配列の中で何番目に小さい値かで決まる

- 分割手続きPARTITIONで，ピボットの選び方を乱数によって決定することで，都合の悪い入力が与えられた時に，常に最悪実行時間となるのを防ぐことができる
  - 乱択版クイックソートによって，ソート済み配列の期待実行時間は，通常のクイックソートの$O(n^2)$から$O(n \lg n)$に改善
  - 配列の値が全て同じ場合は，乱択版クイックソートを利用しても実行時間は$O(n^2)$のまま（全ての入力に関して実行時間のオーダが改善する訳ではない）

乱択版クイックソートの擬似コード①

- 乱択版PARTITIONの擬似コードを以下に示す．ピボットの選択を乱数で行う以外は，通常のPARTITIONと同じ

RANDOMIZED-PARTITION(A, p, r)

1. $i = \text{RANDOM}(p, r)$ ▷ ピボットに使う値の添字を乱数で選択
2. $A[r]$と$A[i]$を交換
3. **return** PARTITION(A, p, r)

乱択版クイックソートの擬似コード②

乱択版PARTITIONを利用する以外は，乱択版クイックソートと通常のクイックソートの擬似コードは同じ：

RANDOMIZED-QUICKSORT(A, p, r)

1. if $p < r$
2. $\quad q =$ RANDOMIZED-PARTITION(A, p, r)
3. $\quad$ RANDOMIZED-QUICKSORT$(A, p, q-1)$
4. $\quad$ RANDOMIZED-QUICKSORT$(A, q+1, r)$

乱択版クイックソートの実行時間

- 乱択版クイックソートの実行時間は，以下のようになる：
  - 最良実行時間と平均実行時間：$O(n \lg n)$
  - 最悪実行時間：$O(n^2)$ ただし，最悪実行時間となるような入力は，通常のクイックソートと比較して大幅に少ない

ソートのまとめ

- 本講義では，これまで4種類のソートアルゴリズムを学んだ．それぞれの特徴を表にまとめると次のようになる(それぞれの特徴をしっかり把握しておくこと)

	最悪実行時間	最良実行時間	平均実行時間	その場ソートか否か	その他の特徴
挿入ソート	$O(n^2)$	$O(n)$	$O(n^2)$	○	
マージソート	$O(n \lg n)$	$O(n \lg n)$	$O(n \lg n)$	×	分割統治法
ヒープソート	$O(n \lg n)$	$O(n \lg n)$	$O(n \lg n)$	○	ヒープはデータ構造としても有用
クイックソート	$O(n^2)$	$O(n \lg n)$	$O(n \lg n)$	○	• 分割統治法 • 実用上は最も高速

https://www.codereading.com/algo_and_ds/algo/quick_sort.html

クイックソート [Quick Sort]

クイックソートはリストにおいて*ピボットと呼ぶ要素を軸に分割を繰り返して整列を行う*アルゴリズムです。

「分割を繰り返して整列を行う」ような手法を**分割統治法 divide-and-conquer** と呼びます。

アルゴリズム分析

1. 要素数が1つかそれ以下なら整列済みとみなしてソート処理を行わない
2. ピボットとなる要素をピックアップする
3. ピボットを中心とした2つの部分に分割する - ピボットの値より大きい値を持つ要素と小さい値を持つ要素
4. 分割された部分(サブリスト)に再帰的にこのアルゴリズムを適用する

分割統治法は手順 4. にあるように再帰処理で実現されます。

分割統治法 divide-and-conquer

分割統治法とは*大きな問題を小さな問題に分割することによって全体を解決*しようとする方法です。

クイックソートではピボットと呼ぶ軸となる要素の値より大きい要素群、小さい要素群という具合にソートの対象となる部分を小さくしてゆきながら全体のソートを完了させます。

図解

ソート前の配列

1. 左端の要素をピボットにして、配列を分割する。

2. 分割されたそれぞれの配列でピボットを設定して、配列を分割する。

3. 分割されたそれぞれの配列の要素が１つなのでソート済みとみなす。

ソート完了

サンプルコード

Python

リストを昇順に整列させる実装例です。

ピボットをリストの先頭要素として、値の小さな要素は `left` リストに、値の大きな要素は `right` リストに格納して最後にピボットを中心としてリストを結合しています。

```
def quicksort(seq):
    if len(seq) < 1:
        return seq
    pivot = seq[0]
    left = []
    right = []
    for x in range(1, len(seq)):
        if seq[x] <= pivot:
            left.append(seq[x])
        else:
            right.append(seq[x])
    left = quicksort(left)
    right = quicksort(right)
    foo = [pivot]
    return left + foo + right
```

Work in Class

Try to run quicksort

https://www.codereading.com/algo_and_ds/algo/source/quick_sort.py

https://www.codereading.com/algo_and_ds/algo/merge_sort.html

マージソート [Merge Sort]

マージソートは*整列されていないリストを2つのリストに分割して、それぞれを整列させた後、それらをマージして整列済みのひとつのリストを作ります。*

マージを利用してリストを整列するのでマージソートという名がついています。

アルゴリズム分析

1. 整列されていないリストを2つのサブリストに分割する
2. サブリストを整列する
3. サブリストをマージしてひとつの整列済みリストにする

分割された部分的なリストを**サブリスト**と呼びます。サブリストもマージソートを使って整列させます。2. の手順にはこのアルゴリズムを再帰的に適用することになります。

マージ(併合)

マージとは*いくつかの整列済みのリストをひとつのリストにまとめる操作*です。

マージの手順

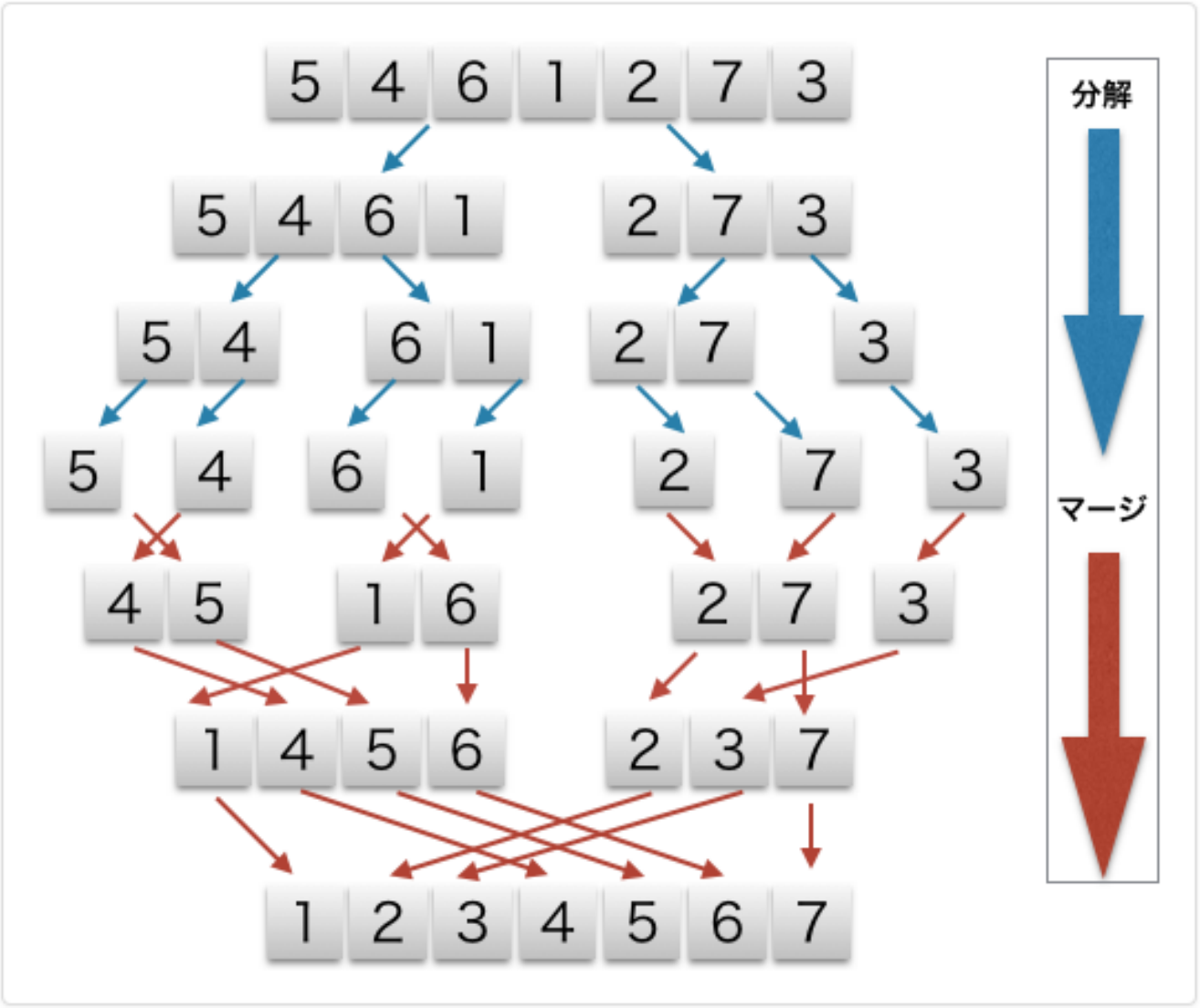
整列済みのリストAとリストBをマージして整列されたリストCを作ると考えます。

1. リストAとリストBの先頭の要素を比較して、小さい方をリストから削除してリストCの末尾に追加する
2. リストA、リストB、どちらか一方の要素がなくなるまで 1. を繰り返す
3. 残った要素をリストCの末尾に順に追加する

以上の手順でマージは完了します。

図解

リストをいったんバラバラにしたあと、マージによってリストを再構築します。



```
# Code for the merge subroutine


def merge(a,b):
    """ Function to merge two arrays """
    c = []
    while len(a) != 0 and len(b) != 0:
        if a[0] < b[0]:
            c.append(a[0])
            a.remove(a[0])
        else:
            c.append(b[0])
            b.remove(b[0])
    if len(a) == 0:
        c += b
    else:
        c += a
    return c
```

```
def mergesort(x):
    """ Function to sort an array using merge sort algorithm """
    if len(x) == 0 or len(x) == 1:
        return x
    else:
        middle = len(x)/2
        a = mergesort(x[:middle])
        b = mergesort(x[middle:])
        return merge(a,b)
```

Exercises

Ex 1

Understand the divide and conquer approach to solve the sorting problem and other problems

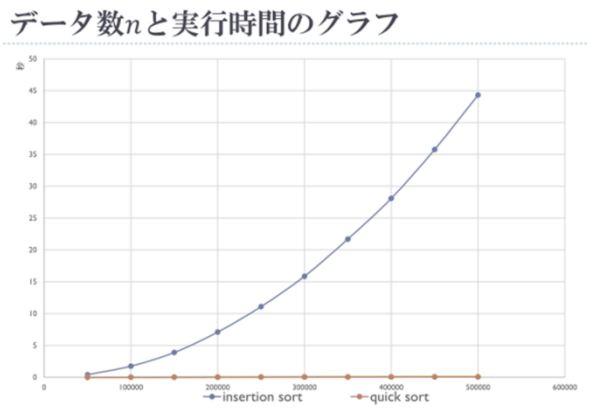
	最悪実行時間	最良実行時間	平均実行時間	その場ソートか否か	その他の特徴
挿入ソート	$O(n^2)$	$O(n)$	$O(n^2)$	○	
マージソート	$O(n \lg n)$	$O(n \lg n)$	$O(n \lg n)$	×	分割統治法
ヒープソート	$O(n \lg n)$	$O(n \lg n)$	$O(n \lg n)$	○	ヒープはデータ構造としても有用
クイックソート	$O(n^2)$	$O(n \lg n)$	$O(n \lg n)$	○	• 分割統治法 • 実用上は最も高速

Ex 2

Refer to the quick-sort in python
https://www.codereading.com/algo_and_ds/algo/quick_sort.html
and merge-sort in python
https://pythonandr.com/2015/07/05/the-merge-sort-python-code/

Please

- measure the execution time of both algorithms,
- run for different data sizes, and
- draw their comparisons in a plotting graph.



https://www.codereading.com/algo_and_ds/algo/source/quick_sort.py

クイックソートのデモンストレーション プログラム: quick_sort.py

https://helloacm.com/quick-demonstration-of-quick-sort-in-python/

(1) Demostration
(2) Add time() for performance

References:

1. http://www.lab2.kuis.kyoto-u.ac.jp/~shuichi/algintro/alg-3s.pdf
2. 2nd year "Data Structure and Algorithm" lecture notes (Lecture 1 and 7)

Reference programs in python

① https://www.codereading.com/algo_and_ds/algo/quick_sort.html
② https://pythonandr.com/2015/07/05/the-merge-sort-python-code/
③ https://www.codereading.com/algo_and_ds/algo/merge_sort.html
④ https://helloacm.com/quick-demonstration-of-quick-sort-in-python/