計算幾何学
Computational Geometry

第一章 基本概念
Basic concepts

教員と教材

- 講義 陳 文西(ちん ぶんし)
  - wenxi@u-aizu.ac.jp
- TA 朝妻 健人(あさつま けんと)、m5201125
  明田川 主(あけたがわ つかさ)、m5201149
- 主な参考書
  1. Computational Geometry—Algorithms and Applications
     M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf
     Springer, 2nd ed. 2000
  2. コンピュータ・ジオメトリー計算幾何学:アルゴリズムと応用
     M. ドバーグ、M. ファン・クリベルド、M. オーバマーズ、O. シュワルツコップ 著、浅野哲夫 訳、近代科学社(上記和訳本)
  3. 計算幾何学入門—幾何アルゴリズムとその応用
     譚学厚、平田富夫、森北出版㈱、2001

授業と評価

- 講義 10/2～11/27、月+木の4限、15回、M6
- 資料 http://i-health.u-aizu.ac.jp/CompuGeo/index.html
- 評価
  - 方法=(① or ②) and ③
  - ① アルゴリズム実装:4題(最多1題/章)、50%
    - 使用言語=C or Java、ウェブベース実装形式=better
    - 提出期限=各章の期限内(約講義終了後1週間)
  - ② プロジェクト研究:1回、50%
    - 研究レポート作成、ウェブベース実装形式=better
    - 提出期限=授業最終日(11/27)
  - ③ 期末試験:基本概念、50%

宿題の提出期限と採点方法

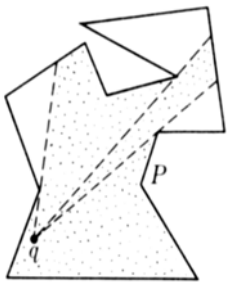
No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
date	10/2	10/5	10/12	10/16	10/19	10/23	10/26	10/30	11/2	11/6	11/9	11/13	11/16	11/20	11/27
lecture chapter	1	1	2	2	3	3/4	4	4	5	5	6	6	7	7	Q&A
deadline for ex			Ex1 10			Ex2 10		Ex3 10			Ex4 20		Ex5 20		report 50

1. 各回演習は100%満点として採点する
2. 結果が正しいものを100%とする
3. 結果が正しくない場合、再提出連絡後一週間内一回のみ再提出が可能
   3-1. 再提出の結果が正しい場合、80%とする
   3-2. 再提出の結果が正しくない場合、20%とする
   3-3. 再提出しない場合、10%とする
4. 期限後一週間以内提出の場合、正解に限り点数を折半する、再提出不可

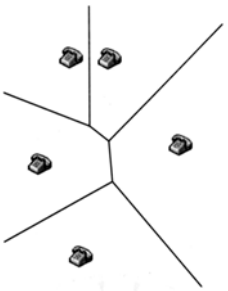
計算幾何学とは

- 幾何学的な問題を取り扱うための効率的なアルゴリズムとデータ構造の設計と解析
- Computational geometry is the branch of computer science that studies algorithms and data structures for solving geometric problems on a computer efficiently

簡単な実例

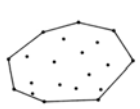


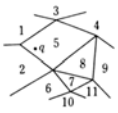
美術館警備



最も近い電話機

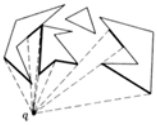
典型的な計算幾何学問題


凸包問題



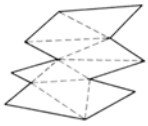
点位置決定問題


最近点問題



可視性問題

最短経路問題


三角形分割問題

応用分野

- 地理情報システム（GIS）
- コンピュータグラフィックス
- ロボティックス
- CAD/CAM
- Molecular Modeling
- Pattern Recognition
- Database

復習

- 高校
  - 基礎幾何学（elementary geometry）
- 大2前期
  - アルゴリズム（algorithm）
  - データ構造（data structure）

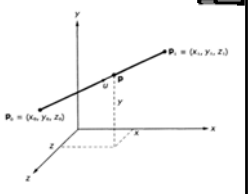
基礎幾何学(elementary geometry)

- 2点間距離
- ベクトル
- 直線方程式
- 三角形面積
- 四角形面積
- 多角形面積

Euclid: Picture courtesy of Lexington High School

2点間距離

- *3*次元の2点$p_0(x_0, y_0, z_0)$と$p_1(x_1, y_1, z_1)$
- *n*次元の2点$\boldsymbol{p}(p_1, p_2, \ldots, p_n)$と$\boldsymbol{q}(q_1, q_2, \ldots, q_n)$



http://reference.wolfram.com/language/guide/DistanceAndSimilarityMeasures.html

Other Distance Metrics

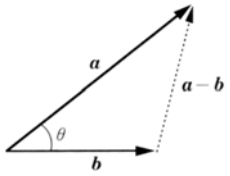
vectors **u** and **v**

No	Metric	Definition
1	euclidean	$\sqrt{\sum(\mathbf{u}-\mathbf{v})^2}$
2	seuclidean	$\sqrt{(\mathbf{u}-\mathbf{v})\mathbf{D}^{-1}(\mathbf{u}-\mathbf{v})'}$
3	mahalanobis	$\sqrt{(\mathbf{u}-\mathbf{v})\mathbf{V}^{-1}(\mathbf{u}-\mathbf{v})'}$
4	cityblock (manhattan)	$\sum\lvert\mathbf{u}-\mathbf{v}\rvert$
5	minkowski	$\left(\sum\lvert\mathbf{u}-\mathbf{v}\rvert^p\right)^{\frac{1}{p}}$
6	cosine	$1-\frac{\mathbf{u}\bullet\mathbf{v}}{\lVert\mathbf{u}\rVert\cdot\lVert\mathbf{v}\rVert}$
7	correlation	$1-\frac{(\mathbf{u}-\bar{\mathbf{u}})\bullet(\mathbf{v}-\bar{\mathbf{v}})}{\lVert\mathbf{u}-\bar{\mathbf{u}}\rVert\cdot\lVert\mathbf{v}-\bar{\mathbf{v}}\rVert}$
8	chebychev (chessboard)	$\max\lvert\mathbf{u}-\mathbf{v}\rvert$
9	canberra	$\sum\lvert\mathbf{u}-\mathbf{v}\rvert/(\lvert\mathbf{u}\rvert+\lvert\mathbf{v}\rvert)$
10	braycurtis	$\sum\lvert\mathbf{u}-\mathbf{v}\rvert/\sum\lvert\mathbf{u}+\mathbf{v}\rvert$

y = pdist(X, metric) computes the distance between columns in the data matrix X.
X: rows correspond to observations, columns correspond to variables

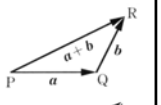
ベクトル

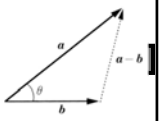
- ベクトル**a**($a_1, a_2, \ldots, a_n$)と**b**($b_1, b_2, \ldots, b_n$)
- 長さ
- 角度

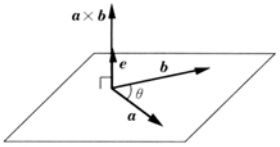


ベクトルの演算

- 和
- 差
- 内積
- 外積





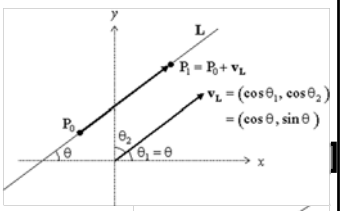


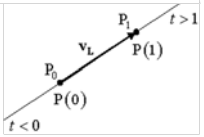
直線方程式

Explicit equation

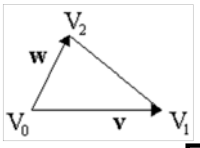
Implicit equation

Parametric equation

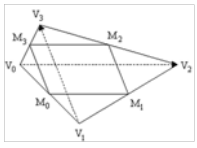




三角形面積

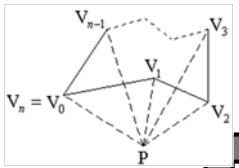


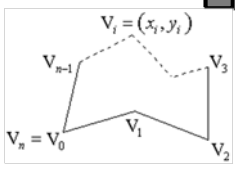
四角形面積



多角形面積

- When P=原点=(0,0)





アルゴリズム

1. 定義
   - 数学などの問題を解くための計算手順・方法
   - A finite set of precise instructions for performing a computation or for solving a problem
2. 評価
   - 領域計算量(space complexity)
   - 時間計算量(time complexity)

実例

1. 線形探索（Linear search）
   - 数列 $a_1, a_2, \ldots, a_n$
   - xを探す
   - 最大計算量＝O(n)
2. 2分探索（Binary search）
   - 数列 $a_1, a_2, \ldots, a_n$, where $a_1 < a_2 < \ldots < a_n$
   - xを探す
   - 最大計算量＝O($\log_2 n$)

計算量の推計

1. 探索 (Searching)、並び替え (Sorting)
   - 比較の回数 (the number of comparisons)
2. 数値計算 (Arithmetic calculation)
   - 乗算の回数 (the number of multiplications)
3. いくつかのステップに分解して、ループを探し、各ステップの計算量を解析する
   1. ループ処理の中身→乗算、比較
   2. 繰り返し回数の上限値
      - 繰り返し回数を影響する処理がある場合→条件判断等

- Loop！Loop！
  - 多くのアルゴリズムは大抵幾つかのループから構成される→ループの解析は一番重要！

計算量の種類

- Worst case → 最大計算量
- Average case → 平均計算量
- Best case → 最小計算量

計算量の記述 ― Big O , Ω , θ

- Big O
  - $f(n) = O(g(n))$ iff there exist positive constants c and n_0 such that $f(n) \le cg(n)$, for all $n \ge n_0$
  - For all sufficiently large n, $g(n)$ is an upper bound for f.
- Ω
  - $f(n) = \Omega(g(n))$ iff there exist positive constants c and n_0 such that $f(n) \ge cg(n)$, for all $n \ge n_0$
  - For all sufficiently large n, $g(n)$ is a lower bound for f.
- Θ
  - $f(n) = \theta(g(n))$ iff there exist positive constants c and n_0 such that $f(n) = cg(n)$, for all $n \ge n_0$
  - For all sufficiently large n, $g(n)$ is a tight bound for f.

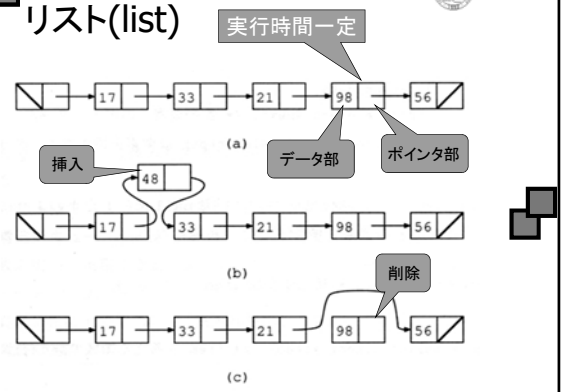
nと伴う計算量関数$f(n)$の変化

f(n)	1	2	4	8	16	32
1	1	1	1	1	1	1
Log n	0	1	2	3	4	5
n	1	2	4	8	16	32
n Log n	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32,768
2^n	2	4	16	256	65,536	4,294,967,296
n!	1	2	24	40320	20,922,789,888,000	$2.6313X10^{35}$

データ構造

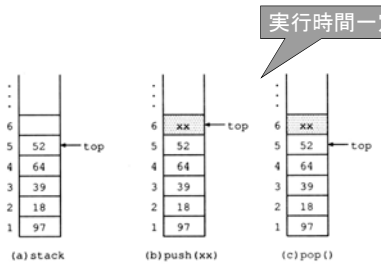
- 定義
  - 基本的な操作（検索、挿入、削除など）を効率よく実行できるために、操作対象となるデータ集合の組織形態
- 基本データ構造
  1. リスト
  2. スタックとキュー
  3. ヒープ
  4. 2分探索木
  5. 平衡2分探索木

リスト(list)



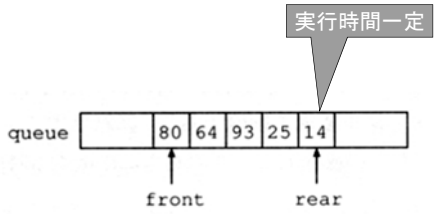
スタック(stack)

- プッシュpush(挿入)とポップpop(削除)
- 後入れ先出し（last-in first-out, LIFO）



キュー(queue)

- 二つのポインタ変数frontとrear
- 先入れ先出し（first-in first-out, FIFO）



木(tree)

- 根、親、子
- 兄弟、先祖、子孫、
- 葉（外点）
- 節点（内点）
- 深さ、高さ（枝の数）

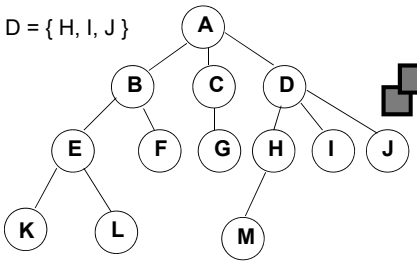
木(tree)

内部節点Internal Nodes = {A, B, C, D, E, H }

葉Leaf Nodes = { K, L, F, G, M, I, J }

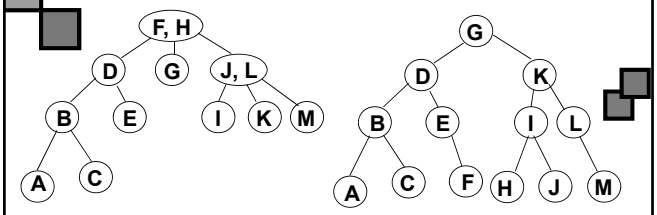
Dの子Children of D = { H, I, J }

根Root = { A }



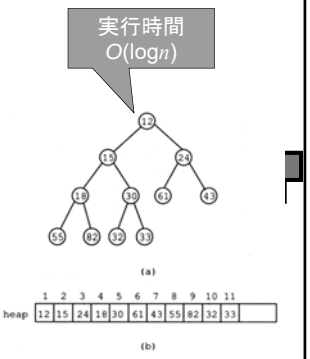
2分木(Binary tree)

- どの節点も2個以下の子を持つ



ヒープ(heap)

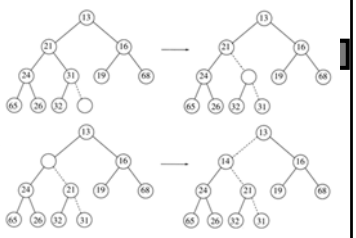
- 条件
  - 親≦子孫
- 根
  - 最小値
- 配列にての配置
  - 親i番
  - 左子2i 右子2i+1
- 操作
  - 挿入、最小値削除



ヒープの挿入(任意値⑭を挿入)

1. 次の位置に穴を用意する
2. If 親≦挿入値？
   - Then 挿入する
   - Else 親を下ろす
3. Repeat from 2

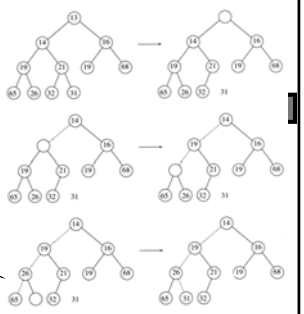
実行時間
$O(\log n)$



ヒープの削除(最小値=根⑬を削除)

1. 根に穴を空ける
2. 穴に小さい子を入れる
3. 最低層まで、Repeat 2
4. 最後の要素を入れる

実行時間
$O(\log n)$

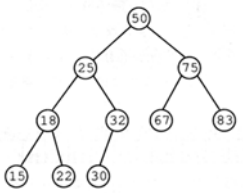


2分探索木(binary search tree)

- データ集合の基本操作→検索、挿入、削除
- 辞書構造＝アルファベット順、昇順、降順
  - 2分探索→$O(\log n)$、挿入、削除→$O(n)$

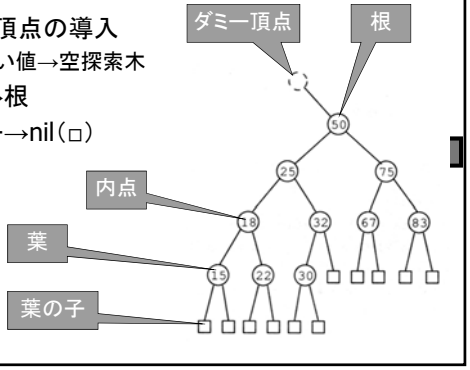
問題:
- 挿入、削除→$O(\log n)$？

答え:
- 2分探索木
  - 左子孫≦親≦右子孫



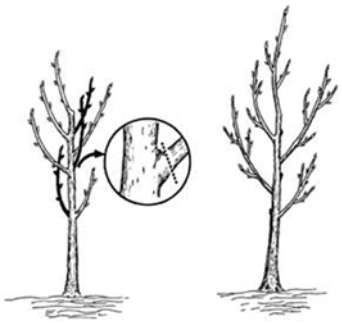
2分探索木の構築

- ダミー頂点の導入
  - 小さい値→空探索木
- 右子→根
- 葉の子→nil(□)



3つの基本操作

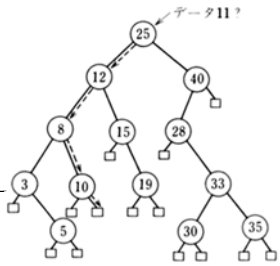
- 検索
  - find(x)
- 挿入
  - insert(x)
- 削除
  - delete(x)



検索find(x)

- 根から始めて検索データxを内点vと比較し、
- $x<v$→左部分木
- $x>v$→右部分木
- $x=v$→found
- $x\neq v$→no found

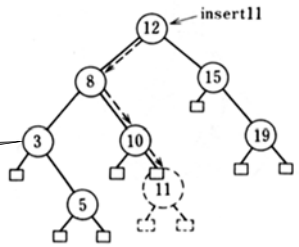
実行時間 O(log$n$)



挿入insert(x)

- find(x)→最後の内点vまで検索
- $x>v$ →右に挿入
- $x<v$ →左に挿入
- $x=v$ →有、無視

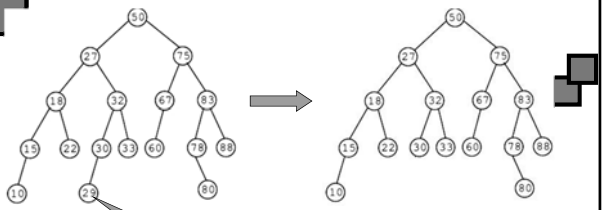
実行時間 O(log$n$)



削除delete(x) －1/3

- find(x)=v

1. v→葉→削除→終了

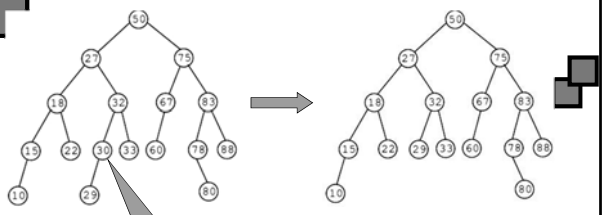


delete(29)

削除delete(x) －2/3

- find(x)=v

2. v→一子→削除→ vの子で置き換える



delete(30)

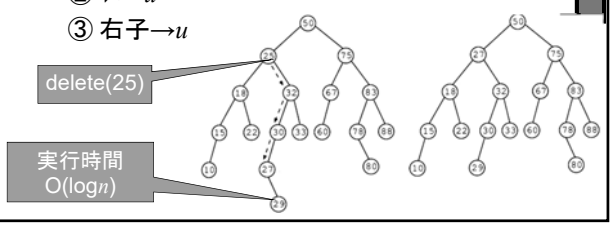
削除delete(x) －3/3

- find(x)=v

3. v→双子
   ① vより大きい最小値uを検索(右子の左子孫)
   ② v← u
   ③ 右子→u

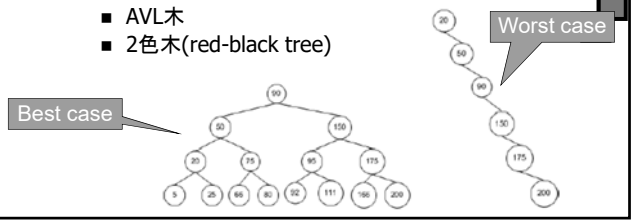
delete(25)

実行時間 O(log$n$)



平衡2分探索木 (balanced binary search tree)

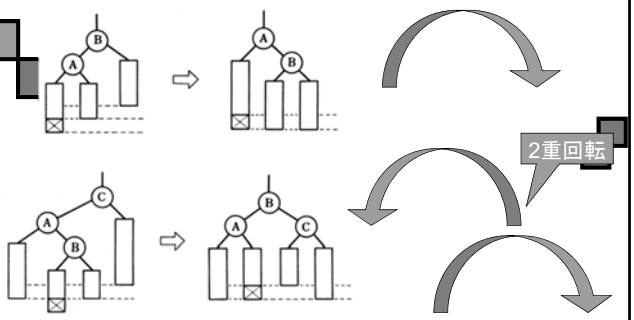
- 一般的
  - 高さ＝O(log$n$)
  - 操作時間＝O(log$n$)
- 木の左右バランスを保つには→How to …?
  - AVL木
  - 2色木(red-black tree)



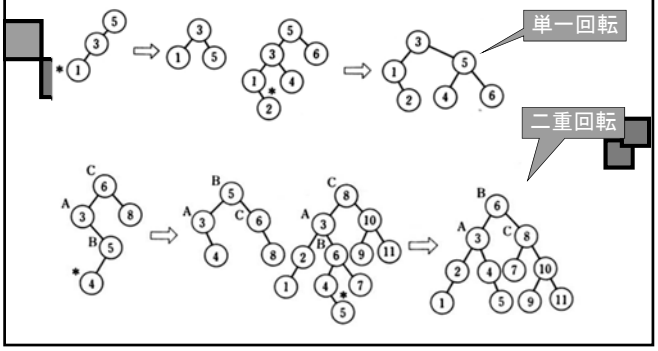
AVL木

- 発明者
  - 二人のロシアコンピュータ学者 Adelson-Velskii と Landis、1962
- 定義
  - どの内点においても、左部分木と右部分木の高さの差は1以下を満たす二分探索木
- 判断指標
  - バランス度＝(右ー左)部分木の高さ
- 操作方法
  - 通常の2分木と同様
  - バランス度の変化を計算する
  - バランス度の復元操作→単一回転・二重回転

AVL木の回転操作

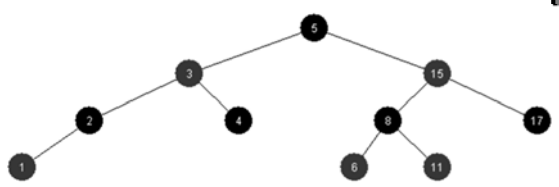


AVL木回転操作の実例



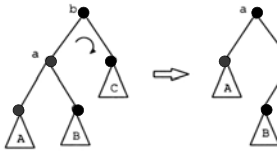
2色木(red-black tree)

- 構築ルール
  - 赤と黒のプロパティを追加
  - 内点＝黒、又は赤
  - 根＝黒、新挿入の子＝赤
  - 赤内点→黒子
  - 根から外点に至るすべてのルートは同じ数の黒子

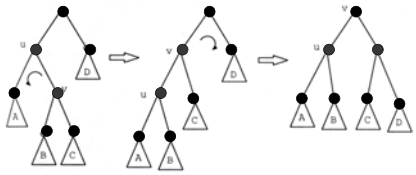


2色木の回転操作

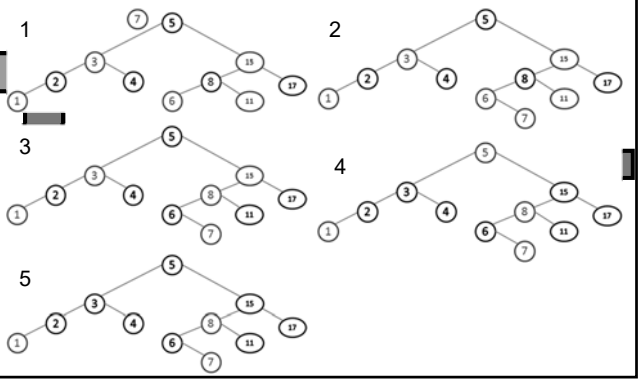
- 単一回転



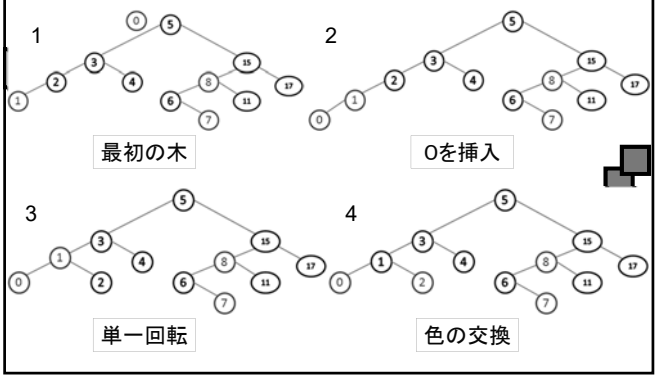
- 二重回転



操作実例(⑦の挿入と色の変化)

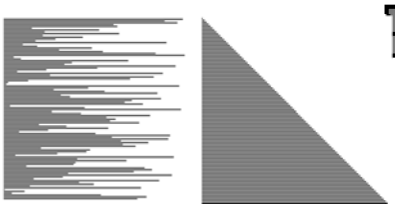


操作実例(⓪の挿入と色の変化)



データ集合の並び替え

- 直接挿入法
- 直接選択法
- バブル法
- 振動法
- 快速法



直接挿入法(Straight Insertion)

- データを一つずつ処理し、該当する位置に直接挿入する

	44	55	12	42	94	18	06	67
$i=2$	44	55	12	42	94	18	06	67
$i=3$	12	44	55	42	94	18	06	67
$i=4$	12	42	44	55	94	18	06	67
$i=5$	12	42	44	55	94	18	06	67
$i=6$	12	18	42	44	55	94	06	67
$i=7$	06	12	18	42	44	55	94	67
$i=8$	06	12	18	42	44	55	67	94

直接選択法(Straight Selection)

- 最小の要素を直接選択して、該当する位置に入れる

44	55	12	42	94	18	06	67
06	55	12	42	94	18	44	67
06	12	55	42	94	18	44	67
06	12	18	42	94	55	44	67
06	12	18	42	94	55	44	67
06	12	18	42	44	55	94	67
06	12	18	42	44	55	94	67
06	12	18	42	44	55	67	94

バブル法(Bubble Sort)

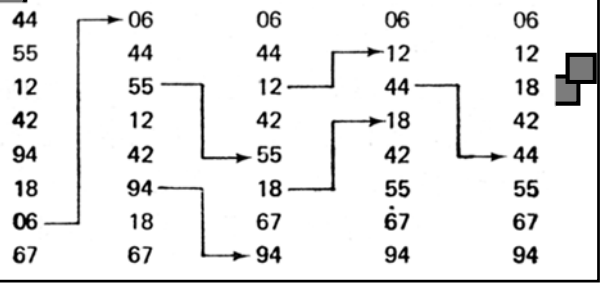
- 一番下から要素を選んで、上の要素と比較
- 上の要素より大きいところまで浮上

	i=2	i=3	i=4	i=5	i=6	i=7	i=8
44	06	06	06	06	06	06	06
55	44	12	12	12	12	12	12
12	55	44	18	18	18	18	18
42	12	55	44	42	42	42	42
94	42	18	55	44	44	44	44
18	94	42	42	55	55	55	55
06	18	94	67	67	67	67	67
67	67	67	94	94	94	94	94

振動法(Shaker Sort)

- 下から比較し、浮上
- 上から比較し、下降

双方向のバブル法



快速法(Quick Sort)

1. 基準値←(約)真中の要素
2. 前半の要素→後半 if ＞基準値
3. 後半の要素→前半 if ＜基準値
4. 2つのサブセットについて1～3を繰り返す