データ構造とアルゴリズム　第6, 7, 8回

1. アルゴリズムの重要性
2. 探索問題
3. 基本的なデータ構造
4. 動的探索問題とデータ構造
5. データの整列
6. グラフのアルゴリズム
7. 文字列のアルゴリズム
8. アルゴリズム設計手法

第５章　データの整列

＊　テキストにない内容

5.1 バブルソート
5.2 セレクションソート
5.3 インサーションソート
5.4 シェルソート
5.5 ヒープソート
5.6 クイックソート
＊　サンプルソート
5.7 マージソート
5.8 ソート問題の計算複雑度
＊　ビンソート，基底法

この章の学習目標

- さまざまな整列アルゴリズムとその計算時間を例を用いて説明できる
  - バブルソート，セレクションソート，インサーションソート，シェルソート，ヒープソート，クイックソート，サンプルソート，マージソート，ビンソート，基底法
- 整列問題の時間計算量の下界について，理由とともに説明できる
- 整列アルゴリズムのプログラムを書ける

整列（ソート）問題の定義

- 整列（ソート）問題の定義
  - 与えられたデータを**昇順（小さい順）**に並べ替え
  - n：データ数
  - データ　$data[0..n-1]$

		→		
0	15		0	5
1	8		1	7
2	31		2	8
3	17		3	9
4	18		4	15
5	22		5	17
6	7		6	18
7	25		7	22
8	9		8	25
9	5		9	31

5.1 バブルソート

- バブルソート
  - 逆順ペアの交換
    - 逆順ペア　$data[i] > data[i+1]$

```
for (k=1 to n-1)
   for (i=0 to n-k-1)
      if (data[i] >data[i+1]) data[i] とdata[i+1] を交換
```


バブルソート：実行例

```
for (k=1 to n-1)
   for (i=0 to n-k-1)
      if (data[i] >data[i+1]) data[i] とdata[i+1] を交換
```

- 実行例

0	15	8	8	8	5
1	8	15	15	15	7
2	31	17	17	17	8
3	17	18	18	7	9
4	18	22	7	18	15
5	22	7	22	9	17
6	7	25	9	5	18
7	25	9	5	22	22
8	9	5	25	25	25
9	5	31	31	31	31

黄色：ソート済で，内のfor文の範囲外

バブルソート：時間計算量

- 計算時間　常に $O(n^2)$
  - 比較回数：$n(n-1)/2$
  - 交換回数：0 ～ $n(n-1)/2$

整列済みのとき 0回
逆順のとき $n(n-1)/2$ 回

```
for (k=1 to n-1)
   for (i=0 to n-k-1)
      if (data[i] >data[i+1]) data[i] とdata[i+1] を交換
```


第５章　データの整列

＊　テキストにない内容

5.1 バブルソート
5.2 セレクションソート
5.3 インサーションソート
5.4 シェルソート
5.5 ヒープソート
5.6 クイックソート
＊　サンプルソート
5.7 マージソート
5.8 ソート問題の計算複雑度
＊　ビンソート，基底法

5.2 セレクションソート

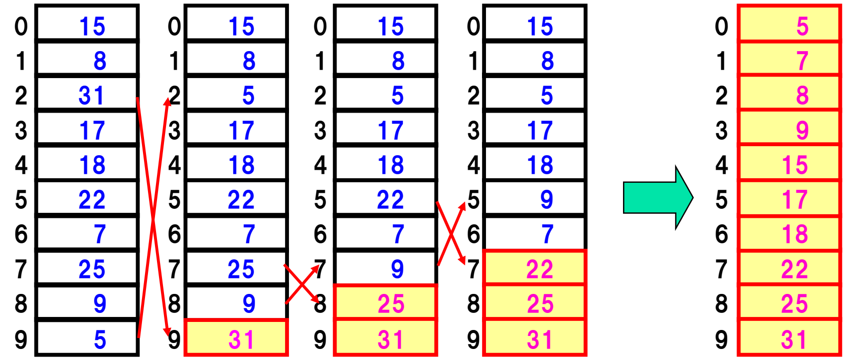
- 最大値から大きい順に選択

```
for (k=n-1 to 1 step -1) {
    data[0]～data[k]の最大値data[m]を求める
    data[m]とdata[k]を交換;
}
```


セレクションソート：実行例

- 実行例

```
for (k=n-1 to 1 step -1) {
    data[0]～data[k]の最大値data[m]を求める
    data[m]とdata[k]を交換;
}
```



黄色：ソート済で，for文での処理が終了

セレクションソート：時間計算量

- 計算時間　常に $O(n^2)$
  - 比較回数：$n(n-1)/2$
  - 交換回数：0 ～ $n-1$

    整列済みのとき 0回

```
for (k=n-1 to 1 step -1) {
    data[0]～data[k]の最大値data[m]を求める
    data[m]とdata[k]を交換;
}
```


第５章　データの整列

＊　テキストにない内容

5.3 インサーションソート

- 整列済みの範囲を配列の前から順に拡大

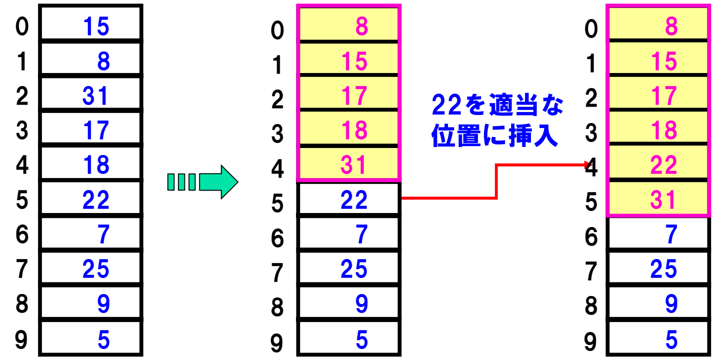
```
for (i=1 to n-1) {
    x=data[i]; j=i;
    while ((j>0) && (data[j-1]>x)) {
        data[j-1]をdata[j]に移動し, j=j-1とする
    }
    data[j]=x;
}
```

テキストの $(j-1)>0$ は誤り

インサーションソート：実行例

- 実行例

```
for (i=1 to n-1) {
    x=data[i]; j=i;
    while ((j>0) && (data[j-1]>x)) {
        data[j-1]をdata[j]に移動し, j=j-1とする }
    data[j]=x; }
```



黄色：この範囲内でソート済

インサーションソート：時間計算量

- 計算時間　最大 $O(n^2)$

  最小 $O(n)$

  - 比較回数：$n-1$ ～ $n(n-1)/2$

    整列済みのとき $n-1$ 回

    逆順のとき $n(n-1)/2$ 回
- 重要な特長
  - ほぼソート済の入力は $O(n)$ 時間で整列できる

第５章　データの整列

＊　テキストにない内容

5.1 バブルソート
5.2 セレクションソート
5.3 インサーションソート
5.4 シェルソート
5.5 ヒープソート
5.6 クイックソート
＊　サンプルソート
5.7 マージソート
5.8 ソート問題の計算複雑度
＊　ビンソート，基底法

5.4 シェルソート

- これまでに紹介した整列アルゴリズムの計算時間
  - バブルソート，セレクションソート　常に $O(n^2)$
  - インサーションソート　最大 $O(n^2)$　　最小 $O(n)$
    - 入力がほぼ整列済みのとき高速
- シェルソートの計算時間
  - パラメタによって異なる．解析は難しい
  - あるパラメタでは
    - 最大 $O(n^{4/3})$，平均 $O(n^{5/4})$

    ＊最大 $O(n \log^2 n)$ のパラメタもある

シェルソート：方針

- 方針（インサーションソートの応用）
  1. 大ざっぱにソート（インサーションソート）
     - 部分列に分割したソートを繰り返す
       - h 個の部分列に分割した場合
         - １つの部分列の長さ　n/h
         - １つの部分列の整列　$O((n/h)^2)$
         - h 個の部分列の整列　$O(n^2/h)$
  2. インサーションソートでソート

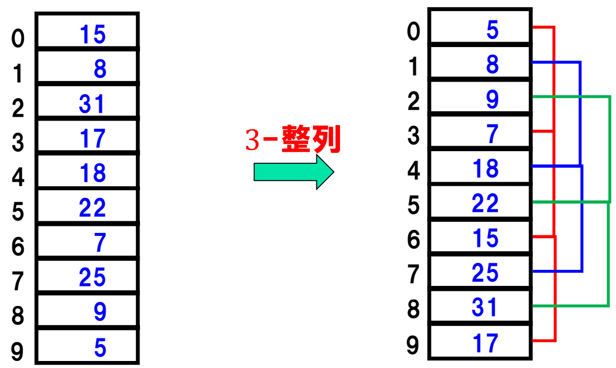
     ほぼソート済みなので高速

シェルソート：h-整列

- 方針（インサーションソートの応用）
  1. 大ざっぱにソート（インサーションソート）

     h-整列
     - 各 i について，$data[i] \leq data[i+h]$
  2. インサーションソートでソート

シェルソート：h-整列



シェルソート

- **シェルソート**
  - h_1-**整列 → h_2-整列 → ・・・ → 1-整列**
    （$h_1 > h_2 > \cdots > 1$）
  - **各整列は，インサーションソートを適用**
  - h-**整列**
    - **各 i について，** $data[i] \leq data[i+h]$
    - **1-整列＝整列済み**

シェルソート：実行例

- **実行例**
  - $h_1 = 5,\ h_2 = 3,\ h_3 = 1$

	初期	→	5-整列後	→	3-整列後	→	1-整列後
0	15		15		8		5
1	8		7		5		7
2	31		25		17		8
3	17		9		9		9
4	18		5		7		15
5	22		22		22		17
6	7		8		15		18
7	25		31		31		22
8	9		17		25		25
9	5		18		18		31

シェルソート：時間計算量

- **計算時間**
  - $h_1,\ h_2, \ldots, h_m$ **の選び方に依存**
    - **うまく選べば**
      - **最大**　$O(n^{4/3})$　（$O(n \log^2 n)$ **も可能**）
      - **平均**　$O(n^{5/4})$
    - **よく利用される系列**
      - $2^k - 1$　　1, 3, 7, 15, 31, 63, …
      - $(3^k - 1)/2$　1, 4, 13, 40, 121, 364, …
    - **プログラムが簡単で，かなり高速**

データ構造とアルゴリズム　第6, 7, 8回

1. **アルゴリズムの重要性**
2. **探索問題**
3. **基本的なデータ構造**
4. **動的探索問題とデータ構造**
5. **データの整列**
6. **グラフのアルゴリズム**
7. **文字列のアルゴリズム**
8. **アルゴリズム設計手法**

第５章　データの整列

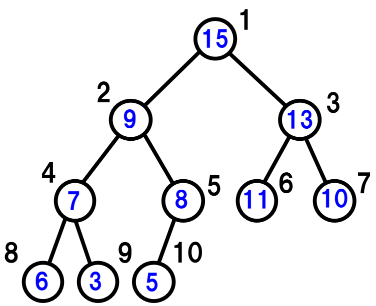
＊　テキストにない内容

5.5 ヒープソート

- ヒープを利用したソート
- ヒープ（第３章で学習済）
  - ２分木を配列 $data[1..n]$ で実現（n：データ数）
    - $data[1]$：根
    - $data[k]$ の左の子：$data[2k]$
    - $data[k]$ の右の子：$data[2k+1]$
  - 親のデータ ≥ 子のデータ
  - データの挿入　$O(\log n)$ 時間
  - 最大データ（根のデータ）の取出し　$O(\log n)$時間

ヒープ

- ヒープの例



1	15
2	9
3	13
4	7
5	8
6	11
7	10
8	6
9	3
10	5

ヒープソート

まずはこちらから

- ヒープソート
  1. ヒープを構成;　$O(n)$ 時間
  2. for (k=n-1; k>0; k--) {
     data[1] と data[k+1] を交換;　$O(1)$ 時間
     /* data[1] は data[1..k+1] の最大値 */
     data[1..k] でヒープを再構成　$O(\log n)$ 時間
     }
- 計算時間　最大 $O(n \log n)$

ヒープソート：ヒープ構成後の動作

アルゴリズム 5.5 の後半

```
for (k=n-1; k>0; k--) {
    x=data[k+1]; data[k+1]=data[1]; i=1; j=2;
    while (j<=k) {
        if (j<k && data[j+1]>data[j]) j=j+1;
        if (x<data[j]) {
            data[i]=data[j]; i=j; j=2*i;
        } else break;
    }
    data[i]=x;
}
```

テキスト $k-$ は誤り

$data[1]$ と $data[k+1]$ を交換

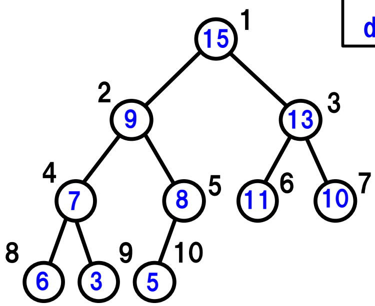
左右の子の大きい方が j

親が子より小さければ入替

ヒープ構成後の動作例 (1)

- ヒープ構成後の動作

ヒープ構成済
data[1..10]



1	15
2	9
3	13
4	7
5	8
6	11
7	10
8	6
9	3
10	5

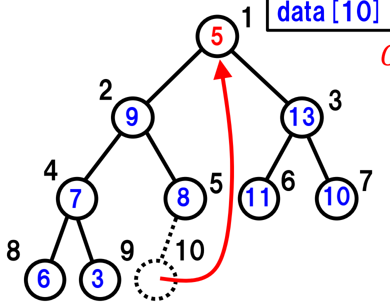
黄色：ヒープ

ヒープ構成後の動作例 (2)

- ヒープ構成後の動作

最大値（根のデータ）
を取出し（削除），
data[10] に移動

$O(1)$ 時間



1	5
2	9
3	13
4	7
5	8
6	11
7	10
8	6
9	3
10	15

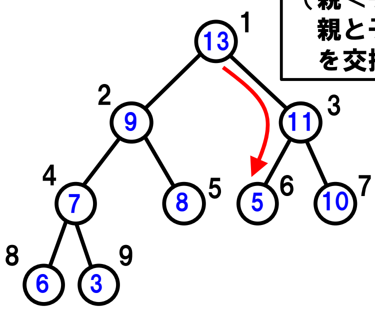
黄色：ヒープ

ヒープ構成後の動作例 (3)

- ヒープ構成後の動作

ヒープの再構成
（親＜子 なら，
親と子の大きい方
を交換）

$O(\log n)$ 時間



1	13
2	9
3	11
4	7
5	8
6	5
7	10
8	6
9	3
10	15

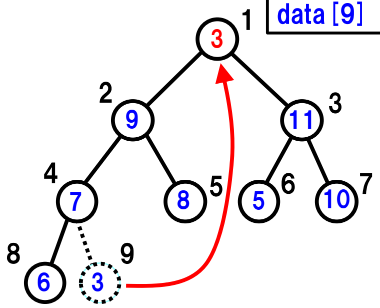
黄色：ヒープ

ヒープ構成後の動作例 (4)

- ヒープ構成後の動作

最大値（根のデータ）を取出し（削除），data [9] に移動

$O(1)$ 時間



1	3
2	9
3	11
4	7
5	8
6	5
7	10
8	6
9	13
10	15

黄色：ヒープ

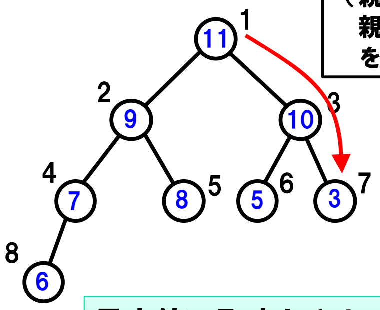
ヒープ構成後の動作例 (5)

- ヒープ構成後の動作

ヒープの再構成

（親＜子 なら，
親と子の大きい方
を交換）

$O(\log n)$ 時間



1	11
2	9
3	10
4	7
5	8
6	5
7	3
8	6
9	13
10	15

最大値の取出しとヒープ再構成を
n 回繰り返せば，ソート完了
$O(n \log n)$ 時間

黄色：ヒープ

ヒープソート

- ヒープソート

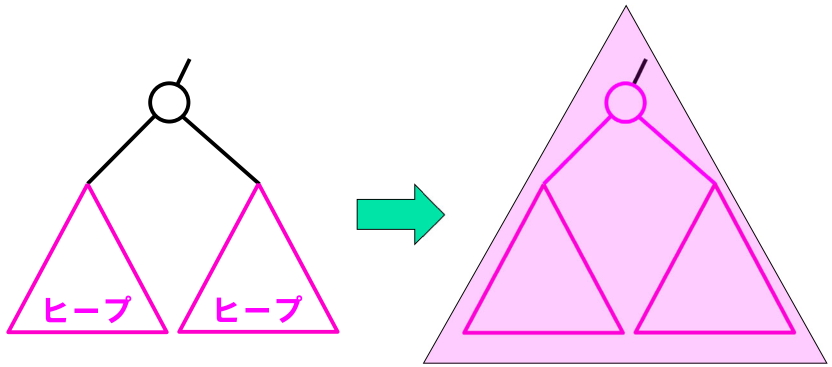
次はこちら

1. ヒープを構成; $O(n)$ 時間
2. for (k=n-1; k>0; k--) {
   data [1] と data [k+1] を交換; $O(1)$ 時間
   /* data [1] は data [1..k+1] の最大値 */
   data [1..k] でヒープを再構成 $O(\log n)$ 時間
   }

- 計算時間　最大 $O(n \log n)$

ヒープソート：ヒープの構成（初期化）

- 全体で $O(n)$ 時間
  - 葉に近い部分からヒープを構成



ヒープ

ヒープの構成（初期化）

- 全体で $O(n)$ 時間
  - 葉に近い部分からヒープを構成

アルゴリズム 5.5 の前半

```
for (k=n/2; k>0; k=k-1) {
    i=k;  j=2*i;  x=data[i];
    while (j<=n) {
        if (j<n && data[j+1]>data[j]) j=j+1;
        if (x<data[j]) {
            data[i]=data[j]; i=j; j=2*i;
        } else break;
    }
    data[i]=x;
}
```

$data[k]$ は子を持つ

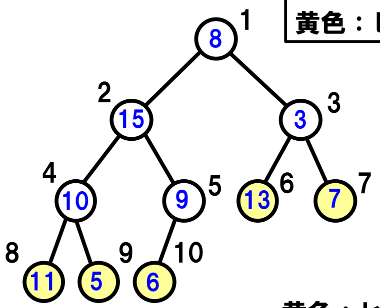
左右の子の大きい方が j

親が子より小さければ入替

ヒープの構成（初期化）：実行例(1)

- 全体で $O(n)$ 時間
  - 葉に近い部分からヒープを構成

初期状態 data[1..10]
黄色：ヒープ条件成立



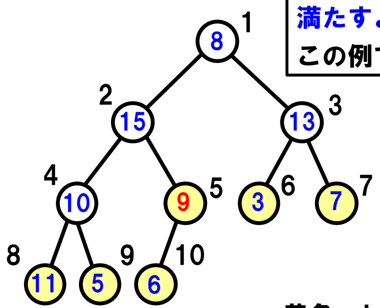
黄色：ヒープ条件成立

1	8
2	15
3	3
4	10
5	9
6	13
7	7
8	11
9	5
10	6

ヒープの構成（初期化）：実行例(2)

- 全体で $O(n)$ 時間
  - 葉に近い部分からヒープを構成

data[5] がヒープ条件を満たすようにする
この例では変更不要



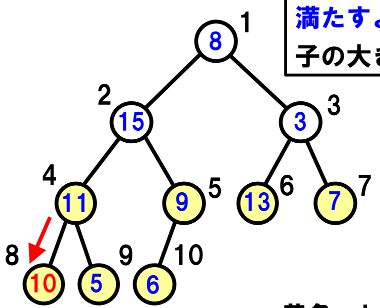
黄色：ヒープ条件成立

1	8
2	15
3	13
4	10
5	9
6	3
7	7
8	11
9	5
10	6

ヒープの構成（初期化）：実行例(3)

- 全体で $O(n)$ 時間
  - 葉に近い部分からヒープを構成

data[4] がヒープ条件を満たすようにする
子の大きい方と交換



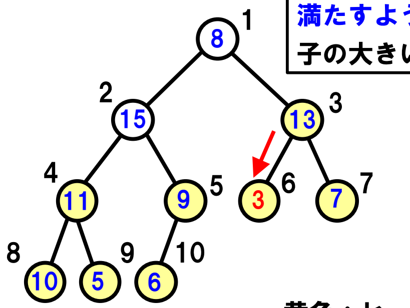
黄色：ヒープ条件成立

1	8
2	15
3	3
4	11
5	9
6	13
7	7
8	10
9	5
10	6

ヒープの構成（初期化）：実行例 (4)

- 全体で $O(n)$ 時間
  - 葉に近い部分からヒープを構成

data [3] がヒープ条件を満たすようにする
子の大きい方と交換



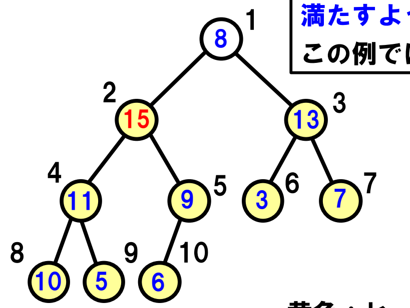
黄色：ヒープ条件成立

1	8
2	15
3	13
4	11
5	9
6	3
7	7
8	10
9	5
10	6

ヒープの構成（初期化）：実行例 (5)

- 全体で $O(n)$ 時間
  - 葉に近い部分からヒープを構成

data [2] がヒープ条件を満たすようにする
この例では変更不要



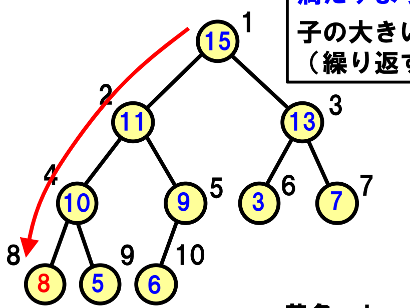
黄色：ヒープ条件成立

1	8
2	15
3	13
4	11
5	9
6	3
7	7
8	10
9	5
10	6

ヒープの構成（初期化）：実行例 (6)

- 全体で $O(n)$ 時間
  - 葉に近い部分からヒープを構成

data [1] がヒープ条件を満たすようにする
子の大きい方と交換
（繰り返す）



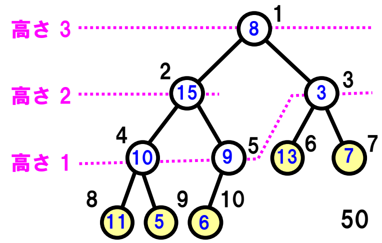
黄色：ヒープ条件成立

1	15
2	11
3	13
4	10
5	9
6	3
7	7
8	8
9	5
10	6

ヒープの構成（初期化）：時間計算量

- 全体で $O(n)$ 時間
  - data $[\lfloor\frac{n}{2}\rfloor]$, data $[\lfloor\frac{n}{2}\rfloor - 1]$,⋯, data [1] の順にヒープを構成
  - data $[i]$ の処理：data $[i..n]$ がヒープ条件を満たすようにする
    - data $[i]$ の節点の高さに比例する時間を要する
      - 高さ：子孫の葉までの距離の最大値
    - 高さ h の節点：$\lceil\frac{n}{2^{h+1}}\rceil$ 個（$1 \le h \le \lfloor\log n\rfloor$）
  - 計算時間
    - $\sum_{h=1}^{\lfloor\log n\rfloor}\lceil\frac{n}{2^{h+1}}\rceil \cdot h = O\left(n\sum_{h=1}^{\log n}\frac{h}{2^{h+1}}\right)$
      $= O(n)$

$\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}$ を利用



ヒープソート：時間計算量

- ヒープソート
  1. ヒープを構成; $O(n)$ 時間
  2. for (k=n-1; k>0; k--) {
     data[1] と data[k+1] を交換; $O(1)$ 時間
     /* data[1] は data[1..k+1] の最大値 */
     data[1..k] でヒープを再構成 $O(\log n)$ 時間
     }
- 計算時間　最大 $O(n \log n)$

第５章　データの整列

＊　テキストにない内容

5.6 クイックソート

- 計算時間
  - 平均　$O(n \log n)$
  - 最大　$O(n^2)$
    - 基本操作が簡単
    - 実際に高速でもっともよく使用されている

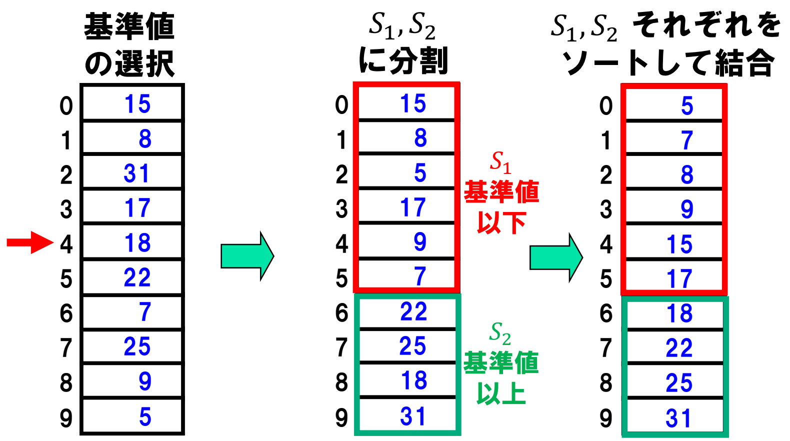
クイックソートの考え方

S：データ集合

1. $|S| \leq 1$ ならば，S を返して終了
2. S から任意に一つの要素 x を選ぶ（基準値）
3. S を，
   x 以下の要素の集合 S_1，
   x 以上の要素の集合 S_2　に分割
   （x は S_1, S_2 のどちらに含めてもよい）
4. S_1，S_2 それぞれを再帰的にソート
5. S_1，S_2 のソート列を連結する

分割統治法（divide and conquer）

クイックソートの考え方



クイックソートの計算時間

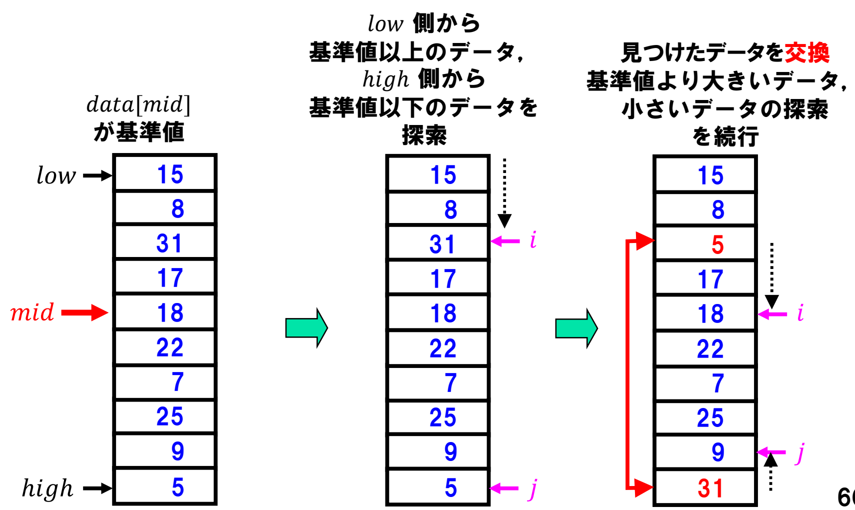
S：データ集合

1. $|S| \leq 1$ ならば，S を返して終了 —— $O(1)$
2. S から任意に一つの要素 x を選ぶ（基準値） —— $O(1)$
3. S を，
   x 以下の要素の集合 S_1，
   x 以上の要素の集合 S_2　に分割 —— $O(n)$
   （x は S_1, S_2 のどちらに含めてもよい）
4. S_1，S_2 それぞれを再帰的にソート —— 再帰の段数は基準値に依存
5. S_1，S_2 のソート列を連結する

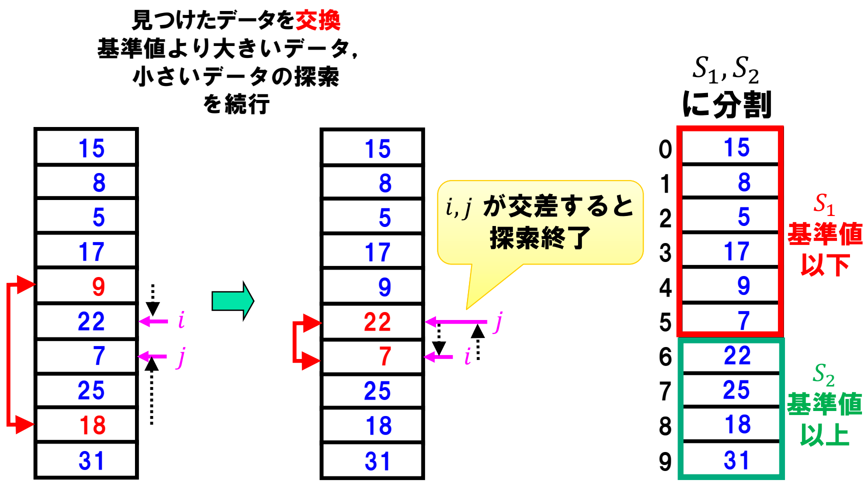
基準値の選び方

- 再帰の段数は基準値に依存
  - a. 毎回、最小値／最大値（に近い値）を選ぶと
    - 再帰の段数　$O(n)$
    - 計算時間　$O(n^2)$　最大
  - b. 毎回、中央値（に近い値）を選ぶと
    - 再帰の段数　$O(\log n)$
    - 計算時間　$O(n \log n)$　最小，平均
- 上記の a. を避けるように基準値を選びたい
  - $data[low..high]$ のソートによく利用する基準値
    - $data[mid]$ ： $mid = (low + high)/2$
    - $data[low]$，$data[mid]$，$data[high]$ の中央値

S_1, S_2 への分割方法 (1)



S_1, S_2 への分割方法 (2)



クイックソート

```
quicksort (int low, int high) {
    if (low<high) {
        int mid= (low + high) / 2;
        int x=data [mid] ;
        int i=low;  int j=high;
        while (i<=j) {
            while (data [i] <x)  i=i+1;
            while (data [j] >x)  j=j-1;
            if (i<=j)  swap (&data [i++] , &data [j--] ) ;
}
        quicksort (low, j) ;
        quicksort (i, high) ;
    }
}
```


クイックソートの最大計算時間

- **最大計算時間：$O(n^2)$**
  - **毎回，data[$low..high$] の最小値を基準値とした場合**
    （最大値を基準値とした場合も同様の議論）
    - $|S_1| = 1,\ |S_2| = high - low$
  - **最大比較回数 $W(n)$**

S_1, S_2 への分割に要する比較回数

$$W(n) = n + 2 + W(n-1)$$

$$W(n) = (n+2) + (n+1) + n + (n-1) + \cdots = O(n^2)$$

クイックソートの平均計算時間

- **平均計算時間：$O(n \log n)$**
  - **仮定：各要素を等確率で基準値とする**
  - **k 番目に小さい値を基準値に選んだ場合**
    - $|S_1| = k,\ |S_2| = n - k$　**（基準値は S_1 に属すると仮定）**
    - **ただし，k が最大値の場合は $|S_1| = n-1,\ |S_2| = 1$**
- **平均比較回数 $C(n)$**

$$C(n) = \Sigma_{k=1}^{n-1}(n + 2 + C(k) + C(n-k))/n$$
$$+(n + 2 + C(n-1) + C(1))/n$$
$$= n + 2 + 2\Sigma_{k=1}^{n-1}C(k)/n + (C(n-1) + C(1))/n$$
$$= 1.38\, n \log_2 n + O(n)$$

第５章　データの整列

＊　テキストにない内容

＊サンプルソート

- クイックソートの拡張
  - 実際的には，
    クイックソート（３値の中央値）と同程度の速度
    ３値（先頭 (low)，中央 (mid)，末尾 $(high)$ ）の中央値
  - より大きい n に対し，
    クイックソート（３値の中央値）より高速
- 平均比較回数（主要項）
  - 単純クイックソート　$1.38n \log_2 n$
  - クイックソート（３値の中央値）　$1.19n \log_2 n$
  - サンプルソート　$1n \log_2 n$

＊サンプルソート

- クイックソート
  - １つの基準値を使って分割
- サンプルソート
  - 複数の基準値（サンプル）を使って分割

サンプルソートの概要と時間計算量

1. $n/\log n$ 個のサンプルを抽出　$O(n/\log n)$ 時間
2. $n/\log n$ 個のサンプルを整列（クイックソート）　平均 $O(n)$
3. サンプルで，入力を $n/\log n + 1$ 個のグループに分割
   - ２分探索　比較回数　$n(\log_2(n/\log n)) \leq n \log_2 n$
4. 各グループを整列　平均 $O(n \log \log n)$
   - グループ数　$n/\log n + 1$
   - 各グループのデータ数　平均 $\log n$
   - 各グループの整列（クイックソート）　平均 $\log n \log \log n$

$\boxed{\text{主要項}\quad 1n \log_2 n}$

サンプルソート：実行例

20	8	27	17	18	15	7	25	9	5	10	3	31	16	19	22

サンプル選択

20	8	27	17	18	15	7	25	9	5	10	3	31	16	19	22

サンプルでグループ分け

7	5	3	8	15	9	10	16	17	18	19	20	25	22	27	31

グループ内でソート

3	5	7	8	9	10	15	16	17	18	19	20	22	25	27	31

＊サンプルソート

- クイックソートの拡張
  - 実際的には，
    クイックソート（３値の中央値）と同程度の速度
    ３値（先頭 (low)，中央 (mid)，末尾 $(high)$ ）の中央値
  - より大きい n に対し，
    クイックソート（３値の中央値）より高速
- 平均比較回数（主要項）
  - 単純クイックソート　$1.38n \log_2 n$
  - クイックソート（３値の中央値）　$1.19n \log_2 n$
  - サンプルソート　$1n \log_2 n$

データ構造とアルゴリズム　第6, 7, 8回

1. アルゴリズムの重要性
2. 探索問題
3. 基本的なデータ構造
4. 動的探索問題とデータ構造
5. データの整列
6. グラフのアルゴリズム
7. 文字列のアルゴリズム
8. アルゴリズム設計手法

第５章　データの整列

＊　テキストにない内容

5.1 バブルソート
5.2 セレクションソート
5.3 インサーションソート
5.4 シェルソート
5.5 ヒープソート
5.6 クイックソート
＊　サンプルソート
5.7 マージソート
5.8 ソート問題の計算複雑度
＊　ビンソート，基底法

5.7 マージソート

- 計算時間
  - 最大　$O(n \log n)$　最適
  - 実際的にはヒープソートより高速
  - 作業用記憶域：大きさ n の配列

マージソート：マージ操作

- マージ（併合）
  - 整列された２本の列（長さ：$m,\ k$）

    ↓　マージ　$O(m+k)$ 時間
  - 整列された１本の列（長さ：$m+k$）

16	23	29	33	41	57	77	91

11	31	55	63	66	71	82	93

⇩

11	16	23	29	31	33	41	55	57	63	66	71	77	82	91	93

マージソート：ボトムアップ式記述

- 最大 $O(n \log n)$ 時間
  - （長さ１の整列された列）$\times n$　：入力

    マージ↓$O(2\times(n/2)) = O(n)$ 時間
  - （長さ２の整列された列）$\times n/2$

    ↓ $O(4\times(n/4)) = O(n)$ 時間
  - （長さ４の整列された列）$\times n/4$

    ↓$O(8\times(n/8)) = O(n)$ 時間

    ・・・

    ↓$O((n/2)\times 2) = O(n)$ 時間
  - （長さ $n/2$ の整列された列）$\times 2$

    ↓$O(n)$ 時間
  - （長さ n の整列された列）$\times 1$

$\log n$ 段

全体で

$O(n \log n)$ 時間

マージソート：実行例 (1)

20	8	27	17	18	15	7	25	9	5	10	3	31	16	19	22

（長さ 1 の整列された列）$\times 16\ (= n)$

⇩ マージ

8	20	17	27	15	18	7	25	5	9	3	10	16	31	19	22

（長さ 2 の整列された列）$\times 8\ (= n/2)$

⇩ マージ

8	17	20	27	7	15	18	25	3	5	9	10	16	19	22	31

（長さ 4 の整列された列）$\times 4\ (= n/4)$

マージソート：実行例(2)

8	17	20	27	7	15	18	25	3	5	9	10	16	19	22	31

（長さ 4 の整列された列）$\times 4\ (= n/4)$

⇩ マージ

7	80	15	27	18	20	25	27	3	5	9	10	16	19	22	31

（長さ 8 の整列された列）$\times 2\ (= n/8)$

⇩ マージ

7	80	15	27	18	20	25	27	3	5	9	10	16	19	22	31

（長さ 16 の整列された列）$\times 1\ (= n/16)$

マージソート：トップダウン式記述

- 最大 $O(n \log n)$ 時間
  - （長さ n の列）のソート　　$T(n)$ 時間
    ↓
  - （長さ $n/2$ の列）$\times 2$ に分割
    ↓
  - （長さ $n/2$ の列）それぞれをソート　$T(n/2)$ 時間 $\times 2$
    マージ ↓ $O(n)$ 時間
  - （長さ n の整列された列）

$T(n) = 2T(n/2) + O(n)$，$T(1) = 0$

- 漸化式を解くと，$T(n) = O(n \log n)$

第５章　データの整列

＊　テキストにない内容

5.1 バブルソート
5.2 セレクションソート
5.3 インサーションソート
5.4 シェルソート
5.5 ヒープソート
5.6 クイックソート
＊　サンプルソート
5.7 マージソート
5.8 ソート問題の計算複雑度
＊　ビンソート，基底法

ソートアルゴリズムの時間計算量

- バブルソート，セレクションソート　常に$O(n^2)$
- インサーションソート　最大，平均 $O(n^2)$，最小 $O(n)$
- シェルソート　最大 $O(n^{4/3})$，平均 $O(n^{5/4})$
- クイックソート，サンプルソート
  最大 $O(n^2)$，平均 $O(n \log n)$
- ヒープソート，マージソート　最大 $O(n \log n)$　最適

5.8 ソート問題の計算複雑度

- **最大計算時間の下界（かかい）** $\Omega(n \log n)$
  - 整列アルゴリズムの**最大時間計算量**は，少なくとも $n \log n$ に比例
  - 最大時間計算量 $O(n \log n)$ の整列アルゴリズム
    - 最大時間計算量に関して**最適**
    - ヒープソート，マージソート
  - アルゴリズムへの制限：**比較アルゴリズム**
    - データの大小比較により，次の動作を決定

Ω記法（オメガ記法）復習 p.29

$\Omega(f(n))$ オメガ $f(n)$，ビッグオメガ $f(n)$

ある正定数 n_0, c が存在し，

$n \geq n_0$ を満たすすべての n について，

$g(n) \geq c \cdot f(n)$ を満たす**関数** $g(n)$ **の集合**

スモールオメガ $\omega(f(n))$ もある

- $g(n) \in \Omega(f(n))$ を $g(n) = \Omega(f(n))$ と書くことも多い
- n が十分大きい場合の $g(n)$ の**下界**（かかい）
- $an + b$ を表すのに正しいのはどれ？

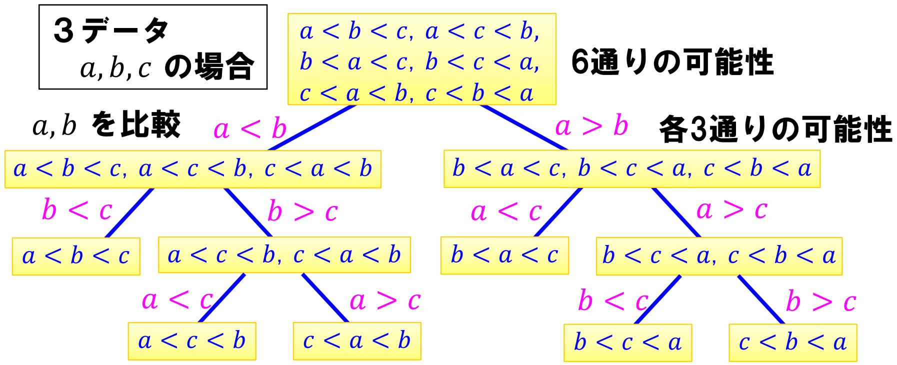
$\Omega(\log n)$ $\Omega(\sqrt{n})$ $\Omega(n)$

$\Omega(5n)$ $\Omega(n^2)$ $\Omega(n + \sqrt{n})$

なるべく大きな簡単な関数で表す $\Omega(n)$

ソート問題の計算時間の下界 (1)

- **最大計算時間の下界** $\Omega(n \log n)$
  - **決定木**による証明
    - 決定木：整列の過程を表す２分木



ソート問題の計算時間の下界 (2)

- **最大計算時間の下界** $\Omega(n \log n)$
  - **決定木**による証明
    - 決定木：整列の過程を表す２分木
  - 葉の数 $n!$
    → 木の高さ（計算時間）$\geq \log n! \approx n \log n$
    - Stirling の公式
      - $n! = (2\pi n)^{1/2} n^n e^{-n}$

第５章　データの整列

＊　テキストにない内容

＊比較以外の方法による整列

- ソートの最大時間計算量の**下界**　$\Omega(n \log n)$
  - アルゴリズムへの制限：**比較アルゴリズム**
    - データの大小比較により，次の動作を決定
- この制限がなければ高速化可能か？
  - ビンソート，基底法　最大時間計算量　$O(n)$

ビンソート

- ビンソート（バケツソート）
  - 最大時間計算量 $O(n)$
  - 基本的アイデア
    - データを 1〜m の整数に制限
    - 値 i 以下のデータの個数を $count[i]$ に求める
    - 値 i のデータは

      $count[i-1]+1$〜$count[i]$ 番目のデータ

ビンソート

データ：1〜8

入力

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

データ i の個数

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

データ i 以下の個数 $count[i]$

1	2
2	3
3	6
4	6
5	6
6	7
7	9
8	10

ソート結果

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

ビンソート：ソート結果の求め方 (1)

データ：1〜8

	入力	データ i の個数	データ i 以下の個数 $count[i]$	ソート結果
1	3	2	2	
2	7	1	3	
3	1	3	6 → 5	
4	1	0	6	
5	3	0	6	
6	6	1	7	3
7	8	2	9	
8	7	1	10	
9	2			
10	3			

ビンソート：ソート結果の求め方 (2)

データ：1〜8

	入力	データ i の個数	データ i 以下の個数 $count[i]$	ソート結果
1	3	2	2	
2	7	1	3 → 2	
3	1	3	5	2
4	1	0	6	
5	3	0	6	
6	6	1	7	3
7	8	2	9	
8	7	1	10	
9	2			
10	3			

ビンソート：ソート結果の求め方 (3)

データ：1〜8

	入力	データ i の個数	データ i 以下の個数 $count[i]$	ソート結果
1	3	2	2	
2	7	1	2	
3	1	3	5	2
4	1	0	6	
5	3	0	6	
6	6	1	7	3
7	8	2	9 → 8	
8	7	1	10	
9	2			7
10	3			

ビンソート：ソート結果の求め方 (4)

データ：1〜8

	入力	データ i の個数	データ i 以下の個数 $count[i]$	ソート結果
1	3	2	2	
2	7	1	2	
3	1	3	5	2
4	1	0	6	
5	3	0	6	
6	6	1	7	3
7	8	2	8	
8	7	1	10 → 9	
9	2			7
10	3			8

ビンソート：ソート結果の求め方 (5)

データ：1〜8

	入力	データ i の個数	データ i 以下の個数 $count[i]$	ソート結果
1	3	2	2	
2	7	1	2	
3	1	3	5	2
4	1	0	6	
5	3	0	6	
6	6	1	7 → 6	3
7	8	2	8	6
8	7	1	9	
9	2			7
10	3			8

ビンソート：ソート結果の求め方 (6)

データ：1〜8

	入力	データ i の個数	データ i 以下の個数 $count[i]$	ソート結果
1	3	2	2	
2	7	1	2	
3	1	3	5 → 4	2
4	1	0	6	
5	3	0	6	3
6	6	1	6	3
7	8	2	8	6
8	7	1	9	
9	2			7
10	3			8

ビンソート：ソート結果の求め方 (7)

データ：1〜8

	入力	データ i の個数	データ i 以下の個数 $count[i]$	ソート結果
1	3	2	2 → 1	
2	7	1	2	1
3	1	3	4	2
4	1	0	6	
5	3	0	6	3
6	6	1	6	3
7	8	2	8	6
8	7	1	9	
9	2			7
10	3			8

ビンソート：ソート結果の求め方 (8)

データ：1〜8

	入力	データ i の個数	データ i 以下の個数 $count[i]$	ソート結果
1	3	2	1 → 0	1
2	7	1	2	1
3	1	3	4	2
4	1	0	6	
5	3	0	6	3
6	6	1	6	3
7	8	2	8	6
8	7	1	9	
9	2			7
10	3			8

ビンソート：ソート結果の求め方 (9)

データ：1〜8

	入力		データ i の個数		データ i 以下の個数 $count[i]$		ソート結果
1	3	1	2	1	0	1	1
2	7	2	1	2	2	2	1
3	1	3	3	3	4	3	2
4	1	4	0	4	6	4	
5	3	5	0	5	6	5	3
6	6	6	1	6	6	6	3
7	8	7	2	7	8 → 7	7	6
8	7	8	1	8	9	8	7
9	2					9	7
10	3					10	8

ビンソート：ソート結果の求め方 (10)

データ：1〜8

	入力		データ i の個数		データ i 以下の個数 $count[i]$		ソート結果
1	3	1	2	1	0	1	1
2	7	2	1	2	2	2	1
3	1	3	3	3	4	3	2
4	1	4	0	4	6	4	3
5	3	5	0	5	6	5	3
6	6	6	1	6	6	6	3
7	8	7	2	7	8 → 7	7	6
8	7	8	1	8	9	8	7
9	2					9	7
10	3					10	8

ビンソートの時間計算量

- ビンソート（バケツソート）
  - 最大時間計算量 $O(n)$
  - 基本的アイデア
    - データを 1〜m の整数に制限
    - 値 i 以下のデータの個数を $count[i]$ に求める
    - 値 i のデータは
      $count[i-1]+1$〜$count[i]$ 番目のデータ
  - 時間計算量
    - $O(n+m)$
    - m が定数なら，$O(n)$

ビンソートの時間計算量

- ビンソート（バケツソート）
  - 最大時間計算量 $O(n)$
  - 基本的アイデア
    - データを 1〜m の整数に制限
    - $count[i]$ の代わりにリストを使用
      - $count[i]$：値 i 以下のデータの数
  - 時間計算量
    - $O(n+m)$
    - m が定数なら，$O(n)$

リストを用いたビンソート

データ：1〜8

	入力	ソート結果
0	3	1
1	7	1
2	1	2
3	1	3
4	3	3
5	6	3
6	8	6
7	7	7
8	2	7
9	3	8

	データ i のリスト
1	1 → 1
2	2
3	3 → 3 → 3
4	
5	
6	6
7	7 → 7
8	8

基底法

- ビンソート
  - データが 1〜m の整数のとき，
    - 時間計算量 $O(n+m)$
    - サイズ m の配列を使用

    → m が大きい（2^{32}）とき，非現実的

⇩

基底法

安定な整列

- ビンソート：安定な整列
  - 同じデータ（キー）は入力の順序を保存

データ：1〜8

	入力	ソート結果
0	3a	1c
1	7b	1d
2	1c	2i
3	1d	3a
4	3e	3e
5	6f	3j
6	8g	6f
7	7h	7b
8	2i	7h
9	3j	8g

	データ i のリスト
1	1c → 1d
2	2i
3	3a → 3e → 3j
4	
5	
6	6f
7	7b → 7h
8	8g

基底法

- データ　k 桁の m 進数（0〜m^k-1）

  32ビット正整数 ＝ 8桁の16進数（4ビット）
- 下位の桁から順にビンソートで整列
  - ビンソート：安定な整列

⇩

上位桁が同じ値なら下位桁の結果を保存

基底法

データ：0～999

	入力	第 1 桁	第 2 桁	ソート結果 第 3 桁
1	124	431	211	124
2	431	321	321	129
3	256	211	124	132
4	132	351	324	211
5	321	132	129	229
6	129	124	229	256
7	324	324	431	321
8	211	256	132	324
9	351	129	351	351
10	229	229	256	431

基底法の時間計算量

- データ　k 桁の m 進数（0～m^k-1）
  32ビット正整数 ＝ 8桁の16進数（4ビット）
- 下位の桁から順にビンソートで整列
  - ビンソート：安定な整列
- 時間計算量
  $O((n+m)k)$
  $m,\ k$ が定数なら $O(n)$

まとめ　第５章　データの整列

＊　テキストにない内容

5.1 バブルソート
5.2 セレクションソート
5.3 インサーションソート
5.4 シェルソート
5.5 ヒープソート
5.6 クイックソート
＊　サンプルソート
5.7 マージソート
5.8 ソート問題の計算複雑度
＊　ビンソート，基底法

この章の学習目標

- さまざまな整列アルゴリズムとその計算時間を例を用いて説明できる
  - バブルソート，セレクションソート，インサーションソート，シェルソート，ヒープソート，クイックソート，サンプルソート，マージソート，ビンソート，基底法
- 整列問題の時間計算量の下界について，理由とともに説明できる
- 整列アルゴリズムのプログラムを書ける