

ООР復習：整列

モデリングとシミュレーション

特論

1

2019年度

只木進一

整列(sort)の方法

- ▶ n^2 回の比較が必要なタイプ
 - ▶ bubble sort, selection sort, insertion sort
- ▶ $n \log n$ 回の比較が必要なタイプ
 - ▶ merge-sort, quick sort
- ▶ 実際に計測する

bubble sort

```
n : dのサイズ
for (i = n; i > 0; i --) {
  for (j = 0; j < i - 1; j ++ ) {
    if (dj+1 < dj) {
      dj+1とdjを入れ替え
    }
  }
}
```

二重ループで要素を比較

insertion sort

```
 $n : d$ のサイズ  
for ( $i = 1; i < n; i++$ ) {  
     $k =$  要素 $i$ より小さい要素の位置  
    if ( $k \neq i$ ) {  
        位置 $i$ の前に要素 $k$ を挿入  
    }  
}
```

小さい要素を探すのが内側ループに相当

selection sort

```
 $n : d$ のサイズ  
for ( $i = 0; i < n - 1; i++$ ) {  
     $m =$  位置 $i$ から最後までで最小の要素の位置  
    if ( $m \neq i$ ) {  
        要素 $i$ と要素 $m$ を入れ替え  
    }  
}
```

小さい要素を探すのが内側ループに相当

Merge Sort

リストの分離

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

3	8	5	2
---	---	---	---

7	6	1	4
---	---	---	---

3	8	5	2
---	---	---	---

7	6
---	---

1	4
---	---

3	8	5	2
---	---	---	---

7	6	1	4
---	---	---	---

Merge Sort

リストの結合

3	8	5	2
---	---	---	---

7	6	1	4
---	---	---	---

3	8	2	5
---	---	---	---

6	7	1	4
---	---	---	---

2	3	5	8
---	---	---	---

1	4	6	7
---	---	---	---

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

- 簡単のため、リスト長を $n = 2^m$ とする。
- 分割の回数は $\log_2 n = m$
- 各階層でのリストへの追加は n 回。
- 分割時の工数 : $n \log_2 n$

サンプルプログラムの取得

➡ “Teams” -> “Clone”

Clone Repository

Steps

1. **Remote Repository**
2. Remote Branches
3. Destination Directory

Remote Repository

Specify Git Repository Location:

Repository URL:
http[s]://host.xz[:port]/path/to/repo.git/

User: (leave blank for anonymous access)

Password: Save Password

Specify Destination Folder:

Clone into:

(Leave empty to specify the destination later)

< Back Next > Finish Cancel Help

- 二つのリポジトリからサンプルコードを取得
 - <https://github.com/modeling-and-simulation-mc-saga/ListSort>
 - <https://github.com/modeling-and-simulation-mc-saga/MyLib>

Object Oriented Programming

- ▶ 例として整列プログラムを考える
- ▶ 何ができるべきか
 - ▶ 整列対象は、要素間の大小関係が分かれば十分
 - ▶ 整列作業に最小限必要な機能
 - ▶ 比較、入れ替え

Object Oriented Programming

■ 整列対象

- 大小関係が定義できれば良い

- `Comparable`を利用

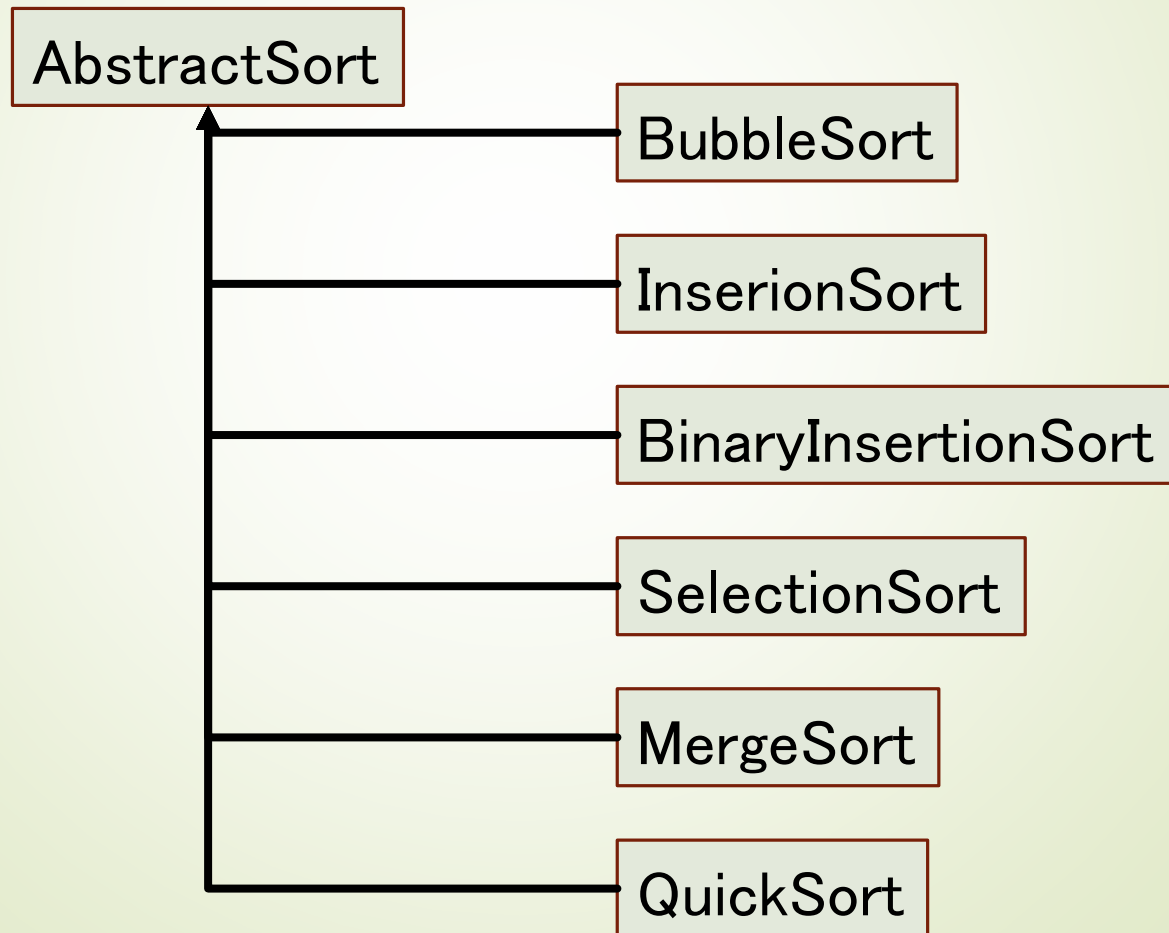
- 他はどうなっている関係無い

- `public class Data implements Comparable<Data>`

- `compareTo()`を実装

■ 整列操作

- `public abstract class AbstractSort<T extends Comparable<T>>`
- 具体的な整列法は、すべて `AbstractSort` のインスタンスに見える
- 整列操作に共通的なメソッド
- 比較回数を計測できる



シミュレーション

- 要素数を n とすると
 - 泡立ち法などは、比較が n^2 回
 - QuickSortなどは、比較が $n \log n$
- 実際に計測

