Introduction

モデル化とシミュレーション特論
2023 年度前期
佐賀大学理工学研究科　只木進一

The purpose of this lecture

- Introducing fundamental methods for simulations
- Improving skills in Object-Oriented-Programming

Examples

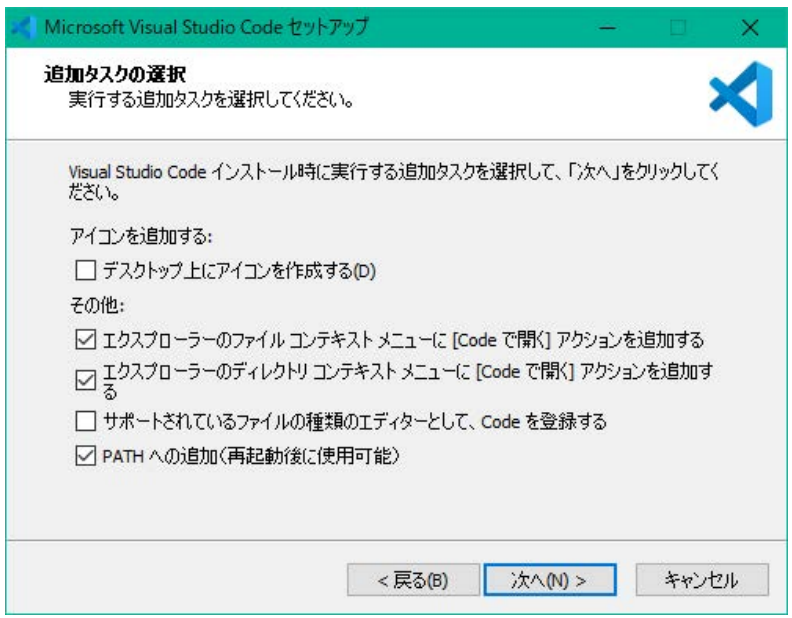
- Differential equations and their numerical solutions
- Properties of random numbers : laws of large numbers, random walk, central limiting theorem
- Monte Carlo method
- Cellular automata
- Neural networks
- Fractals
- Chaos

Preparing tools : Java 17

- Amazon-Corretto
  `https://aws.amazon.com/jp/corretto/`
- Select an adequate installer depending on your platform.
- The installers with the default settings install the application into `C:\Program Files\Amazon Corretto`. You do not need to change the settings.

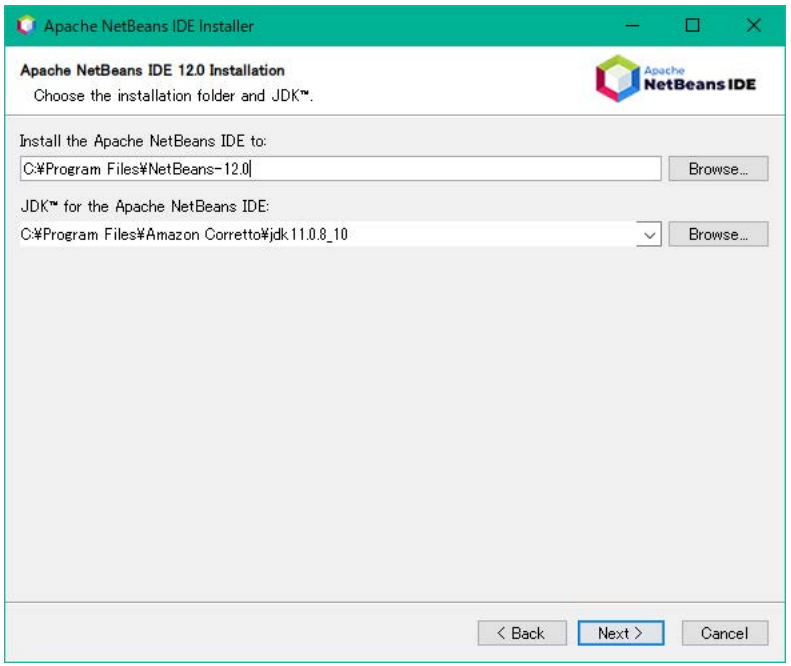
Text editors

- Do you have a good text editor in your PC?
- Visual Studio Code
  `https://azure.microsoft.com/ja-jp/products/visual-studio-code/`
- Do not use the installer downloaded from *Microsoft Store*.
- Important point in installation
  - Check two checkboxes of「Code で開く」 in your context menu.



NetBeans

- Apache NetBeans
  `https://netbeans.apache.org/download/nb17/`
- Check that `Amazon Corretto` installed being assigned to be jdk



- You may use your favorite IDE, such as Eclipse, IntelliJ, etc.

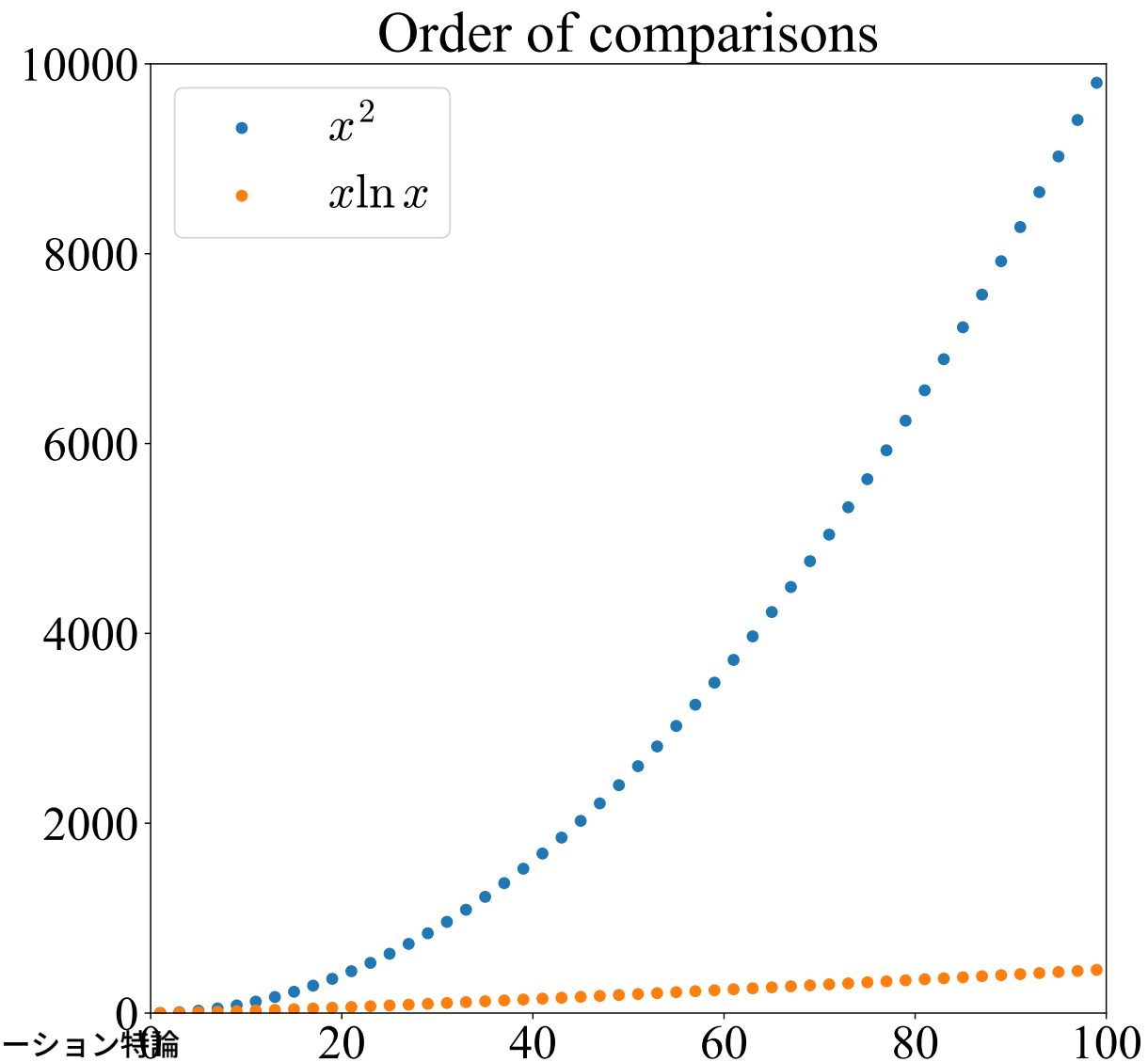
Reviewing OOP

- Defining classes
  - Fields and methods
  - Access controls: `private`, `protected`, and `public`
  - `static` and `final`
- Class inheritance
- Abstract Classes
- Sorting as an example

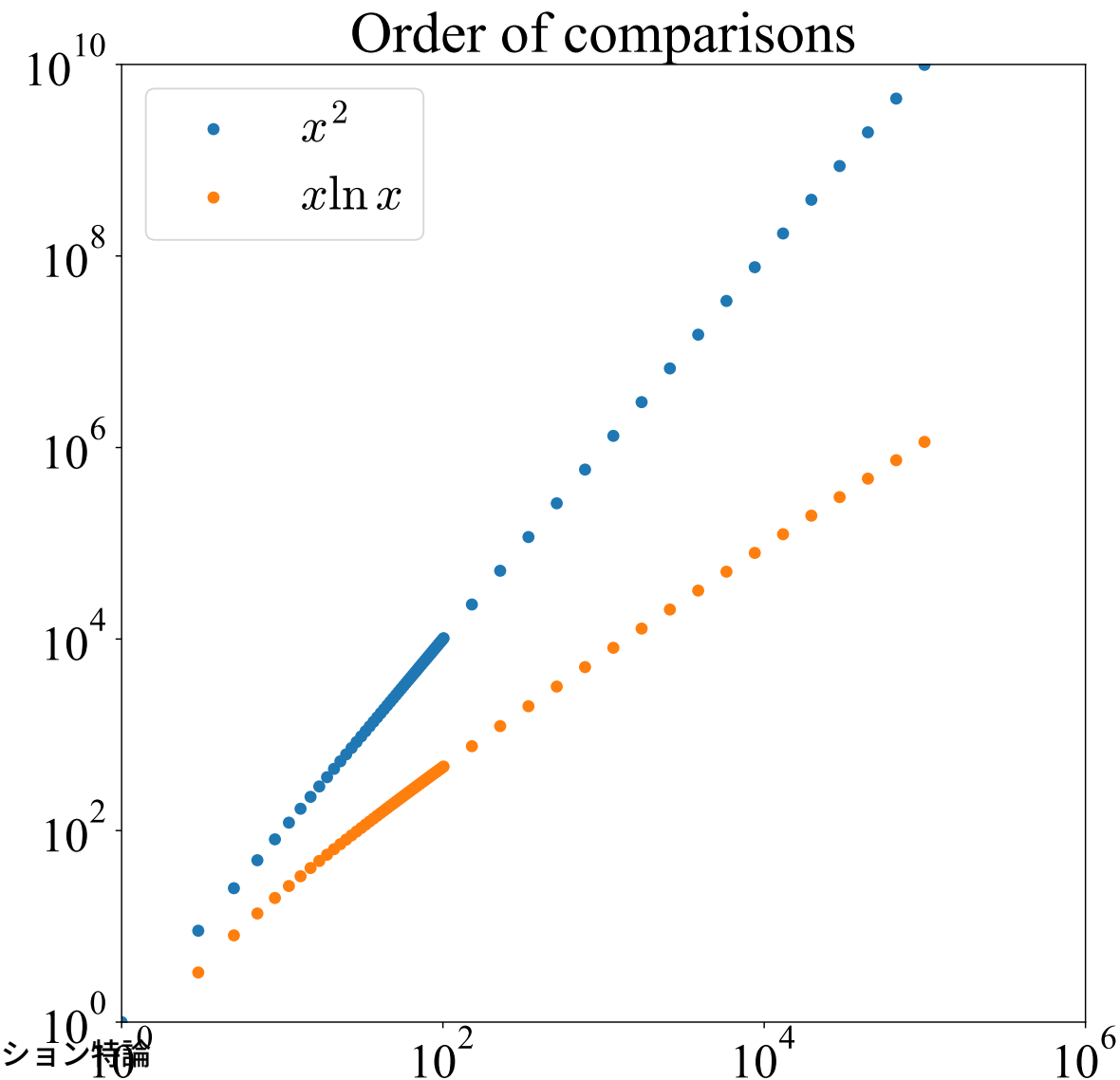
Various sort methods

- Sort method with n^2 comparisons
  - bubble sort, selection sort, insertion sort
- Sort method with $n \log n$ comparisons
  - merge sort, quick sort
- $n^2 \gg n \log n$ for $n \gg 1$
- Observe the number of comparisons for various sorting methods

Order of comparisons



Order of comparisons in logalismic scales



Bubble sort

Algorithm 1 Bubble sort

n is the size of the array d
for $i = n$; $i > 0$; $i--$ **do**
  for $j = 0$; $j < i - 1$; $j++$ **do**
    if $d_{j+1} < d_j$ **then**
      swap d_{j+1} with d_j
    end if
  end for
end for

Attention: two nested loops require $O(n^2)$ comparisons.

Bubble sort in action

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

the largest element is sorted out at the rightmost by one execution of the inner loop with $n - 1$ comparisons

selection sort

Algorithm 2 selection sort

n is the size of the array d
for $i = 0$; $i < n - 1$; $i++$ **do**
    m is the position of the smallest element between i and the last
    if $m \neq i$ **then**
        swap the element at i with that at m
    end if
end for

Attention : searching the smallest element is in the inner loop and requires $O(n)$ comparisons.

insertion sort

Algorithm 3 insertion sort

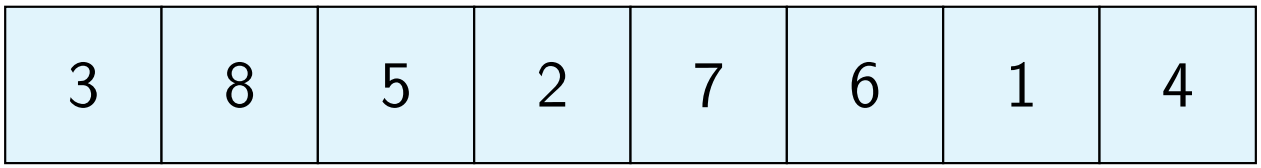
n is the size of the array d
for $i = 0$; $i < n$; $i++$ **do**
    m is the index of the smallest element between i and the last
    if $m \neq i$ **then**
        insert element at m into i
    end if
end for

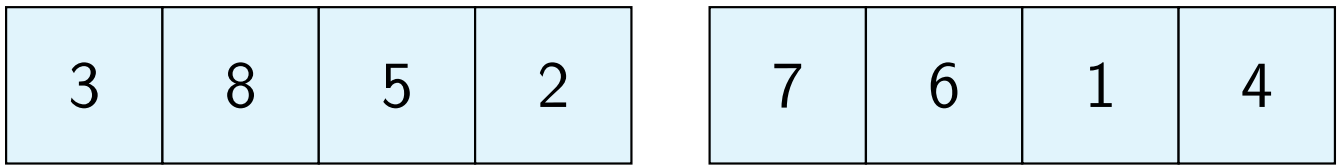
Attention : searching the smallest element is the inner loop and requires $O(n)$ comparisons.

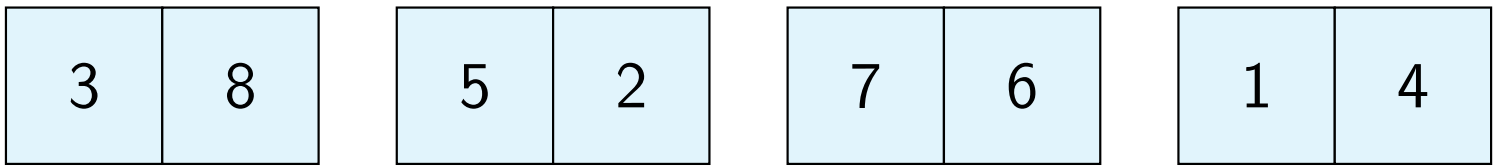
Merge sort

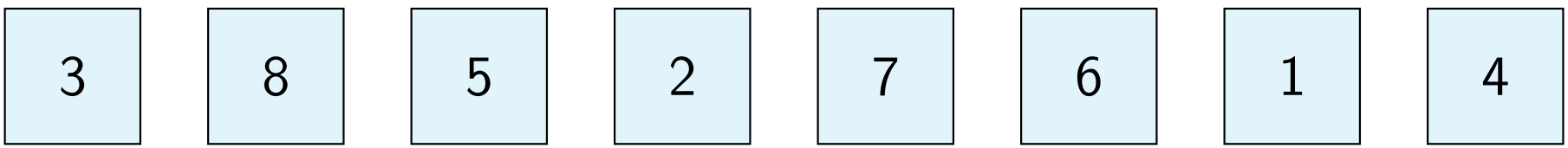
- Dividing list into the smallest size
  - Needs $O(\ln n)$ steps
- Merging sorted lists
  - Merging two sorted lists with n elements requires $O(n)$ comparisons

Merge sort: dividing list into the smallest size

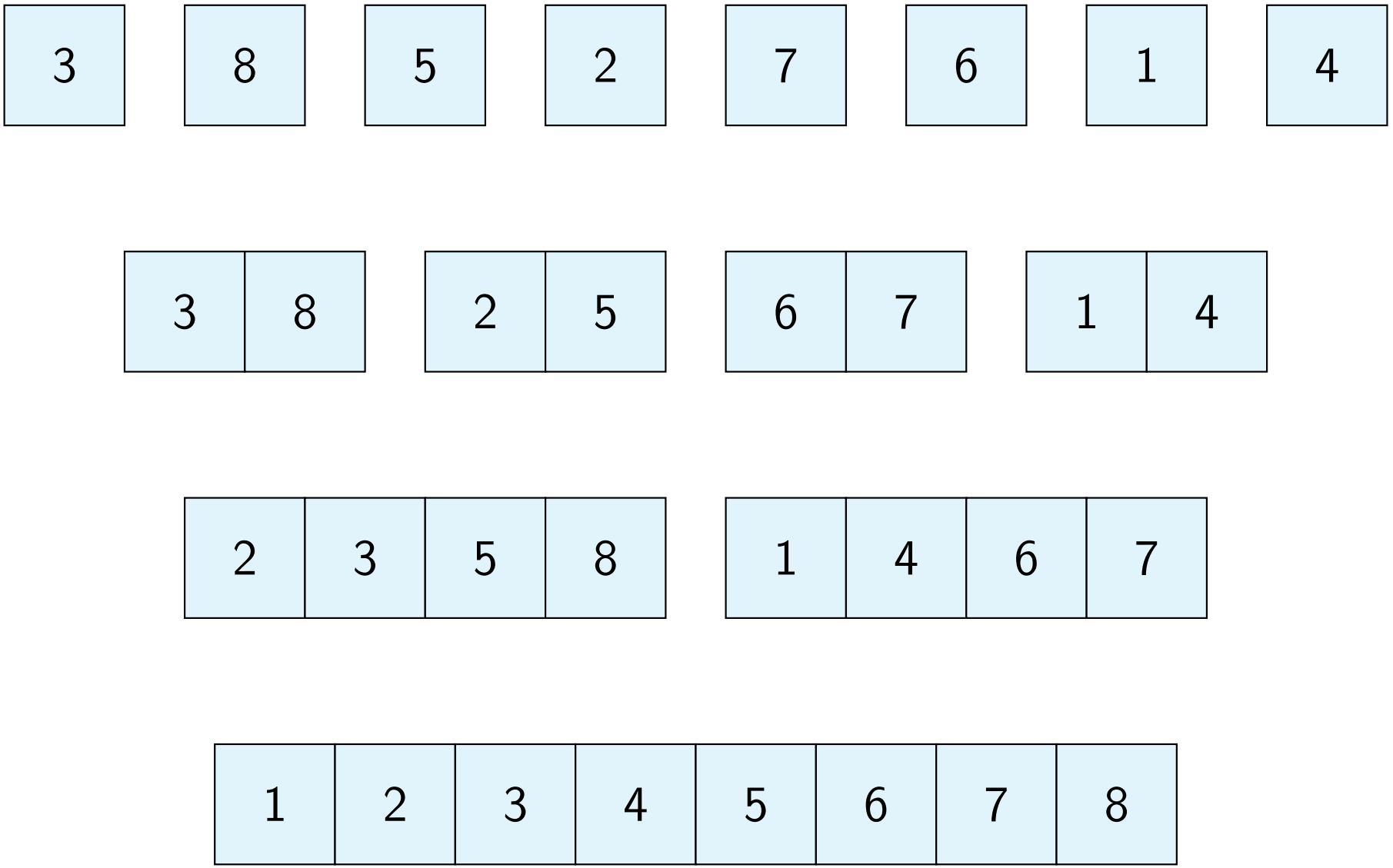








Merge sort: merging lists



merge sort

Algorithm 4 merge sort

n: the size of the array d
$k_{\text{left}} = 0,\ k_{\text{right}} = n$
procedure $\textsc{sortSub}(k_{\text{left}}, k_{\text{right}})$
    $k_{\text{middle}} = (k_{\text{left}} + k_{\text{right}})/2$ (truncate to integer)
    $\textsc{sortSub}(k_{\text{left}}, k_{\text{middle}})$
    $\textsc{sortSub}(k_{\text{middle}}, k_{\text{right}})$
    Combining two sorted lists
end procedure

- Merging operations in horizontal direction needs $O(n)$ comparisons
- Number of layers in vertical direction is $O(\log n)$

Quick sort

- Select one element called *pivot*.
- Divide the list into two parts: a list with smaller elements than the pivot and the remaining.
  - Dividing a list requires $O(n)$ steps
  - The number of division is $O(\ln n)$

Quick sort

Algorithm 5 quick sort

n: the size of the array d

$k_{\mathsf{left}} = 0, k_{\mathsf{right}} = n$

procedure $\textsc{sortSub}(k_{\mathsf{left}}, k_{\mathsf{right}})$

  $k_{\mathsf{middle}} = \textsc{partition}(k_{\mathsf{left}}, k_{\mathsf{right}})$

  $\textsc{sortSub}(k_{\mathsf{left}}, k_{\mathsf{middle}})$

  $\textsc{sortSub}(k_{\mathsf{middle}}, k_{\mathsf{right}})$

end procedure

Quick sort : continued

Algorithm 6 partition

procedure PARTITION(k, ℓ)

$v = d_{\ell-1}$ ▷ pivot

$i = k, j = \ell - 1$

while $i < j$ **do**

Search an element greater than or equal v from the left. Its position is i.

Search an element less than or equal v from the right. Its position is j.

if $i < j$ **then**

Swap d_i with d_j

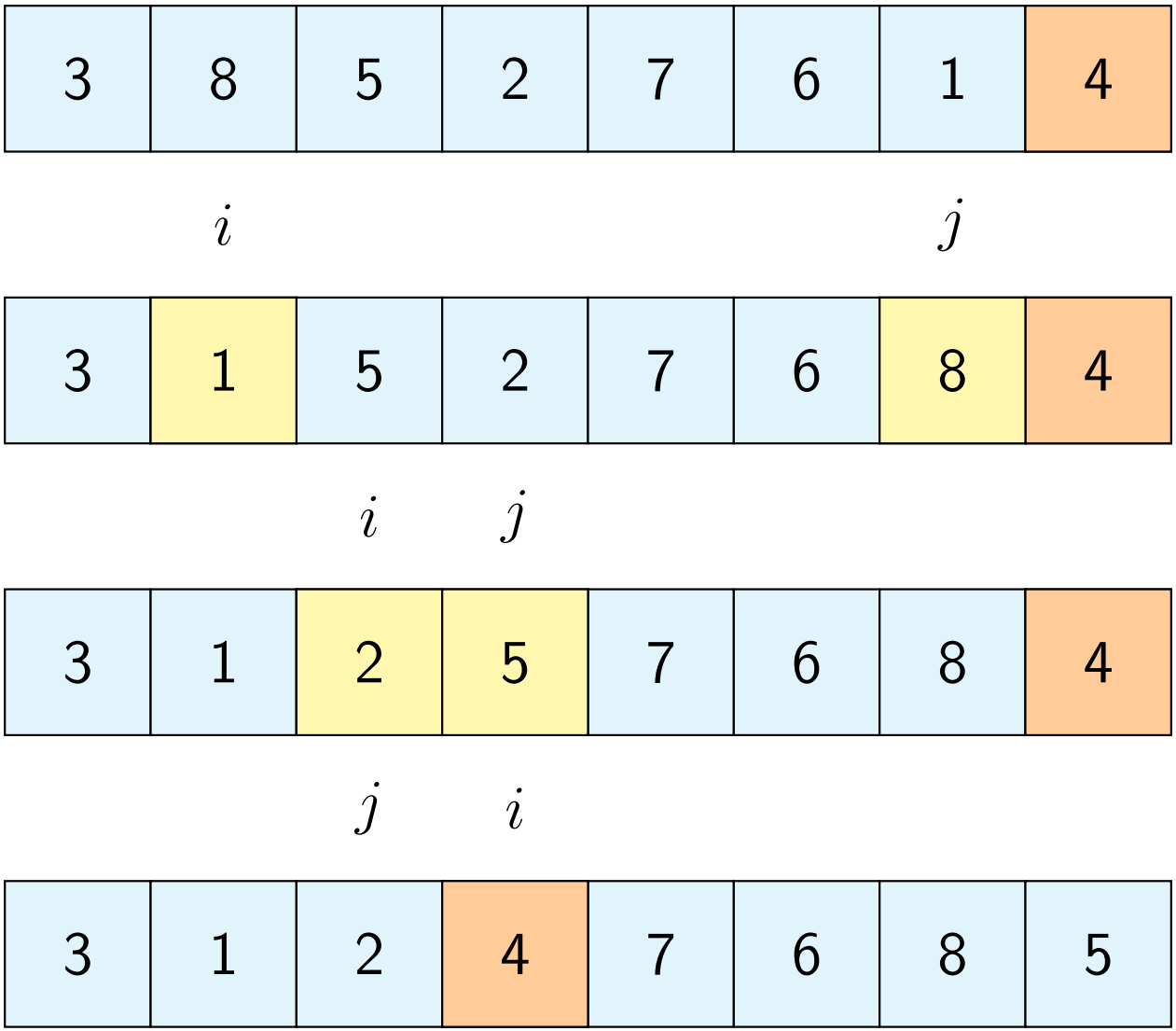
end if

end while

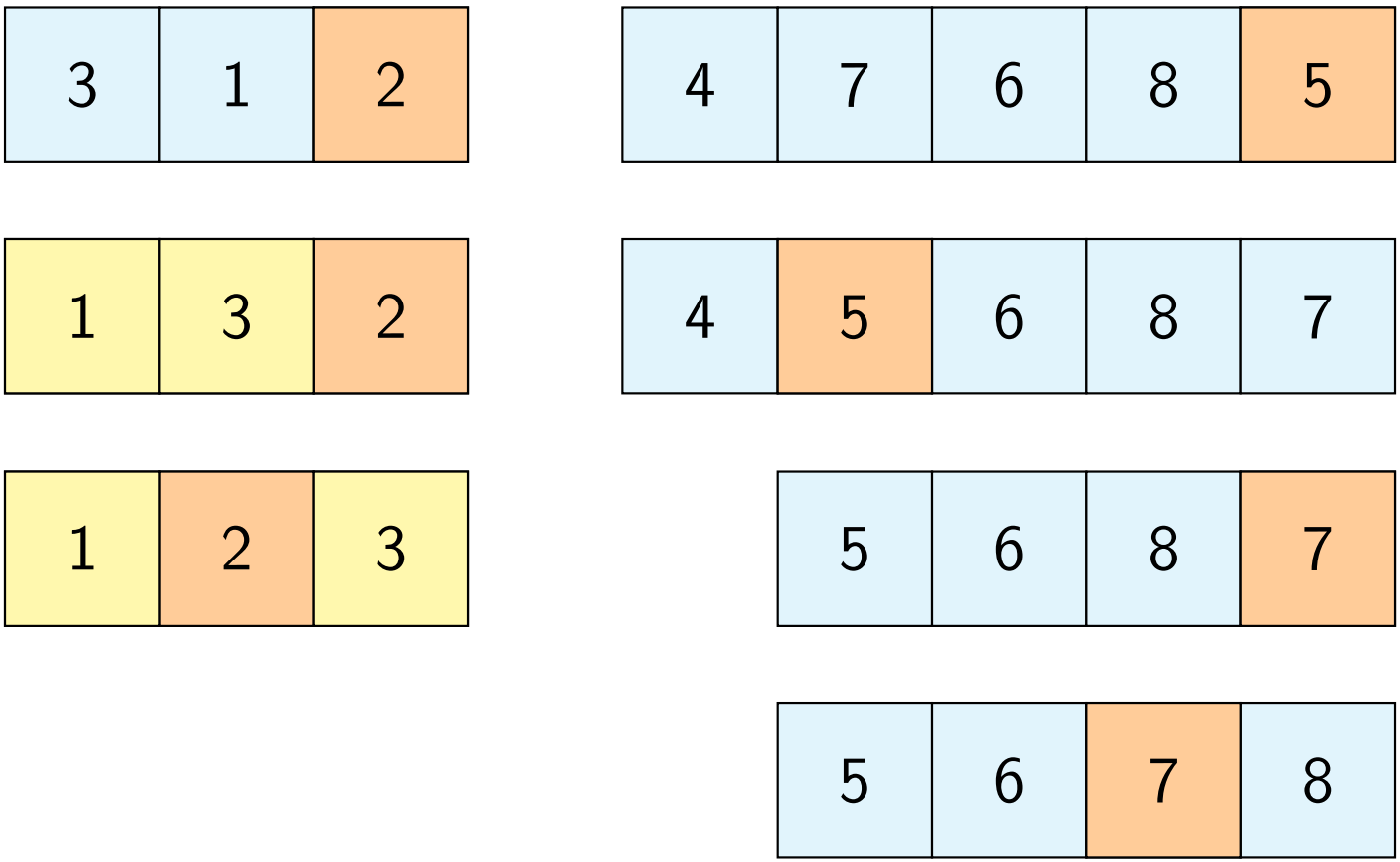
Swap d_i with $d_{\ell-1}$

Return i

Example: quick sort

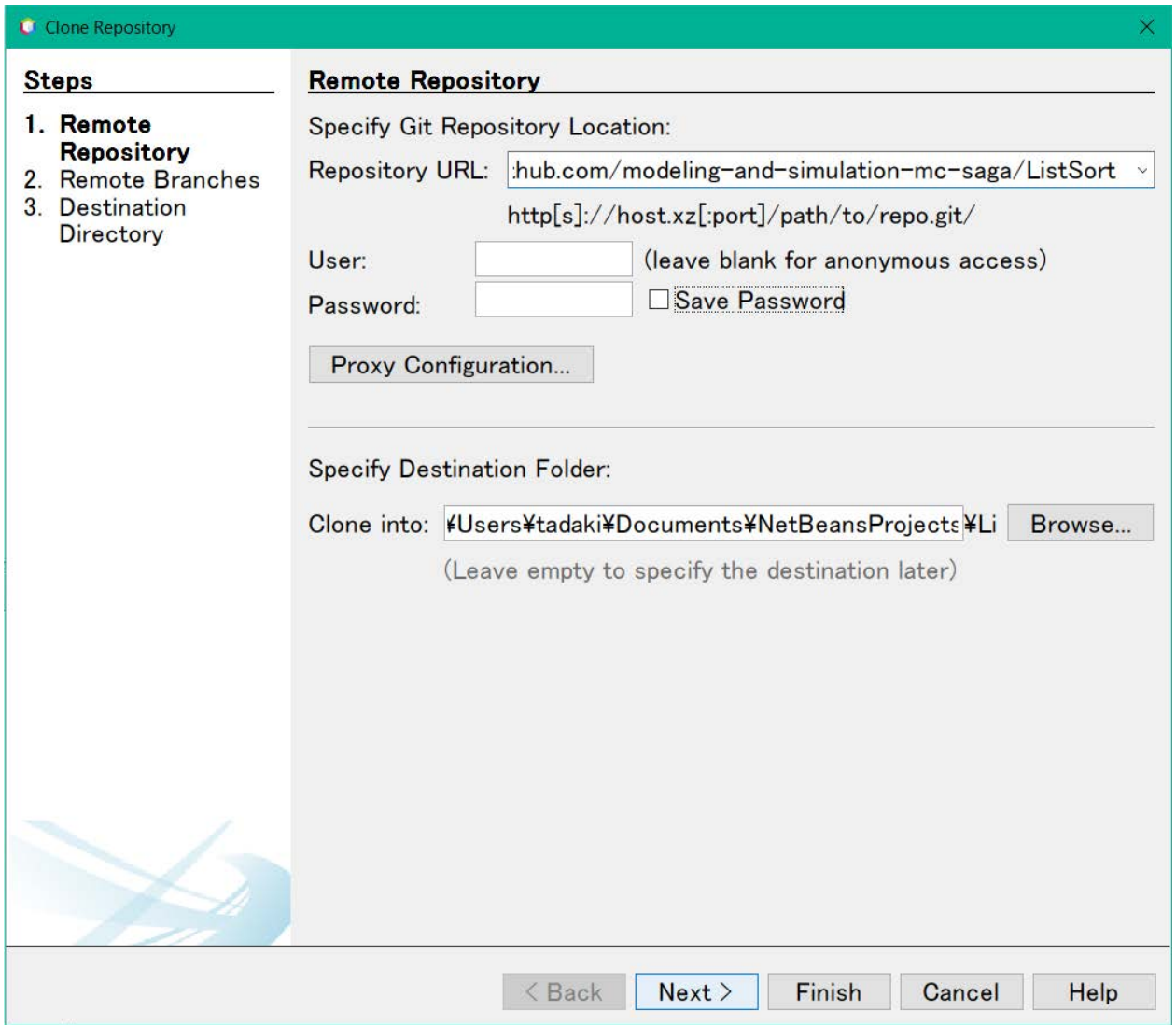


Example: quick sort, continued



Get Sample Programs by NetBeans

"Teams" → "Git" → "Clone"



Get sample programs by Git command

- Obtain Git from `https://git-scm.com/downloads`
- Use command

  `git clone repository`

Repository

- https://github.com/modeling-and-simulation-mc-saga/ListSort
- python library for drawing graphs
  https://github.com/modeling-and-simulation-mc-saga/lib

Review sorting in the viewpoint of OOP

- Minimum common functions for sorting
  - Target objects are required to have large-and-small relationship
  - Minimum functions for sorting : compare and swap
- `Comparable` interface for target objects
  - Interface: special purpose abstract classes
    - Only defining abstract methods and constants
  - Classes with `Comparable` interface
    - Method `compareTo()` defines how to compare with another instance.
  - See `Data` class

Extended for loops

```
1  /*
2  for (int i = 0 ; i < list.size() ; i++) {
3      T t = list.get(i);
4      do something
5  }
6  */
7  for (T t : list){
8      do something
9  }
```

Lambda expressions and `List`

- Lambda expression: anonymous implementation of interfaces
- `Consumer` interface: an operation accepting a single input and returning nothing.

```
1  List<T> list;
2  list.forEach(t -> {do something});
3  /*
4  for (T t: list){
5      do something
6  }
7  */
```

Inheritance

- Subclasses inherit fields and methods of their superclass.
  - Private fields and methods are not accessible directly.
- Subclasses extending their superclass by adding fields and methods, or overriding them.
- `abstract` methods must be implemented.

AbstractSort

- Sorting objects which implement `Comparable` interface
- Not implementing concrete sorting methods
- Implementing common methods required for sorting
- Function for counting comparisons
- Derived classes
  BubbleSort, InsertionSort, SelectionSort, MergeSort, QuickSort

Simulation results

- n: the number of elements
  - bubble sort and etc. need $O\left(n^2\right)$ comparisons
  - merge sort and etc. need $O\left(n \log n\right)$ comparisons