Applied programming and numerical analysis

Lecture 2: Class and sort

Abstract

- Object and class
- Sorting algorithm

3.10 Object and class

- Objects are core things that Python program can manipulate.
- They contains data and processes related to the data.
- Python is designed as an object oriented programming language from the beginning.
  - Procedural programming language: Fortran, C, etc.
  - Functional programming language: Lisp, scheme, etc.
  - Logical programming language: Prolog
  - Object oriented programming language: Small talk, C++, java, C#, Objective C, python, etc.
- Define a class by `class`
- A class is a blueprint of object.
- Methods are defined as function. But the first parameter should be `self`.
- The name of constructor is `__init__`
- Attribute can be specified by attaching `self.` at the head of variable.
- We can handle various types of data at once.
- We will not discuss about objects in Python in detail. But it is used for sort program

Example

```
import copy # Import library w.r.t. copy
# Define a class
class Student():
    def __init__(self, stdNo, name, score):
        self.name = name
        self.stdNo = stdNo
        self.score = score

    def compareScore(self, stdx):
        if (self.score > stdx.score):
            print("The score of {0} is better than that of {1}". format(self.name, stdx.name))
        else:
            print("The score of {0} is not better than that of {1}". format(self.name, stdx.name))

    def printStd(self):
        print("{0:>5}  {1:<20}  {2:>4}".format(self.stdNo, self.name, self.score))

std1 = Student("Kirito", 11, 92)
std2 = Student("Asuka", 12, 96)



std1.compareScore(std2)
std2.compareScore(std1)

std3 = std2
std3.printStd()
```

```
std3.score = 30
std2.printStd()

std1.printStd()
std4 = copy.copy(std1)
std4.score = 30
std1.printStd()
```

3.11 Handle lists

- append()
- insert()
- extend()
- +
- in()
- remove()
- len()

```
# append
list1 = [1, 2]
list1.append(4)
print(list1)
list1.append([2, 3, 5])
print(list1)
print(list1[3])
print(list1[3][2])
list1.append("TSE")
print(list1)
```

```
# insert
list1.insert(2, 100)
print(list1)
list1.insert(-2, 200)
print(list1)

# +
list2 = [6, 7, 8]
list3 = [10, 11]
list4 = list2 + list3
print(list4)

# extend()
list2.extend(list3)
print(list2)

# in
print(6 in list2)
print(3 in list2)
```

```
# remove()
list2.remove(10)
print(list2)

# len()
print(len(list1))
```

3.12 Sorting algorithm

Purpose

- Reorder the data in ascending order or in descending order.
- When a data consists of several items, we decide key items in it. Data with other items are simultaneously reordered according to the values of key data.
  For example, a data for a student consists of student number, name, and score. The data may be sorted according to score.

Well known sorting algorithm

- Stupid sort
- Selection sort
- Bubble sort
- Quick sort
- Merge sort
- Heap sort

We introduce sorting algorithm in ascending order.
The data to be sorted are in a list of N integers (From `A[0]` to `A[N-1]`).

3.12.1 Stupid sort

Algorithm

1. Reorder the list in all orders. (N ! orders)
2. Choose a list that is in ascending order.

Computational complexity

Order of N !

3.12.2 Selection sort

Algorithm

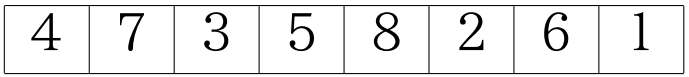
1. Let `l = 0`.
2. From `A[l]` to `A[N-1]`, we search the minimum element. (Assume that it is `A[m]`.)
3. Exchange `A[l]` and `A[m]`.
4. Let `l = l + 1`.
5. If `l == N - 1`, we stop this algorithm. Otherwise go to 2.

Computational complexity

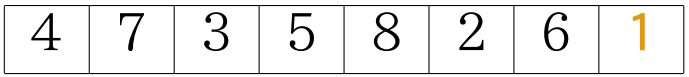
Order of N^2.

Example of process of selection sort.

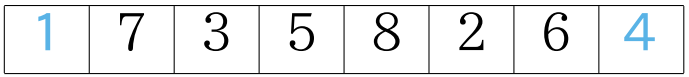
- List to be sorted:



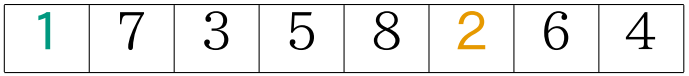
- Select the minimum element. (`l = 0`)



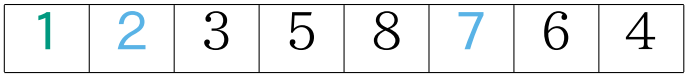
- Exchange the first and the minimum elements.



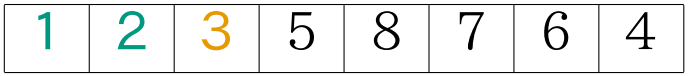
- Select the minimum element from the second to the last. (`l = 1`)



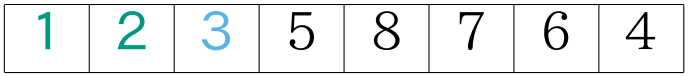
- Exchange the second and the minimum elements.



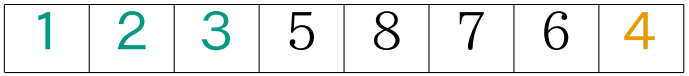
- Select the minimum element from the third to the last. (`l = 2`)



- Exchange the third and the minimum elements.



- Select the minimum element from the fourth to the last. (`l = 3`)



- Exchange the fourth and the minimum elements.

- Select the minimum element from the fifth to the last. (`l = 4`)

- Exchange the fifth and the minimum elements.

- Select the minimum element from the sixth to the last. (`l = 5`)

- Exchange the sixth and the minimum elements.

- Select the minimum element from the seventh to the last. (`l = 6`)

- Exchange the seventh and the minimum elements.

3.12.3 Program of selection sort

- Reorder a list of integer in ascending order.

- Practice: Make a program to reorder a list of integers in descending order.

3.12.4 Bubble sort

1. Let `l = 0` and `m = 1`.
2. If `l` is negative, let `l = m` and `m = m + 1`.
3. If `l == N - 1`, stop the program. Otherwise go to 4.
4. If `A[l] <= A[l + 1]`, let `l = m` and `m = m + 1`.
   Otherwise
   - Exchange `A[l]` and `A[l + 1]`.
   - Let `l = l - 1`.
5. Go to 2.

Computational complexity

Order of N^2.

- Given list. : (`l = 0` and `m = 1`)

4	7	3	5	8	2	6	1

- Compare the first and the second elements. (`l = 0` and `m = 1`)

4	7	3	5	8	2	6	1

- Since the order is correct, compare the second and the third elements. (`l = 1` and `m = 2`)

4	7	3	5	8	2	6	1

- Because the order is opposite, exchange them. (`l = 0` and `m = 2` hold.)

4	3	7	5	8	2	6	1

- Compare the first and the second elements. (`l = 0` and `m = 2`)

4	3	7	5	8	2	6	1

- Because the order is opposite, exchange them. (`l = -1` and `m = 2` hold.)

3	4	7	5	8	2	6	1

- Because it has reached to the first (`l = -1`), go to the next element (`l = 2`) of the element (`l = 1`) that we started exchanging, and compare the third and the fourth elements. (`l = 2` and `m = 3`)

3	4	7	5	8	2	6	1

- Because the order is opposite, exchange them. (`l = 1` and `m = 3` hold)

3	4	5	7	8	2	6	1

- Compare the second and the third elements. (`l = 1` and `m = 3`)

3	4	5	7	8	2	6	1

- Because the order is correct, go to the next element (`l = 3`) of the element (`l = 2`) that we started exchanging, and compare the fourth and the fifth elements. (`l = 3` and `m = 4`)

3	4	5	7	8	2	6	1

- Since the order is correct, compare the fifith and the sixth elements. (`l = 4` and `m = 5`)

3	4	5	7	8	2	6	1

- Because the order is opposite, exchange them. (`l = 3` and `m = 5` hold.)

3	4	5	7	2	8	6	1

- Compare the fourth and the fifth elements. (`l = 3` and `m = 5`)

3	4	5	7	2	8	6	1

- Because the order is opposite, exchange them. (`l = 2` and `m = 5` hold.)

3	4	5	2	7	8	6	1

- Compare the third and the fourth elements. (`l = 2` and `m = 5`)

3	4	5	2	7	8	6	1

- Because the order is opposite, exchange them. (`l = 1` and `m = 5` hold.)

3	4	2	5	7	8	6	1

From here, we show only the results.

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

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

3.12.5 Quick sort

Algorithm

1. At first, the whole list is set to be targets for sorting.
2. If the length of list is not larger than 1, `return`.
3. Let a value be a pivot (If the value is the median, the algorithm is the most efficiency.)
4. Extract an element from the list. It it is smaller than the pivot, we put it from the front in the list. If larger, we put it from the back. If equal, we put it from front and from back alternately. (As a result, smaller and larger values are in the front part and in the back part respectively
5. For each part of the two, we perform the process from 2 respectively.
6. End.

The program from 2 to 5 is performed recursively.

Computational complexity

From $N \log N$ to N^2. It depends on selection of pivot .

- In the next example **the first element in the list is chosen as the pivot**
- Given list:

4	7	3	5	8	2	6	1

- Set the pivot to the first element 4.

4		4	7	3	5	8	2	6	1

- For splitting, extract the first element 4.

4	4		7	3	5	8	2	6	1

- Because 4 is equal to 4 and it is the first time, extract the last element 1 and put 4 there.

4	1		7	3	5	8	2	6	4

- Because 1 is smaller than 4, put it in the first position and extract the element 7 in the next.

4	7	1		3	5	8	2	6	4

- Because 7 is larger than 4, extract the second last element 6 and put 7 there.

4	6	1		3	5	8	2	7	4

- Because 6 is larger than 4, extract the third last element 2 and put 6 there.

4	2	1		3	5	8	6	7	4

- Because 2 is smaller than 4, put it in the second position and extract the element 3 in the next. 2

4	3	1	2		5	8	4	7	4

- Because 3 is smaller than 4, put it in the third position and extract the element 5 in the next.

4	5	1	2	3		8	6	7	4

- Because 5 is larger than 4, extract the fourth last element 8 and put 5 there.

4	8	1	2	3		5	6	7	4

- Because 8 is larger than 4, put 8 in the fifth last position.

4		1	2	3	8	5	6	7	4

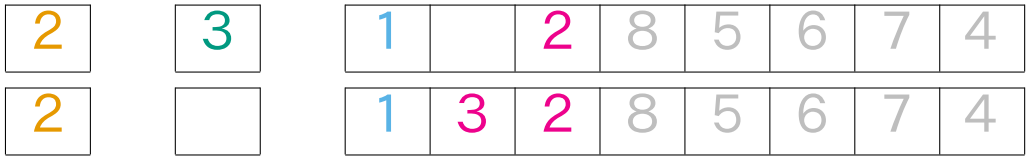
By the splitting, the elements in front and back parts are not larger and not smaller than 4, respectively. We recursively split each part. From here, we show only results.

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

The list has not split.

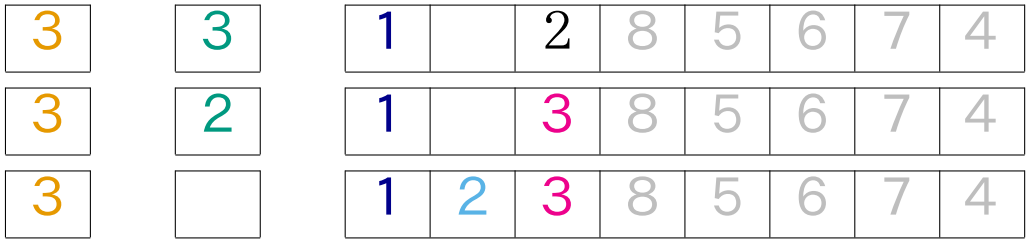
- Split with a pivot 2.

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



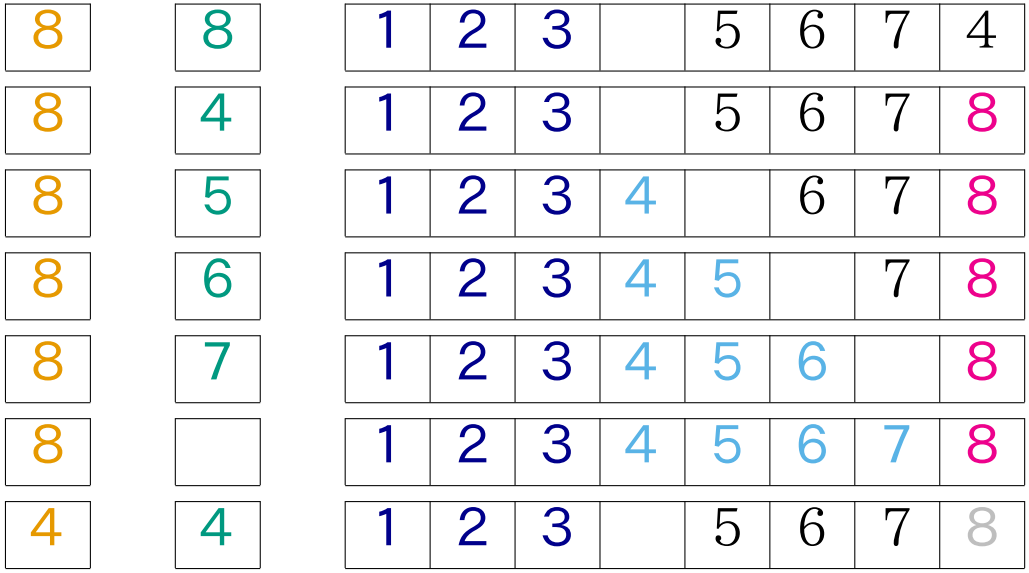
Because the number of element in the front part becomes 1, the splitting for the part has finished.

- Split `[2, 3]`

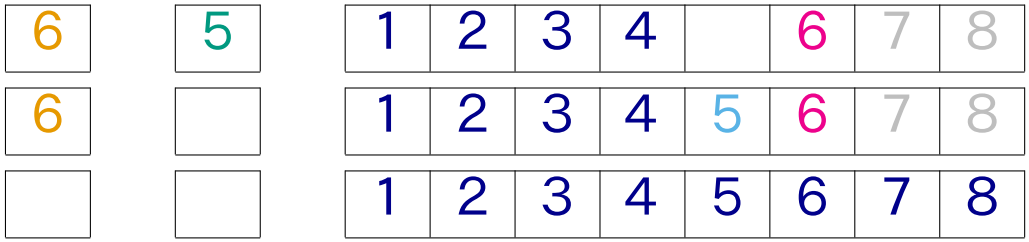


The number of split lists is one, splitting has finished.

- The latter part of the list is split.



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

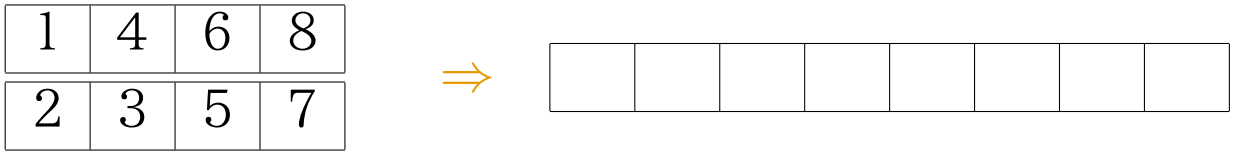


- The process always handles data from the beginning of the list. **Depth-first search**.

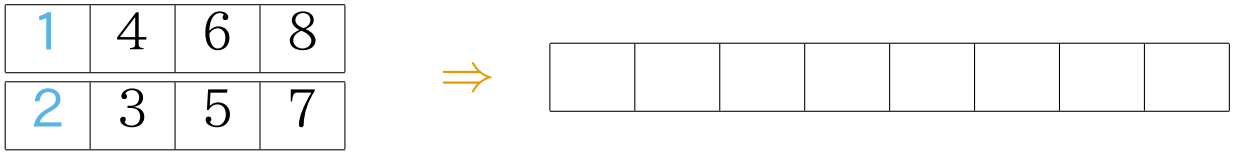
3.12.6 Merge sort

Alogorithm

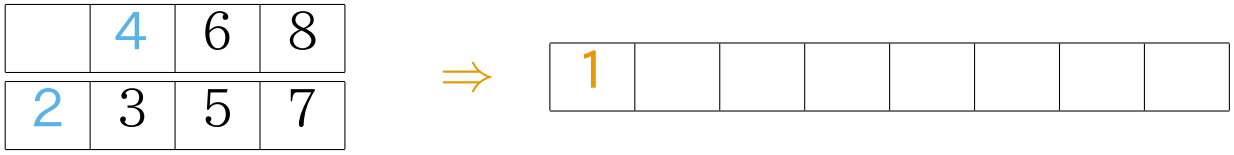
- Let `n = 1`.
- Take two sorted list of length n, and make a sorted list of length $2n$.
- When the number of list is one, we stop the process.
- Computational complexity to merge two sorted lists to a sorted list is the number of elements. **We only handle the two first elements of the two list.**
- Example: The following two list is merged.



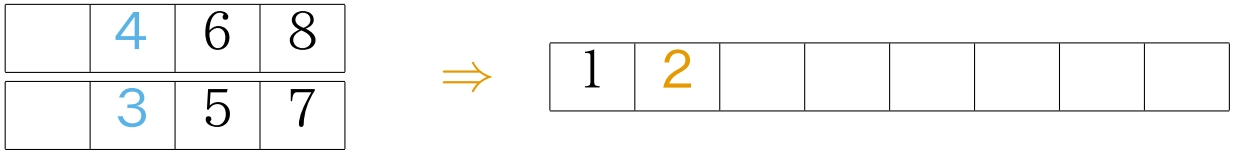
- Compare the two first elements 1 and 2.



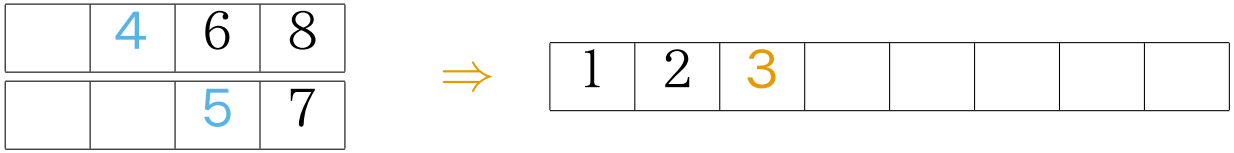
- Because 1 is smaller, put 1 to the output list. And compare 4 and 2.



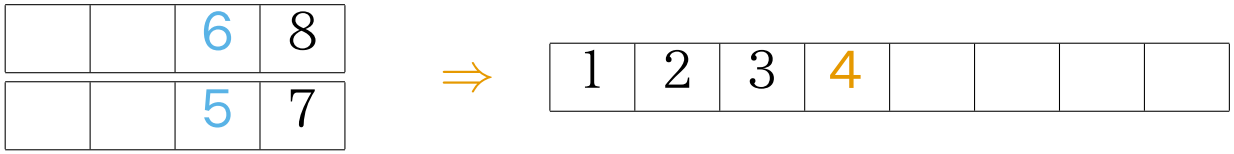
- Because 2 is smaller, put 2 to the output list. And compare 4 and 3.



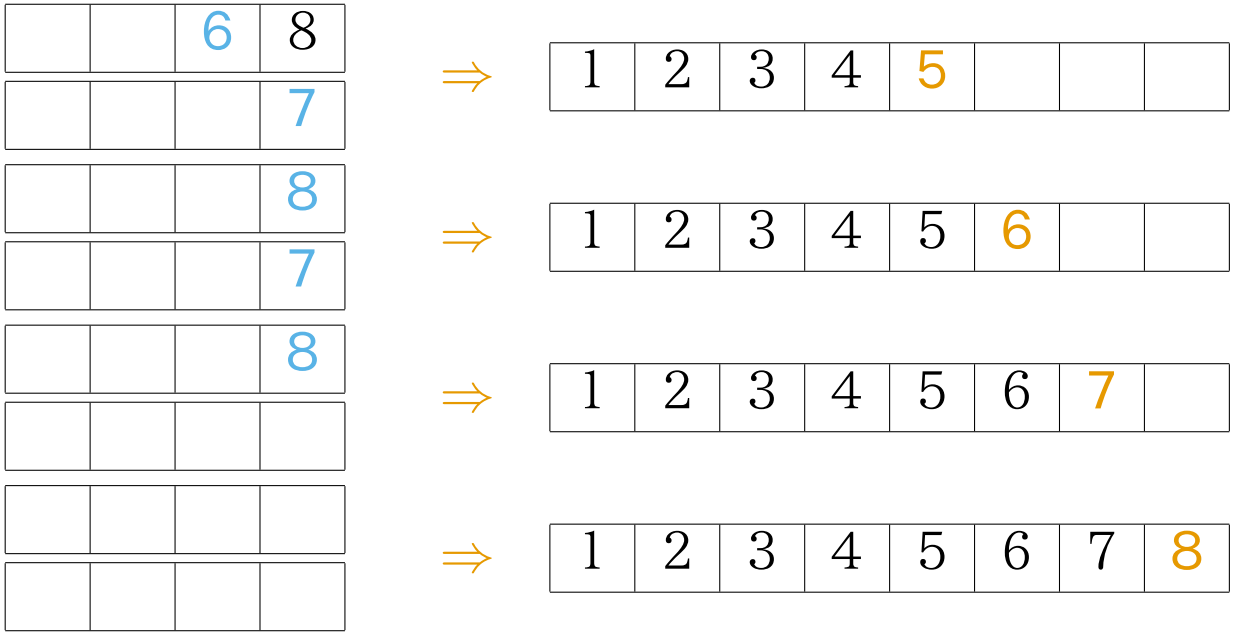
- Because 3 is smaller, put 3 to the output list. And compare 4 and 5.



- Because 4 is smaller, put 4 to the output list. And compare 6 and 5.



From here, we show only results.



Example of merge sort

- Given list:

4	7	3	5	8	2	6	1

- 4 set of pairs of list of which number of element is 1.

4	7	3	5	8	2	6	1

- Execute the sorted merge on each pair.

4	7	3	5	2	8	1	6

- Make pairs with merged lists. (2 pairs are constructed.)

4	7	3	5	2	8	1	6

- Execute the sorted merge on each pair.

3	4	5	7	1	2	6	8

- Make a pair with merged lists. (One pair is constructed.)

3	4	5	7	1	2	6	8

- Execute the sorted merge on the pair.

1	2	3	4	5	6	7	8

Computational complexity

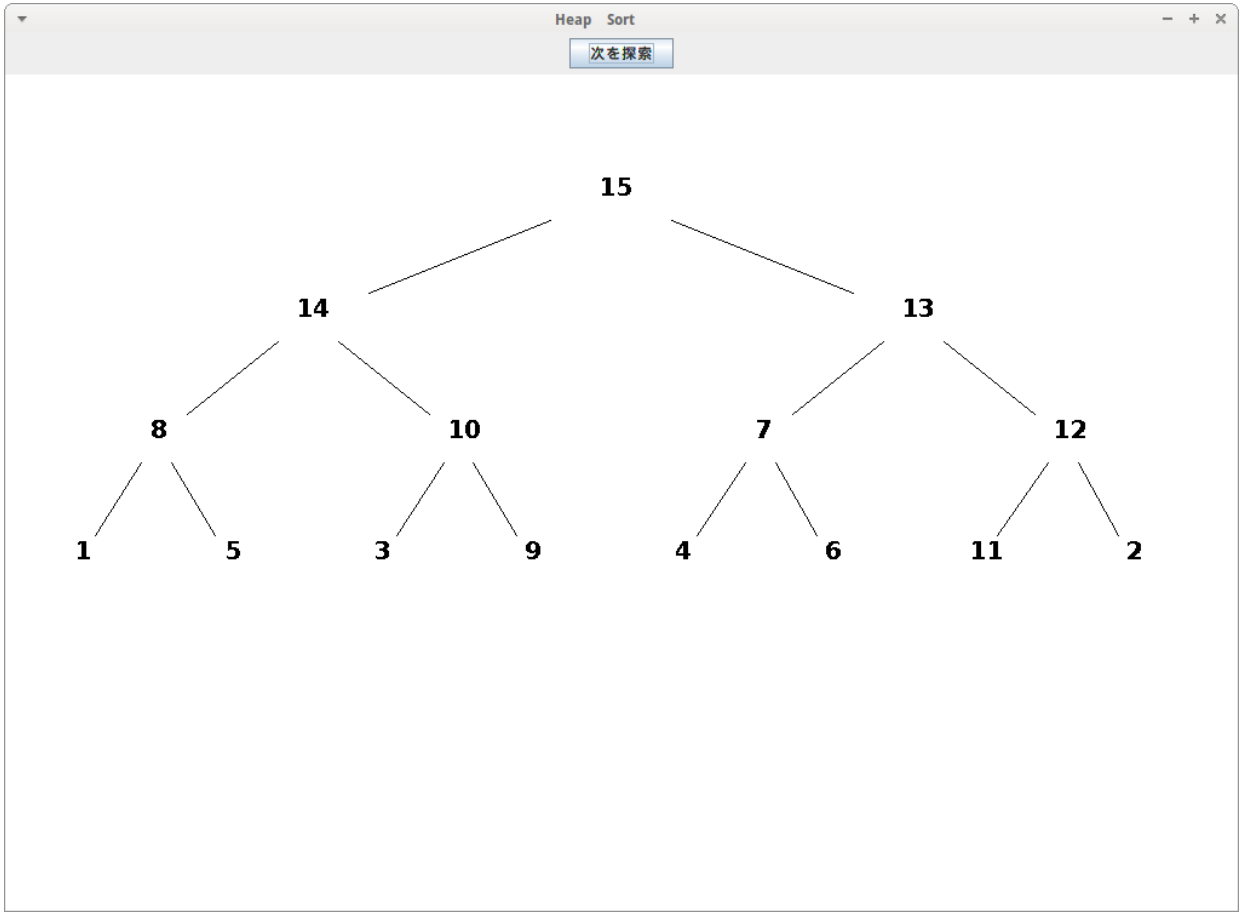
Order of $N \log N$. (Another list of length N is necessary.)

3.12.7 Heap sort

Heap : a tree where the value of each parent is lager than those of children.

Algorithm

- From a tree, we construct a heap. Its process is done from leaf to root.
- Extract root and reconstruct a heap from remained data.



The algorithm is explained by a demonstration.

Computational complexity

Order of $N \log N$.

3.13 Practice

See Lec02S.ipynb

Because time is limited, one third of next lecture, you can you for this practice.

I would like to ask make programs of selection sort and at least anther sort.

3.14 Conclusion

- Object oriented programming:
- Several sorting algorithm: Calculation speed is very different for the same task.