Objectives for this lecture

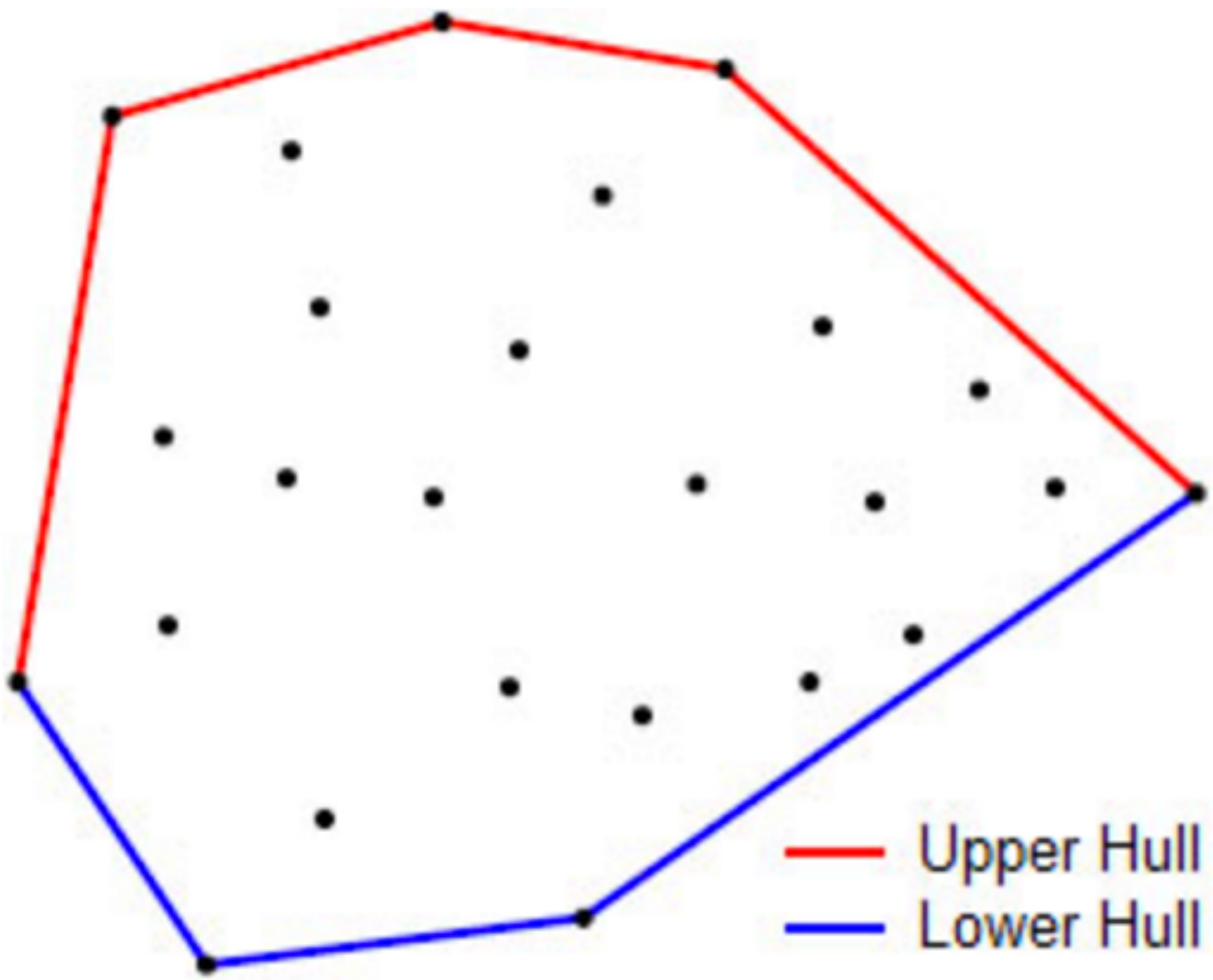
- To understand the basic algorithms:
    - **Bubble** Sort
    - **Selection** Sort
    - **Insertion** Sort

- To implement them in Python

Sorting

Example:

$[6,4,2,1,3,5] \longrightarrow [1,2,3,4,5,6]$



2.2 Graham's scan

The Graham's Algorithm first explicitly sorts the points in O(nlogn) and then applies a linear-time scanning algorithm to finish building the hull.

http://www.tcs.fudan.edu.cn/rudolf/Courses/Algorithms/Alg_cs_07w/Webprojects/Zhaobo_hull/

Sorting Lists

Input:

- A list (not necessarily sorted) of ‘orderable’ element types
- For example, in Python:
  - `the_list = [5,1.5,3,-4.0]` is fine
  - `the_list = [1,'hj',0,'j']` is not
    - Unless you define your own comparison function

Output:

- A list with the same elements as the input list BUT sorted in increasing order.

Bubble Sort

Main idea:
Lighter bubbles rise to the top,
Heavier ones sink to the bottom.

smaller elements “bubble” to the front of the list, larger sink to the end.

Bubble Sort: Python Code

Algorithm BubbleSort(L)
// Sorts a list using bubble sort
// Input: A list of orderable items
// Output: A list sorted in increasing order

```
n ⟵ length(L)
i ⟵ 0
while i< n-1 {
    j ⟵ 0
    while j< n-1 {
        if L[j] > L[j+1] {
            swap L[j] and L[j+1]
        }
        j ⟵ j + 1
    }
    i ⟵ i + 1
}
```

```python
def bubble_sort(the_list):
    n = len(the_list)
    i = 0
    while i < n - 1:
        j = 0
        while j < n - 1:
            if the_list[j] > the_list[j + 1]:
                swap(the_list, j, j + 1)
            j += 1
        i += 1
```

```python
def swap(the_list, i, j):
    tmp = the_list[i]
    the_list[i] = the_list[j]
    the_list[j] = tmp
```

best case: [1, 2, 3, 4, 5, …, n]

no swaps

```
    def bubble_sort(the_list):
t1      n = len(the_list)
t2      for i in range(n - 1):
t3          for j in range(n - 1):
t4              if the_list[j] > the_list[j + 1]:
t5                  swap(the_list, j, j + 1)
```

$\mathbf{c} = t_2$

$\mathbf{d} = t_3 + t_4$

$$\underbrace{\mathbf{c} + (n-1)\mathbf{d}}_{i=0} + \underbrace{\mathbf{c} + (n-1)\mathbf{d}}_{i=1} + \underbrace{\mathbf{c} + (n-1)\mathbf{d}}_{i=2} + \ldots + \underbrace{\mathbf{c} + (n-1)\mathbf{d}}_{i=n-2}$$

$$(n-1)[\mathbf{c} + (n-1)\mathbf{d}]$$

$$n^2\mathbf{d} + n(\mathbf{c}-2\mathbf{d})+(\mathbf{d}-\mathbf{c})$$

$$O(n^2)$$

worst case: [n, n-1,n-2, …, 2, 1]

every swap

```
    def bubble_sort(the_list):
t1      n = len(the_list)
t2      for i in range(n - 1):
t3          for j in range(n - 1):
t4              if the_list[j] > the_list[j + 1]:
t5                  swap(the_list, j, j + 1)
```

$\mathbf{c} = t_2$

$\mathbf{k} = t_3 + t_4 + t_5$

$$\underset{i=0}{\mathbf{c} + (n-1)\mathbf{k}} + \underset{i=1}{\mathbf{c} + (n-1)\mathbf{k}} + \underset{i=2}{\mathbf{c} + (n-1)\mathbf{k}} + \ldots + \underset{i=n-2}{\mathbf{c} + (n-1)\mathbf{k}}$$

$$(n-1)[\mathbf{c} + (n-1)\mathbf{k}]$$

$$n^2\mathbf{k} + n(\mathbf{c}-2\mathbf{k})+(\mathbf{k}-\mathbf{c})$$

$$O(n^2)$$

Improved bubble sort

```
def bubble_sort(the_list):
    n = len(the_list)
    for mark in range(n - 1, 0, -1):
        swapped = False
        for i in range(mark):
            if the_list[i] > the_list[i + 1]:
                swap(the_list, i, i + 1)
                swapped = True
        if not swapped:
            break
```

- Can you leave any of the two loops early?
- Best case $\neq$ Worst case
- **Best case** is a sorted list: **O(n)**
- **Worst case** is list in reverse order: **$O(n^2)$**

Selection Sort

(find minimum,
put it where it belongs,
reduce)

Selection Sort: Code

Algorithm SelectionSort(L)

// Sorts a list using selection sort

// Input: A list of orderable items

// Output: A list sorted in increasing order

n ⟵ length(L)

k ⟵ 0

while k < n {

  Find the minimum item in L[k:n-1] {

    Put the item in the correct position

  }

  k ⟵ k + 1

}

```python
def selection_sort(the_list):
    n = len(the_list)
    for k in range(n):
        min_position = find_minimum(the_list, k)
        swap(the_list, k, min_position)
```

```python
def find_minimum(the_list, starting_index):
    min_position = starting_index
    n = len(the_list)
    for i in range(starting_index, n):
        if the_list[i] < the_list[min_position]:
            min_position = i
    return min_position
```

Selection Sort: Code

n-1 times

```
def selection_sort(the_list):
    n = len(the_list)
    for mark in range(n - 1):
        min_index = find_minimum(the_list, mark)
        swap(the_list, mark, min_index)
```

n-mark-1 times each

fixed

```
def find_minimum(the_list, mark):
    position_minimum = mark
    n = len(the_list)
    for i in range(mark + 1, n):
        if the_list[i] < the_list[position_minimum]:
            position_minimum = i
    return position_minimum
```

Is selection sort better than bubble sort?

Stable sorting

A sorting algorithm is stable if it **maintains the relative order among elements.**

8	3	8	6	3
a	b	c	d	e

3	3	6	8	8
b	e	d	a	c

The **relative order** is preserved
(b before e, a before c)

3	3	6	8	8
e	b	d	a	c

The **relative order**
may not be preserved

Name	Mark
Ann	100
Brendon	90
Cheng	100
Daniel	50

Name	Mark
Daniel	50
Brendon	90
Cheng	100
Ann	100

Cheng before Ann

Name	Mark
Daniel	50
Brendon	90
Ann	100
Cheng	100

Ann before Cheng

stable:

the relative order of elements with the same value is maintained.

Summary

You need to understand and be able to implement the following simple sorting algorithms knowing their time complexity and stability properties:

- Bubble sort
- Selection sort
- Insertion sort