Data Structures in Python
Chapter 5

- Bubble sort
- **Selection sort**
- Insertion sort
- Merge sort
- Quick sort Algorithm
- Quick sort Analysis
- Empirical Analysis

Prof. Youngsup Kim, idebtor@gmail.com, CSEE Dept., Grace School Rm204 Handong Global University

Agenda & Readings

- Agenda
  - Selection sort algorithm
  - Time complexity
  - Bubble sort vs Selection sort
- Reference:
  - Problem Solving with Algorithms and Data Structures
    - Chapter 5 Search, Sorting and Hashing

Selection Sort - Algorithm

- Given is a list L of n value {L[0], ⋯ , L[n−1]}
  - Divide list into unsorted (left) and sorted part (right - initially empty):
    `Unsorted: {L[0], … , L[n-1]}` **`Sorted: {}`**
  - In each pass find **largest value and place it to the right of the unsorted part** using a **single swap**.
  - Reduce size of unsorted part by one and increase size of sorted part by one.
    After i-th pass:
    `Unsorted: {L[0], … , L[n-1-i]}` **`Sorted: {L[n-i],…,L[n-1]}`**
  - Repeat until unsorted part has a size of 1, then all elements are sorted

29	10	14	37	13	`List to sort`
10	14	29	13	37	PASS 1 (4 Comp, 1 Swap)
10	14	13	29	37	PASS 2 (3 Comp, 1 Swap)
10	13	14	29	37	PASS 3 (2 Comp, 0 Swap)
10	13	14	29	37	PASS 4 (1 Comp, 1 Swap)

Selection Sort – Exercise

54	26	93	17	77	31	44	55	20

List to sort

PASS 1 (Comp, Swap)

PASS 2 (Comp, Swap)

PASS 3 (Comp, Swap)

PASS 4 (Comp, Swap)

PASS 5 (Comp, Swap)

PASS 6 (Comp, Swap)

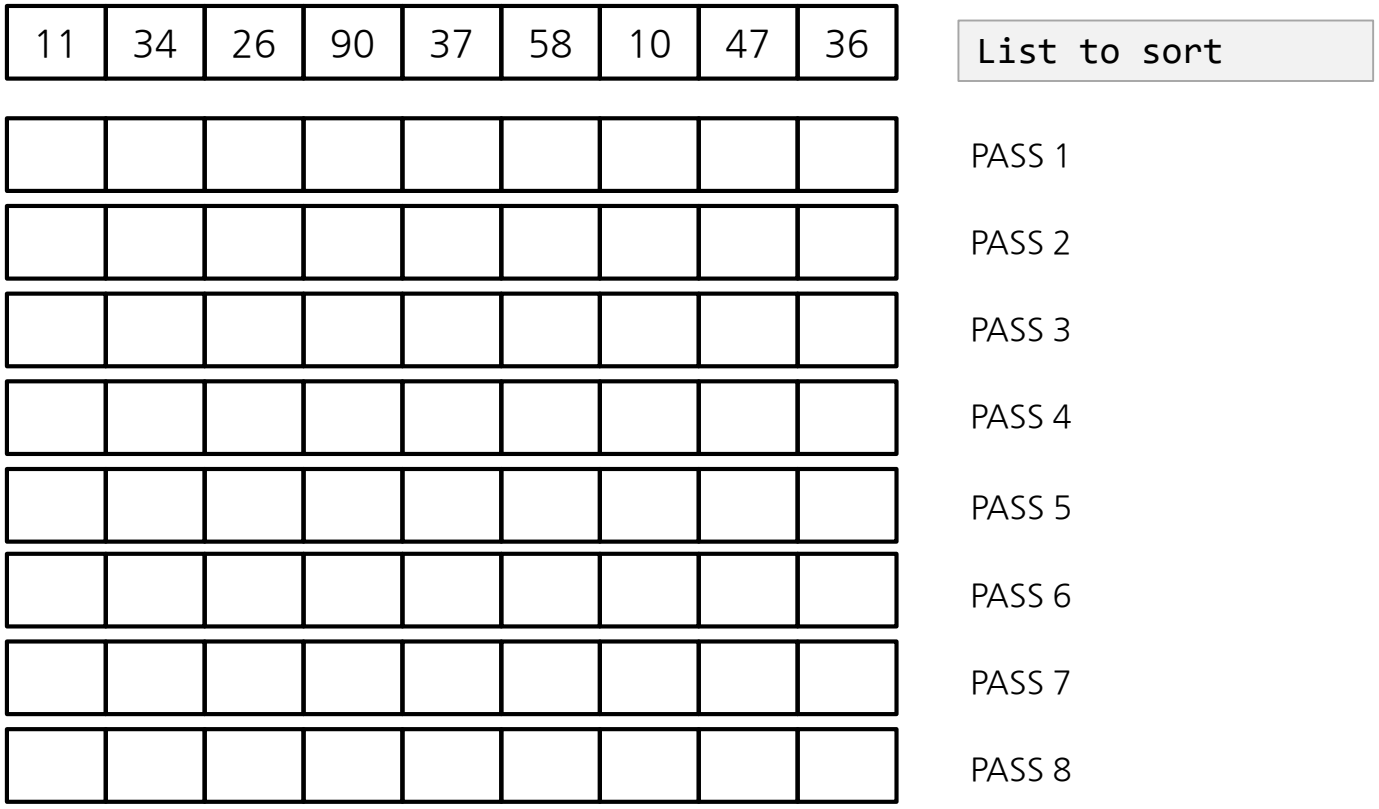
PASS 7 (Comp, Swap)

17	20	26	31	44	54	55	77	93

PASS 8 (Comp, Swap)

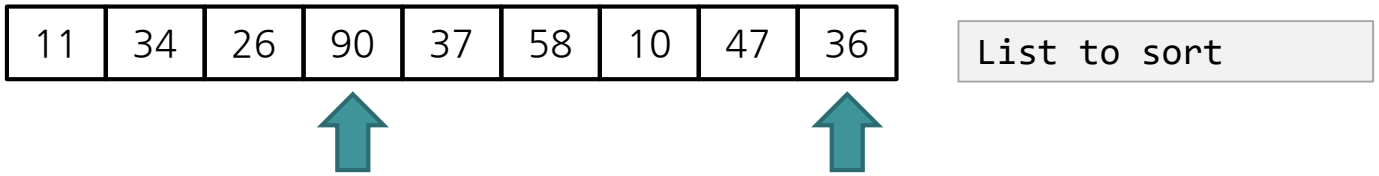
Total PASS 8 (Comp, Swap)

Selection Sort – Exercise

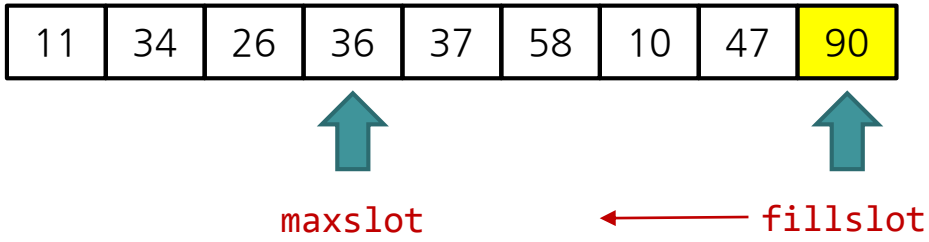


Selection Sort - swap elements

- Each pass we need to swap two elements of the list.
  For example, at the end of the first pass we want to swap the element at position 3 with the element at position 8.



- After the first pass:



Selection Sort Code

- Code

```
def selection_sort(a):
    for fillslot in range(len(a) - 1, 0, -1):
        maxslot = 0
        for slot in range(1, fillslot+1):
            if a[slot] > a[maxslot]:
                maxslot = slot
                a[fillslot], a[maxslot] = a[maxslot], a[fillslot]
                #print(fillslot, "-", a) # enable to see each pass

if __name__ == '__main__':
    a = [54, 26, 93, 17, 77, 31, 44, 55, 20]
    print("before: ", a)
    selection_sort(a)
    print(" after: ", a)
```

no check whether swap necessary

```
before:  [54, 26, 93, 17, 77, 31, 44, 55, 20]
 after:  [17, 20, 26, 31, 44, 54, 55, 77, 93]
```

Selection Sort - Big O

- For a list with n elements:
  - The number of comparisons?
  - pass 1 pass 2 pass 3 … last pass
    n–1 n–2 n–3 … 1

    $1 + 2 + \cdots + (n-3) + (n-2) + (n-1) = \frac{1}{2}(n^2 - n)$

- Big O of the selection sort is $O(n^2)$
  - The number of data increases 10 times, then it takes a 100 times longer.

- Note: **one swap each pass**
  (The current swap() implementation swaps elements even if indices are the same, i.e., no swap necessary.)

Selection Sort - Big O

- What if the data is already sorted?
  - Swaps?
  - Comparisons?

29	10	14	37	13
10	14	29	13	37
10	14	13	29	37
10	13	14	29	37
10	13	14	29	37

```
List to sort
```

PASS 1 (4 Comp, 1 Swap)

PASS 2 (3 Comp, 1 Swap)

PASS 3 (2 Comp, 0 Swap)

PASS 4 (1 Comp, 1 Swap)

5	10	14	32	35
5	10	14	32	35
5	10	14	32	35
5	10	14	32	35
5	10	14	32	35

```
List to sort
```

PASS 1 (Comp, Swap)

PASS 2 (Comp, Swap)

PASS 3 (Comp, Swap)

PASS 4 (Comp, Swap)

Selection Sort - Big O

- What if the data is in reverse order?
  - Swaps?
  - Comparisons?

29	10	14	37	13
10	14	29	13	37
10	14	13	29	37
10	13	14	29	37
10	13	14	29	37

```
List to sort
```

PASS 1 (4 Comp, 1 Swap)

PASS 2 (3 Comp, 1 Swap)

PASS 3 (2 Comp, 0 Swap)

PASS 4 (1 Comp, 1 Swap)

35	32	14	10	5
5	32	14	10	35
5	10	14	32	35
5	10	14	32	35
5	10	14	32	35

```
List to sort
```

PASS 1 (Comp, Swap)

PASS 2 (Comp, Swap)

PASS 3 (Comp, Swap)

PASS 4 (Comp, Swap)

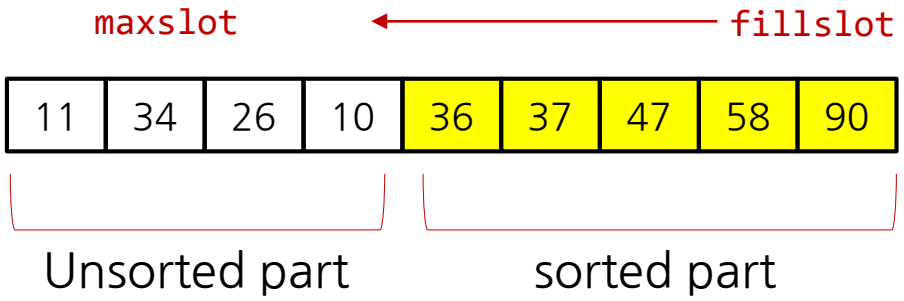
Comparison Bubble Sort vs. Selection Sort

- Bubble and Selection sort use the same number of comparisons.
- Bubble sort does $O(n)$ swaps per pass on average, but Selection sort only 1 swap per pass.
- Selection sort typically executes faster than bubble sort.

- **How can we do better?**
  IDEA: Reduce number of comparisons by inserting into sorted array.

Summary

- Divide array into unsorted (left) and sorted part (right, initially empty)
- Find largest value in unsorted part and place at end - after each pass sorted part increases by one and unsorted part reduces by one.



- Lots of comparisons $O(n^2)$, one swap per pass $O(n)$