COMP1819
Algorithms and Data Structures

Lecture 05: Sorting – Bubble, Selection and Insertion (for a small collection)

Dr. Tuan Vuong

16/02/2021

Content

- Review Lab 04
- Sorting: Bubble
- Selection
- Insertion
- Reinforcement

Lab 04

1. Compare Linear and Binary Search

Generate a sorted sequence of numbers. Set up a five experiment to test the difference between a sequential search and a binary search with the time complexity measurement.

Examples:

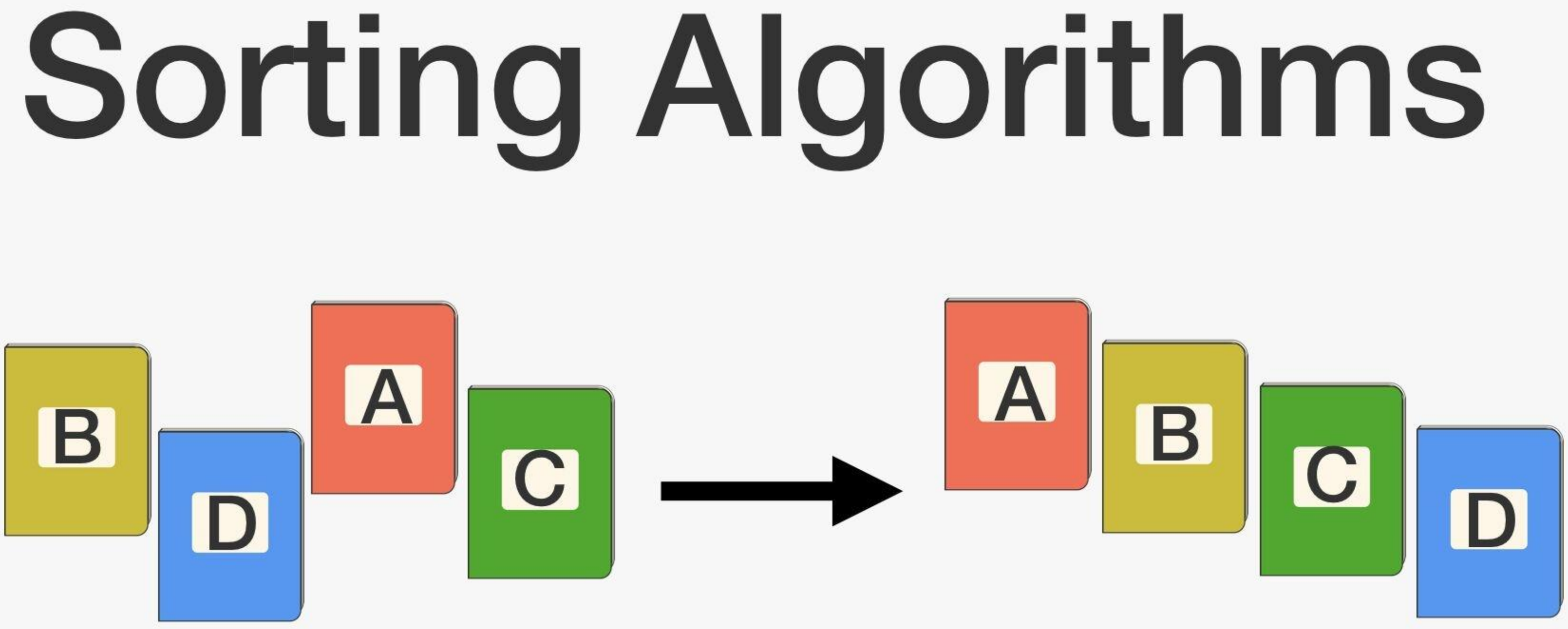
Input	Output
`1 3 9 10 18 …`	`Time execution for linear and binary search`

Hints

- Code for these searches given in the lecture slides
- You should create a large list, maybe 100000 items.
- Plot the graph to see the difference in running time.

How much faster?

Today



What is sorting? Ex: arranging in an ordered sequence

Why sorting?

- Consider last lecture on search
  - Un-sorted list
  - Sorted list
- Sorting books in library
- Sorting Movies in Blockbuster
- Sorting Numbers

Sorting examples: A, E, F, G, B -> A, B, E, F, G

Type of Sorting

There are many, many different types of sorting algorithms, but some are:

- Bubble Sort
- Selection Sort
- Insertion Sort
- Merge Sort
- Shell Sort
- Heap Sort
- Quick Sort
- Radix Sort
- Swap Sort
- Introsort
- Odd-even sort
- Cocktail shaker sort
- Cycle sort
- Merge-insertion sort
- Smooth sort
- Timsort

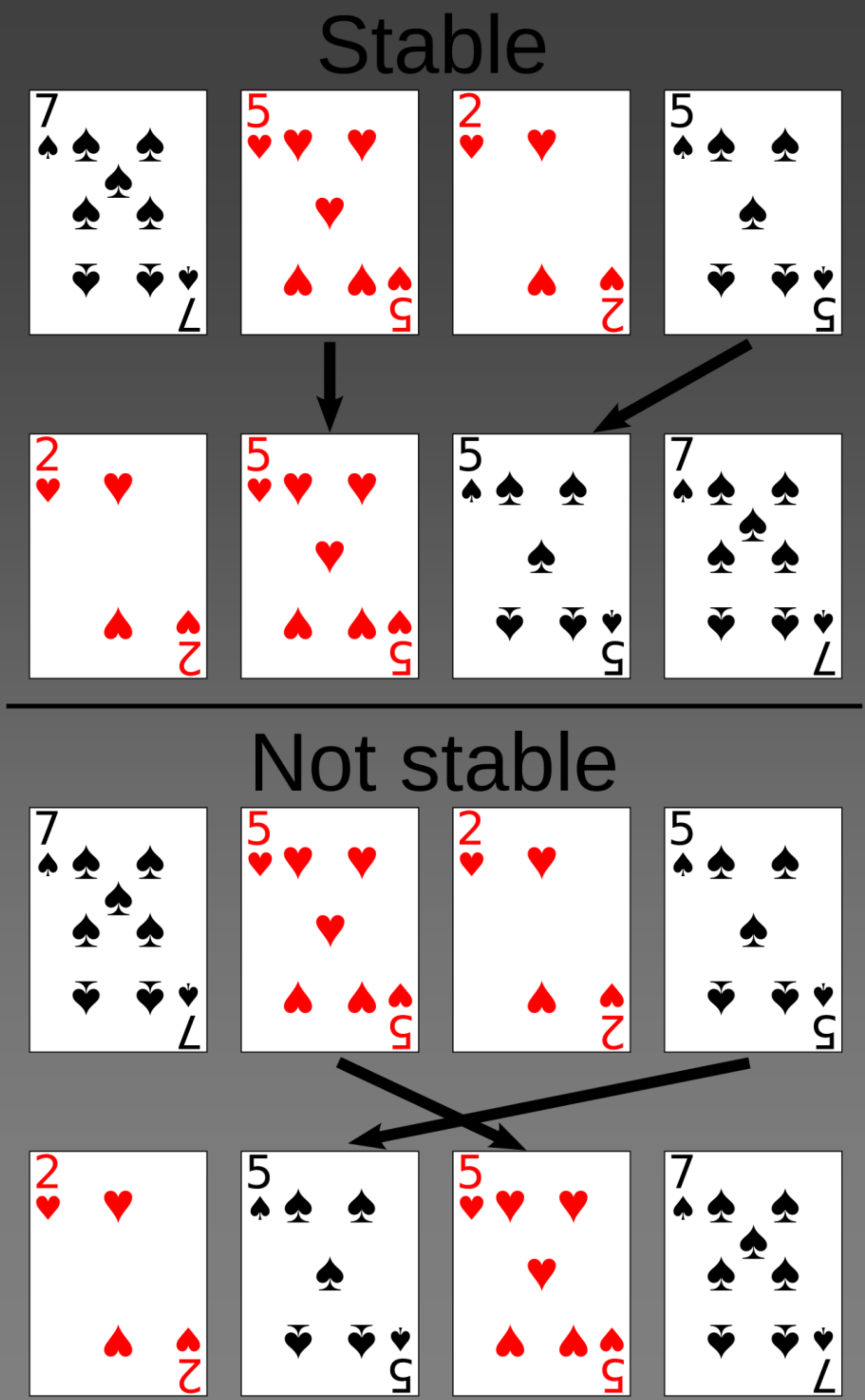
15 Sorting Algorithms (link)

Sorting complexity

- Most of the primary sorting algorithms run on different space and time complexity
- Time Complexity is defined to be the time the computer takes to run a program (or algorithm in our case).
- Space complexity is defined to be the amount of memory the computer needs to run a program.

What is the complexity of Linear and Binary Search?

Stable vs. no stable sort



Bubble

Original: 1 3 6 9 11 14 15 17

Steps: 10 1 3 6 9 11 14 15 17

1 10 3 6 9 11 14 15 17

1 3 10 6 9 11 14 15 17

1 3 6 10 9 11 14 15 17

1 3 6 9 10 11 14 15 17

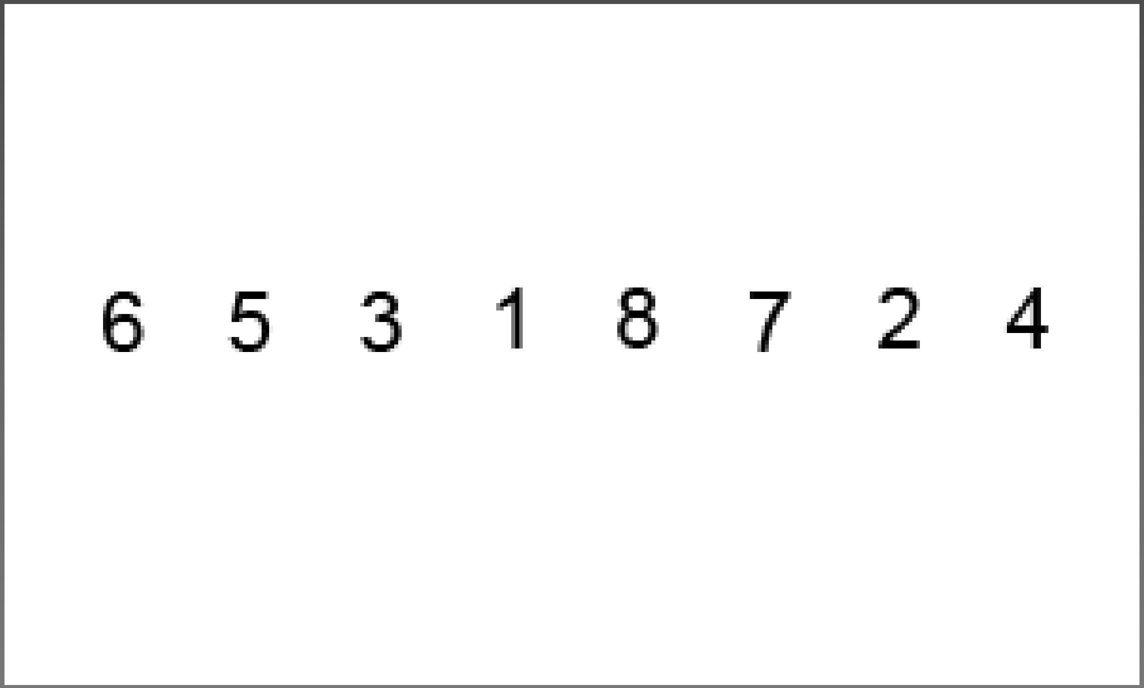
How about adding 10 to the sorted (original) list from the beginning?

Bubble Sorting

- Outer loop: Traverse through all array elements (i)
  - Traverse the array from 0 to n-i-1
    - Swap if the element found is greater than the next element

```
Input = [6, 5, 3, 1, 8, 7, 2, 4]
First pass: (iteration 0)
Swap.. :  6 5 [5, 6, 3, 1, 8, 7, 2, 4]
Swap.. :  6 3 [5, 3, 6, 1, 8, 7, 2, 4]
Swap.. :  6 1 [5, 3, 1, 6, 8, 7, 2, 4]
Swap.. :  8 7 [5, 3, 1, 6, 7, 8, 2, 4]
Swap.. :  8 2 [5, 3, 1, 6, 7, 2, 8, 4]
Swap.. :  8 4 [5, 3, 1, 6, 7, 2, 4, 8]
```

Bubble sort



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

Bubble sort: repeatedly swapping the adjacent items if they are in wrong order. Each pass is bubbling the biggest item to the end.

```python
# Python program for implementation of Bubble Sort

def bubbleSort(arr):
    n = len(arr)

    # Traverse through all array elements
    for i in range(n):

        # Last i elements are already in place
        for j in range(0, n-i-1):

            # traverse the array from 0 to n-i-1
            # Swap if the element found is greater
            # than the next element
            if arr[j] > arr[j+1] :
                arr[j], arr[j+1] = arr[j+1], arr[j]

# Driver code to test above
arr = [64, 34, 25, 12, 22, 11, 90]

bubbleSort(arr)

print ("Sorted array is:")
for i in range(len(arr)):
    print ("%d" %arr[i]),
```

Selection Sort in Python

Analysis of Algorithms

- What does analysis of algorithms involve ?
  - element comparisons
  - the number of element comparisons
- What is the best/worst/average case? When?
- Can you improve from traverse through all array elements (the outer loop)?
- Big-O of Bubble Sort?
- Stable or not stable?

Selection

Original: 6 5 3 1 8 7 2 4

Selection (min.): 6 5 3 1 8 7 2 4

Which item should we swap 6 with?

Selection sort

8 5 2 6 9 3 1 4 0 7

Selection sort animation. Red is current min. Yellow is sorted list. Blue is current item.

Selection sort: repeatedly finding the minimum element from unsorted part and putting it at the beginning of the unsort part.

	8
	5
	2
	6
	9
	3
	1
	4
	0
	7

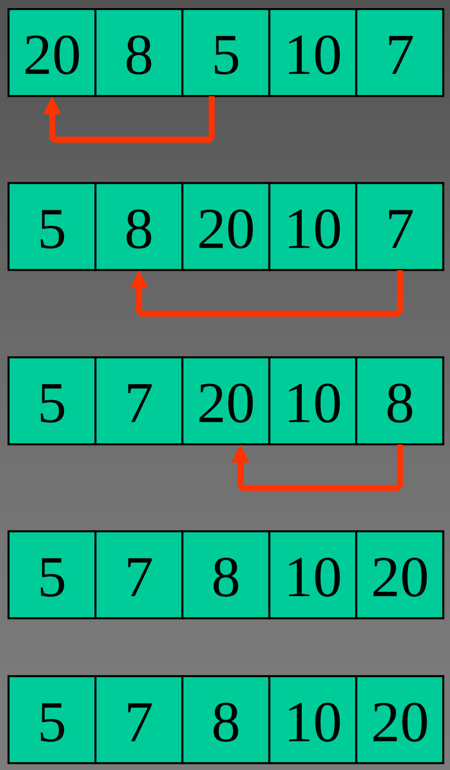
Selection Sorting

Outer loop: Traverse through all array elements (i)

1. find the smallest element

   among *A[i]~ A[len(A)-1];*

2. swap it with *A[i]*;



```python
# Python program for implementation of Selection
# Sort
import sys
A = [64, 25, 12, 22, 11]

# Traverse through all array elements
for i in range(len(A)):

    # Find the minimum element in remaining
    # unsorted array
    min_idx = i
    for j in range(i+1, len(A)):
        if A[min_idx] > A[j]:
            min_idx = j

    # Swap the found minimum element with
    # the first element
    A[i], A[min_idx] = A[min_idx], A[i]

# Driver code to test above
print ("Sorted array")
for i in range(len(A)):
    print("%d" %A[i]),
```

Selection Sort in Python

Analysis of Algorithms

- What is the best/worst/average case? When?
- Big-O?
- Stable or not stable?

Insertion

Original: 1 3 5 6 2 7 8 4

After inserting 2: 1 2 3 5 6 7 8 4

Keep moving 2 left until when?

Insertion Sorting

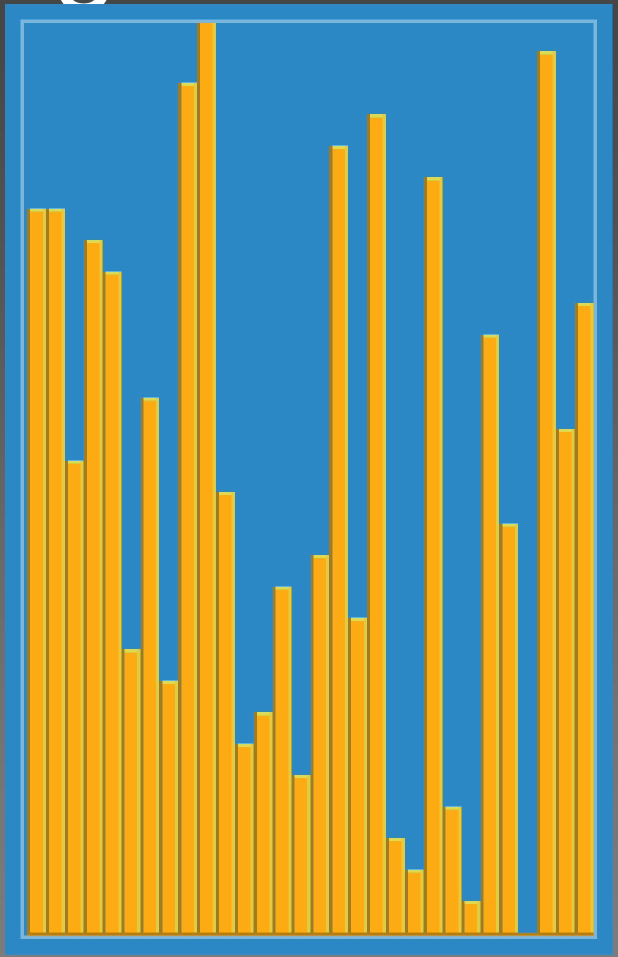
Traverse through 1 to len(arr)

#Place i*th* item in proper position:

key = arr[i]

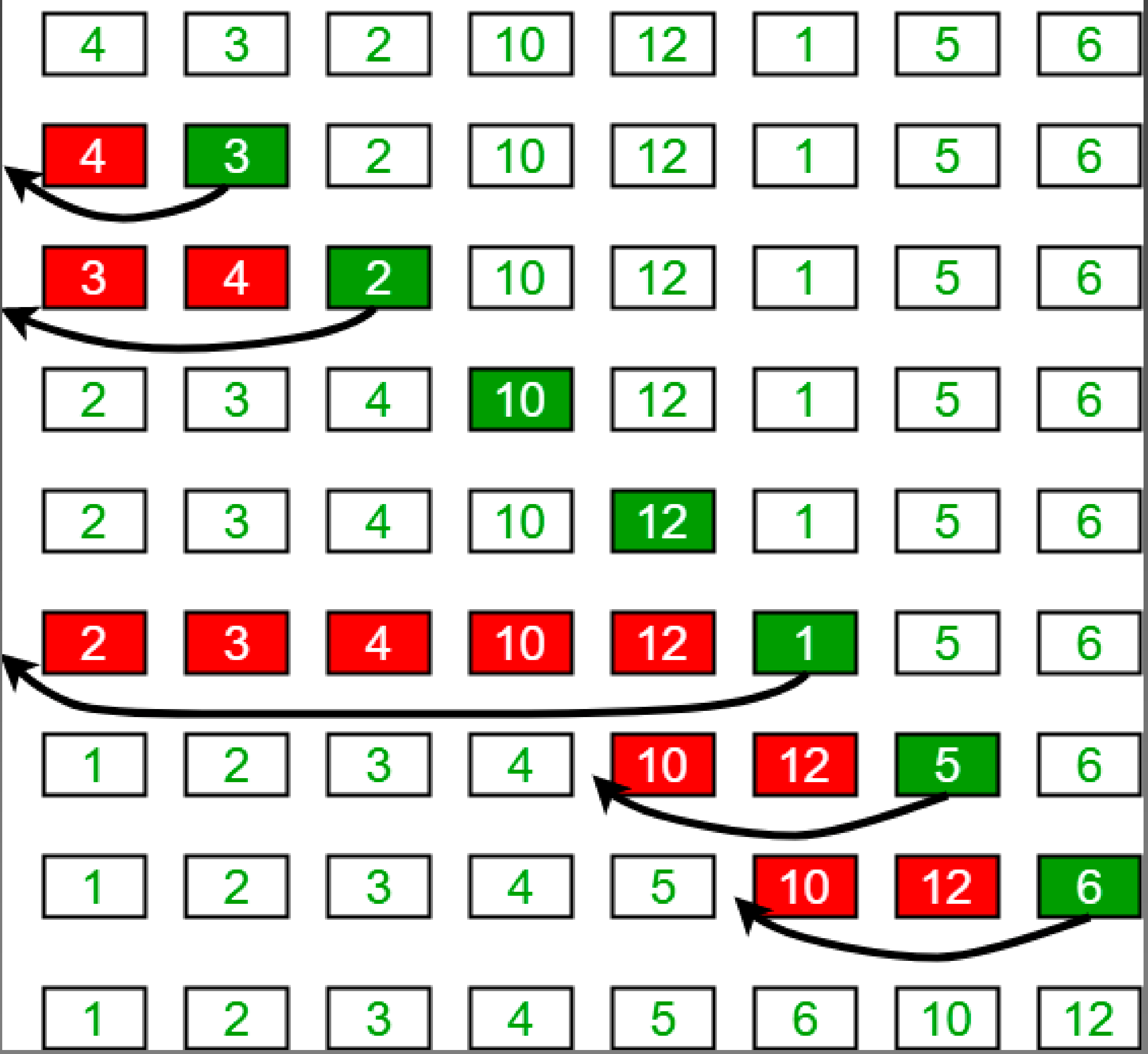
shift those elements *arr[j]* which greater than *key* to right by one position

place *key* in its proper position



Insertion Sort: shifting the values from unsorted part and placed at the correction positions at the sort part.

Insertion Sort Execution Example



```python
# Python program for implementation of Insertion Sort

# Function to do insertion sort
def insertionSort(arr):

    # Traverse through 1 to len(arr)
    for i in range(1, len(arr)):

        key = arr[i]

        # Move elements of arr[0..i-1], that are
        # greater than key, to one position ahead
        # of their current position
        j = i-1
        while j >= 0 and key < arr[j] :
                arr[j + 1] = arr[j]
                j -= 1
        arr[j + 1] = key


# Driver code to test above
arr = [12, 11, 13, 5, 6]
insertionSort(arr)
for i in range(len(arr)):
    print ("% d" % arr[i])
```

Insertion Sort in Python

Analysis of Algorithms

- What is the best/worst/average case? When?
- Big-O?
- Stable or not stable?
- Binary Insertion Sort?

Time and Space Complexity:

SORTING ALGORITHM	TIME COMPLEXITY			SPACE COMPLEXITY
	Best Case	Average Case	Worst Case	Worst Case
Bubble Sort	**$O(N)$**	**$O(N^2)$**	**$O(N^2)$**	**$O(1)$**
Selection Sort	**$O(N^2)$**	**$O(N^2)$**	**$O(N^2)$**	**$O(1)$**
Insertion Sort	**$O(N)$**	**$O(N^2)$**	**$O(N^2)$**	**$O(1)$**

Reinforcement

Discussion 1

Consider the following list of integers: [1,2,3,4,5,6,7,8,9,10]. Show how this list is sorted by the following algorithms:

- bubble sort
- selection sort
- insertion sort

Discussion 2

Consider the following list of integers: [10,9,8,7,6,5,4,3,2,1]. Show how this list is sorted by the following algorithms:

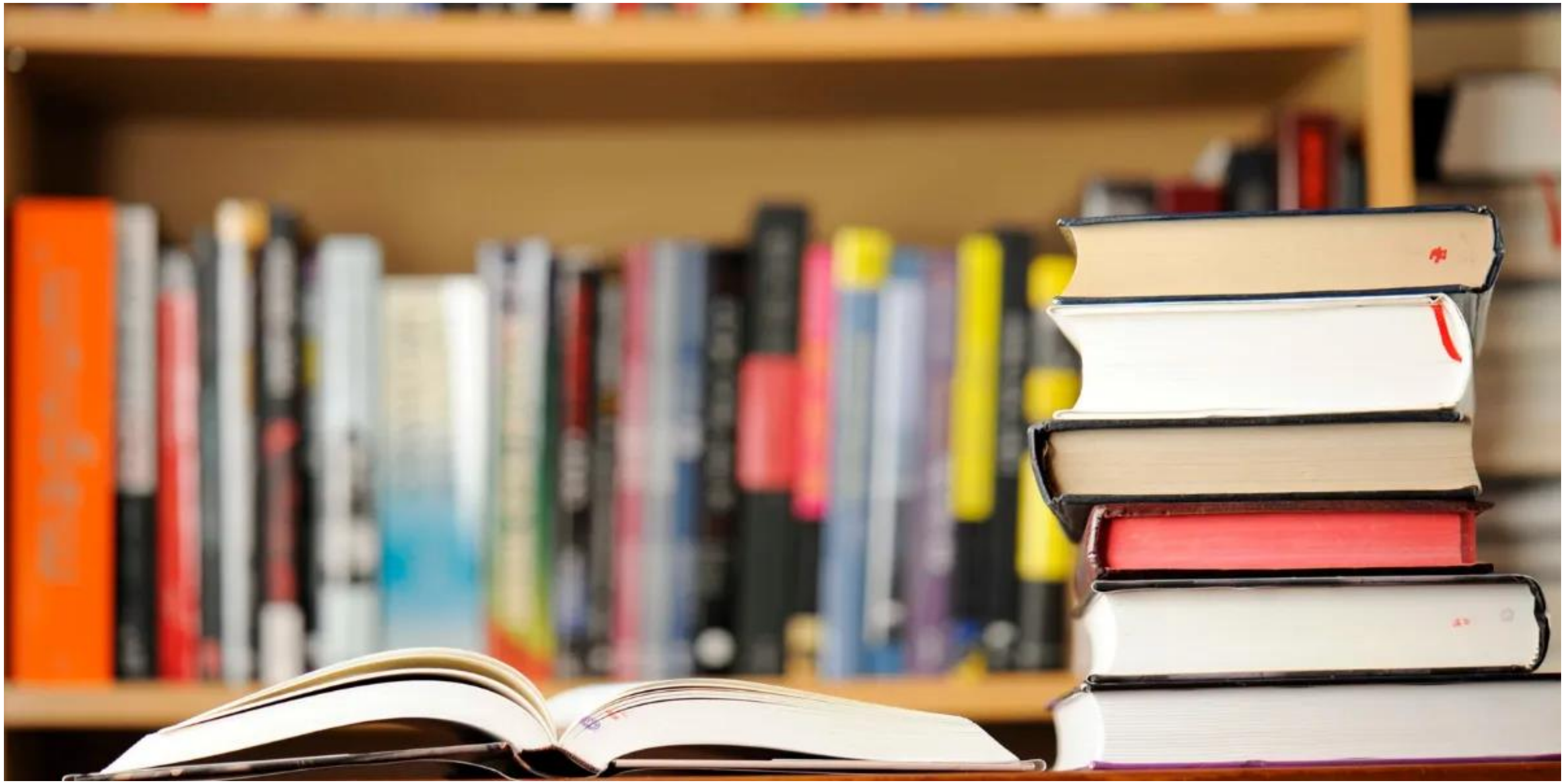
- bubble sort
- selection sort
- insertion sort

Quick overview

- Sorting algorithms
- Stable vs. not-stable sort
- Bubble sort, selection sort, insertion sort
- O(n^2) sort algorithms

Extra reading

- Binary Insertion Sort
- Shell sort



Next week

Maps, dictionaries in Python