Sorting

Week 13

Sorting problem

- The **sorting problem** means ***arranging*** the elements of a list in a specific order, typically in ascending or descending order.
- Sorting is a fundamental task in computer science and has been extensively studied since the early days of computing.
- The input to a sorting algorithm is a list of comparable elements, and the output is the same list with elements arranged in the desired order.
- The size `n` of the input is the number of the elements in the list.

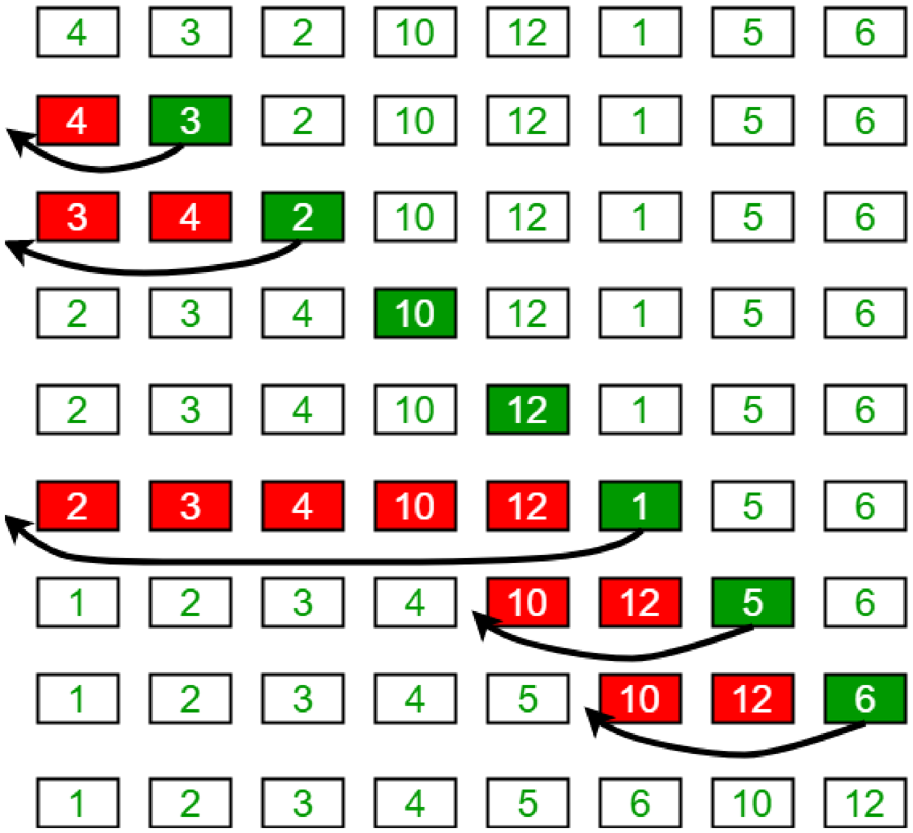
Sorting methods

We consider two famous sorting methods briefly:

- **Insertion Sort**: This algorithm builds the sorted array **one element at a time** by repeatedly picking the next element and **inserting** it into its **correct** position.
- **Quicksort**: A **recursive** algorithm that picks a **pivot** element and **partitions** the array into two sub-arrays, which are then sorted recursively.

Insertion sort

Insertion Sort Execution Example



4 3 2 10 12 1 5 6

4 3 2 10 12 1 5 6

3 4 2 10 12 1 5 6

2 3 4 10 12 1 5 6

2 3 4 10 12 1 5 6

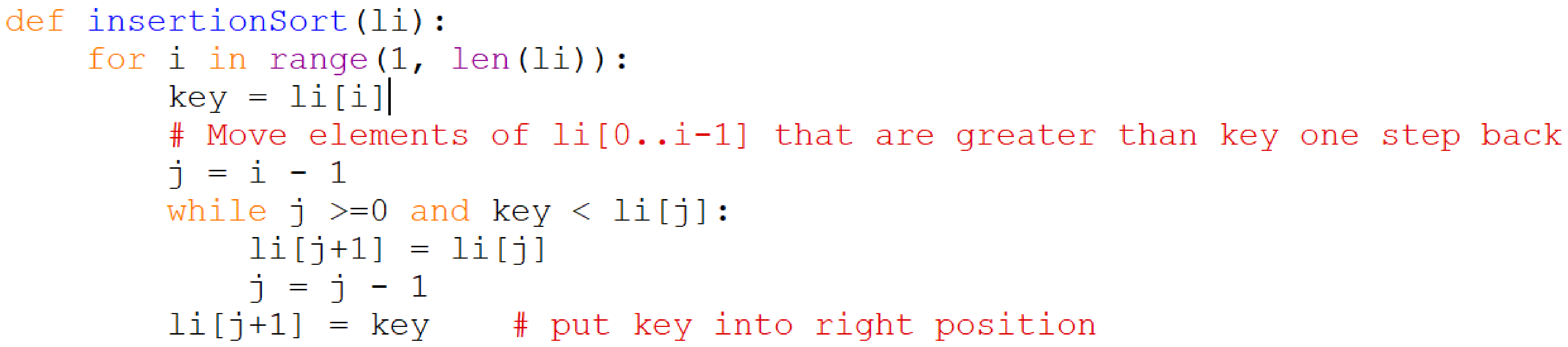
2 3 4 10 12 1 5 6

1 2 3 4 10 12 5 6

1 2 3 4 5 10 12 6

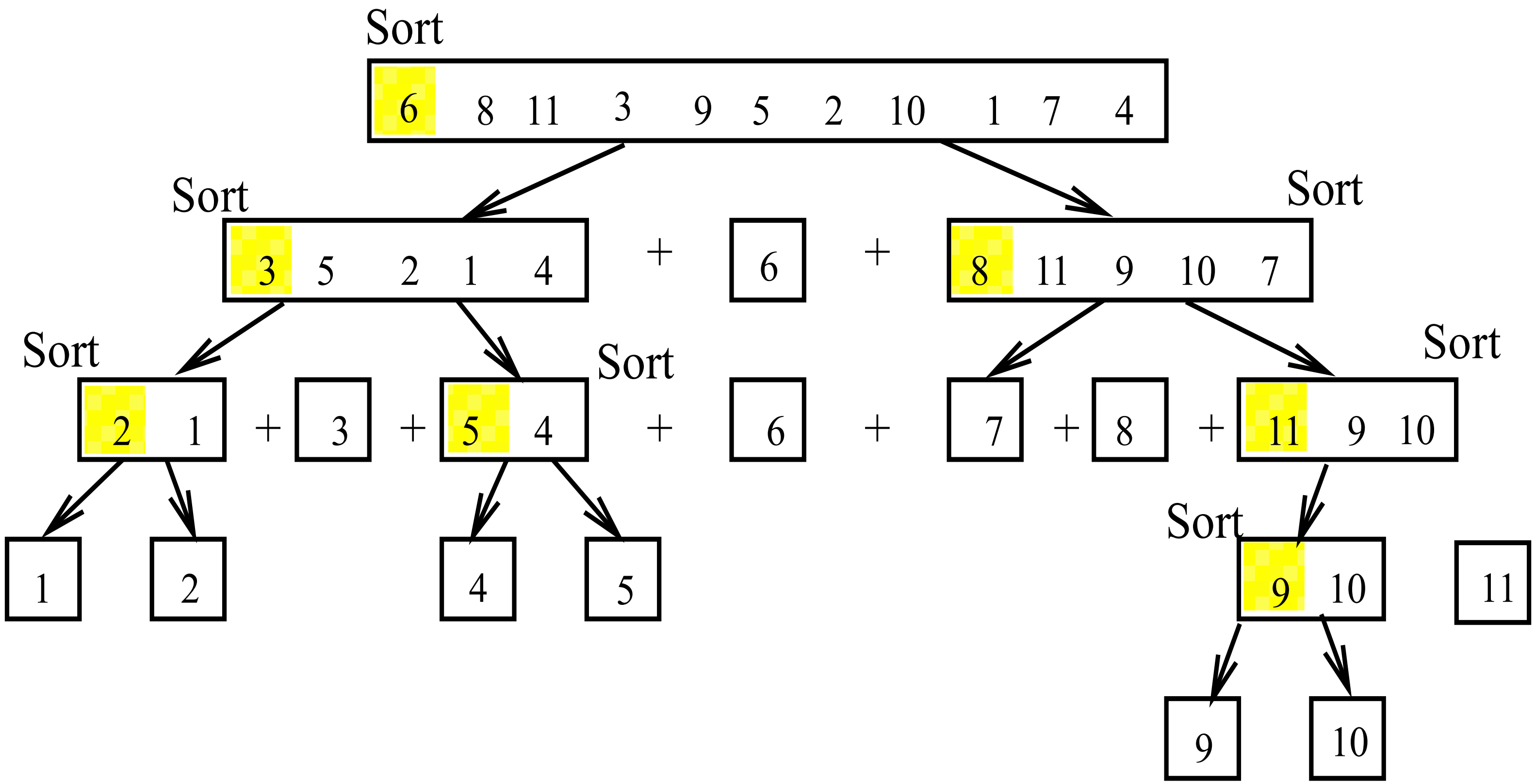
1 2 3 4 5 6 10 12

Insertion sort



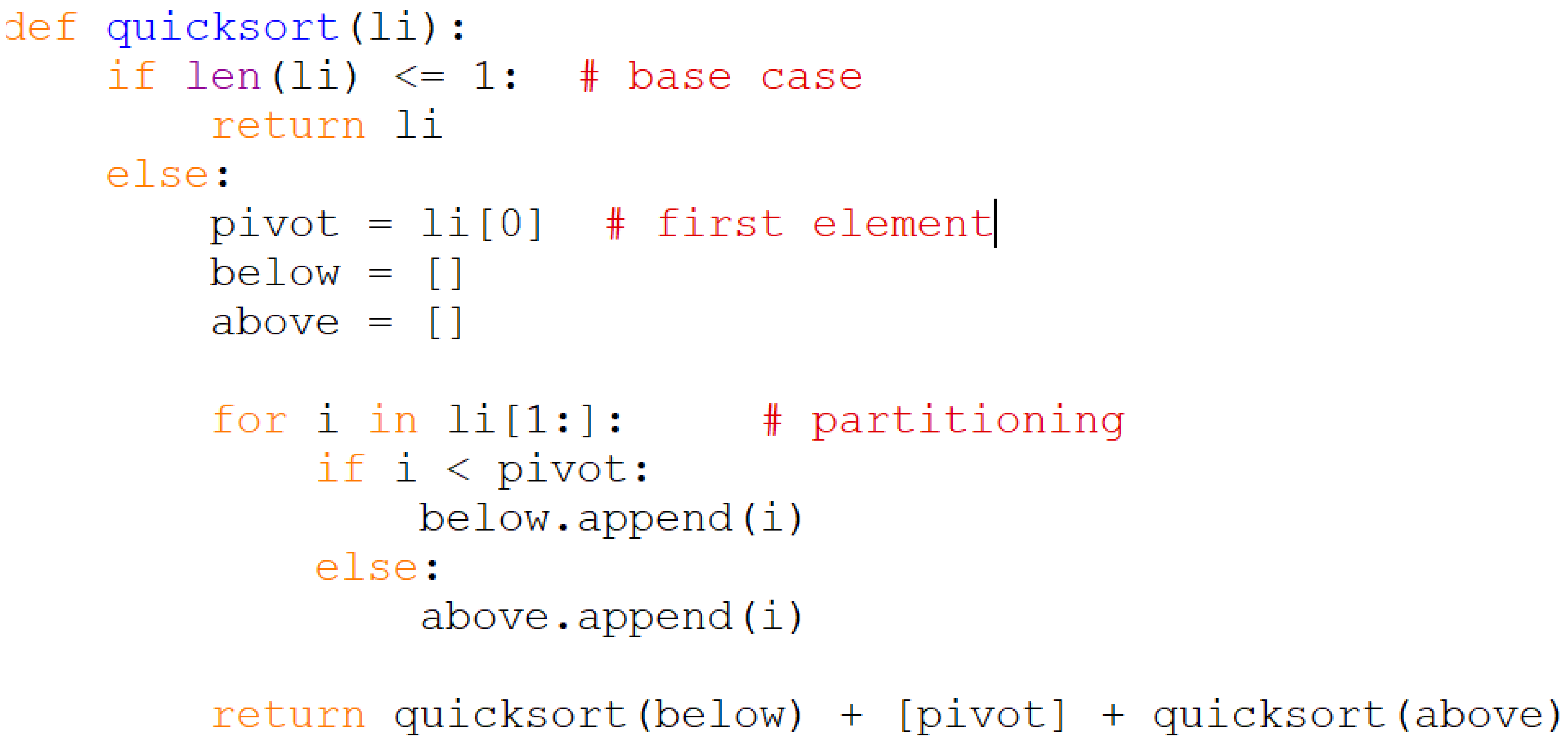
```python
def insertionSort(li):
    for i in range(1, len(li)):
        key = li[i]
        # Move elements of li[0..i-1] that are greater than key one step back
        j = i - 1
        while j >=0 and key < li[j]:
            li[j+1] = li[j]
            j = j - 1
        li[j+1] = key    # put key into right position
```

Quicksort



Sort
6 8 11 3 9 5 2 10 1 7 4
Sort
3 5 2 1 4
+
6
+
Sort
8 11 9 10 7
Sort
2 1
+
3
+
5 4
Sort
+
6
+
7
+
8
+
Sort
11 9 10
1
2
4
5
Sort
9 10
11
9
10

Quicksort



```python
def quicksort(li):
    if len(li) <= 1:  # base case
        return li
    else:
        pivot = li[0]  # first element
        below = []
        above = []

        for i in li[1:]:     # partitioning
            if i < pivot:
                below.append(i)
            else:
                above.append(i)

        return quicksort(below) + [pivot] + quicksort(above)
```

Comparison of running times

Size of list	Insertion sort (seconds)	Quicksort (seconds)
10000	1.57	0.01
15000	3.60	0.02
20000	6.41	0.02
25000	10.03	0.03
50000	42.35	0.07

File: `sort_comparison.py`