CO1107
Data Structure

Sorting Algorithm

Sorting Lists

- Input:
  - A list (not necessarily sorted) of ‘orderable’ element types
  - For example, in Python:
    - `the_list = [5,1.5,3,-4.0]` can be sorted
    - `the_list = [‘Bob’,`Alice’,`Cathy’] can be sorted`
    - `the_list = [1,'hj',0,'j']` cannot be sorted unless you define your own comparison function for different type of objects
- Output:
  - A list with the same elements as the input list BUT sorted in increasing or decreasing order

Lots of approaches

- There are many ways to sort a list of numbers.
- Including:
  - **Selection sort,**
  - **Insertion sort,**
  - Merge sort,
  - Quick sort,
  - Heap sort, and
  - **Bubble sort**
- There are many programs available.

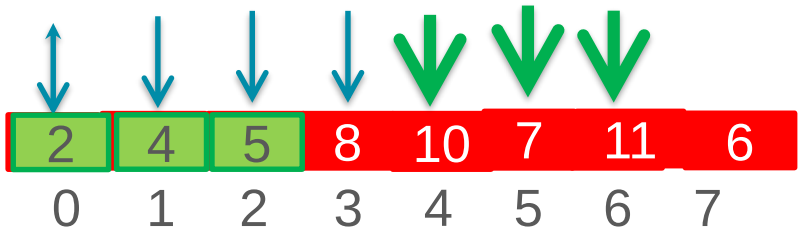
Selection Sort

Idea

- **Select** the smallest number in the list and move it to index 0.
- **Select** the 2nd smallest number in the list and move it to index 1.
- **Select** the 3rd smallest number in the list and move it to inc
- …
- **Select** the n-th smallest n
  the list and move it to ind

Representing Selection Sort as an algorithm

- Select the smallest number in the list and swap it with the number at index 0.
- Select the smallest number in the list between index 1 to n-1 and swap it with number at index 1.
- Select the smallest number in the list between index 2 to n-1 and swap it with number at index 2.
- …



Class Activity

- Sort the list [10,−4,−1,20,13,−3] into ascending order using selection sort
- 10, -4, -1, 20, 13, -3 — Original List
- -4, 10, -1, 20, 13, -3 — **1st Iteration**
- -4, -3, -1, 20, 13, 10 — **2nd Iteration**
- -4, -3, -1, 20, 13, 10 — **3rd Iteration**
- -4, -3, -1, 10, 13, 20 — **4th Iteration**
- -4, -3, -1, 10, 13, 20 — 5th Iteration & Sorted List

Implementing Selection Sort

```
def getMinIndex(myList, start, stop):
    min_index = start
    for i in range(start+1,stop):
        if myList[i] < myList[min_index]:
            min_index = i
    return min_index
def swapElements(myList,i,j):
    temp = myList[i]
    myList[i] = myList[j]
    myList[j] = temp

def selectionSort(aList):
    n = len(aList)
    for index in range(n):
       #Find position of the smallest number in
aList[index:]
       min_position = getMinIndex(aList,index,n)

      #Swap numbers at "index" and "min_position"
       swapElements(aList,index,min_position)

aList = [7,2,11,8,4,2,5,6]
selectionSort(aList)
print(aList)
```

Sample Selection Sort Implementation for list of lists (Task 2 from Week 2 workshop)

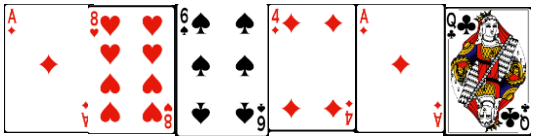
```
def selectionSortDistance ( array ):
    vendor = len( array )
    for position in range(vendor-1):
        minRow = position
        for temp in range( position + 1 , vendor ):
            if array [temp ] [ 0 ] < array [minRow ] [ 0 ] :
                minRow = temp
        array [ position ] , array [minRow] = array [minRow] , array [ position ]

    return array


table=[[120, 150.12], [140, 180.1], [70, 250.02], [99, 398.72], [144, 205.42]]
selectionSortDistance(table)
```

Analysis of Selection Sort

- The order of elements in the list does not affect the sorting time. meaning, even if the list is partially sorted, still each element is compared and there is no breaking out early.
- Selection sort is applicable for smaller dataset.
- Selection sort is useful when the list is NOT partially sorted.
- Should be used in those algorithm where the cost of swapping does not matter.

Insertion Sort

Insertion Sort

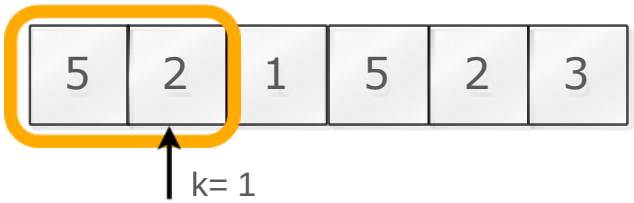


Idea: The way you normally sort playing cards

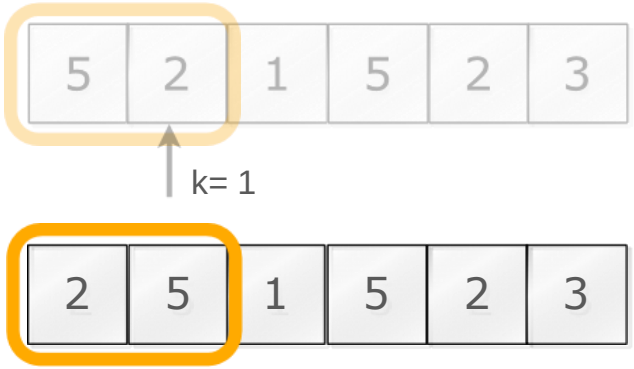
- #Assume all cards are lying face down on the table
- Pick 1st card
- Pick 2nd card and **insert** it such that the 2 picked cards are sorted
- Pick 3rd card and **insert** it such that the 3 picked cards are sorted
- Pick 4th card and **insert** it such that the 4 picked cards are sorted
- …
- Pick n-th card and **insert** it such that the n picked cards are sorted

UNIVERSITY OF LEICESTER

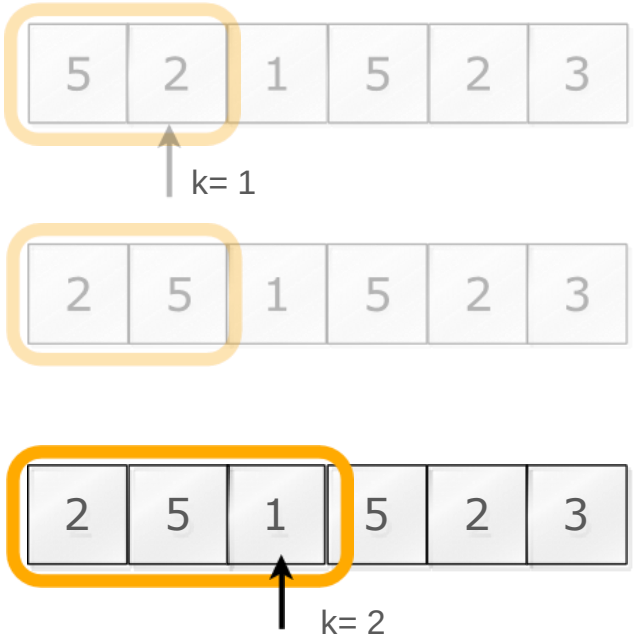
L



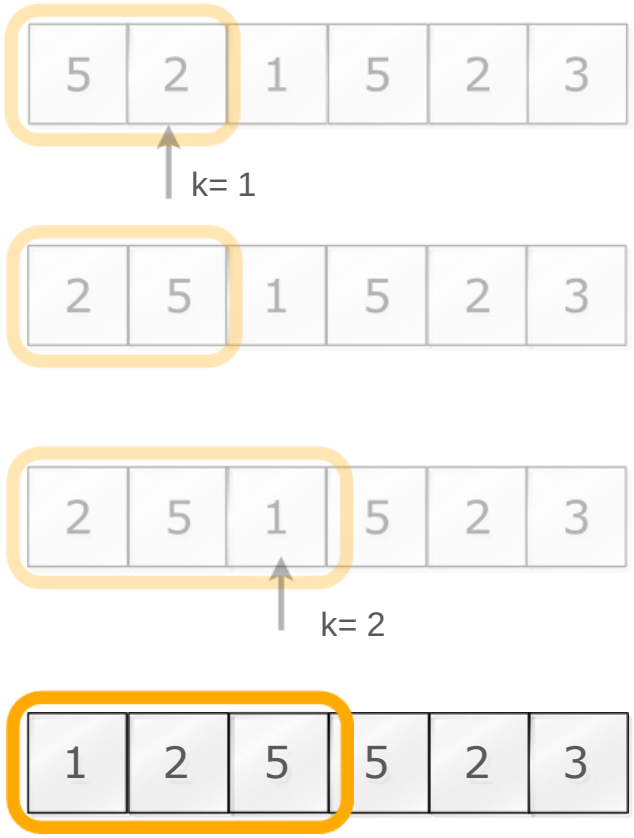
L



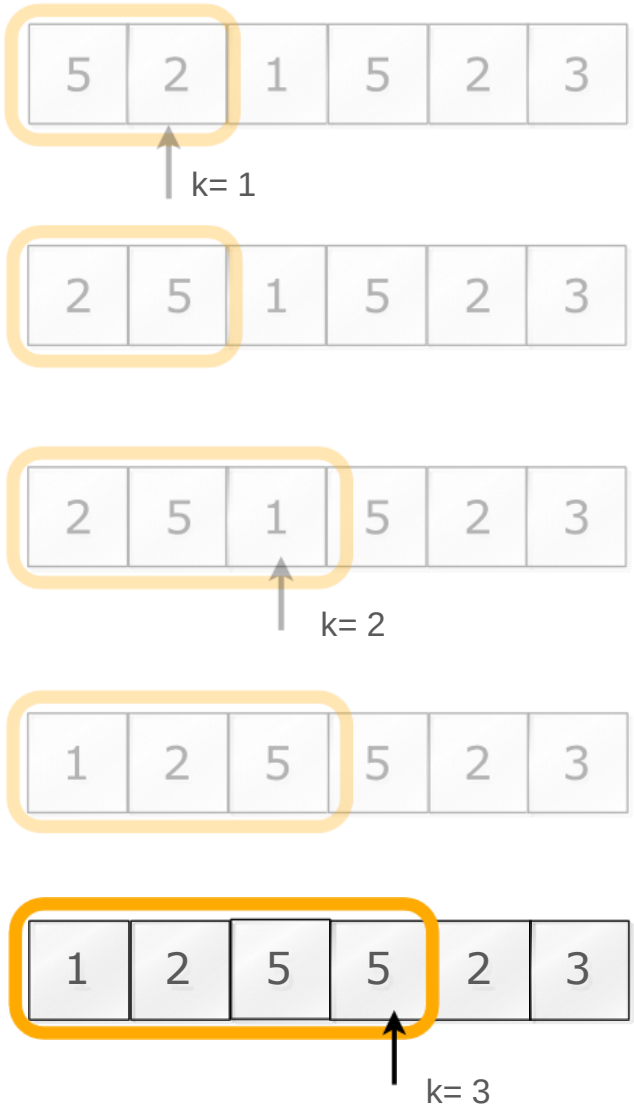
L



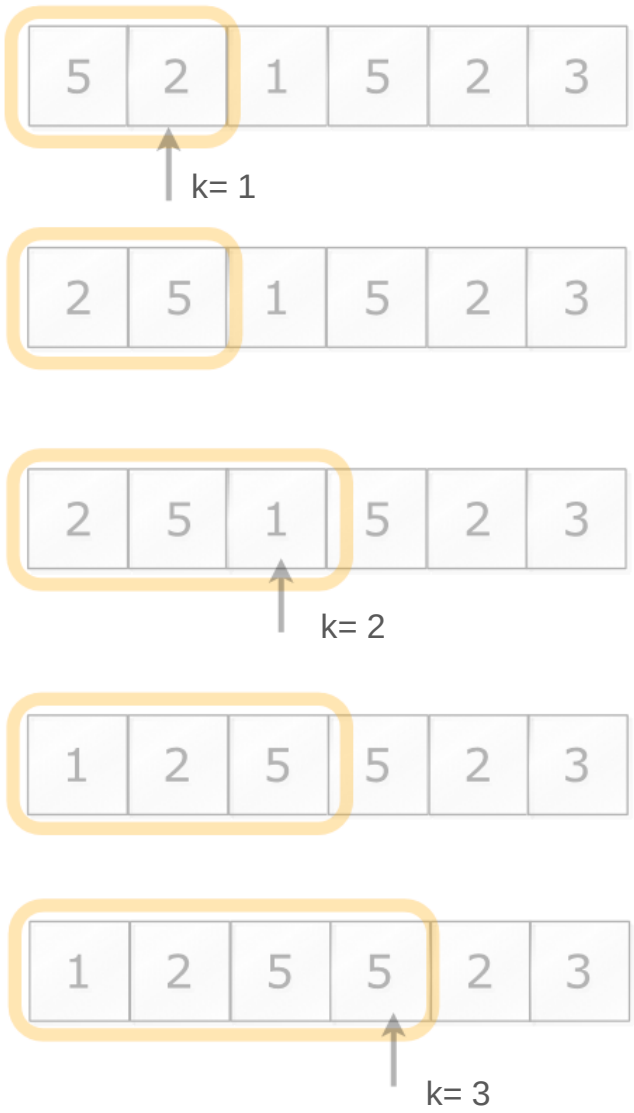
L



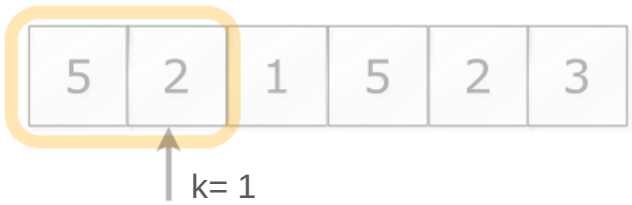
L



L

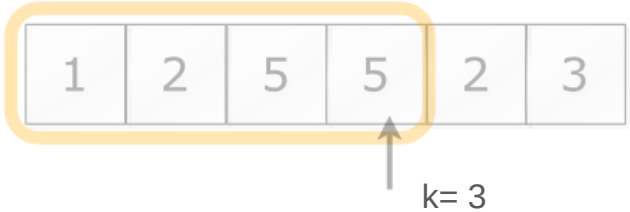


L



k= 1

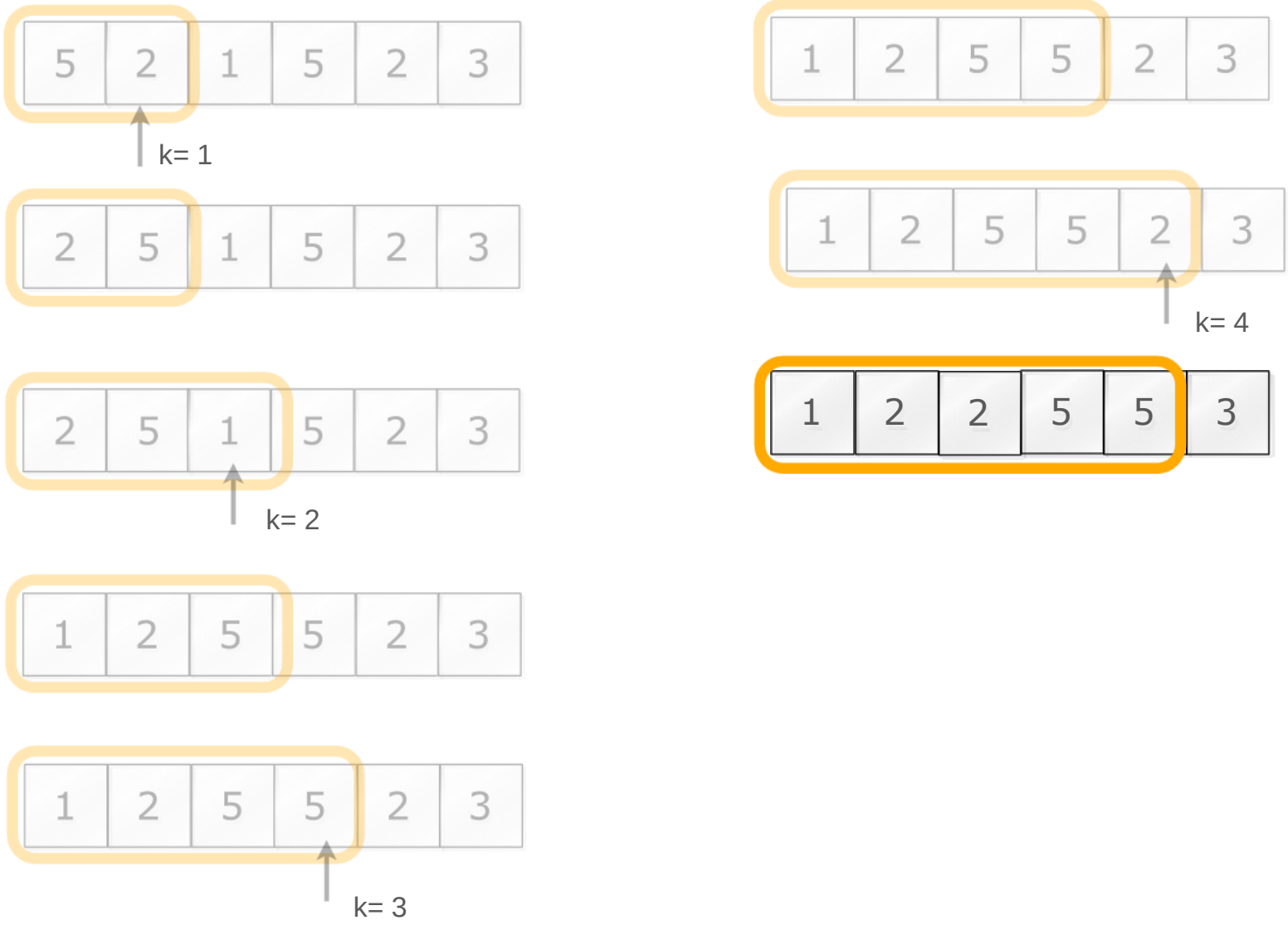
k= 2



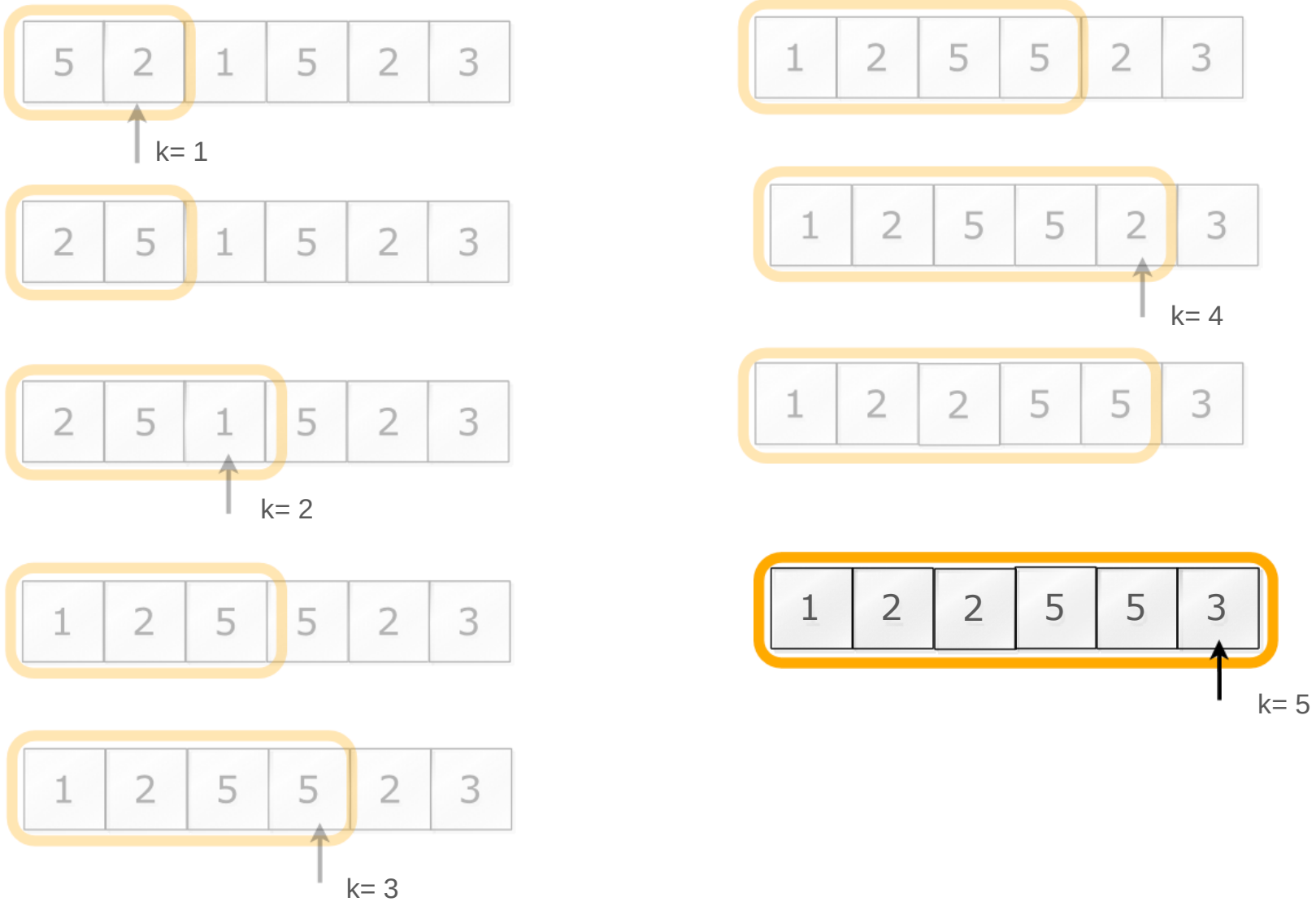
k= 3

k= 4

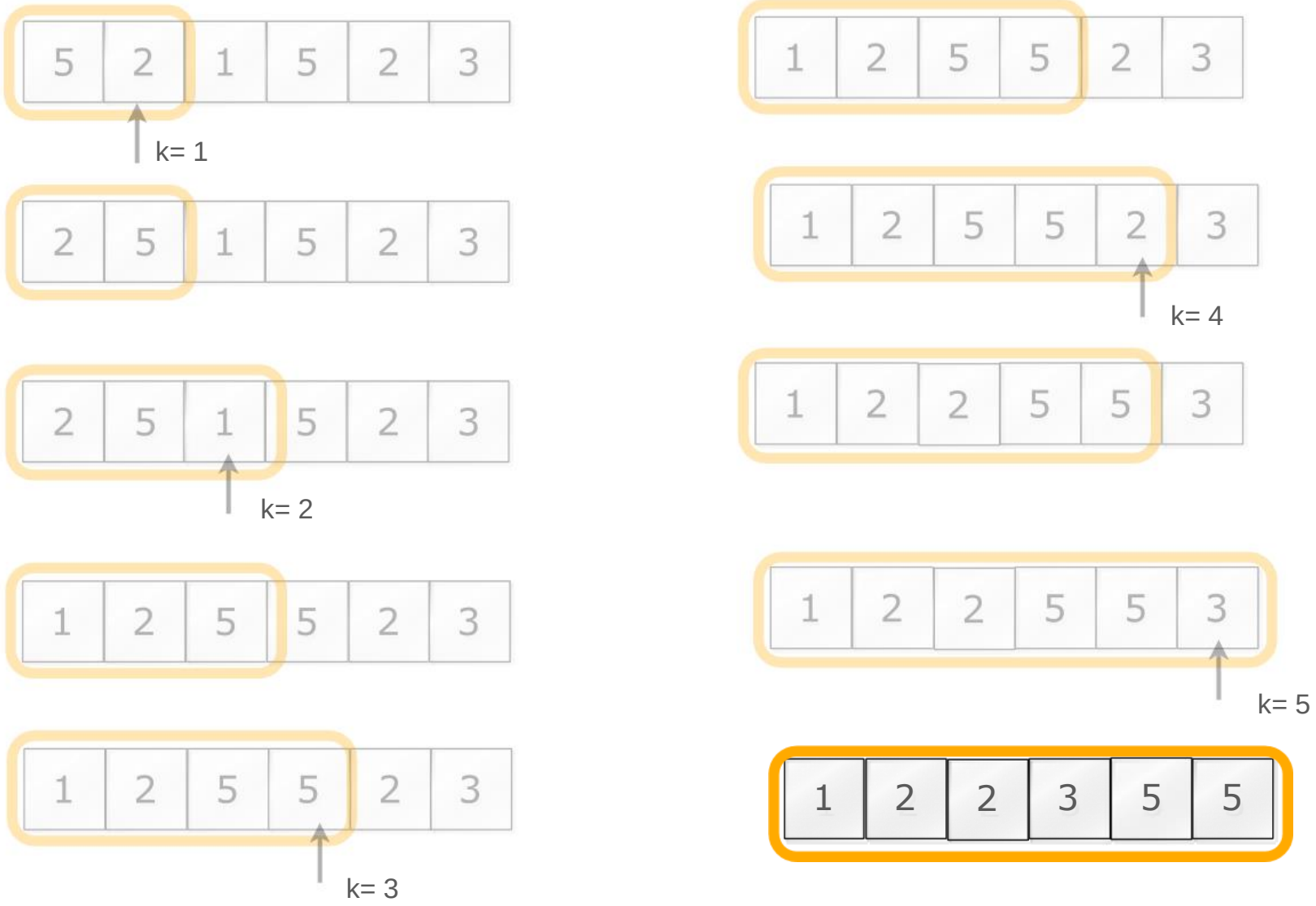
L


5 2 1 5 2 3
k= 1
2 5 1 5 2 3
2 5 1 5 2 3
k= 2
1 2 5 5 2 3
1 2 5 5 2 3
k= 3
1 2 5 5 2 3
1 2 5 5 2 3
k= 4
1 2 2 5 5 3

L

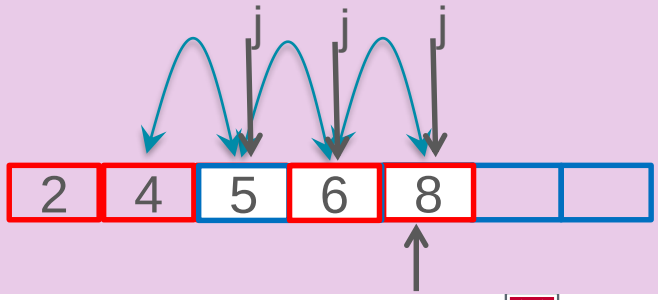

5 2 1 5 2 3
k= 1
2 5 1 5 2 3
2 5 1 5 2 3
k= 2
1 2 5 5 2 3
1 2 5 5 2 3
k= 3
1 2 5 5 2 3
1 2 5 5 2 3
k= 4
1 2 2 5 5 3
1 2 2 5 5 3
k= 5

L


5 2 1 5 2 3
k= 1
2 5 1 5 2 3
2 5 1 5 2 3
k= 2
1 2 5 5 2 3
1 2 5 5 2 3
k= 3
1 2 5 5 2 3
1 2 5 5 2 3
k= 4
1 2 2 5 5 3
1 2 2 5 5 3
k= 5
1 2 2 3 5 5

Insertion Sort

```
#swap elements in a list at indices i and j
def swapElements(myList,i,j):
    temp = myList[i]
    myList[i] = myList[j]
    myList[j] = temp

def shiftback(the_list,j):
    while aList[j-1] > aList[j] and j>0:
            #swap elements at aList[j] and aList[j1]
            swapElements(aList,j-1,j)
            j = j-1

def insertionSort(aList):
    n = len(aList)
    for k in range(1,n):
        #insert aList[k] in aList[0:k] in sorted order
        shiftback(aList,k)

aList = [6,5,3,1,8,7, 2, 4]
insertionSort(aList)
print(aList)
```



Class Activity

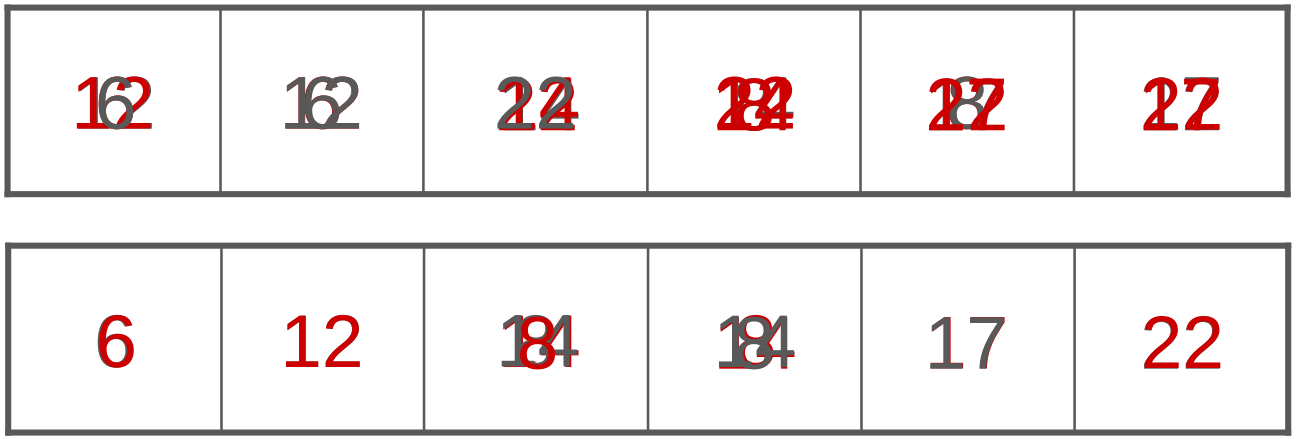
- Sort the list [10,−4,−1,20,13,−3] into ascending order using insertion sort
- 10, -4, -1, 20, 13, -3 Original List
- -4, 10, -1, 20, 13, -3 **1st Iteration**
- -4, -1, 10, 20, 13, -3 **2nd Iteration**
- -4, -1, 10, 20, 13, -3 **3rd Iteration**
- -4, -1, 10, 13, 20, -3 **4th Iteration**
- -4, -3, -1, 10, 13, 20 5th Iteration and Sorted List

Bubble Sort

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

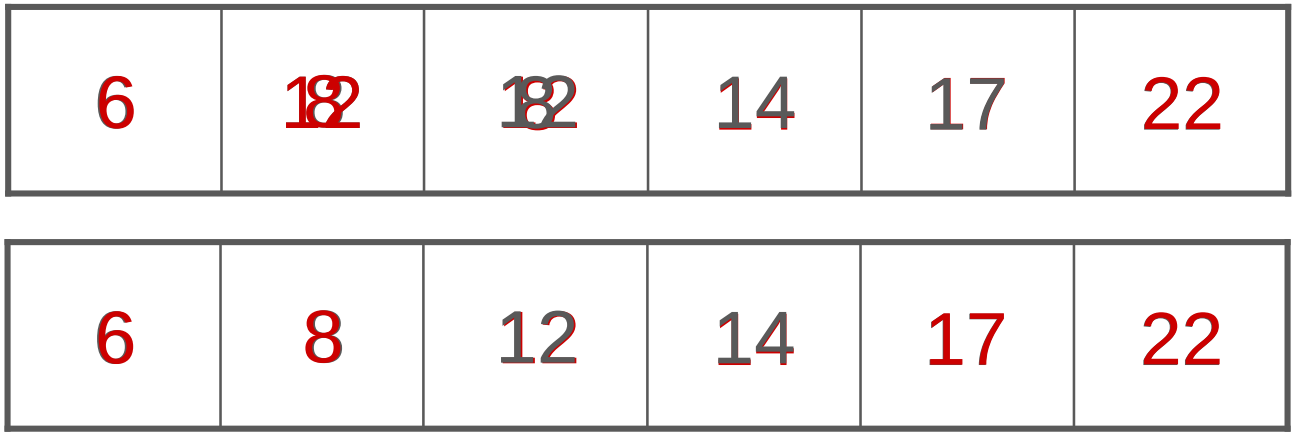
smaller elements “bubble” to the front of the list, larger sink to the end.

Bubble Sort



- Given n numbers to sort:
- Repeat the following n-1 times:
  - For each pair of adjacent numbers:
    - If the number on the left is greater than the number on the right, swap them.

Bubble Sort



- Given n numbers to sort:
- Repeat the following n-1 times:
  - For each pair of adjacent numbers:
    - If the number on the left is greater than the number on the right, swap them.

Bubble Sort Algorithm

```
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
}
```

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
```

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


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