

# CSIT881

## Programming and Data Structures

### Sorting Algorithm



UNIVERSITY  
OF WOLLONGONG  
AUSTRALIA

*Dr. Joseph Tonien*

# Objectives

- Insertion sort
- Selection sort
- Bubble sort

# Insertion sort

Have a look at this example, can you figure out the process of insertion sort?

round 1 start	[ 60, 10, 90, 50, 100, 80, 70, 30, 40, 20]
round 1 finish	[ 10, 60, 90, 50, 100, 80, 70, 30, 40, 20]
round 2 start	[ 10, 60, 90, 50, 100, 80, 70, 30, 40, 20]
round 2 finish	[ 10, 60, 90, 50, 100, 80, 70, 30, 40, 20]
round 3 start	[ 10, 60, 90, 50, 100, 80, 70, 30, 40, 20]
round 3 finish	[ 10, 50, 60, 90, 100, 80, 70, 30, 40, 20]
round 4 start	[ 10, 50, 60, 90, 100, 80, 70, 30, 40, 20]
round 4 finish	[ 10, 50, 60, 90, 100, 80, 70, 30, 40, 20]
round 5 start	[ 10, 50, 60, 90, 100, 80, 70, 30, 40, 20]
round 5 finish	[ 10, 50, 60, 80, 90, 100, 70, 30, 40, 20]
round 6 start	[ 10, 50, 60, 80, 90, 100, 70, 30, 40, 20]
round 6 finish	[ 10, 50, 60, 70, 80, 90, 100, 30, 40, 20]
round 7 start	[ 10, 50, 60, 70, 80, 90, 100, 30, 40, 20]
round 7 finish	[ 10, 30, 50, 60, 70, 80, 90, 100, 40, 20]
round 8 start	[ 10, 30, 50, 60, 70, 80, 90, 100, 40, 20]
round 8 finish	[ 10, 30, 40, 50, 60, 70, 80, 90, 100, 20]
round 9 start	[ 10, 30, 40, 50, 60, 70, 80, 90, 100, 20]
round 9 finish	[ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

# Insertion sort

At each round  $i$ : {item 0, item 1, ..., item  $i$ } are sorted

round 1 start	[60, 10, 90, 50, 100, 80, 70, 30, 40, 20]
round 1 finish	[10, 60, 90, 50, 100, 80, 70, 30, 40, 20]
round 2 start	[10, 60, 90, 50, 100, 80, 70, 30, 40, 20]
round 2 finish	[10, 60, 90, 50, 100, 80, 70, 30, 40, 20]
round 3 start	[10, 60, 90, 50, 100, 80, 70, 30, 40, 20]
round 3 finish	[10, 50, 60, 90, 100, 80, 70, 30, 40, 20]
round 4 start	[10, 50, 60, 90, 100, 80, 70, 30, 40, 20]
round 4 finish	[10, 50, 60, 90, 100, 80, 70, 30, 40, 20]
round 5 start	[10, 50, 60, 90, 100, 80, 70, 30, 40, 20]
round 5 finish	[10, 50, 60, 80, 90, 100, 70, 30, 40, 20]
round 6 start	[10, 50, 60, 80, 90, 100, 70, 30, 40, 20]
round 6 finish	[10, 50, 60, 70, 80, 90, 100, 30, 40, 20]
round 7 start	[10, 50, 60, 70, 80, 90, 100, 30, 40, 20]
round 7 finish	[10, 30, 50, 60, 70, 80, 90, 100, 40, 20]
round 8 start	[10, 30, 50, 60, 70, 80, 90, 100, 40, 20]
round 8 finish	[10, 30, 40, 50, 60, 70, 80, 90, 100, 20]
round 9 start	[10, 30, 40, 50, 60, 70, 80, 90, 100, 20]
round 9 finish	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

# Insertion sort

At each round  $i$ : {item 0, item 1, ..., item  $i$ } are sorted

round 1 start	[60, 10, 90, 50, 100, 80, 70, 30, 40, 20]
round 1 finish	[10, 60, 90, 50, 100, 80, 70, 30, 40, 20]
round 2 start	[10, 60, 90, 50, 100, 80, 70, 30, 40, 20]
round 2 finish	[10, 60, 90, 50, 100, 80, 70, 30, 40, 20]
round 3 start	[10, 60, 90, 50, 100, 80, 70, 30, 40, 20]
round 3 finish	[10, 50, 60, 90, 100, 80, 70, 30, 40, 20]
round 4 start	[10, 50, 60, 90, 100, 80, 70, 30, 40, 20]
round 4 finish	[10, 50, 60, 90, 100, 80, 70, 30, 40, 20]
round 5 start	[10, 50, 60, 90, 100, 80, 70, 30, 40, 20]
round 5 finish	[10, 50, 60, 80, 90, 100, 70, 30, 40, 20]

helicopter view pseudocode of Insertion Sort Algorithm of a list of integers

```
n = length-of(intList)
FOR i from 1 to (n-1)
    // sort intList[0], intList[1], ..., intList[i]
END FOR
```

# Insertion sort

Look at each round in details:

round 1 start [60, 10, 90, 50, 100, 80, 70, 30, 40, 20]

are 10 and 60 in order? No

swap 10 and 60: [10, 60, 90, 50, 100, 80, 70, 30, 40, 20]

round 1 finish [10, 60, 90, 50, 100, 80, 70, 30, 40, 20]

round 2 start [10, 60, 90, 50, 100, 80, 70, 30, 40, 20]

are 90 and 60 in order? Yes

round 2 finish [10, 60, 90, 50, 100, 80, 70, 30, 40, 20]

round 3 start [10, 60, 90, 50, 100, 80, 70, 30, 40, 20]

are 50 and 90 in order? No

swap 50 and 90: [10, 60, 50, 90, 100, 80, 70, 30, 40, 20]

are 50 and 60 in order? No

swap 50 and 60: [10, 50, 60, 90, 100, 80, 70, 30, 40, 20]

are 50 and 10 in order? Yes

round 3 finish [10, 50, 60, 90, 100, 80, 70, 30, 40, 20]

# Insertion sort

Look at each round in details:

```
round 4 start    [10, 50, 60, 90, 100, 80, 70, 30, 40, 20]
```

```
are 100 and 90 in order? Yes
```

```
round 4 finish   [10, 50, 60, 90, 100, 80, 70, 30, 40, 20]
```

```
round 5 start    [10, 50, 60, 90, 100, 80, 70, 30, 40, 20]
```

```
are 80 and 100 in order? No
```

```
swap 80 and 100: [10, 50, 60, 90, 80, 100, 70, 30, 40, 20]
```

```
are 80 and 90 in order? No
```

```
swap 80 and 90:  [10, 50, 60, 80, 90, 100, 70, 30, 40, 20]
```

```
are 80 and 60 in order? Yes
```

```
round 5 finish   [10, 50, 60, 80, 90, 100, 70, 30, 40, 20]
```

# Insertion sort

Look at each round in details:

```
round 6 start      [10, 50, 60, 80, 90, 100, 70, 30, 40, 20]

are 70 and 100 in order? No
swap 70 and 100: [10, 50, 60, 80, 90, 70, 100, 30, 40, 20]
are 70 and 90 in order? No
swap 70 and 90:  [10, 50, 60, 80, 70, 90, 100, 30, 40, 20]
are 70 and 80 in order? No
swap 70 and 80:  [10, 50, 60, 70, 80, 90, 100, 30, 40, 20]

are 70 and 60 in order? Yes
round 6 finish    [10, 50, 60, 70, 80, 90, 100, 30, 40, 20]
```



# Insertion sort

Look at each round in details:

```
round 7 start      [10, 50, 60, 70, 80, 90, 100, 30, 40, 20]

are 30 and 100 in order? No
swap 30 and 100: [10, 50, 60, 70, 80, 90, 30, 100, 40, 20]
are 30 and 90 in order? No
swap 30 and 90:  [10, 50, 60, 70, 80, 30, 90, 100, 40, 20]
are 30 and 80 in order? No
swap 30 and 80:  [10, 50, 60, 70, 30, 80, 90, 100, 40, 20]
are 30 and 70 in order? No
swap 30 and 70:  [10, 50, 60, 30, 70, 80, 90, 100, 40, 20]
are 30 and 60 in order? No
swap 30 and 60:  [10, 50, 30, 60, 70, 80, 90, 100, 40, 20]
are 30 and 50 in order? No
swap 30 and 50:  [10, 30, 50, 60, 70, 80, 90, 100, 40, 20]

are 30 and 10 in order? Yes
round 7 finish    [10, 30, 50, 60, 70, 80, 90, 100, 40, 20]
```

# Insertion sort

Look at each round in details:

```
round 8 start      [10, 30, 50, 60, 70, 80, 90, 100, 40, 20]

are 40 and 100 in order? No
swap 40 and 100: [10, 30, 50, 60, 70, 80, 90, 40, 100, 20]
are 40 and 90 in order? No
swap 40 and 90:  [10, 30, 50, 60, 70, 80, 40, 90, 100, 20]
are 40 and 80 in order? No
swap 40 and 80:  [10, 30, 50, 60, 70, 40, 80, 90, 100, 20]
are 40 and 70 in order? No
swap 40 and 70:  [10, 30, 50, 60, 40, 70, 80, 90, 100, 20]
are 40 and 60 in order? No
swap 40 and 60:  [10, 30, 50, 40, 60, 70, 80, 90, 100, 20]
are 40 and 50 in order? No
swap 40 and 50:  [10, 30, 40, 50, 60, 70, 80, 90, 100, 20]

are 40 and 30 in order? Yes
round 8 finish    [10, 30, 40, 50, 60, 70, 80, 90, 100, 20]
```

# Insertion sort

Look at each round in details:

```
round 9 start      [ 10, 30, 40, 50, 60, 70, 80, 90, 100, 20 ]

are 20 and 100 in order? No
swap 20 and 100: [10, 30, 40, 50, 60, 70, 80, 90, 20, 100]
are 20 and 90 in order? No
swap 20 and 90:  [10, 30, 40, 50, 60, 70, 80, 20, 90, 100]
are 20 and 80 in order? No
swap 20 and 80:  [10, 30, 40, 50, 60, 70, 20, 80, 90, 100]
are 20 and 70 in order? No
swap 20 and 70:  [10, 30, 40, 50, 60, 20, 70, 80, 90, 100]
are 20 and 60 in order? No
swap 20 and 60:  [10, 30, 40, 50, 20, 60, 70, 80, 90, 100]
are 20 and 50 in order? No
swap 20 and 50:  [10, 30, 40, 20, 50, 60, 70, 80, 90, 100]
are 20 and 40 in order? No
swap 20 and 40:  [10, 30, 20, 40, 50, 60, 70, 80, 90, 100]
are 20 and 30 in order? No
swap 20 and 30:  [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

are 20 and 10 in order? Yes
round 9 finish    [ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 ]
```

# Insertion sort

helicopter view pseudocode of Insertion Sort Algorithm of a list of integers

```
n = length-of(intList)
FOR i from 1 to (n-1)
    // sort intList[0], intList[1], ..., intList[i]
END FOR
```

# Insertion sort

At each round  $i$ : {item 0, item 1, ..., item  $i$ } are sorted

pseudocode of round  $i$

```
// sort intList[0], intList[1], ..., intList[i]
k = i
WHILE k > 0 and intList[k-1] > intList[k]
    swap intList[k] and intList[k-1]
    k = k - 1
END WHILE
```

# Insertion sort

Put it together, we have the algorithm for Insertion Sort:

pseudocode

```
n = length-of(intList)
FOR i from 1 to (n-1)
    // sort intList[0], intList[1], ..., intList[i]
    k = i
    WHILE k > 0 and intList[k-1] > intList[k]
        swap intList[k] and intList[k-1]
        k = k - 1
    END WHILE
END FOR
```

# Insertion sort

Python implementation

```
def insertionSort(intList):  
    n = len(intList)  
    for i in range(1, n):  
        #{  
            # sort intList[0], intList[1], ..., intList[i]  
            k = i  
            while (k > 0) and (intList[k-1] > intList[k]):  
                #{  
                    # swap intList[k] and intList[k-1]  
                    temp = intList[k]  
                    intList[k] = intList[k-1]  
                    intList[k-1] = temp  
  
                    k = k - 1  
                #}  
            #}  
        #}
```

# Insertion sort

Suggested activities:

- Make up a list of integers and write down in details each step in sorting this list of integers;
- Sort a list of integers in descending order;
- Sort a list of decimal numbers;
- Sort a list of strings.



# Python List

A list/array is used to hold a list of items:

```
animal_list = ["dog", "cat", "frog"]
```

```
fibonacci_numbers = [0, 1, 1, 2, 3, 5, 8, 13]
```

```
prime_numbers = [2, 3, 5, 7, 11, 13, 17]
```

```
subject_list = ["MATH101", "CS222", "PHY102", "ACCY203"]
```

```
selected_products = [] # this is an empty list
```

This is how we define a list:

```
list_variable = [item1, item2, ..., itemN]
```

# Python List

List items can be accessed via **index**:

```
fibo_numbers = [0, 1, 1, 2, 3, 5, 8, 13]
```

```
print(fibo_numbers[0])    → 0
print(fibo_numbers[1])    → 1
print(fibo_numbers[2])    → 1
print(fibo_numbers[3])    → 2
print(fibo_numbers[4])    → 3
print(fibo_numbers[5])    → 5
print(fibo_numbers[6])    → 8
print(fibo_numbers[7])    → 13
```

items can be **appended** to the end of the list:

```
fibo_numbers.append(21)
fibo_numbers.append(34)
fibo_numbers.append(55)
fibo_numbers.append(89)
```

# Python List

using `len` to find out how many items in the list:

```
animal_list = ["dog", "cat", "frog"]
```

```
animal_count = len(animal_list) → 3
```

Note that `len(animal_list)` is 3, but the **last index** is 2 because the index start at 0.

```
print(animal_list[0])    → "dog"  
print(animal_list[1])    → "cat"  
print(animal_list[2])    → "frog"
```

# Selection sort

Have a look at this example, can you figure out the process of selection sort?

round 0 start	[ 60, 10, 90, 50, 100, 80, 70, 30, 40, 20 ]
round 0 finish	[ 10, 60, 90, 50, 100, 80, 70, 30, 40, 20 ]
round 1 start	[10, 60, 90, 50, 100, 80, 70, 30, 40, 20]
round 1 finish	[10, 20, 90, 50, 100, 80, 70, 30, 40, 60]
round 2 start	[10, 20, 90, 50, 100, 80, 70, 30, 40, 60]
round 2 finish	[10, 20, 30, 50, 100, 80, 70, 90, 40, 60]
round 3 start	[10, 20, 30, 50, 100, 80, 70, 90, 40, 60]
round 3 finish	[10, 20, 30, 40, 100, 80, 70, 90, 50, 60]
round 4 start	[10, 20, 30, 40, 100, 80, 70, 90, 50, 60]
round 4 finish	[10, 20, 30, 40, 50, 80, 70, 90, 100, 60]
round 5 start	[10, 20, 30, 40, 50, 80, 70, 90, 100, 60]
round 5 finish	[10, 20, 30, 40, 50, 60, 70, 90, 100, 80]
round 6 start	[10, 20, 30, 40, 50, 60, 70, 90, 100, 80]
round 6 finish	[10, 20, 30, 40, 50, 60, 70, 90, 100, 80]
round 7 start	[10, 20, 30, 40, 50, 60, 70, 90, 100, 80]
round 7 finish	[10, 20, 30, 40, 50, 60, 70, 80, 100, 90]
round 8 start	[10, 20, 30, 40, 50, 60, 70, 80, 100, 90]
round 8 finish	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

# Selection sort

At each round  $i$ : find the minimum in  $\{\text{item } i, \text{item } i+1, \dots\}$   
and swap it to the position  $i$

round 0 start	[ 60, 10, 90, 50, 100, 80, 70, 30, 40, 20 ]
round 0 finish	[ 10, 60, 90, 50, 100, 80, 70, 30, 40, 20 ]
round 1 start	[10, 60, 90, 50, 100, 80, 70, 30, 40, 20]
round 1 finish	[10, 20, 90, 50, 100, 80, 70, 30, 40, 60]
round 2 start	[10, 20, 90, 50, 100, 80, 70, 30, 40, 60]
round 2 finish	[10, 20, 30, 50, 100, 80, 70, 90, 40, 60]
round 3 start	[10, 20, 30, 50, 100, 80, 70, 90, 40, 60]
round 3 finish	[10, 20, 30, 40, 100, 80, 70, 90, 50, 60]
round 4 start	[10, 20, 30, 40, 100, 80, 70, 90, 50, 60]
round 4 finish	[10, 20, 30, 40, 50, 80, 70, 90, 100, 60]
round 5 start	[10, 20, 30, 40, 50, 80, 70, 90, 100, 60]
round 5 finish	[10, 20, 30, 40, 50, 60, 70, 90, 100, 80]
round 6 start	[10, 20, 30, 40, 50, 60, 70, 90, 100, 80]
round 6 finish	[10, 20, 30, 40, 50, 60, 70, 90, 100, 80]
round 7 start	[10, 20, 30, 40, 50, 60, 70, 90, 100, 80]
round 7 finish	[10, 20, 30, 40, 50, 60, 70, 80, 100, 90]
round 8 start	[10, 20, 30, 40, 50, 60, 70, 80, 100, 90]
round 8 finish	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

# Selection sort

Look at each round in details:

```
round 0 start    [ 60, 10, 90, 50, 100, 80, 70, 30, 40, 20 ]
```

```
find the minimum item in intList[0..9]
```

```
found minimum item: intList[1] = 10
```

```
swap intList[0] and intList[1]
```

```
round 0 finish   [ 10, 60, 90, 50, 100, 80, 70, 30, 40, 20 ]
```

```
round 1 start    [ 10, 60, 90, 50, 100, 80, 70, 30, 40, 20 ]
```

```
find the minimum item in intList[1..9]
```

```
found minimum item: intList[9] = 20
```

```
swap intList[1] and intList[9]
```

```
round 1 finish   [ 10, 20, 90, 50, 100, 80, 70, 30, 40, 60 ]
```

```
round 2 start    [ 10, 20, 90, 50, 100, 80, 70, 30, 40, 60 ]
```

```
find the minimum item in intList[2..9]
```

```
found minimum item: intList[7] = 30
```

```
swap intList[2] and intList[7]
```

```
round 2 finish   [ 10, 20, 30, 50, 100, 80, 70, 90, 40, 60 ]
```

# Selection sort

Look at each round in details:

```
round 3 start    [10, 20, 30, 50, 100, 80, 70, 90, 40, 60]
```

```
find the minimum item in intList[3..9]
```

```
found minimum item: intList[8] = 40
```

```
swap intList[3] and intList[8]
```

```
round 3 finish   [10, 20, 30, 40, 100, 80, 70, 90, 50, 60]
```

```
round 4 start    [10, 20, 30, 40, 100, 80, 70, 90, 50, 60]
```

```
find the minimum item in intList[4..9]
```

```
found minimum item: intList[8] = 50
```

```
swap intList[4] and intList[8]
```

```
round 4 finish   [10, 20, 30, 40, 50, 80, 70, 90, 100, 60]
```

```
round 5 start    [10, 20, 30, 40, 50, 80, 70, 90, 100, 60]
```

```
find the minimum item in intList[5..9]
```

```
found minimum item: intList[9] = 60
```

```
swap intList[5] and intList[9]
```

```
round 5 finish   [10, 20, 30, 40, 50, 60, 70, 90, 100, 80]
```

# Selection sort

Look at each round in details:

```
round 6 start    [10, 20, 30, 40, 50, 60, 70, 90, 100, 80]
```

```
find the minimum item in intList[6..9]
```

```
found minimum item: intList[6] = 70
```

```
swap intList[6] and intList[6] (so basically: do nothing)
```

```
round 6 finish   [10, 20, 30, 40, 50, 60, 70, 90, 100, 80]
```

```
round 7 start    [10, 20, 30, 40, 50, 60, 70, 90, 100, 80]
```

```
find the minimum item in intList[7..9]
```

```
found minimum item: intList[9] = 80
```

```
swap intList[7] and intList[9]
```

```
round 7 finish   [10, 20, 30, 40, 50, 60, 70, 80, 100, 90]
```

```
round 8 start    [10, 20, 30, 40, 50, 60, 70, 80, 100, 90]
```

```
find the minimum item in intList[8..9]
```

```
found minimum item: intList[9] = 90
```

```
swap intList[8] and intList[9]
```

```
round 8 finish   [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```



# Selection sort

Algorithm for Selection Sort:

pseudocode

```
n = length-of(intList)
FOR i from 0 to (n-2)
    among intList[i], intList[i+1], ..., intList[n-1]
    find the minimum item intList[kMin]

    swap intList[i] and intList[kMin]
END FOR
```

# Selection sort

Python implementation

```
def selectionSort(intList):
    n = len(intList)
    for i in range(0, n-1):
        #{
            # find the minimum item in intList[i .. n-1]
            kMin = i
            for k in range(i+1, n):
                if (intList[k] < intList[kMin]):
                    kMin = k

            # swap intList[i] and intList[kMin]
            if (kMin != i):
                temp = intList[i]
                intList[i] = intList[kMin]
                intList[kMin] = temp

        #}
```

# Bubble sort

Bubble (up) sort algorithm:

- Go through the list, compares adjacent items and swaps them if they are in the wrong order;
- Repeat this process until the list is sorted.

The name of the algorithm is derived from the fact that: after each round, the largest items are **bubbled up** towards the end of the list.

# Bubble sort

Let's look at each round in details:

*(compares adjacent items and swaps them if they are in the wrong order)*

```
round 0 start [60, 10, 90, 50, 100, 80, 70, 30, 40, 20]
               [10, 60, 90, 50, 100, 80, 70, 30, 40, 20]
               [10, 60, 90, 50, 100, 80, 70, 30, 40, 20]
               [10, 60, 50, 90, 100, 80, 70, 30, 40, 20]
               [10, 60, 50, 90, 100, 80, 70, 30, 40, 20]
               [10, 60, 50, 90, 80, 100, 70, 30, 40, 20]
               [10, 60, 50, 90, 80, 70, 100, 30, 40, 20]
               [10, 60, 50, 90, 80, 70, 30, 100, 40, 20]
               [10, 60, 50, 90, 80, 70, 30, 40, 100, 20]
round 0 end    [10, 60, 50, 90, 80, 70, 30, 40, 20, 100]
```

*Observe the movement of the largest item 100*

We can see that after the 1st round, the largest item 100 is **bubbled up** to the end of the list.

# Bubble sort

Let's look at each round in details:

*(compares adjacent items and swaps them if they are in the wrong order)*

round 1 start	[10, 60, 50, 90, 80, 70, 30, 40, 20, 100]
	[10, 60, 50, 90, 80, 70, 30, 40, 20, 100]
	[10, 50, 60, 90, 80, 70, 30, 40, 20, 100]
	[10, 50, 60, 90, 80, 70, 30, 40, 20, 100]
	[10, 50, 60, 80, 90, 70, 30, 40, 20, 100]
	[10, 50, 60, 80, 70, 90, 30, 40, 20, 100]
	[10, 50, 60, 80, 70, 30, 90, 40, 20, 100]
	[10, 50, 60, 80, 70, 30, 40, 90, 20, 100]
round 1 end	[10, 50, 60, 80, 70, 30, 40, 20, 90, 100]

We can see that after the 2st round, the 2nd largest item 90 is **bubbled up** to the right place towards the end of the list.

# Bubble sort

Let's look at each round in details:

*(compares adjacent items and swaps them if they are in the wrong order)*

round 2 start	[10, 50, 60, 80, 70, 30, 40, 20, 90, 100]
	[10, 50, 60, 80, 70, 30, 40, 20, 90, 100]
	[10, 50, 60, 80, 70, 30, 40, 20, 90, 100]
	[10, 50, 60, 80, 70, 30, 40, 20, 90, 100]
	[10, 50, 60, 70, 80, 30, 40, 20, 90, 100]
	[10, 50, 60, 70, 30, 80, 40, 20, 90, 100]
	[10, 50, 60, 70, 30, 40, 80, 20, 90, 100]
round 2 end	[10, 50, 60, 70, 30, 40, 20, 80, 90, 100]

We can see that after the 3rd round, the 3rd largest item **80** is **bubbled up** to the right place towards the end of the list.

# Bubble sort

Let's look at each round in details:

*(compares adjacent items and swaps them if they are in the wrong order)*

round 3 start	[10, 50, 60, 70, 30, 40, 20, 80, 90, 100]
	[10, 50, 60, 70, 30, 40, 20, 80, 90, 100]
	[10, 50, 60, 70, 30, 40, 20, 80, 90, 100]
	[10, 50, 60, 70, 30, 40, 20, 80, 90, 100]
	[10, 50, 60, 30, 70, 40, 20, 80, 90, 100]
	[10, 50, 60, 30, 40, 70, 20, 80, 90, 100]
round 3 end	[10, 50, 60, 30, 40, 20, 70, 80, 90, 100]

We can see that after the 4th round, the 4th largest item **70** is **bubbled up** to the right place towards the end of the list.

# Bubble sort

Let's look at each round in details:

```
round 4 start [10, 50, 60, 30, 40, 20, 70, 80, 90, 100]
               [10, 50, 60, 30, 40, 20, 70, 80, 90, 100]
               [10, 50, 60, 30, 40, 20, 70, 80, 90, 100]
               [10, 50, 30, 60, 40, 20, 70, 80, 90, 100]
               [10, 50, 30, 40, 60, 20, 70, 80, 90, 100]
round 4 end   [10, 50, 30, 40, 20, 60, 70, 80, 90, 100]
```

After the 5th round, the 5th largest item **60** is **bubbled up** to the right place towards the end of the list.

```
round 5 start [10, 50, 30, 40, 20, 60, 70, 80, 90, 100]
               [10, 50, 30, 40, 20, 60, 70, 80, 90, 100]
               [10, 30, 50, 40, 20, 60, 70, 80, 90, 100]
               [10, 30, 40, 50, 20, 60, 70, 80, 90, 100]
round 5 end   [10, 30, 40, 20, 50, 60, 70, 80, 90, 100]
```

After the 6th round, the 6th largest item **50** is **bubbled up** to the right place towards the end of the list.



# Bubble sort

Let's look at each round in details:

```
round 6 start [10, 30, 40, 20, 50, 60, 70, 80, 90, 100]
               [10, 30, 40, 20, 50, 60, 70, 80, 90, 100]
               [10, 30, 40, 20, 50, 60, 70, 80, 90, 100]
round 6 end   [10, 30, 20, 40, 50, 60, 70, 80, 90, 100]
```

After the 7th round, the 7th largest item **40** is **bubbled up** to the right place towards the end of the list.

```
round 7 start [10, 30, 20, 40, 50, 60, 70, 80, 90, 100]
               [10, 30, 20, 40, 50, 60, 70, 80, 90, 100]
round 7 end   [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

After the 8th round, the 8th largest item **30** is **bubbled up** to the right place towards the end of the list.

```
round 8 start [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
round 8 end   [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

After the 9th round, the 9th largest item **20** is **bubbled up** to the right place towards the end of the list - and the sorting is **DONE!!!**

# Bubble sort

pseudocode

```
n = length-of(intList)
FOR i from 0 to (n-2)
    FOR j from 1 to (n-i-1)
        // compare adj items, swap if in wrong order
        IF intList[j-1] > intList[j]:
            swap intList[j-1] and intList[j]
        END IF
    END FOR
END FOR
```

# Bubble sort

Python implementation

```
def bubbleSort(intList):  
    n = len(intList)  
    for i in range(0, n-1):  
        #{  
            for j in range(1, n-i):  
                #{  
                    # compare adj items, swap if in wrong order  
                    if intList[j-1] > intList[j]:  
                        # swap intList[j-1] and intList[j]  
                        temp = intList[j-1]  
                        intList[j-1] = intList[j]  
                        intList[j] = temp  
                #}  
            #}  
        #}
```

# Bubble sort

Let's look at another example:

```
round 0 start [ 90, 100, 10, 20, 30, 40, 50, 60, 70, 80]
               [90, 100, 10, 20, 30, 40, 50, 60, 70, 80]
               [90, 10, 100, 20, 30, 40, 50, 60, 70, 80]
               [90, 10, 20, 100, 30, 40, 50, 60, 70, 80]
               [90, 10, 20, 30, 100, 40, 50, 60, 70, 80]
               [90, 10, 20, 30, 40, 100, 50, 60, 70, 80]
               [90, 10, 20, 30, 40, 50, 100, 60, 70, 80]
               [90, 10, 20, 30, 40, 50, 60, 100, 70, 80]
               [90, 10, 20, 30, 40, 50, 60, 70, 100, 80]
round 0 end   [90, 10, 20, 30, 40, 50, 60, 70, 80, 100]
```

```
round 1 start [ 90, 10, 20, 30, 40, 50, 60, 70, 80, 100]
               [10, 90, 20, 30, 40, 50, 60, 70, 80, 100]
               [10, 20, 90, 30, 40, 50, 60, 70, 80, 100]
               [10, 20, 30, 90, 40, 50, 60, 70, 80, 100]
               [10, 20, 30, 40, 90, 50, 60, 70, 80, 100]
               [10, 20, 30, 40, 50, 90, 60, 70, 80, 100]
               [10, 20, 30, 40, 50, 60, 90, 70, 80, 100]
               [10, 20, 30, 40, 50, 60, 70, 90, 80, 100]
round 1 end   [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

# Bubble sort

round 2 start	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
round 2 end	[10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

Notice that in round 2, NOT a single swap is needed. It means that the list has already been sorted. NO NEED TO GO ANY FURTHER TO round 3, round 4, round 5, ...

# Bubble sort

Better algorithm:

pseudocode

```
n = length-of(intList)
FOR i from 0 to (n-2)
    swapped = false
    FOR j from 1 to (n-i-1)
        // compare adj items, swap if in wrong order
        IF intList[j-1] > intList[j]:
            swap intList[j-1] and intList[j]
            # remember that swap is needed
            swapped = true
        END IF
    END FOR
    BREAK IF swapped = false
END FOR
```

# Bubble sort

Python implementation

better algorithm

```
def bubbleSort(intList):  
    n = len(intList)  
    for i in range(0, n-1):  
        swapped = False  
        for j in range(1, n-i):  
            # compare adj items, swap if in wrong order  
            if intList[j-1] > intList[j]:  
                # swap intList[j-1] and intList[j]  
                temp = intList[j-1]  
                intList[j-1] = intList[j]  
                intList[j] = temp  
                # remember that swap is needed  
                swapped = True  
        if not swapped:  
            # swap is NOT needed, so list is SORTED  
            break
```

# Bubble sort

Suggested activities:

- Make up a list of integers and write down in details each step in sorting this list of integers;
- Sort a list of integers in descending order;
- Write a Bubble (down) sort algorithm, so that after each round, the smallest items are **bubbled down** towards the start of the list;



## Suggested activities:

- Write a program to generate a random list of integers of length  $N$ ;
- Write a program to count how many comparison operations, and how many swap operations are needed to sort this random list using each sorting algorithms;
- Repeat this program many times with a large sample of random lists of integers and display the statistics.

# References

- Python 3 documentation  
<https://docs.python.org/3/>
- NumPy Reference  
<https://numpy.org/doc/stable/reference/>