

# Sorting

BBM 101 - Introduction to Programming I

Hacettepe University  
Fall 2016

Fuat Akal, Aykut Erdem, Erkut Erdem

Slides based on material prepared by Ruth Anderson, Michael Ernst and Bill Howe in the course CSE 140  
University of Washington

1

# Sorting

```
hamlet = "to be or not to be that is the  
question whether tis nobler in the mind to  
suffer".split()
```

```
print("hamlet:", hamlet)
```

```
print("sorted(hamlet):", sorted(hamlet))  
print("hamlet:", hamlet)
```

```
print("hamlet.sort():", hamlet.sort())  
print("hamlet:", hamlet)
```

- Lists are **mutable** – they can be changed
  - including by functions

2

# Sorting

```
hamlet: ['to', 'be', 'or', 'not', 'to', 'be', 'that',  
'is', 'the', 'question', 'whether', 'tis', 'nobler',  
'in', 'the', 'mind', 'to', 'suffer']
```

```
sorted(hamlet): ['be', 'be', 'in', 'is', 'mind',  
'nobler', 'not', 'or', 'question', 'suffer', 'that',  
'the', 'the', 'tis', 'to', 'to', 'to', 'whether']
```

```
hamlet: ['to', 'be', 'or', 'not', 'to', 'be', 'that',  
'is', 'the', 'question', 'whether', 'tis', 'nobler',  
'in', 'the', 'mind', 'to', 'suffer']
```

```
hamlet.sort(): None
```

```
hamlet: ['be', 'be', 'in', 'is', 'mind', 'nobler',  
'not', 'or', 'question', 'suffer', 'that', 'the',  
'the', 'tis', 'to', 'to', 'to', 'whether']
```

3

# Customizing the Sort Order

Goal: sort a list of names *by last name*

```
names = ["Isaac Newton", "Albert Einstein", "Niels  
Bohr", "Marie Curie", "Charles Darwin", "Louis  
Pasteur", "Galileo Galilei", "Margaret Mead"]
```

```
print("names:", names)
```

```
sorted(names): ['Albert Einstein', 'Charles  
Darwin', 'Galileo Galilei', 'Isaac Newton',  
'Louis Pasteur', 'Margaret Mead', 'Marie  
Curie', 'Niels Bohr']
```

This does NOT work:

```
print("sorted(names):", sorted(names))
```

When sorting, how should we compare these names?

```
"Niels Bohr"  
"Charles Darwin"
```

4

## Sort Key

A **sort key** is a different value that you use to sort a list, instead of the actual values in the list

```
def last_name(str):  
    return str.split(" ")[1]
```

```
print('last name("Isaac Newton"):',  
      last_name("Isaac Newton"))
```

Two ways to use a sort key:

1. Create a new list containing the sort key, and then sort it
2. Pass a key function to the sorted function

5

## 1. Use a Sort Key to Create a New List

Create a **different list** that contains the sort key, sort it, then extract the relevant part:

```
names = ["Isaac Newton", "Fred Newton", "Niels Bohr"]  
# keyed_names is a list of [lastname, fullname] lists  
keyed_names = []  
for name in names:  
    keyed_names.append([last_name(name), name])
```

1) Create the new list.

Take a look at the list you created, it can now be sorted:

```
print("keyed_names:", keyed_names)  
print("sorted(keyed_names):", sorted(keyed_names))  
print("sorted(keyed_names, reverse = True):")  
print(sorted(keyed_names, reverse = True), sorted_names: ["Isaac Newton", "Fred Newton", "Niels Bohr"]  
(This works because Python compares two elements that are lists elementwise.)
```

2) Sort the list new list.

```
sorted_keyed_names = sorted(keyed_names, reverse = True)  
sorted_names = []  
for keyed_name in sorted_keyed_names:  
    sorted_names.append(keyed_name[1])  
print("sorted_names:", sorted_names)
```

3) Extract the relevant part.

6

## 1. Use a Sort Key to Create a New List

Create a **different list** that contains the sort key, sort it, then extract the relevant part:

```
names = ["Isaac Newton", "Fred Newton", "Niels Bohr"]  
# keyed_names is a list of [lastname, fullname] lists  
keyed_names = []  
for name in names:  
    keyed_names.append([last_name(name), name])
```

```
keyed_names: [['Newton', 'Isaac Newton'], ['Newton', 'Fred Newton'],  
              ['Bohr', 'Niels Bohr']]  
sorted(keyed_names): [['Bohr', 'Niels Bohr'], ['Newton', 'Fred Newton'],  
                     ['Newton', 'Isaac Newton']]  
sorted(keyed_names, reverse = True): [['Newton', 'Isaac Newton'],  
                                     ['Newton', 'Fred Newton'], ['Bohr', 'Niels Bohr']]
```

Take a look at the list you created, it can now be sorted:

```
print("keyed_names:", keyed_names)  
print("sorted(keyed_names):", sorted(keyed_names))  
print("sorted(keyed_names, reverse = True):")  
print(sorted(keyed_names, reverse = True), sorted_names: ["Isaac Newton", "Fred Newton", "Niels Bohr"]  
(This works because Python compares two elements that are lists elementwise.)
```

2) Sort the list new list.

```
sorted_keyed_names = sorted(keyed_names, reverse = True)  
sorted_names = []  
for keyed_name in sorted_keyed_names:  
    sorted_names.append(keyed_name[1])  
print("sorted_names:", sorted_names)
```

3) Extract the relevant part.

7

## 2. Use a Sort Key as the Key Argument

Supply the **key argument** to the **sorted** function or the **sort** function

```
def last_name(str):  
    return str.split(" ")[1]  
  
names = ["Isaac Newton", "Fred Newton", "Niels Bohr"]  
print("sorted(names, key = last_name):")  
print(sorted(names, key = last_name))  
  
print("sorted(names, key = last_name, reverse = True):")  
print(sorted(names, key = last_name, reverse = True))  
  
print(sorted(names, key = len))  
  
def last_name_len(name):  
    return len(last_name(name))  
  
print(sorted(names, key = last_name_len))
```

8

## 2. Use a Sort Key as the Key Argument

Supply the **key** argument to the `sorted` function or the `sort` function

```
def last_name(str):
    return str.split(" ")[1]
names = ["Isaac Newton", "Fred Newton", "Niels Bohr"]
print("sorted(names, key = last_name):")
print(sorted(names, key = last_name))
print("sorted(names, key = last_name, reverse=True):")
print(sorted(names, key = last_name, reverse=True))
print("sorted(names, key = last_name_len):")
print(sorted(names, key = last_name_len))

def last_name_len(name):
    return len(last_name(name))
```

9

## itemgetter is a Function that Returns a Function

```
import operator
All: ('m','i','k','e')

print(operator.itemgetter(2, 7, 9, 10)("dumbstricken"))
operator.itemgetter(2, 5, 7, 9)("homesickness")
operator.itemgetter(2, 7, 9, 10)("pumpernickel")
operator.itemgetter(2, 3, 6, 7)("seminaked")
operator.itemgetter(1, 2, 4, 5)("smirker")

operator.itemgetter(9, 7, 6, 1)("beatnikism")
operator.itemgetter(14, 13, 5, 1)("Gedankenexperiment")
operator.itemgetter(12, 10, 9, 5)("mountebankism")
```

10

## Using itemgetter

```
from operator import itemgetter
student_score = ('Robert', 8)
itemgetter(0)(student_score) ➤ "Robert"
itemgetter(1)(student_score) ➤ 8

student_scores = [('Robert', 8), ('Alice', 9), ('Tina', 7)]
```

- Sort the list by **name**:  
`sorted(student_scores, key=itemgetter(0) )`
- Sort the list by **score**:  
`sorted(student_scores, key=itemgetter(1) )`

11

## Two Ways to Import itemgetter

```
from operator import itemgetter
student_score = ('Robert', 8)
itemgetter(0)(student_score) ➤ "Robert"
itemgetter(1)(student_score) ➤ 8
```

Or

```
import operator
student_score = ('Robert', 8)
operator.itemgetter(0)(student_score) ➤ "Robert"
operator.itemgetter(1)(student_score) ➤ 8
```

12

## Sorting Based on Two Criteria

Two approaches:

Approach #1: Use an itemgetter with two arguments

Approach #2: Sort twice (most important sort *last*)

```
student_scores = [('Robert', 8), ('Alice', 9),  
                  ('Tina', 10), ('James', 8)]
```

**Goal:** sort based on score;  
if there is a tie within score, sort by name

Approach #1:

```
sorted(student_scores, key=itemgetter(1,0))
```

Approach #2:

```
sorted_by_name = sorted(student_scores, key=itemgetter(0))  
sorted_by_score = sorted(sorted_by_name, key=itemgetter(1))
```

13

## Sort on Most Important Criteria LAST

- Sorted by score (ascending), when there is a tie on score, sort using name

```
from operator import itemgetter  
student_scores = [('Robert', 8), ('Alice', 9), ('Tina', 10),  
                  ('James', 8)]
```

```
sorted_by_name = sorted(student_scores, key=itemgetter(0))  
>>> sorted_by_name  
[('Alice', 9), ('James', 8), ('Robert', 8), ('Tina', 10)]
```

```
sorted_by_score = sorted(sorted_by_name, key=itemgetter(1))  
>>> sorted_by_score  
[('James', 8), ('Robert', 8), ('Alice', 9), ('Tina', 10)]
```

14

## More Sorting Based on Two Criteria

If you want to sort different criteria in different directions, you must use multiple calls to `sort` or `sorted`

```
student_scores = [('Robert', 8), ('Alice', 9), ('Tina', 10),  
                  ('James', 8)]
```

**Goal:** sort score from **highest to lowest**; if there is a tie within score, sort by name alphabetically (= **lowest to highest**)

```
sorted_by_name = sorted(student_scores, key=itemgetter(0) )  
sorted_by_hi_score = sorted(sorted_by_name,  
                             key=itemgetter(1), reverse=True)
```

15

## Sorting: strings vs. numbers

- Sorting the powers of 5:

```
>>> sorted([125, 5, 3125, 625, 25])  
[5, 25, 125, 625, 3125]  
>>> sorted(["125", "5", "3125", "625", "25"])  
['125', '25', '3125', '5', '625']
```

16

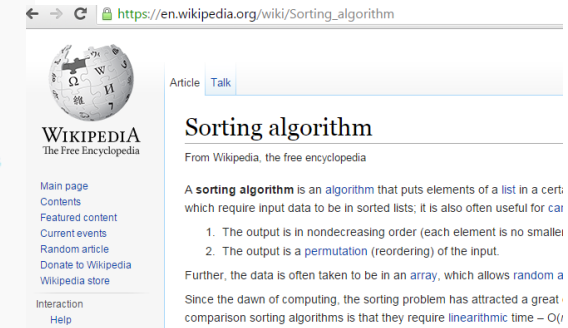
# Sorting



from BBC Documentary: *The Secret Rules of Modern Living Algorithms* 17

# Different sorting algorithms

- 3.1 Simple sorts
  - 3.1.1 Insertion sort
  - 3.1.2 Selection sort
- 3.2 Efficient sorts
  - 3.2.1 Merge sort
  - 3.2.2 Heapsort
  - 3.2.3 Quicksort
- 3.3 Bubble sort and variants
  - 3.3.1 Bubble sort
  - 3.3.2 Shell sort
  - 3.3.3 Comb sort
- 3.4 Distribution sort
  - 3.4.1 Counting sort
  - 3.4.2 Bucket sort
  - 3.4.3 Radix sort



# Bubble Sort

- It repeatedly steps through the list to be sorted,
- compares each pair of adjacent items and swaps them if they are in the wrong order.
- The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted.
- The algorithm, which is a comparison sort, is named for the way smaller elements "bubble" to the top of the list.

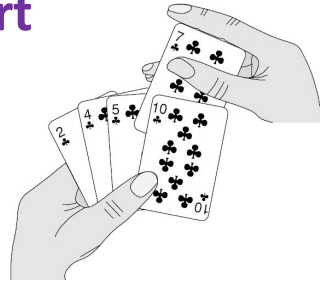
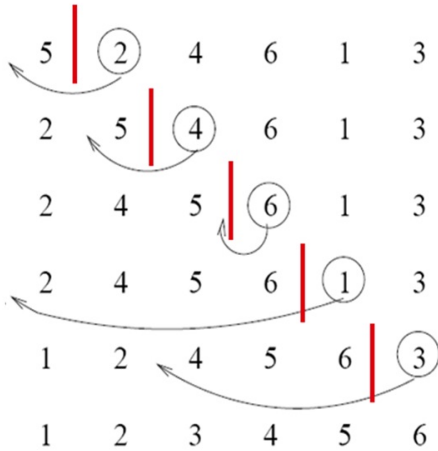


# Bubble Sort

```
def bubbleSort(alist):  
    for passnum in range(len(alist)-1,0,-1):  
        for i in range(passnum):  
            if alist[i]>alist[i+1]:  
                temp = alist[i]  
                alist[i] = alist[i+1]  
                alist[i+1] = temp  
  
alist = [54,26,93,17,77,31,44,55,20]  
bubbleSort(alist)  
print(alist)
```

## Insertion sort

• Idea:



- maintain a sorted sublist in the lower positions of the list.
- Each new item is then "inserted" back into the previous sublist such that the sorted sublist is one item larger.

Done !

21

## Insertion sort

```
def insertionSort(alist):
    for index in range(1,len(alist)):
        currentvalue = alist[index]
        position = index

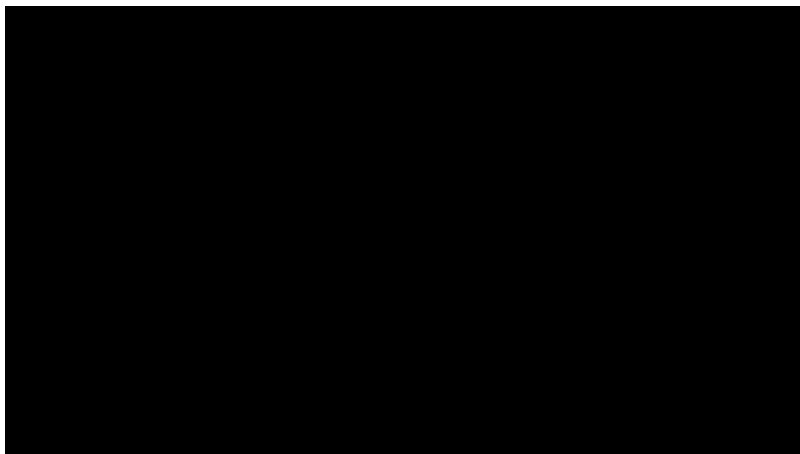
        while position>0 and alist[position-1]>currentvalue:
            alist[position]=alist[position-1]
            position = position-1

        alist[position]=currentvalue

alist = [54,26,93,17,77,31,44,55,20]
insertionSort(alist)
print(alist)
```

22

## Insertion sort



<https://www.youtube.com/watch?v=ROaIU379I3U>

23

www.sorting-algorithms.com

### Sorting Algorithm Animations

Problem Size: [20](#) · [30](#) · [40](#) · [50](#) Magnification: [1x](#) · [2x](#) · [3x](#)

Algorithm: [Insertion](#) · [Selection](#) · [Bubble](#) · [Shell](#) · [Merge](#) · [Heap](#) · [Quick](#) · [Quick3](#)

Initial Condition: [Random](#) · [Nearly Sorted](#) · [Reversed](#) · [Few Unique](#)

	Insertion	Selection	Bubble	Shell	Merge	Heap	Quick	Quick3
<a href="#">Random</a>								
<a href="#">Nearly Sorted</a>								
<a href="#">Reversed</a>								
<a href="#">Few Unique</a>								

24