Algorithms

Algorithms

Sorting Lists

Summary of Course Topics

Summary of Course Topics

1. Basics of computers and how to create/run a Python script

Summary of Course Topics

1. Basics of computers and how to create/run a Python script
2. Basic Python commands and program structure

Summary of Course Topics

1. Basics of computers and how to create/run a Python script
2. Basic Python commands and program structure
3. Basic datatypes, operators, and I/O

Summary of Course Topics

1. Basics of computers and how to create/run a Python script
2. Basic Python commands and program structure
3. Basic datatypes, operators, and I/O
4. if-statements, while-loops, and Boolean logic

Summary of Course Topics

1. Basics of computers and how to create/run a Python script
2. Basic Python commands and program structure
3. Basic datatypes, operators, and I/O
4. if-statements, while-loops, and Boolean logic
5. for-loops and nested loops

Summary of Course Topics

1. Basics of computers and how to create/run a Python script
2. Basic Python commands and program structure
3. Basic datatypes, operators, and I/O
4. if-statements, while-loops, and Boolean logic
5. for-loops and nested loops
6. functions, modules, and procedural programming

Summary of Course Topics

1. Basics of computers and how to create/run a Python script
2. Basic Python commands and program structure
3. Basic datatypes, operators, and I/O
4. if-statements, while-loops, and Boolean logic
5. for-loops and nested loops
6. functions, modules, and procedural programming
7. lists and tuples (including multidimensional lists)

Summary of Course Topics

1. Basics of computers and how to create/run a Python script
2. Basic Python commands and program structure
3. Basic datatypes, operators, and I/O
4. if-statements, while-loops, and Boolean logic
5. for-loops and nested loops
6. functions, modules, and procedural programming
7. lists and tuples (including multidimensional lists)
8. strings, sets, and dictionaries

Summary of Course Topics

1. Basics of computers and how to create/run a Python script
2. Basic Python commands and program structure
3. Basic datatypes, operators, and I/O
4. if-statements, while-loops, and Boolean logic
5. for-loops and nested loops
6. functions, modules, and procedural programming
7. lists and tuples (including multidimensional lists)
8. strings, sets, and dictionaries
9. Object-oriented programming, inheritance, and polymorphism

Summary of Course Topics

1. Basics of computers and how to create/run a Python script
2. Basic Python commands and program structure
3. Basic datatypes, operators, and I/O
4. if-statements, while-loops, and Boolean logic
5. for-loops and nested loops
6. functions, modules, and procedural programming
7. lists and tuples (including multidimensional lists)
8. strings, sets, and dictionaries
9. Object-oriented programming, inheritance, and polymorphism
10. File I/O

What's next?

What's next?

How do we use these commands and techniques to build complex systems?

What's next?

How do we use these commands and techniques to build complex systems?

Get creative!

What's next?

How do we use these commands and techniques to build complex systems?

Get creative!

- algorithms

What is an algorithm?

What is an algorithm?

An *algorithm* is a list of instructions, simple enough to be easily converted into computer code, which can be followed to solve a problem

What is an algorithm?

An *algorithm* is a list of instructions, simple enough to be easily converted into computer code, which can be followed to solve a problem

Written in *pseudo-code*!

Example

Example

Convert Fahrenheit to Kelvins:

Example

Convert Fahrenheit to Kelvins:

1. **input** temperature in degrees Fahrenheit

Example

Convert Fahrenheit to Kelvins:

1. **input** temperature in degrees Fahrenheit
2. temp in degrees C $=$ (temp in degrees F - 32) $\times \frac{5}{9}$

Example

Convert Fahrenheit to Kelvins:

1. **input** temperature in degrees Fahrenheit
2. temp in degrees C $=$ (temp in degrees F - 32) $\times \frac{5}{9}$
3. temp in K $=$ temp in deg C $+$ 273.15

Example

Convert Fahrenheit to Kelvins:

1. **input** temperature in degrees Fahrenheit
2. temp in degrees C $=$ (temp in degrees F - 32) $\times \frac{5}{9}$
3. temp in K $=$ temp in deg C $+$ 273.15
4. **output** temperature in K

Sorting

Sorting

Given a list of values:

```
myList = [6, 3, 0, 3, 10, 5]
```

Sorting

Given a list of values:

```
myList = [6, 3, 0, 3, 10, 5]
```

Sort `myList` from low-to-high:

```
myList = [0, 3, 3, 5, 6, 10]
```

Writing a Sorting algorithm

Writing a Sorting algorithm

Create an algorithm for sorting myList

Writing a Sorting algorithm

Create an algorithm for sorting myList

- `myList` could contain any numbers

Writing a Sorting algorithm

Create an algorithm for sorting myList

- `myList` could contain any numbers
- `myList` could have any length n

Writing a Sorting algorithm

Create an algorithm for sorting myList

- `myList` could contain any numbers
- `myList` could have any length n
- only **two** operations allowed:

Writing a Sorting algorithm

Create an algorithm for sorting myList

- `myList` could contain any numbers
- `myList` could have any length n
- only **two** operations allowed:
    1. **Compare** 2 items in `myList`: `item1 < item2`

Writing a Sorting algorithm

Create an algorithm for sorting myList

- `myList` could contain any numbers
- `myList` could have any length n
- only **two** operations allowed:
  1. **Compare** 2 items in `myList`: `item1 < item2`
  2. **Swap** 2 items in `myList`: `item1 = item2, item2 = item1`

Writing a Sorting algorithm

Create an algorithm for sorting myList

- `myList` could contain any numbers
- `myList` could have any length n
- only **two** operations allowed:
  1. **Compare** 2 items in `myList`: `item1 < item2`
  2. **Swap** 2 items in `myList`: `item1 = item2, item2 = item1`

```
myList = [8, 5, 10, 1, 4]
```

Bubble Sort

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them:

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them:

```
myList = [8, 5, 10, 1, 4]
```

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them:

```
myList = [8, 5, 10, 1, 4]
```

Start at index 0:

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them:

```
myList = [8, 5, 10, 1, 4]
```

Start at index 0:

```
myList[1] = 5, myList[0] = 8
```

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them:

```
myList = [8, 5, 10, 1, 4]
```

Start at index 0:

```
myList[1] = 5, myList[0] = 8
5 < 8 is False
```

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them:

```
myList = [8, 5, 10, 1, 4]
```

Start at index 0:

```
myList[1] = 5, myList[0] = 8
5 < 8 is False
swap(myList[0], myList[1])
```

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them:

```
myList = [8, 5, 10, 1, 4]
```

Start at index 0:

```
myList[1] = 5, myList[0] = 8
5 < 8 is False
swap(myList[0], myList[1])
myList = [5, 8, 10, 1, 4]
```

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them:

```
myList = [5, 8, 10, 1, 4]
```

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them:

```
myList = [5, 8, 10, 1, 4]
```

Now at index 1:

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them:

```
myList = [5, 8, 10, 1, 4]
```

Now at index 1:

```
myList[1] = 8, myList[2] = 10
```

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them:

```
myList = [5, 8, 10, 1, 4]
```

Now at index 1:

```
myList[1] = 8, myList[2] = 10
8 < 10 is True
```

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them:

```
myList = [5, 8, 10, 1, 4]
```

Now at index 1:

```
myList[1] = 8, myList[2] = 10
8 < 10 is True
```

do nothing

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them:

```
myList = [5, 8, 10, 1, 4]
```

Now at index 1:

```
myList[1] = 8, myList[2] = 10
8 < 10 is True
```

do nothing

```
myList = [5, 8, 10, 1, 4]
```

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them:

```
myList = [5, 8, 10, 1, 4]
```

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them:

```
myList = [5, 8, 10, 1, 4]
```

myList = [5, 8, **10, 1**, 4]

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them:

myList = [5, 8, 10, 1, 4]

myList = [5, 8, **10, 1**, 4]

myList = [5, 8, 1, 10, 4]

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them:

`myList = [5, 8, 10, 1, 4]`

`myList = [5, 8,` **10, 1**`, 4]`

`myList = [5, 8, 1, 10, 4]`

`myList = [5, 8, 1,` **10, 4**`]`

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them:

myList = [5, 8, 10, 1, 4]

myList = [5, 8, **10, 1**, 4]

myList = [5, 8, 1, 10, 4]

myList = [5, 8, 1, **10, 4**]

myList = [5, 8, 1, 4, 10]

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them:

```
myList = [5, 8, 10, 1, 4]

myList = [5, 8, 10, 1, 4]

myList = [5, 8, 1, 10, 4]

myList = [5, 8, 1, 10, 4]

myList = [5, 8, 1, 4, 10]
```

One item (10) in the right place!

Bubble Sort

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them.

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them.

Each pass puts one item in the right place

Bubble Sort

Pass over every item in the list, and if it is bigger than its neighbor to the right, swap them.

Each pass puts one item in the right place

Do n passes, to put n items in place

Bubble Sort

```
myList = [8, 5, 10, 1, 4]
```

Bubble Sort

myList = [**8, 5**, 10, 1, 4]

Bubble Sort

myList = [**5, 8**, 10, 1, 4]

Bubble Sort

myList = [5, **8**, **10**, 1, 4]

Bubble Sort

myList = [5, 8, **10, 1**, 4]

Bubble Sort

myList = [5, 8, **1, 10**, 4]

Bubble Sort

myList = [5, 8, 1, **10, 4**]

Bubble Sort

myList = [5, 8, 1, **4, 10**]

Bubble Sort

myList = [**5**, **8**, 1, 4, 10]

Bubble Sort

myList = [5, **8**, **1**, 4, 10]

Bubble Sort

```
myList = [5, 1, 8, 4, 10]
```

Bubble Sort

myList = [5, 1, **8**, **4**, 10]

Bubble Sort

`myList = [5, 1,` **`4, 8`**`, 10]`

Bubble Sort

myList = [5, 1, 4, **8, 10**]

Bubble Sort

myList = [**5**, **1**, 4, 8, 10]

Bubble Sort

myList = [**1**, **5**, 4, 8, 10]

Bubble Sort

```
myList = [1, 5, 4, 8, 10]
```

Bubble Sort

myList = [1, **4, 5**, 8, 10]

Bubble Sort

```
myList = [1, 4, 5, 8, 10]
```

Bubble Sort

myList = [1, 4, 5, **8, 10**]

Bubble Sort

```
myList = [1, 4, 5, 8, 10]
```

Bubble Sort

myList = [1, **4**, **5**, 8, 10]

Bubble Sort

myList = [1, 4, **5, 8**, 10]

Bubble Sort

`myList = [1, 4, 5,` **8, 10**`]`

Bubble Sort

```
myList = [1, 4, 5, 8, 10]
```

Bubble Sort

`myList = [1, 4, 5, 8, 10]` ✓

Bubble Sort

Bubble Sort

- Loop until no swap is done:

Bubble Sort

- Loop until no swap is done:
- In each pass, loop over every index in list

Bubble Sort

- Loop until no swap is done:
- In each pass, loop over every index in list
- Compare to the item to its right

Bubble Sort

- Loop until no swap is done:
- In each pass, loop over every index in list
- Compare to the item to its right
- If bigger, swap

Bubble Sort

- Loop until no swap is done:
- In each pass, loop over every index in list
- Compare to the item to its right
- If bigger, swap
- Otherwise, do nothing

Bubble Sort Code

Bubble Sort Code

```
def swap(myList, ind1, ind2):
    tmp = myList[ind1]
    myList[ind1] = myList[ind2]
    myList[ind2] = tmp
    return
```

Bubble Sort Code

```
def swap(myList, ind1, ind2):
    tmp = myList[ind1]
    myList[ind1] = myList[ind2]
    myList[ind2] = tmp
    return

def BubbleSort(myList):
    didSwap = True
    while didSwap:
        didSwap = False
        for i in range(len(myList - 1)):
            if myList[i+1] < myList[i]:
                swap(myList, i, i+1)
                didSwap = True
    return
```

Selection Sort

Selection Sort

Modify Bubble Sort:

- Loop until no swap is done:
- In each pass, loop over every item in list
- Compare to the item to its left
- *If bigger, swap*
- Otherwise, do nothing

Selection Sort

Modify Bubble Sort:

- Loop until no swap is done:
- In each pass, loop over every item in list
- Compare to the item to its left
- **If bigger, save for later**
- Otherwise, do nothing

Selection Sort

Modify Bubble Sort:

- Loop until no swap is done:
- In each pass, loop over every item in list
- Compare to the item to its left
- **If bigger, save for later**
- Otherwise, do nothing
- **Swap biggest item to end of list**

Selection Sort Code

Selection Sort Code

```
def swap(myList, ind1, ind2):
    tmp = myList[ind1]
    myList[ind1] = myList[ind2]
    myList[ind2] = tmp
    return
```

Selection Sort Code

```
def swap(myList, ind1, ind2):
    tmp = myList[ind1]
    myList[ind1] = myList[ind2]
    myList[ind2] = tmp
    return

def SelectionSort(myList):
    for i in range(len(myList)):
        end = len(myList) - i - 1
        selection = end
        for j in range(end):
            if myList[selection] < myList[j]:
                selection = j
        swap(myList, selection, end)
    return
```

Selection Sort

```
myList = [8, 5, 10, 1, 4]
```

Selection Sort

`myList = [8, 5, 10, 1, 4]` **max ind = 4, max val = 4**

Selection Sort

myList = [**8**, 5, 10, 1, 4] **max ind = 4, max val = 4**

Selection Sort

myList = [**8**, 5, 10, 1, 4] **max ind = 0, max val = 8**

Selection Sort

myList = [8, **5**, 10, 1, 4] **max ind = 0, max val = 8**

Selection Sort

`myList = [8, 5,` **10**`, 1, 4]` **max ind = 0, max val = 8**

Selection Sort

myList = [8, 5, **10**, 1, 4] **max ind = 2, max val = 10**

Selection Sort

`myList = [8, 5, 10,` **1**`, 4]` **max ind = 2, max val = 10**

Selection Sort

`myList = [8, 5, 10, 1, 4]` **max ind = 2, max val = 10**

Selection Sort

```
myList = [8, 5, 10, 1, 4]
```

Selection Sort

myList = [8, 5, **4**, 1, **10**]

Selection Sort

`myList = [8, 5, 4, 1, 10]` **max ind = 3, max val = 1**

Selection Sort

myList = [**8**, 5, 4, 1, 10]   **max ind = 3, max val = 1**

Selection Sort

`myList = [`**`8`**`, 5, 4, 1, 10]` **max ind = 0, max val = 8**

Selection Sort

`myList = [8,` **`5`**`, 4, 1, 10]` **max ind = 0, max val = 8**

Selection Sort

myList = [8, 5, **4**, 1, 10]   **max ind = 0, max val = 8**

Selection Sort

`myList = [8, 5, 4, 1, 10]` **max ind = 0, max val = 8**

Selection Sort

```
myList = [8, 5, 4, 1, 10]
```

Selection Sort

```
myList = [1, 5, 4, 8, 10]
```

Selection Sort

`myList = [1, 5, 4, 8, 10]` **max ind = 2, max val = 4**

Selection Sort

`myList = [`**`1`**`, 5, 4, 8, 10]` **max ind = 2, max val = 4**

Selection Sort

`myList = [1,` **`5`**`, 4, 8, 10]` **max ind = 1, max val = 5**

Selection Sort

```
myList = [1, 5, 4, 8, 10]
```

Selection Sort

myList = [1, **4**, **5**, 8, 10]

Selection Sort

`myList = [1, 4, 5, 8, 10]` **max ind = 1, max val = 4**

Selection Sort

`myList = [`**`1`**`, 4, 5, 8, 10]` **max ind = 1, max val = 4**

Selection Sort

```
myList = [1, 4, 5, 8, 10]
```

Selection Sort

`myList = [1, 4, 5, 8, 10]` ✓

Insertion Sort

Insertion Sort

Insert each item into the already sorted sub-list

Insertion Sort

Insert each item into the already sorted sub-list

- Loop until all items have been inserted:

Insertion Sort

Insert each item into the already sorted sub-list

- Loop until all items have been inserted:
- In iteration i, insert the ith item into the sublist `myList[:i]`

Insertion Sort

Insert each item into the already sorted sub-list

- Loop until all items have been inserted:
- In iteration i, insert the ith item into the sublist `myList[:i]`
    - Easy to insert into an already-sorted sub-list

Insertion Sort

Insert each item into the already sorted sub-list

- Loop until all items have been inserted:
- In iteration i, insert the ith item into the sublist `myList[:i]`
  - Easy to insert into an already-sorted sub-list
  - sub-list `myList[:i]` is already sorted

Insertion Sort

Insert each item into the already sorted sub-list

- Loop until all items have been inserted:
- In iteration i, insert the ith item into the sublist `myList[:i]`
    - Easy to insert into an already-sorted sub-list
    - sub-list `myList[:i]` is already sorted
    - so coding this insertion is *easy*

Insertion Sort

```
myList = [8, 5, 10, 1, 4]
```

Insertion Sort

```
myList = [8, 5, 10, 1, 4]
```

Insertion Sort

```
myList = [8, 5, 10, 1, 4]
```

Insertion Sort

myList = [**5**, **8**, 10, 1, 4]

Insertion Sort

myList = [**5**, **8**, *10*, 1, 4]

Insertion Sort

myList = [**5**, **8**, **10**, 1, 4]

Insertion Sort

`myList = [`**5, 8, 10** , *1*, `4]`

Insertion Sort

myList = [**5, 8, 1, 10**, 4]

Insertion Sort

myList = [**5, 1, 8, 10** , 4]

Insertion Sort

myList = [**1, 5, 8, 10** , 4]

Insertion Sort

`myList = [`**1, 5, 8, 10** , *4*]

Insertion Sort

```
myList = [1, 5, 8, 4, 10]
```

Insertion Sort

```
myList = [1, 5, 4, 8, 10]
```

Insertion Sort

```
myList = [1, 4, 5, 8, 10]
```

Insertion Sort

```
myList = [1, 4, 5, 8, 10]
```

Insertion Sort

```
myList = [1, 4, 5, 8, 10]
```
✓

Insertion Sort Code

Insertion Sort Code

```
def insert(myList, nextInd):
    i = nextInd
    while i > 0 and myList[i] < myList[i-1]:
        swap(myList, i, i-1)
        i = i - 1
    return
```

Insertion Sort Code

```
def insert(myList, nextInd):
    i = nextInd
    while i > 0 and myList[i] < myList[i-1]:
        swap(myList, i, i-1)
        i = i - 1
    return

def InsertionSort(myList):
    for i in range(len(myList)):
        insert(myList, i)
    return
```

Algorithm Complexity

Algorithm Complexity

Which *algorithm* is best?

Algorithm Complexity

Which *algorithm* is best?

Run-time isn't a fair comparison because the runtime is different, depending on who codes it.

Algorithm Complexity

Which *algorithm* is best?

Run-time isn't a fair comparison because the runtime is different, depending on who codes it.

Use $\mathcal{O}$ complexity instead:

Algorithm Complexity

Which *algorithm* is best?

Run-time isn't a fair comparison because the runtime is different, depending on who codes it.

Use $\mathcal{O}$ complexity instead:

- About how many times do we need to look at each data entry in `myList`?

Algorithm Complexity

Which *algorithm* is best?

Run-time isn't a fair comparison because the runtime is different, depending on who codes it.

Use $\mathcal{O}$ complexity instead:

- About how many times do we need to look at each data entry in `myList`?
- All 3 algorithms have 2 nested loops, so we look at each of the n entries about n times...

Algorithm Complexity

Which *algorithm* is best?

Run-time isn't a fair comparison because the runtime is different, depending on who codes it.

Use $\mathcal{O}$ complexity instead:

- About how many times do we need to look at each data entry in `myList`?
- All 3 algorithms have 2 nested loops, so we look at each of the n entries about n times...
- $n \times n = n^2$, so we say these algorithms have $\mathcal{O}(n^2)$ complexity

Algorithm Complexity

Which *algorithm* is best?

Run-time isn't a fair comparison because the runtime is different, depending on who codes it.

Use $\mathcal{O}$ complexity instead:

- About how many times do we need to look at each data entry in `myList`?
- All 3 algorithms have 2 nested loops, so we look at each of the n entries about n times...
- $n \times n = n^2$, so we say these algorithms have $\mathcal{O}(n^2)$ complexity
- learn more in a real *algorithms* class

Algorithm Complexity

Which *algorithm* is best?

Run-time isn't a fair comparison because the runtime is different, depending on who codes it.

Use $\mathcal{O}$ complexity instead:

- About how many times do we need to look at each data entry in `myList`?
- All 3 algorithms have 2 nested loops, so we look at each of the n entries about n times...
- $n \times n = n^2$, so we say these algorithms have $\mathcal{O}(n^2)$ complexity
- learn more in a real *algorithms* class
- We can do better! Sorting can be done in $\mathcal{O}(n \log n)$ time!

Merge Sort

Merge Sort

You can *merge* two alread-sorted lists of length n in $\mathcal{O}(n)$ time

Merge Sort

```
def merge(list1, list2):
    i = 0
    j = 0
    newList = []
    while i < len(list1) and j < len(list2):
        if i >= len(list1):
            newList.append(list2[j])
        elif j >= len(list2):
            newList.append(list1[i])
        elif list1[i] < list2[j]:
            newList.append(list1[i])
            i = i + 1
        else:
            newList.append(list2[j])
            j = j + 1
    return newList
```

Merge Sort

Merge Sort

```
myList = [8, 5, 10, 1, 4]
```

Merge Sort

```
myList = [8, 5, 10, 1, 4]
```

8 5 10 1 4

Merge Sort

```
myList = [8, 5, 10, 1, 4]
```

8 5 10 1 4

5, 8 1, 10 4

Merge Sort

```
myList = [8, 5, 10, 1, 4]
```

8 5 10 1 4

5, 8 1, 10 4

5, 8 1, 4, 10

Merge Sort

```
myList = [8, 5, 10, 1, 4]
```

8 5 10 1 4

5, 8 1, 10 4

5, 8 1, 4, 10

1, 4, 5, 8, 10

Merge Sort

```
myList = [8, 5, 10, 1, 4]
```

8 5 10 1 4

5, 8 1, 10 4

5, 8 1, 4, 10

1, 4, 5, 8, 10

- $\mathcal{O}(\log n)$ levels

Merge Sort

```
myList = [8, 5, 10, 1, 4]
```

8 5 10 1 4

5, 8 1, 10 4

5, 8 1, 4, 10

1, 4, 5, 8, 10

- $\mathcal{O}(\log n)$ levels
- $\mathcal{O}(n)$ work to merge per level

Merge Sort

```
myList = [8, 5, 10, 1, 4]
```

8 5 10 1 4

5, 8 1, 10 4

5, 8 1, 4, 10

1, 4, 5, 8, 10

- $\mathcal{O}(\log n)$ levels
- $\mathcal{O}(n)$ work to merge per level
- $\mathcal{O}(n \log n)$ total work

Coding Merge Sort

Coding Merge Sort

Merge Sort is tricky to code...

Coding Merge Sort

Merge Sort is tricky to code...

We need to learn about *recursion*

Coding Merge Sort

Merge Sort is tricky to code...

We need to learn about *recursion* (next lecture)