Introduction

An algorithm is a set of instructions that perform a particular task (or problem), they are at the heart of Computer Science. As a concept, algorithms have existed long before computers, particularly in mathematics, but also in our day-to-day lives. You can actually consider much of what we do on a daily basis (such as making breakfast, or travelling to school) as algorithms.

For example, this is an algorithm:

```
Make Toast
1  Get bread from cupboard, and open packet.
2  Put 2 slices of bread in toaster.
3  Turn on the toaster by pressing down on the leaver.
4  Put bread away in the cupboard.
5  Get out a plate, a knife and the butter.
6  Wait for the toast to pop up.
7  Put toast on plate.
8  Use knife to put butter on toast.
9  Enjoy.
```

We design write algorithms specifically for use on a computer, and then convert the algorithm into computer code by using a particular programming language, e.g. Python, Java, Ruby, Pascal, Haskell, JavaScript etc... Algorithms that we design for computers need to follow a specific structure to make it easy to convert directly into a programming language.

Pseudocode

Pseudocode is a way to write algorithms using natural language (like english) by describing steps precisely, and usually looks quite similar to a lot of *real* programming languages.

For example this is **pseudocode** for an algorithm that finds the largest of two numbers:

```
max(x:number, y:number) → number
1  IF x > y THEN
2      RETURN x
3  OTHERWISE
4      RETURN y
```

You can see that there are some similarities between **pseudocode** and **python**, including:

- We put line numbers at the side.
- We can use similar **keywords** *(which keywords we use in **pseudocode** don't really matter as long as they are in english and make sense).*
- We use **indentation** (space at the start of a line).

There are many different styles of **pseudocode**, and all of them are valid, for example you don't need to use capital letters, or make things bold etc...

Here's what the above algorithm would look like in python:

```
1  def max(x, y):
2      if x > y:
3          return x
4      else:
5          return y
```

You will have noticed in our **pseudocode** example above, that we also had this extra bit at the top:

```
max(x:number, y:number) → number
```

This is called the **type signature** of the algorithm, and details the inputs that the algorithm accepts, and the type of the thing it will output (if it has an output). If an algorithm has a type signature, then we can think of it as a **function**, like in any programming language.

In this case, the algorithm inputs two values, both of which are numbers, and called them **x** and **y**. It also outputs a **number**. Look at the similarities between this and the first line of our **python** version.

Sorting Algorithms

Quite often when designing algorithms and writing software, we find that we want to sort lists of items into some kind of order (for example, we may have a list of words we want in **alphabetical** order, or a list of numbers we want in **ascending** (increasing) order).

For now, we'll just focus on putting lists of numbers in ascending order.

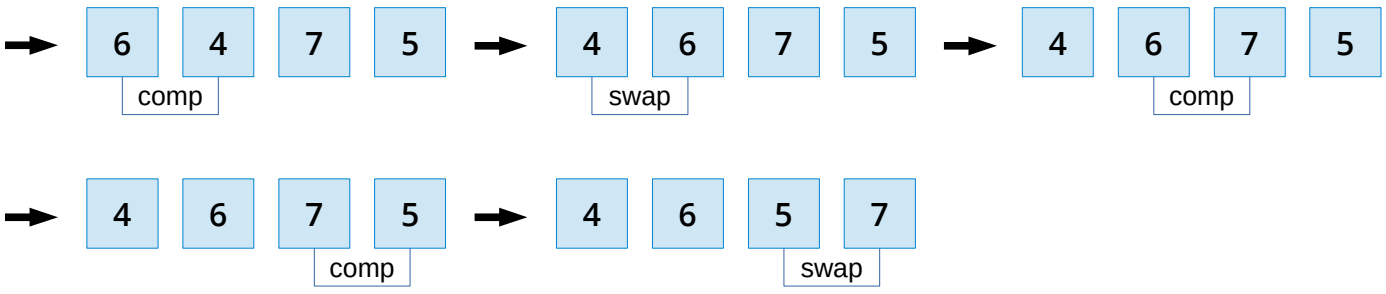
Bubble Sort

Bubble sort is a simple sorting algorithm that works like so:

- Go along the list **comparing** each pair of **consecutive** numbers (numbers that are next to each other in the list).
- If the second number is smaller than the first, **swap** them.
- Do this **sweep** for the whole length of the list, then again but ignoring the last number, and again but ignoring the last 2 numbers, then ignoring the last 3 numbers etc...
- When you reach a point where you are ignoring all numbers, the list will be sorted.

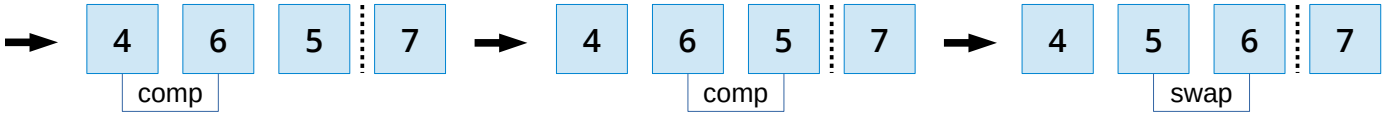
Take for example this list [6, 4, 7, 5], bubble sort would work like this:
*Note: "comp" mean **comparison**.*

Sweep 1:



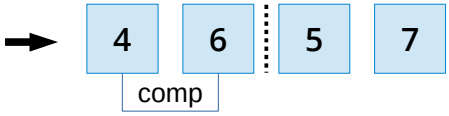
3 comparisons, and 2 swaps.

Sweep 2:



2 comparisons, 1 swap.

Sweep 3:



1 comparison, 0 swaps.

You can see that 1 sweep is not enough, because the numbers 5 and 6 need to be swapped to make the list ordered.

Here is pseudocode for bubble sort: (**Remember:** indexes start at **zero**)
Note: array is a special kind of list

```
bubble_sort(arr:array of numbers) → nothing
1  SET n = length of arr
2  WHILE n > 1 DO
3      FOR i IN [0, 1, … , n-3, n-2] DO
4          IF arr[i] > arr[i+1] THEN
5              swap arr[i] and arr[i+1]
6      SET n = n - 1
```

Questions

1.1:
Draw and run through the steps of bubble sort, like above, for the list [4, 5, 7, 6]

1.2:
Draw and run through the steps of bubble sort for the following lists, and count the number of comparisons and swaps done in each case:

(a) [3, 2, 1]
(b) [2, 7, 1, 6, 4]
(c) [4, 6, 6, 18, 20]

1.3:
Find and write down an order for the list with numbers 4,5,6,7 so that running bubble sort on it will result in no **swaps being** made.

1.4:
Find and write down a general rule that any list has to follow so that bubble sort **never makes any swaps** when run on it.

1.5:
Find and write down an order for the list with numbers 4,5,6,7 so that running bubble sort on it will result in a swap **after every comparison**.

1.6:
Find and write down a general rule that any list has to follow so that bubble sort **always makes a swap after every comparison** when run on it.

1.7:
Write down python code (on paper) for the bubble sort algorithm, using the pseudocode as a starting point. The function definition should look like: `def bubble_sort(arr):`

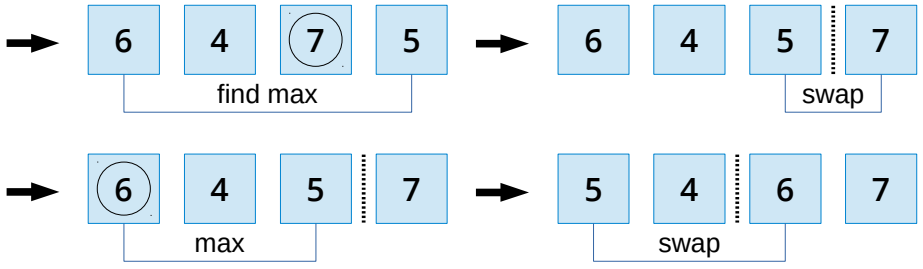
Remember: the python function `range(n):` makes a list like `[0, 1, …, n-1]`

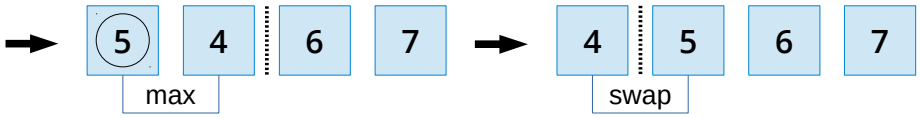
Selection Sort

Selection sort is another basic sorting algorithm, and it works like this:

- find the maximum number in the list
- swap that number with the last element in the list
- find the maximum number in the list, ignoring the last element
- swap that number with the second last element in the list
- find the maximum number in the list, ignoring the last two
- etc...

For example on the list [6, 4, 7, 5], it would work like this:





Here is the pseudo code for selection sort:

```
selection_sort(arr:array of numbers) → nothing
1   SET n = length of arr
2   WHILE n > 1 DO
3       SET max = arr[0]
4       SET max_i = 0
5       FOR i IN [1, 2, … , n-3, n-1] DO
6           IF arr[i] > max THEN
7               SET max = arr[i]
8               SET max_i = i
9       swap arr[max_i] and arr[n-1]
10      SET n = n - 1
```

Questions

2.1:
Draw and run through the steps of selection sort, like above, for the list [5, 4, 7, 6]

2.2:
What is the pseudocode on lines 3-8 doing?

2.3:
Write down python code (on paper) for the selection sort algorithm, using the pseudocode as a starting point. The function definition should look like: `def selection_sort(arr):`

Remember: the python function `range(n):` makes a list like `[0, 1, …, n-1]`