Unit 11 Exercises – Sorting and Complexity

Aims and objectives

- Working with dictionaries and nested lists
- Sorting
- Program planning; coding to specification

This week's exercises

These exercises are based on ideas covered briefly in Chapters 11 and 20 of the book, and in recent lectures which you should re-read.
You will find it useful to do some planning on paper before you start working on the machines.

Task 1 – Word frequencies

Define a function which is given a string and returns a dictionary with words (strings) as keys and numbers as values, showing how many times each word occurs in the original string. For example, when given the string:

```
"The first test of the function."
```

the function *wordFreq()* should return

```
{ "the":2, "first":1, "test":1, "of":1, "function":1 }
```

although of course the items might be in a different order. The function should ignore non-letters, and convert everything to lower case.

Test the function.

The function *split* from the string module will be very useful for this exercise. It separates a string into blocks, using a given string to indicate where the blocks begin and end. For example,

```
"The first test of the function.".split(" ")
```

(argument is a space) returns

```
[ "The", "first", "test", "of", "the", "function." ]
```

Think about how you are going to do this exercise, especially how you are going to convert from the original string into a list of individual words, in lower case, with all the non-letters removed. You might find it useful to develop some small functions to carry out the individual steps of this process.

Task 2 – Swap sort

SwapSort is a bit similar to *BubbleSort*. In *SwapSort,* during the first pass the first and second elements are compared (and swapped if necessary), but then the third and fourth elements are compared, and so on until the end of the list is reached. During the second pass, the first element is skipped and the second and third elements are compared, then the fourth and fifth, and so on. Write a function that implements the swapSort algorithm for sorting a list of integers.

Task 3 - Modified Selection sort

In the lecture you were introduced to the Selection Sort algorithm. For this problem, you need to implement a modified version of the Selection Sort algorithm where:

- Instead of finding the smallest element in the list and swapping it with an element at the beginning of the list,
- you need to find both the smallest and the largest elements and swap them with elements at the beginning and end of the list respectively.

a) You are required to select the and largest numbers in one pass. In other words, your implementation should only include a single loop for finding both the smallest and largest numbers.

Hint: There are some special cases that you need to figure out and handle in your implementation. Make sure that your solution works for these lists:

[6, 4, 5] and [2, 6, 8, 1, 7, 9, 4, 3, 5]

b) What is the invariant of this sorting algorithm?