Unit 12 Exercises – Sorting and Complexity

Aims and objectives

- Sorting
- Program planning; coding to specification

This week's exercises

These exercises are based on ideas covered 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 0 – Predicting the outcome of python code snippets (optional)

Go to http://142.93.32.146/index.php/268686?lang=en and work through the exercises there.

- these exercises are not part of the course.
- they are entirely optional, yet relevant if you want to improve your programming skills.
- the results are anonymous.
- a group of academics (Fionnuala Johnson, Stephen McQuistin, John O'Donnell and John Williamson) hope to publish anonymized findings based on the collected results.

Task 1 – The *join* function (and *split*) (in join.py)

a) Write a function *join* which, given two lists, it returns a list in which each element is a list of two elements, one from each of the given lists. For example:

```
                    join( [1,2,3] , ["a","b","c"] )

returns            [ [1, "a"], [2, "b"], [3, "c"] ]
```

We assume that the given lists both have the same length.

b) *Write tests for the function*: Each test should print two lists and the result of applying *join* to them. You might want to use the idea of generating test cases randomly. Also, make sure that you test the case of two empty lists, the case of two lists with just one element each, and a range of other lengths.

c) Write *split*, the opposite of *join* which, given a nested list where each element is itself a list of two elements, it breaks it down to two different lists.

Task 2 – Shaker Sort

ShakerSort is an optimization of the BubbleSort, that performs passes in both directions (simultaneously), allowing out-of-place items to move fast across the whole list.

ShakerSort performs the following steps:

1. Examines pairs of items, starting from the beginning of the list and performs swapping if necessary.
2. Examines pairs of items, starting from the end of the list and performs swapping if necessary.
3. Repeats the previous two steps until the list is sorted.

Consider the following list: `mylist = [7,5,11,10,8]`

Example:

- From beginning:
  - Examine `mylist[0]` and `mylist[1]`, compare the contents of this pair (7 and 5): swap elements. Resulting list: `[5,7,11,10,8]`
  - Examine `mylist[1]` and `mylist[2]`, compare the contents of this pair (7 and 11): do not swap elements.
  - Examine `mylist[2]` and `mylist[3]`, compare the contents of this pair (11 and 10): swap elements. Resulting list: `[5,7,10,11,8]`
  - Examine `mylist[3]` and `mylist[4]`, compare the contents of this pair (11 and 8): swap elements. Resulting list: `[5,7,10,8,11]`
- From end:
  - Examine `mylist[4]` and `mylist[3]`, compare the contents of this pair (11 and 8): do not swap elements.
  - Examine `mylist[3]` and `mylist[2]`, compare the contents of this pair (8 and 10): swap contents. Resulting list: `[5,7,8,10,11]`

  Repeat until list is sorted.

Write a method *ShakerSort* to implement this algorithm. Make sure it works for the following:

```
mylist1 = [2, 6, 8, 1, 7, 9, 4, 3, 5]
mylist2 = [10, 11, 3, 2, 5, 8, 1]
mylist3 = [10, 3, 2, 5, 8, 1]
```

Task 3 – The odd-even Sort

The *odd-even sort* algorithm is based on repeatedly making two passes on a list:

1. The first pass examines pairs of items `a[j]`, `a[j+1]` where j is odd.
2. The second pass examines pairs of items `a[i]`, `a[i+1]` where i is even.
3. In each pass, the following occurs: if the contents of the pair being examined are unordered, then the elements are swapped.
4. Repeat passes until list is sorted.

Consider the following list: `[1,7,5,2,6]`

First pass

- Start with odd position
  - Examine `a[1]` and `a[2]`, compare contents of this pair (7 and 5): swap elements. Resulting list: `[1,5,7,2,6]`
  - Examine `a[3]` and `a[4]`, compare contents of this pair (2 and 6): do not swap elements.
- Start with even position
  - Examine `a[0]` and `a[1]`, compare contents of this pair (1 and 5): do not swap elements.
  - Examine `a[2]` and `a[3]`, compare contents of this pair (7 and 2): swap elements. Resulting list: `[1,5,2,7,6]`

Repeat until list is sorted.

Write a function *oddEvenSort* that implements the Odd-Even sort algorithm. Make sure it works for the following lists:

```
mylist1 = [2, 6, 8, 1, 7, 9, 4, 3, 5]
mylist2 = [10, 11, 3, 2, 5, 8, 1]
mylist3 = [10, 3, 2, 5, 8, 1]
```