Week 9: Interview Preparation (Coding)

Introduction

Last week we went through some common interview topics for technical exams. Sometimes you will be asked to discuss the solution, however sometimes you will be asked to write code.

When asked to write code for an interview, you can normally do it in any language you wish. If you feel comfortable enough then Python is always a good one as the syntax is so easy and you are less likely to make a mistake!

Here are some questions that I have seen that come up in interviews, try implement them in Python in the most efficient solution you can think of! **Feel free to discuss your solutions!**

I have provided template code for all the functions you will write in this sheet :)

A good website for practice for these types of questions is `www.hackerrank.com` or `www.leetcode.com`.

TASKS

Task 1: Lists

Given a list **of integers** and 2 indices, i and j, how would you swap the i^{th} and j^{th} elements **in place** and without using a 3rd variable? **Complete the 'Swap' function** in the template - this should use this method to return the list with the two elements swapped.

Task 2: Find Pair Sum

Given a **sorted** list of integers and a variable x, complete the function 'FindPairSum' that returns $True$ if there exists two elements in the list that add up to x and returns $False$ otherwise.

There is no one solution for this, however try to think of the one that would be most optimal (in terms of space and time complexity)!

For example:

```
FindPairSum( [2,5,7,10,14] , 12 ) = True  #as 10+2=12 (also 5+7)
FindPairSum( [2,5,7,10,14] , 13 ) = False
```

Task 3: Fibonacci

For this task you will implement 2 different versions to compute the Fibonacci sequence. One way using recursion (this may be the way you have implemented it before) and one using a method called dynamic programming.

In an interview, if you get a question where you think recursion may be involved, always stop to think about if dynamic programming can be used - it is normally a lot faster and and will give you a much more optimal solution.

Recursion: Complete the 'Fibonacci' function which inputs a number n and returns the n^{th} Fibonacci number *using recursion*.

Dynamic Programming: Dynamic programming can be used as an improvement upon recursion. Instead of recursively calling the function, you calculate all things needed within the function and save it into a data structure (normally an array). This data can then be accessed through the array instead of calling the function again. **Try implementing your Fibonacci function using dynamic programming instead within the 'FibonacciDP' function.** For an example of dynamic program see below.

Comparing: Try comparing the runtime of each of these functions for the same number (try $n = 40$). A function has been written for you called 'TimeFunction' - input (as a string) the call you want to make, e.g 'TimeFunction('Fibonacci(40)')'. You can uncomment the last two lines in the template to do this. **Which is quicker and why?**

An example of dynamic programming can be seen below for calculating the factorial of an integer n. The first function uses recursion and the second uses dynamic programming.
Now, in this case DP might not be quicker as you only need to call each case once - this is just an example to show the implementation.

```
def Factorial(n):
    if n == 1:
        return 1
```

```
    else:
        return n * Factorial(n-1)

def FactorialDP(n):
    # DPArray[i] will store i!
    DPArray = [0,1]
    for i in range(2,n+1):
        LastNumber = DPArray[i-1]
        DPArray.append(LastNumber * i)
    # Return DPArray[n] to get n!
    return DPArray[n]
```

Task 4: SubText

Complete the function 'SubText' - this function should take 2 strings, $s1$ and $s2$ and return $True$ if all letters of $s1$ are present in $s2$ (ordering does not matter) and $False$ otherwise. In other words, you could make the text of $s1$ out of some (or maybe all) of the letters in $s2$ (can only use each occurence of each letter once).

Example:

```
SubText('acer','racecar') = True
SubText('babble','probable') = False
```

There are many ways in which you can do this, however try think about more optimal methods and which data structures will be useful! *Hint: ordering does not matter.*

Hint: the first function given in ***last week's worksheet (week 8) in question 1*** *will prove to be very useful.*
