Practice Questions on Recursion (Week 5)

Tutorial Questions

1. **Pascal's triangle** is made up of multiple levels of integers as shown below.

Level	List of Numbers
0	1
1	1 1
2	1 2 1
3	$\boxed{1}$ $\boxed{3}$ 3 1
4	1 $\boxed{4}$ 6 4 1

At any given level **n**, there are **n**+1 numbers. Let **pt(n, k)** represent the **k**th number at level **n** of the Pascal's triangle (range of **k** is from 1 to **n**+1).
The value **pt**(**n**, **k**) can be computed recursively as follows:

- For **k** = 1 or **n**+1, coefficient is 1;
- For any other value of **k**, its value is the sum of two numbers from the immediate previous level – the number to the left and the number to the right. Written formally:
  pt(**n**, **k**) = **pt**(**n**-1, **k**-1) + **pt**(**n**-1, **k**).

In the example above, the number 4 at level 4 is the sum of 1 and 3 from level 3, i.e.:
pt (4, 2) = **pt** (3, 1) + **pt** (3, 2)

a) Based on the above definition, complete the following recursive function design.

```
# Compute kth number at level n, n ≥ 0, k ≥ 1
def pt(n, k):
  # base case 1
  if ________________:
      return 1
  # base case 2
  if _________________:
      return 1

  # reduction step
  
```

b) Trace the sequence of recursive function calls for **pt**(4, 2). Hint: recall how Merge Sort's calling sequence was traced.

2. You are given the following algorithm to determine the value of **x** raised to the power of **n**.

```
def power3(x, n):
    if n == 0:
        return 1
    elif n % 2 == 0:
        temp = power3(x, n//2)
        return temp * temp
    else:
        temp = power3(x, n//2)
        return x * temp * temp
```

   a) How many multiplications does **power3** perform for **x** = 3 and **n** = 16?
   b) How many multiplications does **power3** perform for **x** = 3 and **n** = 19?
   c) What is the complexity of **power3**?

3. Suppose that **sub_function** has linear complexity. What is the complexity of **my_function** below?

   a)
```
def my_function(n):
    if (n > 0):
        sub_function(n)
        my_function(n//2)
```
   Hint:

$$\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{16}+\cdots=\sum_{n=1}^{\infty}\left(\frac{1}{2}\right)^n=\frac{\frac{1}{2}}{1-\frac{1}{2}}=1.$$

   b)
```
def my_function(k, n):
    if (n > 0):
        sub_function(k)
        my_function(k, n//2)
```

   c)
```
def my_function(n):
    if (n > 0):
        sub_function(n)
        my_function(n-1)
```

4. Write a recursive algorithm **max_array(a)** that takes in an array of positive integers and returns the biggest integer in the array. Your solution should not rely on sorting the array first.

5. A palindrome is a word that has the same spelling forwards and backwards, like “MADAM”. Write a recursive algorithm **is_palindrome(s)** to check if a string is a palindrome. For example, **is_palindrome("madam")** returns **True**, but **is_palindrome("madman")** returns **False**.

6. Given a recursive algorithm **f**(**n**) that takes a non-negative integer **n** as input.

```
def f(n):
    if n == 0 or n == 1:
        return 1
    return -f(n-1) - f(n-2)
```

   a) Specify the output values of the following expressions: **f**(1), **f**(5), **f**(6), **f**(330).
   b) What is the worst-case complexity of the algorithm **f**(n)? Show your working.

Extra Practice Questions

7. Rewrite the following Dijkstra's algorithm to calculate the greatest common divisor of two integers using recursion.

```
def dijkstra(a, b):
    while a != b:
        if a > b:
            a = a - b
        else:
            b = b - a
    return a
```

8. Rewrite the following Euclid's algorithm to calculate the greatest common divisor of two integers using recursion.

```
def euclid(a, b):
    while b != 0:
        t = b
        b = a % b
        a = t
    return a
```

9. Write a recursive algorithm **repeat_string(s,n)** that returns a concatenation of n copies of the string **s**. For example, **repeat_string("apple", 3)** will return **"appleappleapple"**.

10. Write a recursive algorithm **sum**(**n**) that computes the sum of the first n positive integers. For example, **sum**(1) returns **1**, **sum**(2) returns **1+2**, **sum**(3) returns **1+2+3**.

11. Write a recursive algorithm **reverse(a)** that returns an array with the same elements as **a**, but in reverse order.

12. The function **f12**(**x**, **n**) is defined like this:

$$f(x,n) = \frac{1}{x^1} + \frac{1}{x^2} + \frac{1}{x^3} + \cdots + \frac{1}{x^n}$$

e.g.:

$$f(2,4) = \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} = 0.9375$$

You are given this power function that returns $\mathbf{x}^{\mathbf{n}}$. Use this in your solution:

```
def power(x, n):
    if n == 0:
        return 1
    if n == 1:
        return x
    return x * power(x, n-1)
```

Write the function **f12_rec**(**x**, **n**) that uses recursion to return the correct value.

13. The function **f13**(**x**, **y**, **n**) is defined like this:

$$f(x, y, n) = 1x + 2y + 3x + 4y + 5x \ldots \;\; (there\ \ are\ \ n\ \ terms\ \ in\ \ the\ \ series)$$

e.g.:

$$f(4, 3, 5) = 1(4) + 2(3) + 3(4) + 4(3) + 5(4) = 54$$

Write the function **f13_rec**(**x**, **y**, **n**) that uses recursion to return the correct value.

14. The function e[x] is approximated by the following infinite series[1]: $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$

The function **f14** takes two arguments **x** and **n** (where **n**>0) and returns the value of the series after **n** iterations, so that: $e^x \sim 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$

e.g.: $f(2,4) \sim 1 + \frac{2}{1!} + \frac{2^2}{2!} + \frac{2^3}{3!} + \frac{2^4}{4!} = 7.0$

```
def f14(x, n):
    sum = 1
    for j in range(1, n+1):
        sum += (power(x, j) / factorial(j))
    return sum

def power(x, n):
    if n == 0:
        return 1
    if n == 1:
        return x
    return x * power(x, n-1)

def factorial(n):
    if n == 1:
        return 1
    return n * factorial(n-1)
```

a) What is the complexity of the iterative version of **f14** given above?

b) Rewrite **f14** using recursion. Call your function **f14_rec**. You will be given more marks if your algorithm's time complexity is lower (better). Can you come up with a recursive algorithm that has O(n) complexity?

~End

[1] For the Mathematically-inclined, see https://www.efunda.com/math/taylor_series/exponential.cfm