**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   


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)

1. 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

1. 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**.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09 | **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 |

1. How many multiplications does **power3** perform for **x** = 3 and **n** = 16?
2. How many multiplications does **power3** perform for **x** = 3 and **n** = 19?
3. What is the complexity of **power3**?
4. Suppose that **sub\_function** has linear complexity. What is the complexity of **my\_function** below?
5. **def** my\_function(n):  
    **if** (n > 0):  
    sub\_function(n)  
    my\_function(n//2)

Hint:

1. **def** my\_function(k, n):  
    **if** (n > 0):  
    sub\_function(k)  
    my\_function(k, n//2)
2. **def** my\_function(n):  
    **if** (n > 0):  
    sub\_function(n)  
    my\_function(n-1)
3. 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.

1. 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**.
2. 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)

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

**Extra Practice Questions**

1. Rewrite the following Dijkstra’s algorithm to calculate the greatest common divisor of two integers using recursion. Hint: base case will be when **a==b**

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07 | **def** dijkstra(a, b):  **while** a != b:  **if** a > b:  a = a – b  **else**:  b = b - a  **return** a |

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

|  |  |
| --- | --- |
| 01  02  03  04  05  06 | **def** euclid(a, b):  **while** b != 0:  t = b  b = a % b  a = t  **return** a |

or

|  |  |
| --- | --- |
| 01  02  03  04 | **def** euclid(a, b):  **while** b != 0:  a, b = b, a%b # swap  **return** a |

1. 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"**.
2. 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**.
3. Write a recursive algorithm **reverse(a)** that returns an array with the same elements as **a**, but in reverse order.
4. The function **f12**(**x**, **n**) is defined like this:

e.g.:

You are given this power function that returns **xn**. 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.

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

e.g.:

Write the function **f13\_rec**(**x**, **y**, **n**) that uses recursion to return the correct value.

1. The function e***x*** is approximated by the following infinite series[[1]](#footnote-1):

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.g.:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17 | **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) |

1. What is the complexity of the iterative version of **f14** given above?
2. Rewrite **f14** using recursion. Call your function **f14\_rec**. You will be given more marks if your algorithm’s time complexity is lower (better).

~End

1. For the Mathematically-inclined, see <https://www.efunda.com/math/taylor_series/exponential.cfm> [↑](#footnote-ref-1)