COMP S265F Design and Analysis of Algorithms
Lab 2: Recursive Algorithms – Suggested Solution

Question 1.

(a) Recursive function `fac(k)`:

```
1 def fac(k):
2     if k == 0:
3         return 1
4     else:
5         return k * fac(k-1)
```

(b) We prove that `fac(n)` correctly returns $n!$ by induction on n.
Base case: When $n = 0$, `fac(n)` $= 1 = 0!$.
Induction step. Assume that `fac(k)` $= k!$ for some integer $k \geq 0$.
When $n = k + 1$, `fac(k+1)` $= (k+1) \times$ `fac(k)` (by the definition of `fac`)

$$= (k+1) \times k! \quad \text{(by the induction hypothesis)}$$
$$= (k+1)! \ .$$

Question 2.

(a) The print statement in line 3 takes $O(1)$ time.
The for-loop iterates for $n = O(n)$ times.
Thus, the time complexity of `function` is

$$O(n \times 1) = O(n) \ .$$

(b) The print statements in lines 3 and 5 take $O(1)$ time.
Each of the for-loops in lines 2 and 4 iterates for $n = O(n)$ times.
Thus, the time complexity of `function` is

$$O(n \times 1 + n \times 1) = O(2n) = O(n) \ .$$

(c) The print statement in line 4 takes $O(1)$ time.
The outer for-loop in line 2 iterates for n times with $i = 0, 1, \ldots, n-1$.
For each iteration, the inner for-loop in line 3 iterates for i times with $j = 0, 1, \ldots, i-1$.
Thus, the time complexity of `function` is

$$(0 + 1 + 2 + \cdots + (n-1)) \cdot O(1) = \frac{n \cdot (n-1)}{2} \cdot O(1) = O(n^2 \cdot 1) = O(n^2) \ .$$

Question 3. The following is a Python recursive function of the Euclid's algorithm:

```
1 def gcd(a, b):
2     if a == b:
3         return a
4     elif a > b:
5         return gcd(a-b, b)
6     else:
7         return gcd(a, b-a)
```

Question 4. The following is the new function for finding the maximum:

```
def max2(num_list):
    m = num_list[0]
    for i in num_list:
        if i > m:
            m = i
    return m
```

The function max2 works, as follows:

1. The variable m keeps the maximum value we have seen, and is initially set as the first number of the list.
2. The for-loop scans the list once. When the list number i is larger than m, we update the maximum value m we have seen to i.
3. After scanning the number list, m will therefore equal to the maximum number in the list.

Step 1 and 3 take $O(1)$ time, and Step 2 takes $O(n)$ time. Therefore, the time complexity of the new function is $O(n+1) = O(n)$.