# COMP S265F Design and Analysis of Algorithms
# Lab 2: Recursive Algorithms

In this lab, we will review the technique of recursion and study how to prove the correctness of a recursive algorithm using mathematical induction. We will also introduce the syntax of defining classes in Python, and analyze time complexity of different algorithms.

## 1. Recursive algorithm

*Recursion* is a method of solving a problem where the solution depends on solutions to smaller instances of the same problem. An *recursive algorithm* is an algorithm that applies recursion. Below is an example of a recursive algorithm that finds the sum of integers from 1 to an user-specified positive integer $n$:

01sum.py

```python
from sys import stdin

def main():
    """Reads a user-specified positive integer n,
    and then prints the sum from 1 to n."""

    n = int(stdin.readline()) # n is a positive integer
    print(f"When n = {n}, 1 + 2 + 3 + ... + n = {sum(n)}.")

def sum(k):
    if k == 1:
        return 1
    else:
        return k + sum(k-1)

if __name__ == "__main__":
    print(main.__doc__)
    main()
```

**Python docstring.** A multi-line string in Python can be defined by surrounding the string with triple quotes or triple double-quotes. The `main` function above has a *docstring* that is a string literal occurs as the first statement in a module, function, class or method definition, providing a short description of the function. Such a docstring becomes the `__doc__` special attribute of that object, and provides a convenient way of associating documentation with Python modules, functions, classes, and methods.

**Python global variable `__name__`.** It refers to the entry point to your program. If you run the program directly, then it is `"__main__"`. Otherwise, it is the program name you import the module by. Therefore, code under the if-block will only run if the module is the entry point to your program.

**Structure of a recursive function.** A recursive function is composed of

1. *Base case:* For simple input, the result can be returned directly.

2. *Recursive step/Recurrence:* For large input, the result is obtained by using the solution on smaller input.

Due to this simple structure, a recursive function is easy to debug, implement, and comprehend. The structure of a recursive function resembles that of a mathematical induction proof, so its correctness can be proved easily by mathematical induction.

**Theorem 1.** *The function* `sum(n)` *returns the sum from 1 to $n$ correctly for any positive integer $n$.*

*Proof.* We prove the statement by induction on $n$.
**Base case:** When $n = 1$, `sum(n)` $= 1$, which is the correct sum from 1 to n.
**Induction step.** Assume that `sum(k)` returns the correct sum from 1 to $k$ for some positive integer $k$, i.e.,

$$\texttt{sum(k)} = \sum_{i=1}^{k} i \ .$$

$$
\begin{aligned}
\text{When } n = k+1,\ \texttt{sum(k+1)} &= (k+1) + \texttt{sum(k)} \quad \text{(by the definition of } \texttt{sum}\text{)} \\
&= (k+1) + \sum_{i=1}^{k} i \quad \text{(by the induction hypothesis)} \\
&= \sum_{i=1}^{k+1} i\ .
\end{aligned}
$$

$\square$

## 2. Python class

The syntax for defining classes in Python is straightforward, as shown below.

02sum.py

```
1  from sys import stdin
2
3  class Summation:
4      # Constructor
5      def __init__(self, n):
6          self.n = n # Create an instance variable
7
8      # Instance method 1
9      def result(self):
10         return self.sum(self.n)
11
12     # Instance method 2
13     def sum(self, k):
14         if k == 1:
15             return 1
16         else:
17             return k + self.sum(k-1)
18
19 def main():
20     """Reads a user-specified positive integer n,
21     and then prints the sum from 1 to n."""
22
23     n = int(stdin.readline()) # n is a positive integer
24     s = Summation(n)          # Create a Summation object
25     print(f"When n = {n}, 1 + 2 + 3 + ... + n = {s.result()}.")
26
27 if __name__ == "__main__":
28     print(main.__doc__)
29     main()
```

The first argument of all instance methods is `self`, which represents the instance of the class. In line 6, `self.n` is an instance variable, but `n` is only a local variable. Similarly, you need to call an instance method on a class instance, e.g., `self.sum` in line 10, and `s.result` in line 25.

## 3. Exercises

**Question 1.** The factorial of a positive integer $n$, denoted by $n!$, is the product of all integers from 1 to $n$. By definition, $0! = 1$.

(a) Write a recursive function `fac(n)` to compute $n!$ for any integer $n \geq 0$.

(b) Prove the correctness of `fac(n)` using mathematical induction.

**Question 2.** Determine the time complexity of the following function on a positive integer $n$.

(a)

```
def fuction(n):
  for i in range(n):
    print(i)
```

(b)

```
def fuction(n):
  for i in range(n):
    print(i)
  for j in range(n):
    print(j)
```

(c)

```
def fuction(n):
  for i in range(n):
    for j in range(i):
      print(i + j)
```

**Question 3.** Complete the following Python recursive function of the Euclid's algorithm.

```
def gcd(a, b):
    if a == b:
        # return ...
    elif a > b:
        # return ...
    else:
        # return ...
```

**Question 4.** The following function finds the maximum number in a list of $n$ numbers, but has a time complexity of $O(n^2)$. Rewrite the function so that its time complexity is $O(n)$.

```
def max1(num_list):
    for i in num_list:
        isMax = True

        for j in num_list:
            if j > i:
                isMax = False

        if isMax:
            return i
```