# ECS 32B: Winter 2019
# Homework Assignment 3

**Due Date:** No later than Saturday, February 16, 9:00pm

Submit your solutions via Canvas in one file named "hw3.py".

Here are your homework problems:

**1.** The sum of the first n terms of the Kleinfeldt sequence is defined as

$1 + 1/4 + 1/9 + 1/16 + 1/25 + ... + 1/n^2$

We can rewrite this definition recursively, so that the sum of the first n terms of the Kleinfeldt sequence can be defined as:

```
kleinfeldt (n) = 1 if n = 1
               = 1/n² + kleinfeldt (n – 1) if n > 1
```

Using Python and the information provided above, construct the function `kleinfeldt` that takes one argument, an integer n that is greater than 0, and returns the sum of the first n terms of the kleinfeldt sequence as a real number. Your solution must be a recursive solution. Here are some examples:

```
>>> kleinfeldt(1)
1
>>> kleinfeldt(2)
1.25
>>> kleinfeldt(100)
1.6349839001848923
```

**2.** Assume you are climbing a ladder. The ladder has n rungs. You can only climb one or two rungs at a time. How many different ways can you climb to the top? For example, if there are three rungs on the ladder, there are three different ways to climb to the top:

1 rung + 1 rung + 1 rung
1 rung + 2 rungs
2 rungs + 1 rung

Use Python and recursion to write a function called `ladder` that calculates the number of different ways you can get to the top. Here are some examples:

```
>>> ladder(3)
3
>>> ladder(5)
8
>>> ladder(10)
89
```

**3.** Write a recursive Python function that expects one argument, a Python list of integers, and returns the largest integer in the list. Thinking recursively, the largest integer is either the first integer in the list or the largest integer in the rest of the list, whichever is larger. If the list has only one integer, then the largest integer is this single value. You may assume that the list has at least one element. You may assume that the list has at least one element. Here are some examples:

```
>>> findLargest([1, 7, 35, 12, 19, 106, 0])
106
>>> findLargest([42])
42
```

(Helpful Python syntax: If A is a list of integers, and you want to set the list B to all of the integers in A except the first one, you can write B = A[1:])

**4.** If a list of size n has an element that appears more than n/2 times, let's call it a "majority element". For example, the list [2,2,5,1,5,5,1,5,5] has a majority element (5) but the list [2,2,5,1,5,5,1,5] does not. Use Python and recursion to write a function that expects one argument, a Python list, and returns the majority element. If there isn't a majority element, your program should return None. Here's a sketch of a recursive algorithm to find the majority element:

Step 1: Find the possible majority elements in the list
Step 2: Verify whether each possibility is actually a majority element

Step 1 is the recursive part. To find a possible majority element in the list A, create a second (empty) list B. Compare A[0] and A[1]. If their values are equal, add one of them to B. Otherwise, don't do anything. Then compare A[2] and A[3]. If they are equal, add one of them to B. Continue on in this way until all of A has been read. Then, recursively find the possible majority elements in B. The recursion will end when there are two or fewer elements in the list. These are the final possible majority elements for the original list.

Step 2 can be accomplished by a simple sequential search through the list A to verify if any possible majority element constitutes more than half of the original list.

Some notes:

Draw some test cases, follow the algorithm, and convince yourself of how it works.

If the list has an odd number of elements, do the pairing comparisons as described above, leaving the last element unpaired. Add this unpaired element to B.

Write two functions:

`findPossibilities(list)` will accomplish step 1. It expects a Python list as an argument and returns a Python list of possible majority elements.

`verifyPossibilities(list, possibilities)` will accomplish step 2. It expects the original Python list and the Python list of possible majority elements as arguments. It returns the majority element if there is one or None otherwise.

For problems 5 and 6, assume that you have constructed a linked list using the `Node()` class from your textbook. Here is one such example:

```
>>> n1 = Node(10)
>>> n2 = Node(20)
>>> n3 = Node(30)
>>> n4 = Node(40)
>>> n2.setNext(n1)
>>> n3.setNext(n2)
>>> n4.setNext(n3)
```

**5.** Use Python and recursion to write a function called `findValue` that determines whether a given data value is in the linked list. The function returns `True` if the value is present, and `False` otherwise. Here are some examples:

```
>>> findValue(10,n4)
True
>>> findValue(40,n4)
True
>>> findValue(50,n4)
False
>>>
```

Note that your function must work on lists other than the example above. You may assume that the list has at least one element.

**6.** Use Python and recursion to write a function called `findLastValue` that returns the last value in a linked list. Here is an example:

```
>>> findLastValue(n4)
10
```

Note that your function must work on lists other than the example above. You may assume that the list has at least one element.