CISC 2200
Recursion Practice Problems
Fall 2019

1. Keys to solving problems recursively

(1) What's the most simple case (e.g., smallest n, or smallest array, or shortest linked list, …) for which a solution is trivially found?

(2) How can I reduce the problem size: i.e. solve the larger problem instance (e.g., larger n, larger array, or longer linked list) by solving the smaller problem instance input, and then use these smaller problem(s) solution to come up solution to larger one?

2. General format of recursive function/algorithm

```
if (some condition for which answer is known trivially)  //base case

   solution statement

else                                        //general case
    some preprocessing

    one or multiple recursive calls

    some postprocessing

    generate solution
```

3. Three-Questions to verify a recursive algorithm/function is correct

(1) **Base-case question**: is there a non-recursive way out of the function, to end the recursive call?
(2) **Smaller-caller question**: does each recursive function call involve a smaller case/instance of the problem? And eventually leading to a base case?
(3) **Generel-case question**: assuming recursive call works correctly (i.e., smaller instances are solved correctly), is the solution to the larger problem composed/pieced together correctly form them?

4. Practice Problems

(a) Calculating factorial

Two different ways to define factorial

n! = n*(n-1)*(n-2) …. 2*1
0! = 1 //a special case

or
n! = n * (n-1)! //relates n! to (n-1)!, a recursive definition, recurrence relation
0! = 1

Recall this arises from calculation of Permutation: P(n,n)=n!

These translates into two different ways to implement the function:

```
/*  calculate n! for n>=0
   pre: n>=0
   post: return n!
*/
int factorial (int n)
{

}
```

(b) Calculating Fibonacci Term

(c) Find largest element in an array

Given an array a[0….99], we can view a[1…13] as a sub-array (a smaller array that is part of a[0….999]).
In general a[i…j] is a subarray of array a[m…n], if i>=m and j<=n

Iterative Solution?

How to solve it recursively?

(d) Binary search

Suppose an array is sorted in asceodning order, how to search for a particular element more efficiently (than linear search, i.e., going through each element one by one)?

0) base case:

1) find the midpoint of array
2) if element in the midpoint is smaller than target
continue to search in second half
if element in the midpoint is larger than target
continue to search in first half
else
we found it!

Now let's translate this into code

(e) Calculating length of linked list

interative solution (midterm)

```
int length (Node * head)
{
   Node * p = head;
  int length = 0;   //set up variables correctly initially

  while (p!=NULL) {   //set up looping condition correctly
       length ++; //we see a node *p

       p = p->next;   //inrement length and mvoe forward p
  }
  return length;

}
```

How to solve it recursively?