EECS 1012: LAB 10 – Recursion (a topic related to computational thinking, Nov 30-Dec 2)

A. IMPORTANT REMINDERS

1) Note that the deadline of remaining labs is on Wednesday night.
2) Each lab including the pre-lab mini quiz is about 2.0 % of your overall grade.
3) You must attend your own lab session (the one you are enrolled in). If you need to change your lab enrollment, you should go to the department. Instructors or TAs cannot change your enrollment. TAs are available via Zoom to help you. The attendance is optional, but is highly recommended. You can also have your work verified and graded during the lab sessions. Feel free to signal a TA for help if you stuck on any of the steps below. Yet, note that TAs would need to help other students too. In case you run out of time, the submission you make over eClass will be marked by the TAs after the lab ends (possibly not by the same TAs who assisted you during the lab session).
4) You can submit your lab work anytime before the deadline. We do not accept late submissions.
5) You must complete the pre-lab quiz posted on eClass no later than the first 15 minutes of your lab time.

B. IMPORTANT PRE-LAB WORKS YOU NEED TO DO BEFORE GOING TO THE LAB

1) Download this lab files and read them carefully to the end.
2) Review and implement the following recursive functions: `Factorial(n)` and `Fibonacci(n) from slides`.
3) Review the following links:

> https://www.freecodecamp.org/news/quick-intro-to-recursion/
> https://www.youtube.com/watch?v=YZcO_jRhvxs
> https://www.youtube.com/watch?v=B0NtAFf4bvU
> https://levelup.gitconnected.com/introduction-to-recursion-7848231b0d1b

4) Trace the following four calls: `Factorial(3)`, `Factorial(4)`, `Fibonacci(4)`, and `Fibonacci(5)`.

C. GOALS/OUTCOMES FOR LAB

1) To practice more computational thinking by using recursion.
2) To become familiar with how to trace recursive functions.

D. TASKS

1) TASK 1: Solving four computational thinking problem using Recursion (not solely iteration) and trace them for some sample inputs.

E. SUBMISSIONS

1) Manual verification by a TA (optional)

You may have one of the TAs verify your lab before submission. The TA will look at your various files in their progression. The TA may also ask you to make minor modifications to the lab to demonstrate your knowledge of the materials. The TA can then record your grade in the system.

2) EClass submission

Create a **folder** named "**Lab10**" and copy your HTML, JS, and Word files; Once you are done, compress the folder and upload the zip (or tar) file to eClass.

F. FURTHER DETAILS

Task 1: In this task, you complete four recursive functions and trace them for some sample inputs.

    a. Create an HTML file that includes four buttons with the following captions: "recursive reverse", "count 7s recursively", "recursive multiply", and "find it recursively".

b. Create a JavaScript file that supports these buttons. The implementations should be recursive. In other word, implementing these functions solely interactively has no point in this lab.

    a. Problem 1. Devise a function that receives a string and computes its reverse with a recursive method.

For instance, if the input is CSisFUN, the function should output NUFsiSC.

Approach:

Let's assume the signature of our function is `reverse(s)` where s is the input string. In the body of your function, check if the length of s is less than 2. If so, just return s because reverse of a string with length less than 2 is the same string. If length of s is greater than 2, make a new string that excludes the first character of the old string. That means the length of the new string is one character less than that of the original string; and now call your reverse function for the new string and concatenate it with the character that you excluded. As a hint, part of your code will look like this:

```
return reverse(newString) + s[0];
```

Writing any explicit loop in your code results a 0 for this question.
Remember to provide pre- and post- conditions.

Now copy/paste the following trace table in your Word file and trace your function `reverse(s)` for when s is initially CSisFUN. As a guideline, we have populated the tables with the values for the first 3 calls and the corresponding returned values for each of those calls. Note that in this example, the first table is populated top down (i.e. time elapses from row x to row x+1) whereas the returned value table is populated bottom-up (i.e. time elapses from row x to row x-1). You may want to use debugging features or simply console.log to observe these values.

call #	S	s[0]		returned value
1	CSisFUN	C		NUFsiSC
2	SisFUN	S		NUFsiS
3	isFUN	i		NUFsi
...	...	...		...

time

    b. Problem 2. Devise a function that receives a natural number and computes how many digits 7 is in that number with a recursive method.

For instance, if the input is 1012, the function should output 0. If the input is 237577, the function should return 3.

Approach:

Let's assume the signature of our function is `howMany7(n)` where n is the input whole number. In the body of your function, check if n is equal to 0. If so, just return 0. If n>0, separate its least significant digit; also make a newN that excludes that digit. If the list significant digit is equal to 7, recursively call your function with newN and add it up to 1 (because you just observed a digit 7). If the least significant digit is not a 7 just recursively call your function with newN.

You should just work with numbers in this question. Converting the number to string results a 0 for this question.

Also, writing any explicit loop in your code results a 0 for this question.
Don't forget to provide pre- and post- conditions.

Now copy/paste the following trace table in your Word file and trace your function howMany7(n) for when n is initially 237577. As a guideline, we have populated the tables with the values for the first 4 calls and the corresponding returned values for each of those calls. Note that in this example, the first table is populated top down (i.e. time elapses from row x to row x+1) whereas the returned value table is populated bottom-up (i.e. time elapses from row x to row x-1). You may want to use debugging features or simply console.log to observe these values.

call #	n	lsd
1	237577	7
2	23757	7
3	2375	5
4	237	7
...	...	...

returned value
3
2
1
1
...

time

c. Problem 3. Devise a function that receives two natural numbers and computes their multiplication with a recursive method.
For instance, if the inputs are 5 and 6, the function should output 30.

You are NOT allowed to use the multiplication operator (i.e. *).

Approach:
Let's assume the signature of our function is multiply(m,n) where m and n are the whole numbers inputted. In the body of your function, check if n is equal to 0 and/or check if n is equal to 1 and return a reasonable value. Hint: what is m*0 or m*1? For the rest of the body of your function use the following theorem:

multiply (m,n)= multiply(m,n-1)+m;

Now copy/paste the following trace table in your Word file and trace your function multiply(m,n) for when m and n are initially 5 and 6, respectively. As a guideline, we have populated the tables with the values for the first 2 calls and the corresponding returned values for each of those calls. Note that in this example, the first table is populated top down (i.e. time elapses from row x to row x+1) whereas the returned value table is populated bottom-up (i.e. time elapses from row x to row x-1). You may want to use debugging features or simply console.log to observe these values.

call #	M	n
1	5	6
2	5	5
...	...	...

returned value
30
25
...

time

d. Problem 4. Assume an array of 20 non-descending numbers exists. Receive a number from user and determine if that number exists in the array or not.
For instance, assume the array is:
[8, 9, 9, 12, 13, 13, 13, 15, 20, 100, 100, 101, 123, 129, 300, 1000, 5001, 20000, 20000, 34511]

Now, assume the user enters 5001, the program should output true. But, if the user enters 299, the program should output no.

Approach:

Let's assume the signature of our function is find(x,A,i,j) where x is the number we are looking for in array A, and the first index of the array is i and its last index is j. That means we want to determine whether x exists in A anywhere from index i to index j.

In the body of your function, you can compare x with the item that is in the middle of the array. The middle of the array is at index (i+j)/2, let's call it mid. If x<=a[mid], you should recursively call your function to search the <u>first half</u> of the array, i.e. from i to mid. If x>a[mid], you should recursively call your function to search the <u>second half</u> of the array, i.e. from mid+1 to j. As a hint, part of your code will look like these:

```
return find(x,A,i,mid)
…
return find(x,A,mid+1,j)
```

Note that i and j are getting closer to each other by every recursive call. Once i becomes equal to j, that means there is only one element in that range. So, you compare x with a[i] (or with a[j]). If they are equal, you return **true**, otherwise you return **false**. If j<i, that also means x does not exist in A and you should return **false**.

Writing any explicit loop in your code results a 0 for this question.
Remember to provide pre- and post- conditions.

Now copy/paste the following trace table in your Word file and trace your function find(x,A,i,j) for when x, i, and j are initially 123, 0, and 19, respectively. Assume A is the array given in the example above. As a guideline, we have populated the tables with the values for the first 2 calls and the corresponding returned values for each of those calls. Note that in this example, the first table is populated top down (i.e. time elapses from row x to row x+1) whereas the returned value table is populated bottom-up (i.e. time elapses from row x to row x-1). You may want to use debugging features or simply console.log to observe these values.

call #	x	i	j	mid		returned value
1	123	0	19	9		true
2	123	10	19	14		true
…	…			…		…

time

Show your <u>recursive functions</u>, <u>pre- post- conditions</u>, and <u>trace tables</u> to your TA before submit. (optional)

G. AFTER-LAB WORKS (THIS PART WILL NOT BE GRADED)

In order to review what you have learned in this lab as well as expanding your skills further, we recommend the following questions and extra practices:

1) Write a recursive function for the balanced-string function that was introduced in lectures.
2) Write a recursive function to implement the mergeSort using the following algorithm.
   a. Assume an array of n unsorted numbers exists. Let's call it A. the first index is 0 and the last one is n-1. Let's call these indices i and j.
   b. Divide the array to two halves. Let's call them A1 and A2. A1 indicates elements of A from index i to index m. A2 indicates elements of A from index m+1 to index j. where m is the midpoint of A.
   c. Recursively call the mergeSort algorithm for A1 and A2.
   d. Merge A1 and A2 (that now each of is ascendingly sorted) into an ascendingly sorted array for A.

Hint. Before implementing `mergeSort`, you may want to implement the `merge` function. The `merge` function receives two arrays that are already sorted and merges them to a bigger sorted array.

For instance, assume A1=[3, 5, 9, 14, 39, 43, 123, 140] and A2=[1, 3, 14, 16]. Then, `merge(A1,A2)` will return [1, 3, 3, 5, 9, 14, 14, 16, 39, 43, 123, 140]. You need to call your `merge` function in Step d of the `mergeSort` algorithm above. Note that the `merge` function is not recursive, but `mergeSort` is.

3) Add all around 8 to 10 new problems that you saw for recursion concept in the lecture notes and in this lab to your Learning Kit. That means, for each of these problems write a solution in Java too.

Now your Learning Kit should contain over 50 problems. If you have worked on these 50+ problems mostly by your own and not with too much help from other resources, you deserve an A/A+ not only in 1012 also you are heading for that in 1022, 2030, and 2011 and any other advance programming course. Good Luck.