**SOFTWARE DESIGN LABORATORY**

PreLab

* Readings

1.1 Recursion:

* Chapters 4: Ref Lab3 Data Structures and Algorithms in Python

Michael T. Goodrich, Roberto Tamassia et.al

OBSERVATION :

A recursion trace closely resembles the computer language's execution of the recursion. When a function (recursive or otherwise) is called in Python, a structure known as an activation record or frame is produced to contain information about the function's progress. Recursion has the capacity to call itself to solve an issue by breaking it down into smaller variants. People frequently feel that an entity cannot be defined in terms of itself. Recursion disproves the idea. And, if used correctly, this strategy has the potential to produce quite powerful outcomes. When it comes to data abstraction, recursion is always a good solution. Recursive definitions are frequently used to define data and related operations. And it is not incorrect to state that recursion is the most natural answer for challenges associated with the implementation of various operations on data.

CONCLUSION :

Most modern programming languages offer functional recursion through the same mechanism that supports older kinds of function calls. When one invocation of the function makes a recursive call, the invocation is suspended until the recursive call completes. Recursion is an important technique in the study of data structures and algorithms.

1.2 Recursion: Part I p.4 to p.69 : Ref Lab4 The Recursive Book of

Recursion Ace the Coding Interview with Python and JavaScript (Early

Access), Al Sweigart

OBSERVATION :

A recursive function is one that calls itself in programming. To comprehend recursion, you must first become acquainted with a less evident data structure known as a stack, which governs the program's flow of execution.

CONCLUSION :

Every recursive function must have at least two cases: the recursive case and the base case. The base case is a simple problem that we know how to solve, and it is the case that ends the recursion. The recursive situation is the more general case of the problem we're attempting to solve. It is always possible to convert a recursive algorithm to an iterative method.

* Answers to Questions

Ref Lab3 Data Structures and Algorithms in Python Michael T. Goodrich,

Roberto Tamassia et.al

Answer : R-4.1 to R-4.5 page 180

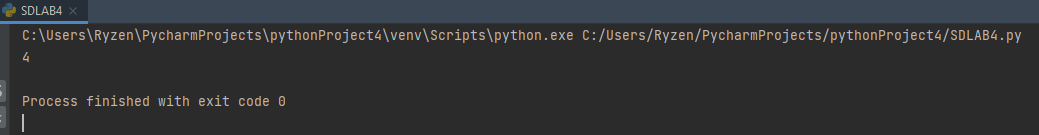
R-4.1 Describe a recursive algorithm for finding the maximum element in a sequence, S, of n elements. What is your running time and space usage?

CODE :

def maxRec(lst):

if(len(lst) == 1):  
 return lst[0]  
 else:  
 maxVal = maxRec(lst[2:])  
 if(maxVal < lst[0]):  
 maxVal = lst[0]  
 return maxVal  
  
lst = [3, 2, 4, 5, 1]  
maxVal = maxRec(lst)  
print(maxVal)

OUTPUT



R-4.2 Draw the recursion trace for the computation of power (2,5) , using the traditional function implemented in Code Fragment 4.11.

CODE :

public static double power(double x, int n)  
if (n == 0)  
return 1;  
else  
return x \* power(x, n-1);  
(in place of x we put 2 and in the place of n 5)

R-4.3 Draw the recursion trace for the computation of power (2, 18), using the repeated squaring algorithm, as implemented in Code Fragment 4.12.

Recursion trace:

power (2,18)

power (2,9)

power (2,4)

power (2,2)

power (2,1)

power (2,0)

power (2,0)

power (2,4)

power (2,2)

power (2,1)

power (2,0)

power (2,0)

2 to the power of 18 is 262144.

R-4.4Draw the recursion trace for the execution of function reverse(S, 0, 5)(Code Fragment 4.10) on S=[4, 3, 6, 2, 6].

R-4.5Draw the recursion trace for the execution of function Puzzle Solve (3,S,U)(Code Fragment 4.14), where S is empty and U={a, b, c, d}.

PuzzleSolver.java  
  
import java.util.Iterator;  
import java.util.LinkedHashSet;  
  
public  
  
  
class PuzzleSolver{  
private static  
  
  
class QueuSet< T > extends  
  
  
LinkedHashSet < T > {  
 private  
static  
final  
long  
serialVersionUID = 9965261432562813003L;  
public  
boolean  
offer(T  
element){  
return super.add(element);  
}  
  
public  
T  
poll()  
{  
Iterator < T > it = iterator();  
if (it.hasNext()) {  
T first = it.next();  
it.remove();  
return first;  
} else {  
return null;  
}  
}  
}  
public  
static  
void  
main(String[]  
args){  
 QueueSet < Character > qs = new  
QueueSet <> ();  
qs.offer(‘a’);  
qs.offer(‘b’);  
qs.offer(‘c’);  
qs.offer(‘d’);  
qs.offer(‘e’);  
  
puzzleSolve(3, “”, qs);  
}  
private  
static  
void  
puzzleSolve(int  
k, String  
s, QueuSet < Character > u) {  
for (int i=0; < u.size();++){  
// Remove e from u  
Character element = u.poll();  
  
// Add e to the end of s  
s = s+element;  
  
if (k == 1) {  
// Test  
whether  
s is a  
configuration  
that  
solves  
the  
puzzle  
// if s solves the puzzle then add s to output  
  
System.out.printIn(s);  
} else {  
puzzleSolve(k – 1, s, u);  
}  
  
// Remove  
e  
from the end  
  
of  
s  
s = s.substring(0.  
S.length() - 1);  
  
// Add  
e  
back  
to  
U  
u.offer(element);  
}  
}  
}