Csci 1523
Study Guide - Recursion

Student (Print): Amethyst O'Connell

This study guide contains 5 pages (including this cover page) and 20 problems. Check to see if any pages are missing. Enter all requested information on the top of this page, and put your initials on the top of every page, in case the pages become separated.

You may use your books, notes, calculator or internet sources while completing this study guide.

Please try to answer the sections clearly and PRINT your answers legibly.

Special Note: The completion of study guides in this course is to supplement your learning of the materials they are not required as a part of normal grading. However they may enhance the extra credit portion of the course if attendance has been regular and laboratories completed satisfactorily.

Chapter 11 Recursion Dierbach Study Guide
Recursive programming is an important technique used to implement algorithms. Recursion is frequently a slower technique at execution time but it is also the most natural technique when addressing particular types of problems.

Recursion finds practical application in a variety of practical programming situations. For example operating system utilities frequently use recursive algorithms to search over directory trees. Sorting applications which implement the popular *Quicksort* algorithm, (which constitutes practically all sort utilities) implement the algorithm recursively. Searching algorithms such as binary search over lists and trees are also typically implemented using recursive formulation. Artificial intelligence applications in fields such as robotics and expert systems also use recursion.
In this section we look at recursive functions and recursive problem solving along with some applications of the technique.

Below you will find a series of questions concerning recursion. The materials required to answer them are in Chapter 1 of Dierbach, course notes and movies.

1. The for and the while are iterative control structures implemented in Python.
2. When implementing a recursive algorithm a large problem is sucessively broken down into similiar subproblems.
3. Python implements recursive functions which support the implementation of recursive algorithms.
4. A recursive function is often called a function which calls itself.
5. There are two types of entities related to any function: 1. function definition and 2. execution instances.
6. While there is only one function definition there may be many execution instances of a function.
7. tf[F] In order for an algorithm to be recursive a function must called itself at least 2 times. F
8. The calling and suspending of executing function instances can theoretically continue forever.
9. A non-terminating sequence of a function calling itself is called infinite recursion.
10. Recursive functions stop when a solution is reached.
11. At execution time a recursive algorithm which uses **precisely** the same number of operations to reach a solution is always slower than an iterative solution to the same problem.

Is this statement generally True or False? In the space provided below explain your reasoning.

True, because function calls are time consuming, so even if you're doing the same number of operations, it's going to take longer to do it recursively because the method itself takes time.

12. ___ For $O(n^2)$ algorithms recursion is always the fastest solution technique.

13. Algorithms for sorting such as *Quicksort* and *Mergesort* exhibit a growth in complexity that is $O(nlog(n))$. Algorithms for searching such as *binary search* exhibit a growth in complexity which is $O(log(n))$. Review the complexity functions for these sorting and searching algorithms. Develop a justification based on the functional form of these equations for using *recursive* problem solving over iterative problem solving approaches.

14. In our text book in Section 11.1.2 a recursive solution to the factorial problem is discussed and a prototype algorithm developed. Using this implementation how many executing instances of the function, *factorial*, would there be at the time the recursion stops if the number input to this function was: 3: _____. In the space provided below provide a sketch similar to Figure 11-2 in the text to justify your answer:

15. The condition that controls when functions in a recursion stop calling themselves is referred to as the _____ case.

16. In the Figure 1 below we see a definition of a recursive function, myfunc(n), for any value n what does this function return, (answer should be in terms of n): _____.

```
1  def myfunc(n):
2      if n == 0:
3          return 0
4      else:
5          return myfunc(n-1) + 2
```

Figure 1: Recursive function

17. In the space provided below write a recursive algorithm which takes a string, s, as an argument and returns the string with its letters appearing in reverse order.

18. F Only problems which are recursively defined can be solved with recursion.

19. F The power of recursion is its speed of execution over iteration.

20. T When a problem can be solved as easily using iteration as it can be with recursion it is preferable to use iteration.