CSE 3101 Design and Analysis of Algorithms
Practice Test for Unit 2
Recursion

Jeff Edmonds

First learn the steps. Then try them on your own. If you get stuck only look at a little of the answer and then try to continue on your own.

1. We discuss two different purposes and definitions of "size". The running time of an algorithm is defined to be a mapping from the size of the input instance to the running time. Here "size" is the number of bits to represent the instance. If the instance is $\langle n, m \rangle$ then size is $s = \log n + \log m$. In the *friends* level of abstracting recursion, you can give your friend any legal instance that is "smaller" than yours according to some measure of "size". You must also solve on your own any instance that is sufficiently small according to this same definition of size. Here you can design your definition of "size" any way you like. Note that the number of bits to represent $\langle n-1, 2m \rangle$ is bigger than the number of bits to represent $\langle n, m \rangle$. Is this then a fair subinstance to give your friend? Maybe if you define size differently. Which of the following recursive algorithms has been designed correctly? If so what is your measure of the size of the instance? On input instance $\langle n, m \rangle$, either bound the depth to which the algorithm recurses as a function of n and m or prove that there is at least one path down the recursion tree that is infinite.

algorithm $R_a(n, m)$
$\langle$***pre−cond***$\rangle$**:** n & m ints.
$\langle$***post−cond***$\rangle$**:** Say Hi

```
begin
    if(n ≤ 0)
        Print("Hi")
    else
        R_a(n − 1, 2m)
    end if
end algorithm
```

algorithm $R_b(n, m)$
$\langle$***pre−cond***$\rangle$**:** n & m ints.
$\langle$***post−cond***$\rangle$**:** Say Hi

```
begin
    if(n ≤ 0)
        Print("Hi")
    else
        R_b(n − 1, m)
        R_b(n, m − 1)
    end if
end algorithm
```

algorithm $R_c(n, m)$
$\langle$***pre−cond***$\rangle$**:** n & m ints.
$\langle$***post−cond***$\rangle$**:** Say Hi

```
begin
    if(n ≤ 0 or m ≤ 0)
        Print("Hi")
    else
        R_c(n − 1, m)
        R_c(n, m − 1)
    end if
end algorithm
```

algorithm $R_d(n, m)$
$\langle$***pre−cond***$\rangle$**:** n & m ints.
$\langle$***post−cond***$\rangle$**:** Say Hi

```
begin
    if(n ≤ 0 or m ≤ 0)
        Print("Hi")
    else
        R_d(n − 1, m + 2)
        R_d(n + 1, m − 3)
    end if
end algorithm
```

algorithm $R_e(n, m)$
$\langle$***pre−cond***$\rangle$**:** n & m ints.
$\langle$***post−cond***$\rangle$**:** Say Hi

```
begin
    if(n ≤ 0 or m ≤ 0)
        Print("Hi")
    else
        R_e(n − 4, m + 2)
        R_e(n + 6, m − 3)
    end if
end algorithm
```

2. Review the problem on Iterative Cake Cutting. (Section 2.3) You are now to write a recursive algorithm for the same problem. You will, of course, need to make the pre and post conditions more general so that when you recurse, your subinstances meet the preconditions. Similar to moving from insertion sort to merge sort, you need to make the algorithm faster by cutting the problem in half.

   (a) You will need to generalize the problem so that the subinstance you would like your friend to solve is a legal instance according to the preconditions and so that the post conditions states the task you would like him to solve. Make the new problem, however, natural. Do not, for example, pass the number of players n in the original problem or the level of recursion. The input should simply be a set of players and a sub-interval of cake. The post condition should state the requirements

on how this subinterval is divided among these players. To make the problem easier, assume that the number of players is $n = 2^i$ for some integer i.

(b) Give recursive pseudo code for this algorithm. As a big hint, towards designing a recursive algorithm, we will tell you the first things that the algorithm does. Each player specifies where he would cut if he were to cut the cake in half. Then one of these spots is chosen. You need to decide which one and how to create two subinstances from this.

(c) Be sure to determine the running time.

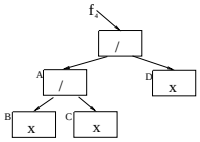
(d) Formally prove all the steps similar to those for an iterative algorithm needed to prove this recursive program correct.

(e) Now suppose that n is not 2^i for some integer i. How would we change the algorithm so that it handles the case when n is odd? I have two solutions. One which modifies the recursive algorithm directly and one that combines the iterative algorithm and the recursive algorithm. You only need to do one of the two (as long as it works and does not increase the bigoh of the running time.)

3. Develop an algorithm that searches for a key within a binary search tree.

4. Trace out the execution of Derivative on the instance

   $f = (x/x)/x$. In other words, draw a tree with a box for each time a routine is called. For each box, include only the function f passed and derivative returned.

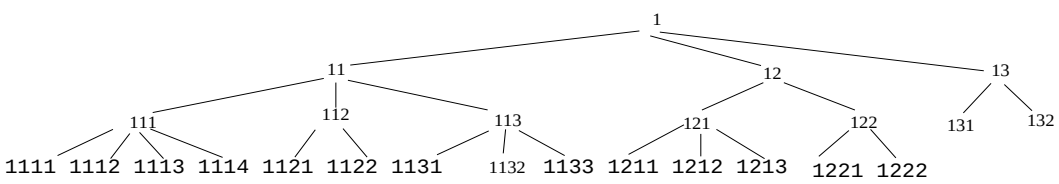


5. Recursion GCD: Write a recursive program to find the GCD of two numbers. The program should mirror the iterative algorithm.

6. Recursion Smallest: Explain the recursive algorithm given in class for finding the k^{th} smallest element in an array. Explain what the running time is.

7. Recursion Smallest in Tree: Our task now is the same as the last question, except that the input is a binary search tree and an integer k. Recall that in a binary search tree all the values smaller than the root are already in the left subtree and all those larger are in the right. As before, it returns the k^{th} smallest element in the tree. Explain what the running time is.

8. The algorithm $TreeToMatrix(tree, M, i, j)$ is passed a non-empty tree $tree$, a matrix M (which is passed both in and out), and two indexes i and j. The result of the algorithm is to place the data items in the tree $tree$ into the matrix M. The data item at the root, denoted $rootData(tree)$, is placed in $M(i, j)$. Its children, denoted $child(tree, 1), child(tree, 2), \ldots, child(tree, nChild(tree))$, are placed one column to the right. Its first child is placed just to the right of the root in $M(i, j+1)$. Its other children are shifted down to make room for the subtrees rooted at the previous children. An example is given below, with the input tree on the left and the resulting matrix on the right. Be sure to document any extensions you make to the pre or post conditions. Only the commented code is required. MORE DISCUSSION ALLOWED.



i/j	1	2	3	4
1	1	11	111	1111
2				1112
3				1113
4				1114
5			112	1121
6				1122
7			113	1131
8				1132
9				1133
10		12	121	1211
11				1212
12				1213
13			122	1221
14				1222
15		13	131	
16			132	

9. Recursion Inversions: Given a sequence of n distinct numbers $A = \langle a_1, a_2, \ldots, a_m \rangle$, we say that a_i and a_j are inverted if $i < j$ but $a_i > a_j$. The number of inversions in the sequence A, denoted $I(A)$, is the number of pairs a_i and a_j that are inverted. $I(A)$ provides a measure of how close A is to being sorted

in increasing order. For example, if A is already sorted in increasing order, then $I(A)$ is 0. At the other extreme, if A is sorted in decreasing order, every pair is inverted, and thus $I(A)$ is $\binom{n}{2}$. Describe a divide and conquer algorithm that finds $I(A)$ in time $\Theta(nlogn)$. You should assume that the sequence A is given to us in an array such that we can test the value of a_i in unit time, for any i. Note that the obvious algorithm is to test all pairs a_i and a_j; that algorithm runs in $\Theta(n^2)$ time. Hint: you should modify merge sort to also compute $I(A)$. The work each stack frame requires an iterative algorithm. Describe this iterative algorithm using loop invariants.