Theory Problems 1

Algorithms

Problem 1:GCD

In the lecture, we introduced an algorithm for greatest common devisor

Algorithm

GCD(m,n)   // Greatest common devisor
if $m < n$ then swap(m,n)
while $n \neq 0$
    $r = m \bmod n$
    $m = n$
    $n = r$
end while
output m

1. Use the algorithm to compute GCD(1,1), GCD(6,8) and GCD(7,5)
2. Rewrite the algorithm to use recursion

Problem 2: GCD proof of correctness

Proving the correctness of an algorithm can be difficult, but using *invariants* and *termination conditions* can be helpful. We will look at the GCD proof in stages, using the recursive version.

Let the new values be m',n' so GCD(m,n) recursively calls GCD(m',n')

$$m' = n, n' = m \bmod n$$

1. Termination condition: Show that if n'=0 and m$\geq$n then n divides both m and n (hence n is a divisor at termination) and is the greatest divisor of both.

2. Invariant: Let d be a divisor of m and n. Show that d also divides m' and n'. It may help to write m=ad, n=bd.

3. Hence prove that GCD(m,n) returns the greatest common divisor.

Problem 3: Write an algorithm

Problem:	Number swap
Input:	$(a \in \mathbb{R}, b \in \mathbb{R})$
Output:	$(a' = b, b' = a)$

You have a computer which can hold only two real numbers, *a* and *b* in memory. It can execute only the following six operations:

a=a+b, a=a-b, a=b-a, b=a+b, b=a-b, b=b-a

1. Write an algorithm to solve the number swap problem on this computer.
2. Choose some test cases for your algorithm
3. Prove the correctness more rigorously

Problem 4: Efficiency

Algorithm

fib($n \in \mathbb{N}$)
// the n^{th} Fibonacci number
if n==1 then return 1
If n==2 then return 1
return fib(n-1)+fib(n-2)

The Fibonacci numbers are defined as

$$f_1 = 1, f_2 = 1,$$
$$f_n = f_{n-1} + f_{n-2} \; (n > 2)$$

How many calls of fib(.) are needed to find f_n for n=1..6? Verify that for large n, the number of calls is approximately $2^{\alpha n}$ and find α

Problem 4: Efficiency

Algorithm

```
fib(n ∈ ℕ)
// the nth Fibonacci number
x=0, y=1
while n>1
    z=x+y
    x=y
    y=z
    n=n-1
end while
return y
```

This is an iterative Fibonacci algorithm.

Assuming a sum or assignment is one step, how many steps are used to find f_n?

Compare this algorithm to the previous one. Why the difference in steps taken?