# Homework 1, CS561, Fall 2014

Aaron Gonzales

September 2, 2014

## 1 exercise 3.1-5 CLRS - Prove theorem 3.1

## 2 Let $f(n) and g(n)$ be two functions that take on nonnegative values and assume $f(n) = O(g(n))$. Prove that $g(n) = \Omega(f(n))$.

By definition, $\Omega(g(n)) = \{f(n)$ : there exist positive constants $c, n_0$ such that $0 \leq cg(n) \leq f(n) \forall n \geq n_0\}$.

By definition, $O(g(n)) = \{f(n)$ : there exist positive constants $c, n_0$ such that $0 \leq f(n) \leq cg(n) \forall n \geq n_0\}$.

if $f(n)$ is bounded above by $g(n)$, $g(n)$ must be at least equal to or lower than $f(n)$ for all $n \geq n_0$.

## 3 Problem 3-2 on Relative asymptotic growths

| A | B | $O$ | $o$ | $\Omega$ | $\omega$ | $\Theta$ |
|---|---|---|---|---|---|---|
| $lg^k$ | $n^\epsilon$ | | | | | |
| $n^k$ | $c^n$ | | | | | |
| $\sqrt{n}$ | $n^{sinn}$ | | | | | |
| $2^n$ | $2^{n/2}$ | | | | | |
| $n^{lgc}$ | $c^{lgn}$ | | | | | |
| $lg(n!)$ | $lg(n^n)$ | | | | | |

## 4 Assume you have functions $f$ and $g$, such that $f(n)$ is $O(g(n))$. for each of the following statements, decide wether you think it is true or false and give either a proof or a counterexample.

**(a)** $log_2 f(n) = O(log_2(g(n))$

**(b)** $2^{f(n)} = O(2^{g(n)})$

**(c)** $f(n)^2 = O(g(n)^2)$

## 5 Problem 7-3: Alternative Quicksort analysis

## 6 Imagine you are doing a stress test on a particular model of smart phones. you have a ladder with $n$ rungs. You want ot determine the highest rung from which you cand rop a phone wihtout it breaking and you want to do it iwth the smallest number of phone drops.

### (a) Imagine that you have exactly 2 phones. Devise an algorithm that can determine the highest safe rung using $o(n)$ drops. (little o).

Let it be stated that a ladder must have at least one one rung. If it has one rung, we only need one phone to test and see if that rung is safe (if it breaks, it's unsafe, if not, it's safe).

If a ladder has $n > 1$ rungs, let us start by dropping the phone on the second rung. If the phone breaks, we drop from the rung below the current rung (in this case, the first rung) and if it doesn't break, we know that it is safe from the current-1 rung.

Stated in pseudocode:

```
for each rung on ladder:
  drop phone from rung+1
  if phone breaks:
  drop phone from rung
  if phone breaks:
  return current_rung - 2
  else return rung
```

we only test at most $n/2+1$ rungs, which dropping the constant is $n/2 < n \forall n \geq 2$. By definition, $o(g(n)) = \{f(n) :$ for any positive constant $c > 0$, there exisists a constant $n_0 > 0$ such that $0 \leq f(n) \leq cg(n) \forall n \geq n_0\}$. as such, the algorithm is $o(n)$.

**(b) Now suppose you have $k$ phones. Devise an algorithm that can determine the highest safe rung with the smallest number of drops. If $f_k(n)$ is the number of drops that your algorithm needs, what is $f_k(n)$ asymptotically? Hint: you shoudl ensure that $f_{k+1}(n) = o(f_k(n))$ for any $k$.**

This is an example of binary search. If we assume that the rungs in the ladder are in a sorted array $rungs[] < ++ >$, the algorithm could be stated as follows:

```

def binary_search(val, left = 0, right = nil)
  right = ladder.size - 1 unless right
  mid = (left + right) / 2

  if left > right
    return null

  if val == ladder[mid]
  return mid
  elif val > ladder[mid]
  binary_search(val, mid + 1, right)
  else
  binary_search(val, left, mid - 1)

```

Proof by induction that $f_k(n) = O(logn)$ follows: Let $P(n)$ be the assertion that our binary search of the ladder's rungs works correctly.

Our base case is a ladder of size 1: If the ladder has one rung, then $n = 1$ and the function returns 1, which is true. If $P(n)$ works for all $n$, then it works for a ladder of size $n+1$.

as we have already assumed a sorted ladder,

## 7 The game of Match.

## 8 Drunken Debutants

## 9 Pairs on a circle