CS3000: Algorithms & Data
Drew van der Poel

Recitation 2:

- Asymptotics
- Divide & Conquer
- Recursive Algorithms

May 17, 2021

Problem 1

Prove or disprove each of the following statements. For a proof, you need to give an argument that works for all f and g. To disprove, it would suffice to give one counterexample. Assume that f and g are functions that are increasing with n and satisfy $f(n), g(n) \geq 2$ for all $n \geq 1$.

$\log_2(f(n)), \log_2(g(n))$

≥ 1

$\forall n \geq 1$

(a) If $f(n) = \Omega(g(n))$, then $2^{f(n)} = \Omega(2^{g(n)})$. False

$f(n) = n, \quad g(n) = 2n$

$n = \Omega(2n), \quad 2^n \neq \Omega(2^{2n} = 4^n)$

(b) If $f(n) = O(g(n))$, then $\log_2(f(n)) = O(\log_2(g(n)))$. True

$f(n) \leq c\, g(n) \quad \forall n \geq n_0$

$\log_2(f(n)) \leq \log_2(c\, g(n)) \quad \forall n \geq n_0$

$= \log_2(g(n)) + \log_2(c)$

$$\log_2(f(n)) \le \underline{\log_2(g(n))} + \underline{\log_2 c} \quad \forall n \ge n_0$$

$$\boxed{\le} \log_2(g(n))\left(1 + \log_2 c\right) \quad \forall n \ge n_0$$

$$= \underline{\log_2(g(n))} + \log_2 c \; \underline{\log_2(g(n))}$$

$$c' = 1 + \log_2 c$$

$$n_0' = n_0$$

$$\Rightarrow$$

$$\log_2(f(n)) \le c' \log_2(g(n)) \quad \forall n \ge n_0'$$

$$\square$$

Problem 2

An array $A[1..n]$ is said to have a *majority element* if more than half of its entries are the same. For instance, the array $[2,5,1,4,4]$ does not have a majority element, while the array $[A, C, C, A, G, C, C]$ has a majority element, which is C.

We would like to determine whether a given array A has a majority element, and if so, find the element. Use a Mergesort-style divide-and-conquer approach to solve the majority element problem in $O(n \log n)$ time.

(*Hint:* Split the array into two arrays A_1 and A_2 of half the size. Use a divide-and-conquer approach that finds the majority element of A, if it exists, using the knowledge of majority elements of A_1 and A_2, if they exist.)

Idea: split array in half (L, R)

if there is a majority elem m in A,

then m must also be a majority elem.

in L and/or R

Majority(A[l, ..., r])

if l = r
return A[l]

n = len(A)/2

L = A[l: n]

R = A[n+1 : r]

$\perp$ m_L = majority(L)

m_R = majority(R)

if freq of m_L in A > n: return m_L

if freq of m_R in A > n: return m_R

return $\perp$ (False)

$$\boxed{\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + O(n) \\ T(1) &= O(1) \end{aligned}}$$

The "Master Theorem"

- Recipe for recurrences of the form:
  - $T(n) = \boldsymbol{a} \cdot T(n/\boldsymbol{b}) + Cn^{\boldsymbol{d}}$

$a = 2$
$b = 2$
$d = 1$

- Three cases:
  - $\left(\frac{a}{b^d}\right) > 1: \; T(n) = \Theta\left(n^{\log_b a}\right)$
  - $\left(\frac{a}{b^d}\right) = 1: \; T(n) = \Theta\left(n^d \log n\right)$ $\leftarrow$ $\Theta(n \log n)$
  - $\left(\frac{a}{b^d}\right) < 1: \; T(n) = \Theta\left(n^d\right)$

$\frac{2}{2^1} = 1$

Problem 3

Consider the following recursive algorithm:

Algorithm 1: Recursive Algorithm

Function RECALG(n):
    If $n == 1$ **:**
        print("Hello")
    Else
        RECALG($n/2$)
        RECALG($n/2$)
        For $i \in \{1, \ldots, \lfloor n^{3/2} \rfloor\}$
            print("Hello")

$a = 2$
$b = 2$
$d = 3/2$

$\frac{2}{2^{3/2}} < 1$

Case 3

(a) State a recurrence which captures the number of times "Hello" is printed. Your recurrence should be of the form "$T(n) = \ldots$". Don't forget your base case(s)!

$T(n) = 2T(\frac{n}{2}) + n^{3/2}$; $T(1) = 1$

(b) Solve your recurrence from part (a) with the Master Theorem. That is, give a simplest function $g(n)$ such that $T(n) = \Theta(g(n))$. $T(n) = \Theta(n^{3/2})$

G