CS 170 Homework 1

Due **9/6/2021, at 10:00 pm (grace period until 11:59pm)**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write none.

In addition, we would like to share correct student solutions that are well-written with the class after each homework. Are you okay with your correct solutions being used for this purpose? Answer "Yes", "Yes but anonymously", or "No"

2 Recurrence Relations

For each part, find the asymptotic order of growth of T; that is, find a function g such that $T(n) = \Theta(g(n))$. In all subparts, you may ignore any issues arising from whether a number is an integer.

(a) $T(n) = 4T(n/2) + 42n$

(b) $T(n) = 4T(n/3) + n^2$

(c) $T(n) = T(\sqrt{n}) + 1$ (You may assume that $T(2) = T(1) = 1$)

1. 3037534078 Ye Tian

3037534676 Shenghan Zheng

2. Apply master thm

(a) $T(n) = 4T(\frac{n}{2}) + \Theta(n) \Rightarrow T(n) = \Theta(n^2)$ $g(n) = n^2$

(b) $T(n) = 4T(\frac{n}{3}) + \Theta(n^2) \Rightarrow T(n) = \Theta(n^2)$ $g(n) = n^2$

(c) $T(n) = T(\sqrt{n}) + 1$ $\because T(2) = T(1) = 1$

∴ we just need to count how many root square should we apply on n to make it less than 2

To avoid situation for not being an integer assume $n = 2^{2^k}$ ($k = \log_2 \log_2 n$, $k \in \mathbb{Z}$)

$T(n) = T(\sqrt{n}) + 1 = T(n^{\frac{1}{2^1}}) + 1 = T(n^{\frac{1}{2^2}}) + 2$

$= \cdots = T(n^{\frac{1}{2^k}}) + k = T(2) + k = 1 + k$

$= 1 + \log_2 \log_2 n$

pick $g(n) = \log_2 \log_2 n$

3 Euclid GCD

Euclid's algorithm is an algorithm that computes the greatest common divisor between two integers $a, b \geq 0$:

```
function EUCLID-GCD(a, b)
    if a > b then
        return EUCLID-GCD(b, a)
    else if a = 0 then
        return b
    else
        return EUCLID-GCD(b mod a, a)
```

(3, b) = (0, 3)

As an introduction to the types of things we'll be doing in CS 170, let's analyze this algorithm's runtime and correctness.

(a) Let $n = a + b$. We will try to write this algorithm's runtime as $\mathcal{O}(f(n))$ for some function f, counting every integer operation (including **mod**) as one operation taking constant time. Starting with EUCLID-GCD(a, b) where $a < b$, what will the arguments be after two recursive calls?

(b) Assume still that $a < b$, and let n' be the sum of a and b after two recursive calls. Show that after two recursive calls, $n' \leq \frac{1}{2}n$. What runtime do we end up with, in the form $\mathcal{O}(f(n))$?

Hint: Use the properties that: (1) b **mod** $a \leq a$, (2) If $b \geq a$, then b **mod** $a \leq b - a$.

```
function EUCLID-GCD(a, b)
    if a > b then
        return EUCLID-GCD(b, a)
    else if a = 0 then
        return b
    else
        return EUCLID-GCD(b mod a, a)
```

(c) Let's look at correctness. We can prove that this algorithm returns the correct answer using a proof by induction. Note that the result of the algorithm does not depend on if $a > b$ or $a \leq b$, so let's focus on the case that $a \leq b$, and the other case follows.

As our base case, let $a = 0$. Argue that EUCLID-GCD$(0, b)$ returns the correct answer for all b.

(d) Now for the inductive step: Assume that EUCLID-GCD(a, b) computes the right answer for all $a \leq b \leq k - 1$. Show that EUCLID-GCD(a, b) computes the right answer for all $a \leq b = k$.

Hint: Use the property that $d \mid a$ and $d \mid b \iff d \mid a$ and $d \mid (b$ **mod** $a)$.

4 Sequences

Suppose we have a sequence of integers A_n, where $A_0, \ldots, A_{k-1} < 50$ are given, and each subsequent term in the sequence is given by some integer linear combination of the k previous

(a) ① if $a=0$ EUCLID-GCD(a,b) will just return b for the first call

② $a>0$ after first call we have EUCLID-GCD $(b \bmod a, a)$ $\because$ $b \bmod a \leq a-1$ $\therefore$ $b \bmod a < a$ we don't need to switch

1) if $a \mid b$ it will return a after 2 recursive call

2) if $a \nmid b$ the arguments became

EUCLID-GCD$(a \bmod (b \bmod a), b \bmod a)$

(b) denote $b = ap + q$, $p = [\frac{b}{a}]$ $q = b \bmod a$

from (a) we know that if the algorithm doesn't return after 2 recursive calls, we get

$n' = a \bmod (b \bmod a) + (b \bmod a) = a \bmod q + q$

$\leq a - q + q = a$

$\because a < b$ $\therefore$ $n' \leq \frac{n}{2} = \frac{1}{2}(a+b)$

(c)

define $r_0 = b$, $r_1 = a$ $\quad a < b$

$r_0 = r_1 p_1 + r_2 \quad p_1 = \left[\frac{r_0}{r_1}\right] \quad r_2 = r_0 \bmod r_1 \in [0, r_1 - 1]$

$r_1 = r_2 p_2 + r_3 \quad p_2 = \left[\frac{r_1}{r_2}\right] \quad r_3 = r_1 \bmod r_2 \in [0, r_2 - 1]$

⋮ ⋮

$r_t = r_{t+1} p_{t+1} + r_{t+2} \quad p_{t+1} = \left[\frac{r_t}{r_{t+1}}\right] \quad r_{t+2} = r_t \bmod r_{t+1} \in [0, r_{t+1} - 1]$

it terminates at r_{t+2}, we show that

$\gcd(a, b) = r_{t+1}$

clearly $r_{t+2} = 0$ because $r_0 > r_1 > \cdots > r_{t+2}$

and they are non-negative integers

$0 \equiv r_0 \equiv r_1 p_1 + r_2 \equiv r_2 \pmod{\gcd(a, b)}$

likewise, we get $r_i \equiv 0 \pmod{\gcd(a, b)} \quad (i = 0, 1, \cdots, t+1)$

if $r_{t+1} = k \cdot \gcd(a, b) \quad (k > 1)$

$r_t \equiv r_{t+1} p_{t+1} + \overset{0}{r_{t+2}} \equiv 0 \pmod{k \gcd(a, b)}$

$\Rightarrow \quad r_{t-1} \equiv r_t p_t + r_{t+1} \equiv 0 \pmod{k \gcd(a, b)}$

$\Rightarrow \quad \cdots$

$\Rightarrow \quad r_0 \equiv r_1 p_1 + r_2 \equiv 0 \pmod{k \gcd(a, b)}$

$\Rightarrow$ $k \gcd(r_0, r_1) \mid \gcd(r_0, r_1)$ contradiction !

$\Rightarrow$ $k=1$ $\Rightarrow$ $r_{t+1} = \gcd(r_0, r_1)$

(Lemma 1 : $a \in \mathbb{Z}^+$ $b \in \mathbb{N}$, $a \mid b$ then $a \mid b-a$)

pf of lemma :

$\because$ $a \equiv 0 \pmod a$ $b \equiv 0 \pmod a$

$\because$ $a \equiv b \pmod a$ $\therefore$ $b-a \equiv 0 \pmod a$

$\therefore$ $a \mid b-a$ proved !

Back to (c)

for the problem E-GCD(0, b) returns b

which it's true from lemma 1

$\Rightarrow$ E-GCD(r_{t+1}, r_{t+2}) returns true value

proved !

(d) Induction

Assume that E-GCD(a,b) compute right answer for all $a \le b \le k-1$ $(k \ge 1)$

for $a, b \le k$

if $a, b \le k-1$ it returns correct answer from assumption

if $a = b = k$ E-GCD(a,b) returns a it's true

if $b = k$ $a < b$

E-GCD(a,b) returns E-GCD(b mod a, a)

otherwise it returns b when a=0 which always get the correct answer from (c)

b mod a $\le a \le k-1$ $a \le k-1$

by assumption, it returns correct answer

By induction E-GCD(a,b) returns true answer for

$\forall a, b \in N$

terms: $A_i = A_{i-1}b_1 + A_{i-2}b_2 + \cdots + A_{i-k}b_k$. You are given as inputs A_0 through A_{k-1} and the coefficients b_1 through b_k.

(a) Devise an algorithm which computes A_n **mod** 50 in $\mathcal{O}(\log n)$ time (**Hint:** use the matrix multiplication technique from class). You should treat k as a constant, and you may assume that all arithmetic operations involving numbers of $\mathcal{O}(\log n)$ bits take constant time. [1]

Give a 3-part solution as described in the homework guidelines.

(b) Devise an even faster algorithm which doesn't use matrix multiplication at all. Once again, you should still treat k as a constant.

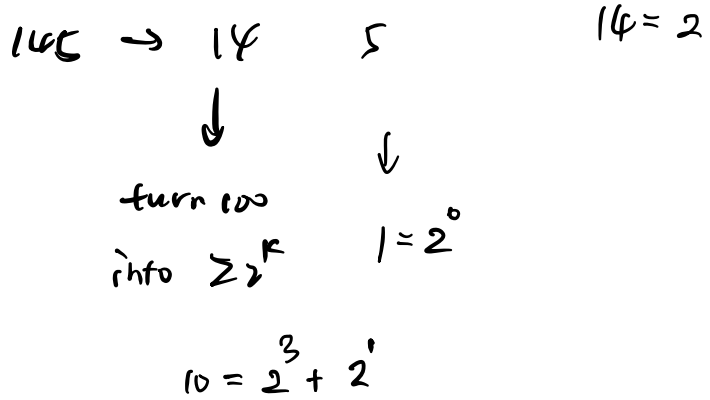
Hint: Exploit the fact that we only want the answer mod a constant (here 50).

Give a 3-part solution as described in the homework guidelines.

5 Decimal to Binary

Given the n-digit decimal representation of a number, converting it into binary in the natural way takes $O(n^2)$ steps. Give a divide and conquer algorithm to do the conversion and show that it does not take much more time than Karatsuba's algorithm for integer multiplication.

Just state the main idea behind your algorithm and its runtime analysis; no proof of correctness is needed as long as your main idea is clear.



[1] A similar assumption – that arithmetic operations involving numbers of $\mathcal{O}(\log N)$ bits take constant time, where N is the number of bits needed to describe the entire input – is known as the *transdichotomous word RAM model* and it is typically at least implicitly assumed in the study of algorithms. Indeed, in an input of size N it is standard to assume that we can index into the input in constant time (do we not typically assume that indexing into an input array takes constant time?!). Implicitly this is assuming that the register size on our computer is at least $\log N$ bits, which means it is natural to assume that we can do all standard machine operations on $\log N$ bits in constant time.

4. ① Algorithm

(a) $$B = \begin{bmatrix} b_1 & b_2 & \cdots & b_k \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & & & \\ 0 & \cdots & 1 & 0 \end{bmatrix} \qquad \begin{pmatrix} A_i \\ A_{i-1} \\ \vdots \\ A_{i-k+1} \end{pmatrix} = B \cdot \begin{pmatrix} A_{i-1} \\ \vdots \\ A_{i-k} \end{pmatrix}$$

$$\begin{pmatrix} A_n \\ \vdots \\ A_{n-k+1} \end{pmatrix} = B^{n-k} \cdot \begin{pmatrix} A_{k-1} \\ \vdots \\ A_0 \end{pmatrix}$$, then get A_n

② Correctness apply induction on B's index

base: when index = 1, we get vector with form $\begin{pmatrix} A_k \\ \vdots \\ A_1 \end{pmatrix}$

$$B \cdot \begin{pmatrix} A_{k-1} \\ \vdots \\ A_0 \end{pmatrix} = \begin{pmatrix} b_1 A_{k-1} + \cdots + A_0 b_k \\ A_{k-1} \\ \vdots \\ A_{k-} \end{pmatrix} = \begin{pmatrix} A_k \\ \vdots \\ A_1 \end{pmatrix}$$

assume when index = j we get the vector $\begin{pmatrix} A_{k-1+j} \\ \vdots \\ A_j \end{pmatrix}$

$$B \cdot \begin{pmatrix} A_{k-1+j} \\ \vdots \\ A_j \end{pmatrix} = \begin{pmatrix} b_1 A_{k-1+j} + \cdots + b_k A_j \\ A_{k-1+j} \\ \vdots \\ A_{j+1} \end{pmatrix}$$ proved!

③ runtime

Because we assume that the operation take constant time. With the given fast matrix powering in lecture, it's $O(\log n)$

(b)

① algorithm

define $B^{(k)}_{i,j} = \sum_{n=1}^{k} B^{(1)}_{i,n} B^{(k-1)}_{n,j} \bmod 50$ ... 1) where

$$B^{(1)}_{i,j} = \begin{cases} b_j & i = 1 \\ 1 & i = j+1 \\ 0 & \text{else} \end{cases} \qquad B^{(0)}_{i,j} = \begin{cases} 1 & i = j \\ 0 & \text{else} \end{cases}$$

Faster:

input: $b_1, \cdots, b_k$, n (suppose ≥ 2), $B^{(1)}_{i,j}$ $B^{(0)}_{i,j}$ $(i, j = 1, 2, \cdots, k)$, $A_0, \cdots, A_{k-1}$, recursion for B

output: result of A_n

power = 0

p = 0

while power < 50^{k^2}

Calculate $B^{(k)}_{i,j}$ $(i,j = 1, 2, \cdots, k)$ using 1)

if $B^{(k)}_{i,j} = B_{i,j}$ for $\{(i,j) \mid i,j \in N^+, i,j \leq k\}$

p = power

time = 0

While time < $n-k \pmod p$

Calculate $B^{(k)}_{i,j}$ $(i,j = 1, 2, \cdots, k)$ using 1)

Return $\sum_{n=1}^{k} B^{(n-k)}_{i,n} A_{n-1} \pmod{50}$

② Correctness

$$B = \begin{pmatrix} b_1 & b_2 & \cdots & b_k \\ 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & \cdots & 1 & 0 \end{pmatrix}$$

for a square matrix in size $k \times k$, it has

50^{k^2} possibilities if we mod 50 for each entry

Thus, $\exists\ S \in N^+$ $F \in 50^{k^2}$, s.t $B^S \bmod 50 = B \bmod 50$

$$\Rightarrow B \pmod{50} = B^{n-k \pmod S} \pmod{50}$$

$$\sum_{q=1}^{k} B^{n-k \pmod S} \pmod{50}$$ 1,q A_{q-1} is the true

answer

(3) runtime

To find p, we need to do a few times which upper bound is 50^{k^2}, and the compare for k^2 value each time is $O(c)$. Because the upper bound, the calculation of $B_{i,j}^{(k)}$ is $O(c)$

$n-k$ is at most $\log n$ bits

from (a) The complexity is $O(\log \log n)$

5. Multiply is the 'divide and conquer' multiplication

① DecimalToBinary (x)

Input: $x_{n-1} \cdots x_0$: decimal integer represented as an array of bits

Output: binary number

if $n = 1$: tranverse $A = \{0_2, 1_2, 10_2, 11_2, 100_2, 101_2, 110_2, 111_2, 1000_2, 1001_2\}$
  if $x_0 = a$, $a \in A$
    return a

$y \leftarrow$ DecimalToBinary $(x_{n-1} \cdots x_{\lceil \frac{n}{2} \rceil})$

$z \leftarrow$ DecimalToBinary $(x_{\lceil \frac{n}{2} \rceil - 1} \cdots x_0)$

$w \leftarrow$ CalculatePowerOf10 $(\lceil \frac{n}{2} \rceil + 1)$

$a \leftarrow$ DecimalToBinary (w)

$L \leftarrow$ Multiply (a, y)

$b \leftarrow$ Add (L, z)

return b

② Correctness

Apply induction on the length of input

base case $n=1$

$x_0 \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ i.e. $\exists\, a \in A$ s.t. $x_0 = a$

Suppose it's true for $n \in K-1$ $(K \geq 2)$

for $n=K$

DecimalToBinary$(x_{n-1} \cdots x_0)$ will return

y=DecimalToBinary$(x_{n-1} \cdots x_{\lceil \frac{n}{2} \rceil})$, z=DecimalToBinary$(x_{\lceil \frac{n}{2} \rceil - 1} \cdots x_0)$ which will both return the true value

$x = yw + z$ ··· 1) $w = 10^{\lceil \frac{n}{2} \rceil + 1}$

from definition w will be turn to binary

thus 1)'s RHS will be all binary

add them result in binary form of x

proved!

(3) runtime

denote it $T(n)$

in each recursive calls, we call $T(\frac{n}{2})$ 3 times

for x, y, a Calculate Power Of 10 just add $\lceil \frac{n}{2} \rceil + 1$ 0 after 1 it's $\Theta(n)$

Multiply is $\Theta(n^{\log_2 3})$, Add is $\Theta(n)$

$\Rightarrow T(n) = 3T(\frac{n}{2}) + \Theta(n^{\log_2 3})$

Apply master thm $T(n) = \Theta(n^{\log_2 3})$

which is same as karatsuba $\Rightarrow$ it doesn't take much more time

proved!