## 1-3

easy parts.

## 4 Recurrence Relations

(a)

$$\begin{aligned}T(n) &= 4T(\frac{n}{4}) + 32n \\ &= 4^{\lceil \log_4 n \rceil} T(\frac{n}{4^{\lceil \log_4 n \rceil}}) + 32 \sum_{d=0}^{\lceil \log_4 n \rceil} \frac{n}{4^d} \cdot 4^d \\ &= \mathcal{O}(n) \cdot \lceil \log_4 n \rceil \\ &= \mathcal{O}(n \log n)\end{aligned}$$

(b)

$$\begin{aligned}T(n) &= 4T(\frac{n}{3}) + n^2 \\ &= 4^{\lceil \log_3 n \rceil} T(\frac{n}{3^{\lceil \log_3 n \rceil}}) + \sum_{d=0}^{\lceil \log_3 n \rceil} (\frac{n}{3^d})^2 \cdot 4^d \\ &= \mathcal{O}(n^2) \cdot \frac{\frac{4}{9}^{\lceil \log_3 n \rceil} - 1}{1 - \frac{4}{9}} \\ &= \mathcal{O}(4^{\log_3 n}) \\ &= \mathcal{O}(n^{\log_3 4})\end{aligned}$$

(c)

$$\begin{aligned}T(n) &= T(\frac{3n}{5}) + T(\frac{4n}{5}) \\ &= T((\frac{3}{5})^2 n) + 2T(\frac{3}{5} \cdot \frac{4}{5} n) + T((\frac{4}{5})^2 n) \\ &= ... \\ &= (\frac{3}{5} + \frac{4}{5})^{-\log_{\frac{3}{5}} n} \cdot T(n) \\ &= \mathcal{O}(n^{\log_{\frac{5}{3}} \frac{7}{3}}) \\ &\approx \mathcal{O}(n^{1.658})\end{aligned}$$

## 5 In Between Functions

$$\exists d \in \mathbb{R}, \forall a > 1, \frac{f}{a^n} \leq d \tag{1}$$

$$\exists h \in \mathbb{R}, \forall c > 0, \frac{n^c}{f} \leq h \tag{2}$$

We assume that:

$$f = a^n + n^c$$

such f will be eligible.

*Proof.* We primarily verify formula (1). So we have

$$\frac{f}{a^n} = 1 + \frac{n^c}{a^n} \tag{3}$$

$$= 1 + a^{c \log_a n - n} \tag{4}$$

Because

$$h(x) = c \log_a n - n$$

get its maximum value at $x_0 = \frac{a}{\ln a}$, therefore $\exists d = 1 + a^{h(x_0)}$, making formula (1) workable.

Likewise, we treat formula (2) as same as formula (1), So we have

$$\frac{n^c}{f} = 1 - \frac{1}{1 + \frac{n^c}{a^n}} \tag{5}$$

$$= 1 - \frac{1}{1 + a^{c \log_a n - n}} \tag{6}$$

therefore $\exists h = \frac{a^{h(x_0)}}{1+a^{h(x_0)}}$, making formula (2) workable. □

# 6 Sequences

(a) Devise an algorithm which computes $r \equiv A_n \mod 50$ in $\mathcal{O}(\log n)$ time:

**Algorithm 1** My algorithm for computing $A_n \mod 50$

**Input:** A value $n$, an array $A = (A_{k-1}, ..., A_0)$ and the another array $b = (b_1, ..., b_k)$
**Output:** A integar value $r \equiv A_n \mod 50$
M $\leftarrow \begin{bmatrix} b[1, ..., k-1] & b_k \\ I_k & 0 \end{bmatrix}$
$A_1 \leftarrow ExpBySquaring(M^n) * Transpose(A)$
r $\leftarrow A_1[0] \mod 50$
return r

*Proof.* Since

$$A_k = \sum_{i=0}^{k} b_i A_{k-1-i} \tag{7}$$

$$= \begin{bmatrix} b_1 & \dots & b_k \end{bmatrix} \begin{bmatrix} A_{k-1} \\ \vdots \\ A_0 \end{bmatrix} \tag{8}$$

We reckon upon vectorising formula (8), then

$$\begin{bmatrix} A_k \\ \vdots \\ A_1 \end{bmatrix} = \begin{bmatrix} b_1 & b_2 & \dots & b_{k-1} & b_k \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix} \begin{bmatrix} A_{k-1} \\ \vdots \\ A_0 \end{bmatrix}$$

Suppose $M = \begin{bmatrix} b_1 & b_2 & \dots & b_{k-1} & b_k \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{bmatrix}$, then

$$\begin{bmatrix} A_n \\ \vdots \\ A_{n-k} \end{bmatrix} = M^{n-1} \begin{bmatrix} A_{k-1} \\ \vdots \\ A_0 \end{bmatrix}$$

Hence we can try out to calculate $M^{n-1}$ for attaining $A_n$. □

**Runtime Analysis.** *With using exponentiation by squaring algorithm to calculate Matrices multiplication, which by the master theorem works out to* $\mathcal{O}(k^3 \log n)$.

(b) Devise an even faster algorithm which doesn't use matrix multiplication at all:

**Algorithm 2** My algorithm for computing $A_n \mod 50$

**Input:** A value $n$, an array $A = (A_{k-1}, ..., A_0)$ and the another array $b = (b_1, ..., b_k)$
**Output:** A integar value $r \equiv A_n \mod 50$
Suppose P(x) $\leftarrow x^k - \sum_{i=0}^{k-1} b_{i+1}x^i$
Suppose D(x) $\leftarrow x^n$
R(x) $\leftarrow ExpBySquaring(D(x), Q(x))$
Get coefficients from R(x) to compute r, then r $\leftarrow r \mod 50$
return r

*Proof.* We try to prove a theorem by means of generating function $G(x) = \sum a_i x^i$:

**Theorem.** *Each term of a series determined by the linear recursion of the k-order order can be obtained by the linear combination of the first k-terms of the series.*

*Proof.* Apparently true when $n < k$. When $n \geq k$, it is obtained by recursion:

$$G(x) = A_n x^n = \sum_{i=1}^{k} b_i x^i A_{n-i} x^{n-i}$$

So the theorem holds. □

Given $F\left(\sum c_i x^i\right) = \sum c_i b_i$ , so the answer is $F\left(x^n\right)$. Due to

$$A_n = \sum_{i=1}^{k} A_{n-i} b_i$$

We can prove these formulas as follows:

$$F\left(x^n\right) = F\left(\sum_{i=1}^{k} b_i x^{n-i}\right)$$

$$\Longrightarrow F\left(x^n - \sum_{i=1}^{k} b_i x^{n-i}\right) = F\left(x^n\right) - F\left(\sum_{i=1}^{k} b_i x^{n-i}\right) = 0$$

Let

$$G(x) = x^k - \sum_{i=1}^{k} b_i x^{k-i}$$

$$A(x) = x^n$$

then $F\left(A(x) + G(x)\right) = F(A(x)) + F\left(G(x)\right) = F(A(x))$ . Then you can reduce the order of $A(x)$ by adding or subtracting multiples of $G(x)$ to $A(x)$ multiplicative times. That is to find $F(A(x) \bmod G(x))$ . The order of $A(x) \bmod G(x)$ does not exceed $k-1$, and $A_{0..k-1}$ has been given, it can be counted. □

**Runtime Analysis.** *The problem is transformed into quickly finding* $x^n \bmod G(x)$ *, as long as the multiplication and modulo in exponentiation by squaring algorithm are replaced by polynomial multiplication and polynomial modulo, it can be done in* $\mathcal{O}(k \log k \log n)$*. This problem is solved within the time complexity.*

# 7 Decimal to Binary

Given the n-digit decimal representation of a number, converting it into binary in the natural way takes $\mathcal{O}(n^2)$ 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:

---

**Algorithm 3** My algorithm for converting decimal to binary: $DivAndCoqD2B(d)$

**Input:** A n-digit decimal representation of a number $d$
**Output:** A binary number $b$
**if** n == 1 **then**
  return $NormalD2B(d)$
**end if**
$d_l, d_r \leftarrow$ leftmost $\lceil \frac{n}{2} \rceil$ , rightmost $\lfloor \frac{n}{2} \rfloor$ digits of d
$b_l, b_r \leftarrow DivAndCoqD2B(d_l), DivAndCoqD2B(d_r)$
return $b_l * DivAndCoqD2B(10^{\lfloor \frac{n}{2} \rfloor}) + b_r$

---

*Proof.* Its context is omitted due to some restriction of this question. □

**Runtime Analysis.** *Since our method for converting n-digit decimals starts by making recursive calls to convert these three pairs of n/2-digit decimals (three subproblems of half the size), and then evaluates the preceding expression in* $\mathcal{O}(n^2)$ *time. Writing* $T(n)$ *for the overall running time on n-digit inputs, we get the recurrence relation*

$$T(n) = 3T(\frac{n}{2}) + \mathcal{O}(n^2)$$

*Hence in the meantime, this particular one works out to* $\mathcal{O}(n^{\log_2 3})$*, the same running time as Karatsuba's algorithm for integer multiplication.*