1. (20 Points) Let $m$ be a randomly chosen non-negative integer having at most $n$ decimal digits, i.e. an integer in the range $0 \leq m \leq 10^n - 1$. Consider the following problem: determine $m$ by asking only 5-way questions, i.e. questions with at most 5 possible responses. For instance, one could ask which of 5 specific sets $m$ belongs to. Prove that any algorithm restricted to such questions, and which correctly solves this problem, runs in time $\Omega(n)$.

Proof:
Since there are exactly $10^n$ possible verdicts, and every question has at most 5 answers, any algorithm solving this problem can be represented by a decision tree whose height $h$ satisfies

$$h \geq \lceil \log_5(10^n) \rceil = \lceil n \cdot \log_5(10) \rceil = \Omega(n).$$

Therefore any algorithm that solves this problem runs in time $\Omega(n)$. ■

2. (20 Points) Recall that the average case run time of RandSelect() satisfies the recurrence relation

$$t(n) = (n - 1) + \frac{n - 1}{n^2} \sum_{k=1}^{n-1} t(k)$$

Prove that $t(n) = \Theta(n)$.

Proof:
First observe that $t(n) \geq n - 1 = \Omega(n)$, so $t(n) = \Omega(n)$. We prove now that $t(n) \leq 2n$ for all $n \geq 1$ by induction. This shows that $t(n) = O(n)$, and hence $t(n) = \Theta(n)$ as required.

I. Observe $t(1) = (1 - 1) + \frac{1-1}{1^2} \cdot \text{(empty sum)} = 0 \leq 2 = 2 \cdot 1$, establishing the base case.

II. Let $n > 1$, and assume for all $k$ in the range $1 \leq k < n$ that $t(k) \leq 2k$. We must show $t(n) \leq 2n$.

$$t(n) = (n - 1) + \frac{n-1}{n^2} \cdot \sum_{k=1}^{n-1} t(k)$$

$$\leq (n - 1) + \frac{n-1}{n^2} \cdot \sum_{k=1}^{n-1} 2k$$

by the induction hypothesis

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

$$= n - 1 + \frac{n^2 - 2n + 1}{n}$$

$$= n - 1 + n - 2 + \frac{1}{n}$$

$$= 2n - 3 + \frac{1}{n}$$

$$\leq 2n$$

since $n > 1 \Rightarrow \frac{1}{n} \leq 1 \Rightarrow -3 + \frac{1}{n} \leq 0$.

It follows that $t(n) \leq 2n$ for all $n \geq 1$. ■
3. (20 Points) Assume we are given 3 gold bars (labeled 1, 2, 3), one of which may be counterfeit: either gold-plated lead (heavier than gold) or gold-plated tin (lighter than gold). Consider again the problem of finding which, if any, of the bars is counterfeit and what it is made of. The only tool at your disposal is a balance scale. Each use of the scale produces one of three possible outcomes: tilt left, balance, or tilt right.

a. (10 Points) Give a decision tree argument establishing a lower bound on the (worst case) number of weighing's performed by any algorithm for this problem. (In your proof, clearly enumerate the set of possible verdicts.)

Claim: Any algorithm solving this problem must use the balance at least 2 times.

Proof: There are 7 possible verdicts to this problem: \{1L, 1H, 2L, 2H, 3L, 3H, AllGold\}, and each test has 3 possible outcomes. Thus any algorithm solving this problem can be represented by a decision tree whose height \( h \) satisfies \( h \geq \lceil \log_3(7) \rceil = 2 \). Therefore 2 weighing's are necessary in worst case.

b. (10 Points) Design an algorithm that solves this problem by using the balance scale (at most) the number of times you found in (a). Express your algorithm as a decision tree.

Solution:
In the figure below, a branch left indicates that the scale tilts left, a vertical branch indicates that the scale balances, and branch right indicates the scale tilts right. Note that some outcomes are not possible at certain points in the algorithm, indicated by the absence of a branch right or left.

![Decision Tree](image-url)
4. (20 Points) Consider the following problem: given a bit string of length \( n \), where \( n \) is an odd positive integer, determine whether the string contains more 1's or 0's, i.e. determine the majority bit. The only operation allowed is to peek at a bit.

a. (10 Points) Describe a simple algorithm that solves this problem using \( n \) peeks in worst case, and \( \lceil n/2 \rceil \) peeks in best case.

**Solution:** Step through the string from left to right keeping a running total of both the number of 1's and the number of 0's encountered. As soon as one of the totals reaches \( \lceil n/2 \rceil \), return the corresponding bit as majority. The total number of peeks is \( \lceil n/2 \rceil \) at best, and \( n \) at worst.

b. (10 Points) Give an adversary argument showing that any algorithm for this problem performs \( n \) peeks, in worst case. Clearly describe the daemon's strategy. Prove that if any algorithm returns an answer after fewer than \( n \) peeks, then the daemon can display an input string that is consistent with all answers given, but which contradicts the algorithm's verdict.

**Proof:** Run any algorithm for this problem against the following adversary, simulating a bit string of length \( n \). Daemon's strategy: answer all questions with alternating bits, that is answer 0 to the \( i \)th question if \( i \) is even, and answer 1 to the \( i \)th question if \( i \) is odd.

Suppose the algorithm halts and returns an answer after fewer than \( n \) peeks. Then at most \( (n - 1)/2 \) of the daemon's answers have been 1, and at most \( (n - 1)/2 \) have been 0. If the algorithm returns 1 as the majority bit, the daemon can claim all remaining bits are 0. This is consistent with all his previous answers, and it implies that the number of 0's is at least \( (n + 1)/2 \), contradicting the algorithm’s verdict. If the algorithm returns 0 as majority bit, the daemon can claim all the remaining bits are 1, implying the number of 1's is at least \( (n + 1)/2 \), again contradicting the algorithm’s verdict. Thus any algorithm performing fewer than \( n \) peeks cannot be correct.
5. (20 Points) Let $k$ be an integer in the range $9 < k < 27$, and assume there exists a method for multiplying two $3 \times 3$ matrices by performing sums and products of the matrix elements, and in which only $k$ of the operations are products (and which product is not assumed to be commutative.)

a. (5 Points) Explain how this method can be used to recursively multiply two $n \times n$ real matrices, where $n$ is an exact power of 3. (You need not write pseudo-code, a verbal description will suffice.)

Solution:
Regard an $n \times n$ square matrix (where $n$ is a power of 3) as a $3 \times 3$ matrix, each of whose 9 elements is a square submatrix of size $\frac{n}{3} \times \frac{n}{3}$. To multiply two $n \times n$ square matrices, we multiply two $3 \times 3$ matrices of matrices. We suppose this multiplication can be done by performing only $k$ multiplications of the underlying $\frac{n}{3} \times \frac{n}{3}$ matrices (which are non-commutative operations). Since $n$ is a power of 3, we can recur on this process down to matrices of size $1 \times 1$, where the recursion halts. At each level, there are $k$ recursive multiplications of $9$ matrices whose size is $1/3$rd that of the current level.

b. (3 Points) Write a recurrence relation for the running time $T(n)$ of the algorithm you described in (a). (Note that this recurrence will contain $k$ as a parameter.)

Solution:
We can write this as either $T(n) = kT(n/3) + \Theta(1)$ or $T(n) = kT(n/3) + \Theta(n^2)$. The first term is the cost of the $k$ recursive calls. In the first recurrence, $\Theta(1)$ is the overhead cost of the current recursive invocation. In the second recurrence, $\Theta(n^2)$ is the cost of the real number additions needed to compute the product. (Note to grader: consider either recurrence to be correct.)

c. (4 Points) Use the Master Theorem to find an asymptotic solution to the recurrence you found in (b). (Note your answer will again depend on $k$.)

Solution:
$T(n) = kT(n/3) + \Theta(1)$:
Compare $1 = n^0$ to $n^{\log_3(k)}$. Let $\epsilon = \log_3(k) - 0$. Then $\epsilon > 0$, and $1 = O(n^{\log_3(k)-\epsilon})$, so by case 1 we have $T(n) = \Theta(n^{\log_3(k)})$.

$T(n) = kT(n/3) + \Theta(n^2)$:
Compare $n^2$ to $n^{\log_3(k)}$. Let $\epsilon = \log_3(k) - 2$. Then $\epsilon > 0$ since $k > 9$, and so $n^2 = O(n^{\log_3(k)-\epsilon})$. Again by case 1 we have $T(n) = \Theta(n^{\log_3(k)})$.

d. (8 Points) Determine the largest integer $k$ for which $T(n) = o(n^{\log_2(7)})$, making your algorithm in (a) better than Strassen's. (Typo Fixed.)

Solution:
We seek the largest integer $k$ such that $n^{\log_3(k)} = o(n^{\log_2(7)})$, or equivalently $\log_3(k) < \log_2(7)$. Therefore $k < 3^{\log_2(7)}$, and hence $k = \lceil 3^{\log_2(7)} \rceil = 21$. ■