Homework 5: Divide & Conquer, Randomized Algorithms, and Fast Fourier Transform

Instructions: **Due 02/18/25 11:59pm**

- Please type your solutions using LaTeX or any other software. Handwritten solutions will not be accepted.
- Please try to write concise responses.
- You should not use only pseudocode to describe your algorithms.
- Unless otherwise stated, saying log means base 2.
- Comparisons and basic arithmetic (addition, subtraction, bit-shifting) operations take $\mathcal{O}(1)$ time.

Q1 A monkey currently has b rotten bananas and seeks to get rid of as many rotten bananas as possible. There are n baskets laid out in order and each basket has a multiplier that can be applied on the current amount of bananas the monkey has. For example, using a basket with a multiplier of 0.5 will halve the amount of rotten bananas the monkey has. These multipliers are stored in the array $M = [m_1, m_2, \ldots, m_n]$. The monkey is allowed to start at some basket i and end at some basket j, where $1 \leq i \leq j \leq n$, applying multipliers to his bananas *without* skipping any baskets. For example, given the multipliers $M = [3, 0.1, 0.2, 100, 0.75, 4, 0.1]$ and $b = 100$ bananas, the monkey's best course of action would be to start at position 2 and stop at position 3. The number of rotten bananas that the monkey would have remaining is then $100(0.1)(0.2) = 2$.

You will design an efficient *divide-and-conquer* algorithm that performs the following task:

Input: An amount of bananas $b > 0$, an array of multipliers $M = [m_1, m_2, \ldots, m_n]$
Output: The maximum amount of bananas the monkey can get rid of.
(doesn't have to be an integer)

(a) Describe your algorithm.

Note: Feel free to merge (a) and (b) if it's easier to explain that way.

> ...

(b) Show the correctness of your algorithm.

> ...

(c) Provide a recurrence for your algorithm and use it to analyze the runtime.

> ...

Q2 Suppose you are walking down a long road, whose length you don't know in advance. Along the road are houses, which you would like to photograph with your very old camera. This old camera allows you to take as many pictures as you want, but only has enough memory to store one picture at a time. The street contains n houses, but you don't know n before you reach the end of the street.

Your goal is to end up with one photo in your camera, where that photo is equally likely to show any of the n houses. One algorithm for achieving this goal is to walk all the way down the street, counting houses, so that we can determine n. Then we roll an n-sided die, where X denotes the roll outcome. Then we walk to the house numbered X and take its picture. However, you're a busy person and you don't want to walk down the street again. Can you achieve your goal by walking up the street only once?

(a) Describe a randomized algorithm to solve this problem.

Hint: Your algorithm will involve replacing the current photo stored in memory with some probability as you walk (only once) down the street.

> ...

(b) Prove that, for all i, $\mathbb{P}(i\text{-th item is output}) = 1/n$.

> ...

Q3 You're a chronic sports gambler and *DraftDuels* releases a promotion regarding the Atlanta Falcons where you think you might win some money. If, throughout the duration of the season, every player on the Falcons roster wins "Player of the Game" for at least one game, you will win a \$100 cash prize. Otherwise, you win nothing. Let's say there's n players on the football team, and each one of them has a $1/n$ chance of earning the "Player of the Game" title for any specific game. Prove that the expected number of games you would need in a season to win your \$100 bet is $\Theta(n \log(n))$.

Hint: Define G_k = number of games to get kth unique player to win "Player of the Game" after $k-1$ different players have already won "Player of the Game". Then, let $G = \sum_k G_k$, and use linearity of expectation.

> …

Q4 We are given n coins which each have their own probability of landing heads c_1,c_2, …, c_n. Now, we flip all the coins and record how many of them landed on heads. We are interested in analyzing the probability that exactly k of the flips landed on heads. You will design an efficient *divide-and-conquer* algorithm that performs the following task:

Input: Coins with probabilities $\mathcal{C} = \{c_1, c_2, \ldots, c_n\}$ of landing heads, and some integer $0 \le k \le n$
Output: The probability exactly k coins landed heads

(a) Lets first try a simple case by hand. If there are 3 coins, with probabilities c_1, c_2, and c_3 of landing on heads, what is the probability that when we flip all of them exactly 2 of them will be heads?

Note: Your answer should be written in terms of c_1, c_2, and c_3.

> …

(b) For each coin c_i, what is a polynomial function $p_i(x)$ such that when you compute

$$p(x) = \prod_{i=1}^{n} p_i(x)$$

the coefficient of the x^kth term of $p(x)$ is the probability that exactly k flips landed on heads?

> …

(c) Use this to describe an algorithm to solve the problem.

> …

(d) Provide a recurrence for your algorithm and use it to analyze the runtime.

Hint: To solve the recurrence, It's easier to divide both sides by n and to try analyzing $\frac{T(n)}{n}$.

> …

Q5 The following is the Fast Fourier Transform algorithm as presented in the textbook.

```
function FFT($a, \omega$):
    Input: An array $a = (a_0, a_1, \ldots, a_{n-1})$, for $n$ a power of 2
    Input: A primitive $n$th root of unity, $\omega$
    Output: $M_n(\omega)a$
    if $\omega = 1$ then
        return $a$
    $(s_0, s_1, \ldots, s_{n/2-1}) \leftarrow \mathrm{FFT}((a_0, a_2, \ldots, a_{n-2}), \omega^2)$
    $(s'_0, s'_1, \ldots, s'_{n/2-1}) \leftarrow \mathrm{FFT}((a_1, a_3, \ldots, a_{n-1}), \omega^2)$
    for $j := 0$ to $n/2 - 1$ do
        $r_j \leftarrow s_j + \omega^j s'_j$
        $r_{j+n/2} \leftarrow s_j - \omega^j s'_j$
    return $(r_0, r_1, \ldots, r_{n-1})$
```

(a) Provide a modification of the algorithm where n is a power of 3, and explain why it works.

Hint: First figure out how to write a polynomial of degree d, where d is a power of 3, as a linear combination of polynomials that are a function of x^3. Next, find a way to group *triplets* of each root of unity instead of the *pairs* that the algorithm from the book follows. Finally, provide a divide and conquer algorithm where each subproblem is inputted with the $(n/3)$rd roots of unity.

> ...

(b) Provide a recurrence for your algorithm and use it to analyze the runtime.

> ...