CS 344: Design and Analysis of Computer Algorithms

Instructor: Kangning Wang

Homework 1: Due on **February 7, 2025**

Acknowledgment: Our team members are Alexander Lin (al1655), Pranav Tikkawar (pt422), and Ivan Zheng (iz72) .

Problem 1

Solve these recurrence relations and represent the answers in the Θ notation. In each of these questions, the given formulas are for $n > 100$, and we know $T(n) = 1$ for all $n \leq 100$. All fractional inputs are rounded down. Other than the first one below, you can write down your answer without providing your proof (which can be tedious), but you should be able to give a proof if asked. Example: $T(n) = 2T(n/2) + \frac{n}{\ln n}$ gives $T(n) = \Theta(n \log \log n)$.

1. $T(n) = T(0.4n)+T(0.3n)+T(0.2n)+n$. (Prove this one formally using strong induction.)
2. $T(n) = T(n/3) + 10$.
3. $T(n) = 7T(n/2) + n^2$.
4. $T(n) = T(n/3) + T(2n/3) + n \ln n$.
5. $T(n) = 2T(n-1) + 1$.
6. $T(n) = T(n-1) + T(n-2) + 1$.
7. $T(n) = T(\sqrt{n}) + 344$.

Solution.

1. $\Theta(n)$ because proof by induction. Base case: for n $\leq$ 100, T(n) = 1. Inductive hypothesis: assume T(k) = $\Theta(k)$ for all k < n. Inductive step: T(n) = $\Theta(0.4n)+\Theta(0.3n)+\Theta(0.2n)+n = \Theta(n)$. Thus, T(n) = $\Theta(n)$.
2. $\Theta(logn)$ because using Master Theorem, a = 1, b = 3, f(n) = 10, c = 0, $\log_b$a = 0. Thus, $\Theta(n^{log_3 1}logn) = \Theta(logn)$.
3. $\Theta(n^{log_2 7})$ because using Master Theorem, a = 7, b = 2, f(n) = n^2, c = 2, $\log_b$a $\approx$ 2.807. Thus, T(n) = $\Theta(n^{log_2 7})$.
4. $\Theta(nlog^2 n)$ because the depth is $\log_3$n and each level's work is nlnn. Thus, $\Theta(nlnn \cdot log_3 n)$ = $\Theta(nlog^2 n)$.

5. $\Theta(2^n)$ because T(n) = 2T(n - 1) + 1 and T(n - 1) = 2T(n -2) + 1. Expanding, 2(2T(n - 2) + 1) + 1 = 4T(n - 2) + 3. T(n - 2) = 2T(n - 3) + 1, expanding again, 4(2T(n - 3) + 1) + 3 = 8T(n - 3) + 7. We can see the formula of T(n) = 2^kT(n - k) + (2^k - 1). Assuming T(0) = 0, T(n) = 2^n - 1. Thus, $\Theta(2^n)$.
6. $\Theta(\phi^n)$ where $\phi = \frac{1+\sqrt{5}}{2}$ because we can see the ressemblance to the Fibonacci sequence in T(n).
7. $\Theta(loglogn)$. Assume n = 2^{2^k}, then $\sqrt{n} = 2^{2^{k-1}}$. T(n) = T($n^{\frac{1}{2}}$) + 344. Expanding, T($n^{\frac{1}{2}}$) = T($n^{\frac{1}{4}}$) + 344 $\rightarrow$ T($n^{\frac{1}{4}}$) = T($n^{\frac{1}{8}}$) + 344. Since n = 2^{2^k}, $log_2(log_2(n))$ = k. Thus, T(n) = 344 $\cdot$ $log_2(log_2(n))$.

Problem 2

Given a directed graph $G = (V, E)$, we want to count the number of walks of length k from u to v for a given pair of vertices (u, v). We are only interested in this number modulo a constant p, so that each basic arithmetic operation only takes constant time. Let A be the adjacency matrix of G, and let $B = A^k$.

1. Prove using induction that B_{uv} is exactly the number of walks of length k from u to v.
2. It is known that $n \times n$ matrix multiplication can be done in $O(n^{2.372})$ time. Design an algorithm that counts the number of walks (modulo p) of length k from u to v in $O(n^{2.372} \log k)$ time, where $n = |V|$.

Solution.

1. Here is my solution.
2.

Problem 3

We have an array of n distinct integers $(a_1, a_2, \ldots, a_n)$. An inversion is a pair of indices (i, j) with $i < j$ and $a_i > a_j$. The weight of an inversion (i, j) is defined as $(a_i - a_j)^2$. Give an algorithm that computes the sum of weights of all inversions in $O(n \log n)$ time.

Solution. Here is my solution.

Problem 4

Given n points $(p_1, p_2, \ldots, p_n)$ on a Euclidean plane in *general position* (that is, no two points coincide and no three points are on the same line), find three different points p_i, p_j, p_k among

them to minimize the perimeter of the triangle with p_i, p_j, p_k as its vertices. Your algorithm should run in $O(n \log n)$ time. You can assume that among the points $(p_1, p_2, \ldots, p_n)$, all x-coordinates are distinct and all y-coordinates are distinct.

Solution. Here is my solution.

Assumptions for Problems 5 and 6

All basic arithmetic operations can be done in constant time, and the fast Fourier transform runs in $O(n \log n)$ time.

Problem 5

The RUCS stock price has an interesting behavior. Every day independently, exactly one of the following events will happen.

- The price doubles ($\times 2$); this happens with probability 0.5.
- The price quadruples ($\times 4$); this happens with probability 0.2.
- The price halves ($\times 0.5$); this happens with probability 0.3.

The price of the RUCS stock is 1 today, and we need to calculate the probability that its price becomes at least p after n days. Design an algorithm for this that runs in $O(n \log n)$ time. $O(n^2)$ running time is also good if $O(n \log n)$ seems out of reach. (Hint: This problem is an application of polynomial multiplication and the fast Fourier transform.)

Solution. Here is my solution.

Problem 6

There are n stars in a different universe, and all of them are negligibly small in terms of volume. They happen to be in the same straight line, and their locations are $1, 2, \ldots, n$. (Thus the distance between each consecutive pair is 1.) The star k has mass $m_k > 0$.

You are interested in knowing the net gravity F_k that each star k receives. It is defined as

$$F_k = -\sum_{j<k} \frac{G m_j m_k}{(k-j)^2} + \sum_{j>k} \frac{G m_j m_k}{(k-j)^2},$$

where G is the gravitational constant.

Design an algorithm to compute all of $F_1, F_2, \ldots, F_k$ in $O(n \log n)$ time. (Hint: This problem is an application of polynomial multiplication and the fast Fourier transform.)

Solution. Here is my solution.