Algorithms: CSE 202 — Homework 1

For each problem, provide a high-level description of your algorithm. Please make sure to include the necessary details that are crucial for its correctness and efficiency. Prove its correctness and analyze its time complexity.

Problem 1: Diameter of a tree (CLRS)

The *diameter* of a tree $T = (V, E)$ is given by

$$\max_{u,v \in V} \delta(u, v)$$

where $\delta(u, v)$ is the shortest path length between the vertices u and v. That is, the diameter of the tree is the length of the longest shortest-path between any two nodes in the tree. Give an efficient algorithm to compute the diameter of a tree, and analyze the running time of your algorithm.

We are expecting a linear (in the number of vertices of the tree) time algorithm for this problem.

Problem 2: Sorted matrix search

Given an $m \times n$ matrix in which each row and column is sorted in ascending order, design an algorithm to find an element.

If the element occurs multiple locations in the matrix, your algorithm is allowed to return the element from any location.

Problem 3: 132 pattern

Given a sequence of n distinct positive integers $a_1, \ldots, a_n$, a 132-pattern is a subsequence a_i, a_j, a_k such that $i < j < k$ and $a_i < a_k < a_j$. For example: the sequence $31, 24, 15, 22, 33, 4, 18, 5, 3, 26$ has several 132-patterns including 15, 33, 18 among others. Design an algorithm that takes as input a list of n numbers and checks whether there is a 132-pattern in the list.

Problem 4: Base conversion

Give an algorithm that inputs an array of n base b_1 digits representing a positive integer in base b_1 in little endian format (that is, the least significant digit is at the lowest array index) and outputs an array of base b_2 digits representing the same integer in base b_2 (again in little endian format). Get as close as possible to linear time. Assume b_1, b_2 are fixed constants.

If we do the conversion digit-by-digit, the number of operations is really $O(n^2)$, since, for example, once we convert each digit we need to add up all the resulting $O(n)$ bit numbers. We can do better using a divide-and-conquer algorithm using a FFT integer multiplication algorithm as a sub-routine. Assume n is a power of 2; otherwise, pad with 0's in the most significant digits.