Algorithms
Homework 4

Saad Mneimneh
Computer Science
Hunter College and the Graduate Center of CUNY

Readings
Based on Lectures 7 and 8 and their assigned readings.

Problem 1
Consider an $n \times n$ 0-1 matrix X, where each entry X_{ij} is either 0 or 1. We square the matrix, then sum up each row. In other words, let

$$A[i] = \sum_{j=1}^{n} X_{ij}^2$$

(a) If each entry in X was chosen independently and uniformly at random, what is the expected value of $A[i]$ for any given i?

(b) Show that $E[\max_i A[i]] = \Theta(n^2)$, and based on that, what is the expected running time of counting sort on A?

(c) Describe a fast way to sort the array A.

Problem 2
We have n people standing on n mountain peaks. The following figure shows an example of these peaks for $n = 6$.

Each person needs to attempt the next challenge, which is to climb the higher mountain closest to their current position and to their right.

The n peaks can be represented by an array of numbers (and let's assume they are integers); for instance, the array for the above example cab be $A = [2, 5, 1, 6, 3, 4]$. The problem becomes as follows: For each index i, find the smallest index j such that $j > i$ and $a[j] > a[i]$. If no such j exists, make $j = i$ (the person stays). For instance, if we imagine another array B, the answer to all the j's for this instance would be $B = [2, 4, 4, 4, 6, 6]$.

(a) Show that the straight forward approach in which we check for each i, every $j = i + 1, i + 2, \ldots, n$, requires $\Omega(n^2)$ time. *Hint*: construct an input that forces the $\Omega(n^2)$ time.

(b) Find an algorithm that performs the task in $O(n)$ time. *Hint*: use a simple data structure.

Problem 3
Given an array $A[1 \ldots n]$ of numbers, assume that the i^{th} smallest element is guaranteed to lie somewhere between position $i-k$ and $i+k$ for all $i = 1 \ldots n$, where k is some fixed constant. Call this condition: almost sorted.

(a) Design an algorithm to sort the array in $O(n \log k)$ time.

(b) Show that any algorithm that sorts an almost sorted array requires $\Omega(n \log k)$ time. *Hint*: think about all possible permutations of $(1, 2, \ldots, n)$ that satisfy the almost sorted condition, and use a decision tree argument.

Problem 4

Given an array A that is not sorted, we want to find two elements, call them x and y, that are "close enough". For instance, consider the average distance between consecutive elements in the sorted order $z_1, z_2, \ldots, z_n$. This average distance can be computed as

$$avgD = \frac{(z_2 - z_1) + (z_3 - z_2) + \ldots + (z_n - z_{n-1})}{n-1}$$

(but it can also be computed without knowing the sorted order)
So let's call two elements x and y close enough if $|x - y| \leq avgD$. Find two such elements in linear time. *Hint 1*: use divide-and-conquer to work with two **balanced subproblems**. *Hint 2*: What can you say about the average distance in a one of the two subproblems?