Homework 3: Greedy and Divide-and-Conquer Algorithms

Handed out Monday, October 2. Due at the start of class Friday, October 13.
Upload your code for Problem 5 to Moodle by 2pm (10/13/2017).

Problem 1. Given a list $A =< a_1, ..., a_n >$ of integers, an oparity inversion is a pair a_i and a_j such that $i < j$, $a_i > a_j$, and a_i and a_j are of different parities. (In other words, it is an inversion in which one number is even and the other is odd.) See the example below in Figure 1.
Design an $O(n\ log\ n)$ algorithm that counts the number of off-parity inversions in a list A containing n elements. Justify your algorithm's correctness and derive its running time.

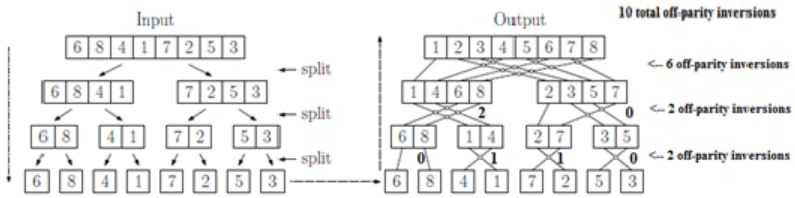

Figure 1: Problem 1.

Problem 2. You are given a set of n lines in the plane. Each line is given by a pair of numbers (a_i, b_i), which represents the line $y = a_i x + b_i$. That is, a_i gives the slope of the line and b_i gives its y-intercept. You are told that $a_i < 0$ (all slopes are negative) and $b_i > 0$ (all intercepts are positive). You are asked to count the number of intersections that occur in the positive (x, y)-quadrant. (For example, in Figure 2 below, there are six intersections in the first quadrant, shown as black dots.)
Of course, this would be easy to do in $O(n^2)$ time, but I want you to develop an answer for this problem that runs in $O(n\ log\ n)$ time.
For simplicity, you may assume that the a_i values are all distinct and the b_i values are also all distinct. You may also assume that no two lines intersect exactly on the x-axis or the y-axis.

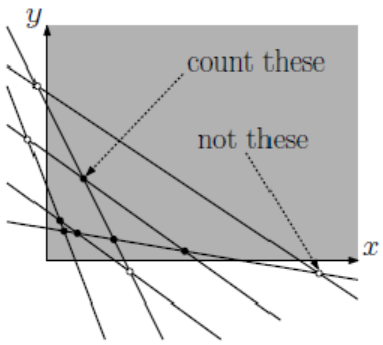

Figure 2: Problem 2.

Problem 3. You are given an undirected graph $G = (V, E)$ in which the edge weights are highly restricted. In particular, each edge has a positive integer weight of either $1, 2, ..., W$, where W is a constant (independent of the number of edges or vertices). Show that it is possible to compute the single-source shortest paths in such a graph in $O(n\ +\ m)$ time, where $n = |V|$ and $m = |E|$.
Hint: Because W is a constant, a running time of $O(W(n\ +\ m))$ is as good as $O(n\ +\ m)$.
Hint: You may make modifications to the original graph.

Problem 4. You are given a connected, undirected graph $G = (V,\ E)$, where each edge $(u, v) \in W$ is labeled with a nonnegative weight $w(u, v)$. Suppose that we run Prim's MST algorithm starting at some source node $s \in V$. Let T be the resulting tree. You would like to know whether T is also the shortest path tree for the single-source node s. Of course, you could run Dijkstra's algorithm and compare the two trees, but this would take additional $O(n\ log\ n\ +\ m)$ time, where $n = |V|$ and $m = |E|$.

Show that (once computed) it is possible to determine whether T is the shortest path tree in only $O(n + m)$ time. Your algorithm takes as input the graph G, a source vertex s, and (as in the output of Prim's algorithm) the tree T is represented by the values $pred[v]$, for each $v \in V$. Your algorithm need only answer 'yes' or 'no'. To avoid messy situations involving ties, you may assume that all edge weights and shortest path lengths are distinct. As always, justify your algorithm's correctness and derive its running time.

Problem 5. Programming - Identifying DAGs Write a Python program that reads in an adjacency matrix representing a graph, either from the command line or as input, and determines if that graph is a DAG (directed acyclic graph) or not. The output for your program should be Yes, it's a DAG, or No, it's not a DAG.

Examples: G1 = [[0,1,1,0,1], [0,0,0,0,1], [0,1,0,1,0], [0,1,0,0,1], [0,0,0,0,0]] Returns "Yes, it's a DAG." G2 = [[0,1,0,1,0], [1,0,1,0,1], [0,1,0,1,1], [1,0,1,0,1], [0,1,1,1,0]] Returns "No, it's not a DAG."

Extra Credit: If the Graph is a DAG, output a valid topological ordering of the nodes.