CSE 101 Homework 1

Background (Order and Recurrence Relations), Speeding up algorithms with restructuring and Data Structures.

Due April 24

100 points total = 10 %

Order (10 points) Is it always the case that $f(2n) \in O(f(n))$? If so, give a proof; if not, give a counter-example.

Triangles(10 points) Let G be an undirected graph with nodes $v_1, ..v_n$. The *adjacency matrix* representation for G is the $n \times n$ matrix M given by: $M_{i,j} = 1$ if there is an edge from v_i to v_j, and $M_{i,j} = 0$. The *adjacency list* representation is an array, which for each node, contains the head of a list of (pointers to) all of the node's neighbors in the graph.

A triangle in the graph is a set of three nodes that are all adjacent. The problem is, given a graph, does it contain a triangle? On the calibration homework, you were asked to give and analyze an $O(nm + n^2)$ time algorithm for this problem in the adjacency matrix representation, where n is the number of nodes and m the number of edges. Can you give an equally fast algorithm if the input is presented in the adjacency list representation? You can use the existence of the algorithm for adjacency matrix representation as a subroutine, without needing details or proof. (Just cite the answer key.)

Forest detection:(20 points) A *forest* is an undirected graph which has no cycles, i.e., no paths from a node to itself. (A forest will be a union of one or more disconnected trees, hence the name.)

Consider the following, high-level algorithm that tells whether an input graph G is a forest:

$IsForest(G : graph)$.

1. While (some node x of G has degree ≤ 1), delete such a node x from G (and an edge adjacent to x if there is one.)
2. If there are any nodes left, return $False$, else return $True$

Prove that $IsForest$ returns $True$ if and only if G is a forest (10 points). Then give an efficient low-level version of the algorithm assuming the graph is given in adjacency list format. (This means specifying the data structures used and giving the algorithm in terms of these data structures.) Analyze the time in terms of both n, the number of nodes, and m the number of edges (10 points).

Top k (20 points) Give an $O(k \log k)$ algorithm that given a max-heap data structure with n elements and an integer k, returns the list of the k largest elements in the heap. Note that you are asked to make the running time independent of n, and just a function of k. (Hint: use a second heap.)

Energy Crisis (20 points) Governor Davis needs long-term contracts for power to keep the lights on. He has a list of n bids $B_I, 1 \leq I \leq N$ from power companies, each with a *rate* (cost per megawatt) r_I and a *capacity* C_I (the maximum number of megawatts the company can guarantee). The governor, to encourage low bids, has guaranteed that he will pay for all accepted bids at the highest rate of any accepted bid. The accepted bids' capacities must sum to at least M, the state's demand for power, to ensure enough power. In addition to the money paid per megawatt, the governor expects each contract to cost a fixed amount F for lawyers, setting up connections to the grid, etc. So he expects the total cost to the state to be $Fk + rM$, where k is the number of accepted bids, and r is the maximum rate of an accepted bid. Give an efficient algorithm, in terms of the number of bids n, that, given F, M and the list of n bids B_I, computes a subset of bids A that minimizes the cost to the state subject to ensuring enough power. (Hint: Say that the rate r was known. Which bids would the governor accept? Then compute this for each possible r, but combine the work for different r's. Try to beat $O(n^2)$.)

Implementation-20 points Implement bubble-sort and heap-sort. Plot their performance on random arrays of n integers with values between 1 and n, for $n = 2^6, 2^8, 2^{10}, 2^{12}, 2^{14}, 2^{16}$. Plot their performance on a log-log scale. Is heap-sort always better than bubble-sort? Why or why not?