Fundamental Algorithms. Sample Final Exam.

In any of the problems in this test, you may use any of the standard data structures discussed in class without giving all the details. For example you may say "Use a 2-3 tree" without further explanation. However, if you are modifying the standard data structure, then you need to explain how the modifications work, and how 2-3 tree operations need to be changed to adapt to this modification.

Use good test-taking strategy. Look quickly through the exam, find the problem that seem easy, and do those first. The answers do not have to be in order in your booklet.

Significant partial credit will be given for answers that are part right. If you cannot find an algorithm with the specified running time, write an algorithm with a slower running time and state what the running time is. If you have an algorithm that is almost but not entirely correct, write that down, and state that you think it is not always correct.

Problem 1: 15 points

An *up-down* sequence is one that first steadily increases then steadily decreases: for instance 2,5,8,9,12,15,10,9 4.

A. Give an $O(\log n)$ algorithm to find the maximum of an up-down sequence.

B. Give an $O(n)$ algorithm to sort an up-down sequence.

Problem 2: 10 points

Consider the following problem: You are given a collection of m sets $s_1 \ldots s_m$ each of size k. The sets are implemented as unordered arrays. You want to find the size of the common intersection of all these; that is the number of elements u that are in all the sets. Find an algorithm to compute in expected time $O(mk)$.

For example, consider the case $m = 3$, $k = 5$,

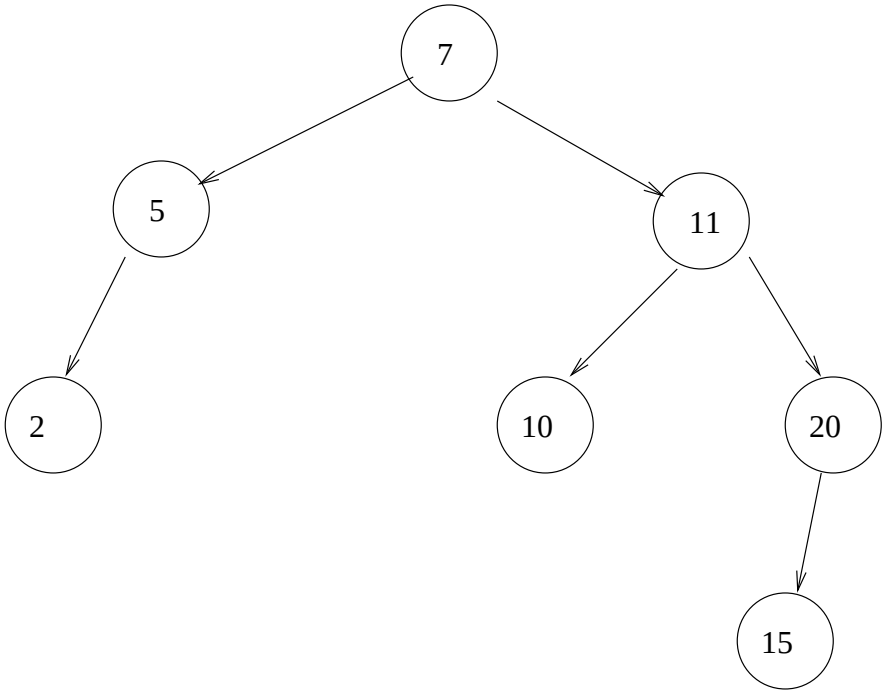
$$s_1 = \{101, 202, 303, 404, 505\}$$
$$s_2 = \{101, 102, 303, 304, 505\}$$
$$s_3 = \{202, 203, 302, 303, 304\}$$

Then the common intersection is $\{303\}$, so the answer is 1.

Problem 3: 10 points

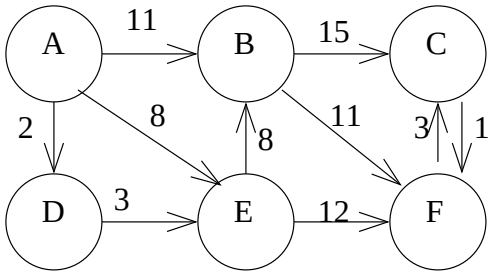
Write an algorithm `CommonAncestor(T,X,Y)` which, given a binary search tree `T`, and two elements `X` and `Y` that are in `T`, finds the lowest common ancestor `A` of `X` and `Y` in `T`. The algorithm does not have to *check* that `X` and `Y` actually are in the tree. It should run in time proportional to the depth of `A`; that is, if `T` is very deep but `A` is very near the root, it should run very quickly.

For example if `T` is the tree shown below, `CommonAncestor(T,2,10)` should return 7. `CommonAncestor(T,10,15)` should return 11. `CommonAncestor(T,15,20)` should return 20.



Problem 4: 10 points

Give a trace of Dijkstra's algorithm running on the graph below, using vertex A as the source. Show the successive values of the chosen vertex v, the set S and the array of distances D; you do not have to show the array used for path recovery.



Problem 5: 20 points

Suppose that you have a directed graph G where the vertices are labelled by positive integers.

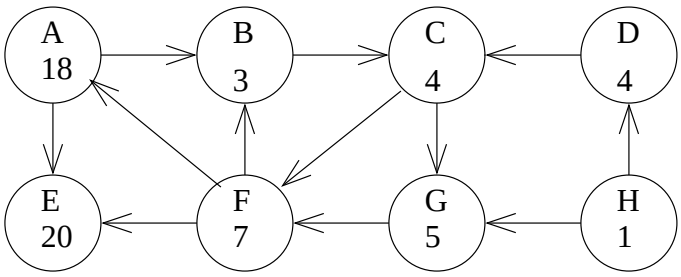
A. An *increasing* path is one where each vertex is greater than its predecessor. Give a linear-time algorithm to compute the longest increasing path through G. Assume that the graph is input as an adjacency-list data structure.

B. A *super-sum* path is one where each vertex is greater than the sum of the preceding nodes on the path. Give a polynomial-time algorithm to find the longest super-sum path through the graph. What is the running time of your algorithm? Assume that the graph is input in terms of an adjacency array.

   Hint: Use dynamic programming. For each vertex V and each integer, record the path of length k ending at V with the lowest total cost.

For instance, in the graph shown below, the longest increasing path is $B \to C \to G \to F \to A \to E$ and the longest super-sum path is $H \to G \to F \to A$.

In both (A) and (B), the algorithm should return the actual path, not just a numerical measure associated with the path.



Problem 6: 10 points

Suppose that you have two sets U and V implemented as two-three trees. Assume that the root of each tree is labelled with the size of the set. Give an algorithm to compute the size of the intersection of the two trees (that is, the number of elements that are in both U and V) in time $O(\min(|U| + |V|, |U| \log(|V|), |V| \log(|U|)$.

Problem 7: 10 points

Suppose that you are given a graph in which there are v vertices, e edges ($e > v$) and the weights on the edges are integers between 1 and e. Give an algorithm for finding the minimum spanning tree problem with a running time that is much faster than $O(e \log e)$.

Problem 8: 15 points

Consider a one-player game of the following structure. The player moves following arcs on a directed graph. Each edge has an associated non-negative toll, which must be spent to cross it.

The arcs have different color: blue, red, and green. Certain vertices are labelled as "upgrade" vertices. Initially the player can only use blue arcs; once he has reached an upgrade vertex the first time, he may use either blue or red arcs; once he has reached an upgrade vertex for the second time, he may use any color arc. He can use the same upgrade vertex twice, but he has to move off it in between (that is, he is not allowed to just stay on the vertex and get both upgrades).

The objective of the game is to get from the "START" vertex to any "GOAL" vertex with the minimum total cost.

For instance, in the graph below, the optimal solution is $A \to B \to E \to C \to B \to E \to F$ for a total cost of 548. (The third upgrade has no effect.) Note that the optimal path uses the edge $B \to E$ twice

Construct an algorithm to find the optimal solution, given a graph of this kind. Hint: Use shortest paths and the method of cloning.

For simplicity, you may assume that, from any vertex U to any other vertex V there is only one edge of any color (e.g. that there is not both a blue and a red edge from A to B), though this assumption is not actually very critical.