CS 331 Test 1 Practice Sheet

These problems are here to guide your study for the midterm. Most of them are not representative of the difficulty of the actual test, though some may incorporate ideas or patterns from problems that will be similar to the ones you will face. Consider each problem a sort of "bullet point" on a list of things you should know.

1. Growth of Functions

For each pair of functions, determine which has a faster asymptotic growth rate. It may be that the two are the same! Do not justify with any formal mathematics. Instead, argue it by a simpler method which you could use to convince your classmates.

Example 1. 2^{2^n} vs e^{e^n}

Example 2. $\left(2^{\lg n}\right)!$ vs 100^n

Problems

Remember that the two functions may have the same asymptotic growth rate! All logarithms are base-2 unless otherwise stated. All functions are functions of n.

The faster-growing function is in red. If neither function is colored, their asymptotic growth rates are the same.

1. n^3 vs $n^2 + 10n$
2. $\log n$ vs $\log(n^{10})$
3. $\log n$ vs $\log_{10} n$
4. e^n vs $e^{\cos n}$
5. e^{2n} vs e^n
6. $\log 2n$ vs $\log n$
7. 100^{100} vs $\log \log \log n$
8. $\sum_{i=0}^{n} i$ vs n^3

2. The Recalcitrant Bachelors

Consider the following algorithm to solve the original Gale-Shapely Marriage problem:

Algorithm 1: Reversed-Order GS Algorithm

```
Initially all m ∈ M and all w ∈ W are free;
while there is a an m who is free and hasn't proposed to every w do
    Choose such a man m;
    Let w be the lowest-ranked woman in m's preference list to whom m has not
     yet offered proposed;
    if w is free then
        (w, m) become engaged
    else
        w is currently engaged with m';
        if w prefers m' to m then
            m remains open;
        else
            w prefers m to m' :
            (m, w) become engaged;
            m' becomes free
return the set S of assigned pairs.
```

Note that the men in this algorithm are rather strange, and will go through their preference list from lowest-to-highest, rather than highest-to-lowest like in the original algorithm. The women behave the same as in the original G-S.

This algorithm quite clearly does not produce stable marriages. (There is at least one counterexample with just two pairs.) Examine the correctness proof for the original G-S algorithm, and identify where it fails for this modified algorithm.

3. Graphs and Trees

Updating MST, Part 1

Suppose that we have a minimum spanning tree, T, of a graph $G = (V, E)$. Now suppose that we add an edge to G. T may no longer be an MST of G. However, recomputing the entire tree is more work than we need to do.

Devise an algorithm to obtain the new MST T' that runs in $O(|V|)$.

Updating MST, Part 2

Instead of adding a new edge, someone decided that they should add a new node instead. Call this new node v.

3. Graphs and Trees continued on next page...

Devise an algorithm to obtain the new MST T'. This algorithm should take roughly $O(dV)$ where d is the degree of the added node (v).

Bridge Builders, MST Application

We have two towns, A and B, that each have their own "significant places."

Your company recently got a contract to service the significant places of both towns. Unfortunately, the roads in these towns are all pretty run-down, so your company will have to make its own road network. In addition, your company needs to build a bridge connecting the two towns. The company would like for the total length of all the roads involved to be as short as possible, while still connecting all points inside the two towns.

You already know that you can create the cheapest road network inside each individual town by using an MST algorithm. All you need to do is decide where to place the bridge.

Assuming that the bridge will always be the same length no matter where you choose to place the endpoints in A and B, what is the optimal way to place the bridge?

Counting Dijkstra

Recall that in Homework 1, we asked you to describe a modification to BFS that allowed it to report both the shortest path and the number of shortest paths in an unweighted graph.

Make a similar modification to Dijkstra's algorithm that allows it to report both the shortest path and the number of distinct shortest paths.

Finding Bipartite Graph

Suggest the required modifications to the following DFS algorithm to determine whether a given undirected graph G is bipartite. You do not have to prove the correctness of your algorithm.

```
DFS(G)
mark all vertices as unvisited
choose some starting vertex s
add s to the list L (stack)
while L nonempty
        visit some vertex v from the list L (pop)
        if v is not visited
            mark v as visited
            for each neighbor w
                add w to the list L
```

3. Greedy Proofs

Carefully study the greedy proofs for both maximum-interval and minimum-lateness scheduling. Make sure you understand both of them on an intuitive level, and could reconstruct them if necessary.

Done that? Good.

Now consider a related problem. The *Interval Coloring Problem* asks how to color a set of intervals such that no two identically-colored intervals overlap, using as few colors as possible. Equivalently, if you think of a multicore computer, it asks how many processing units a computer would need to have to execute the given schedule of tasks, if only one task can run on a processing unit at a given time.

It turns out that for interval coloring (which is superficially quite similar to interval scheduling), Earliest Starting Time is actually a greedy optimal choice.

Sketch a proof of this fact.

More on Exchange Argument..

Consider the following scheduling problem. Each job has a *length* and a *preferred finish time*, and you must schedule all the jobs on a single machine without overlapping. (There is a single start time at which all the jobs become available at once.) If you complete a job before the preferred time, you get a reward equal to the time we have saved. If you complete it after the preferred time, you pay a penalty equal to the amount of time that you are late. Your total reward (which we want to maximize) is the sum of all rewards for early completion minus the sum of all penalties for late completion.

a. Argue carefully that there is an optimal schedule that has no idle time (actually, we want to argue that every optimal schedule has no idle time). That is, every optimal schedule always has one job start at the same time the previous job finishes. (Hint: Show how to change any schedule with idle time to another schedule that has no idle time and gets at least more reward.)

b. Consider the greedy algorithm where we sort the jobs by length and schedule them in that order, shortest job first, with no idle time. Prove that this achieves a net reward at least as great as that of any schedule. (Note: In lecture we proved that a different greedy algorithm is optimal for a different goal, that of minimizing the maximum lateness. You cannot simply quote that result because it does not apply to this algorithm or to these rewards and penalties. But a similar exchange argument will work in this new case.