Algorithms
Computer Science 140 & Mathematics 168
Instructor: B. Thom
Fall 2004
Homework 8b
Due on Tudes, 10/26/04 (beginning of class)

1. [**35 Points**] **Some Prim and Kruskal Games!**

   Suppose that all the edges in a graph are *integers* in the range from 1 to $|V|$. Knowing this, how fast can you make Kruskal's algorithm run? What about Prim's? How about if the edge weights are integers in the range from 1 to W for some *constant* W? In answering these questions, outline how you'd modify each algorithm to achieve a particular runtime, and explain why this runtime is correct. Feel free to rely on any results you've already seen in class, but if you choose to do so, explain what these results are and why you are using them. If you are doing clever things with your data-structures to achieve these run-times, be sure to address these issues (you don't need to write a novel...a picture and an accompanying sentence might nicely suffice).

   After you've answered these questions, enter your runtime results in the table provided below. (This provides useful vantage points from which to view performance). The top row of this table is filled in for you with the run-times derived in class (i.e. no restrictions on edge weights are made). As a result of these entries, you can infer what the definitions of sparse and dense graphs are (sparse: $E \in O(V)$; dense: $E \in O(V^2)$). You'll also want to use this information to reduce your algorithms' run-times to functions of one variable. Finally, you'll want to provide a couple sentences commenting on why your modified run-times behave as they do—what aspects of the algorithms themselves result in the behaviors you see given various graph and weight conditions?

	Prim		Kruskal	
	Sparse Graph	Dense Graph	Sparse Graph	Dense Graph
$w \in \mathcal{R}$	$O(V \log V)$	$O(V^2 \log V)$	$O(V \log V)$	$O(V^2 \log V)$
$w \in \{1, 2, \cdots, W\}$				
$w \in \{1, 2, \cdots, V\}$				

2. [**30 Points**] **Barůvka's Algorithm!** In class we mentioned that the first algorithm for computing minimum spanning trees was published by the Czech mathematician Otakar Barůvka in 1926. The algorithm works like this:

```
E_f = {};  /* Comment: E_f is a subset of a MST */
Treat the V vertices in the graph as V connected components;
while E_f contains fewer than V-1 edges
{
```

```
    for each connected component C
    {
       Find the least weight edge (u,v) with one
         vertex in C and one vertex not in C;
       Add edge (u,v) to E_f;
    }
    Compute the new connected components;
 }
 return E_f /* Comment:  E_f is now a MST! */
```

Notice that if C_1 and C_2 are two different connected components before we begin the for loop, then inside the for loop the algorithm will choose the least weight edge coming out of component C_1 and also the least weight edge coming out of C_2. The edge chosen by C_1 might join C_1 and C_2 into a new connected component, but this new connected component will not be discovered until the for loop has ended! In other words, both C_1 and C_2 will each get an opportunity to choose the least weight edges coming out of their components!

(a) Give a counter-example that shows that Borůvka's Algorithm doesn't work! Show your counter-example graph and explain carefully why Borůvka's Algorithm would not compute a minimum spanning tree in this case.

(b) Now assume that no two edges in the graph have the same weight. By the OPTIONAL BONUS PROBLEM below, such a graph has exactly one minimum spanning tree (you may just use this fact here, although you are encouraged to prove it in the bonus problem!). Under this assumption, prove that Borůvka's Algorithm is correct.

(c) Why doesn't your proof from part (b) work if some edges in the graph have the same weights?

(d) How could Borůvka's Algorithm be modified slightly to work in the most general case that edge weights are not necessarily distinct? Your modification should be simple and elegant.

(e) Describe an implementation (how the algorithm would be implemented using appropriate data structures) for Borůvka's Algorithm and derive its running time. Try to make your implementation as fast as you can.

(f) **[15 Points OPTIONAL BONUS PROBLEM (due Tuesday in class as well)]** Let G be a connected undirected graph in which each edge has a distinct edge weight (that is, no two edges have the same weight). Show that there is a *unique* minimum spanning tree in the graph.