Assignment 10. **Sets and graphs.**

Given April 13, due Monday, April 20.

1. The *binomial tree* (related to binomial heap, CLR, chapter 20) of height $h+1$, T_h, is defined inductively. T_0 is a single element. We have been saying that a tree with a single element has height 1, although one could make a case for saying height 0. T_{h+1} is made from two copies of T_h, call them T_h' with root x', and T_h'' with root x''. In T_{h+1}, none of the pointers is changes except that the parent pointer of x'' is set to point to x', and a child pointer of x' points to x'', if there are child pointers.

a. Show that T_h has height $h+1$ and $n = 2^h$ nodes.

b. The *binomial coefficients*, $\binom{h}{k}$, satisfy the relation $\binom{h}{k} = \binom{h-1}{k} + \binom{h-1}{k-1}$. From this, find a formula for the number of nodes at depth k in T_h,

c. Show that T_h has the fewest elements among all trees of height h that can be made using `make_set` and `union` operations, assuming that we are using the union by rank heuristic. Hint: Let S_h be the sparsest tree of height h that can be made. Show that S_{h+1} is made by joining two copies of S_h, because any other join would either not increase the height, or have more elements than necessary.

2. Suppose we have a family of disjoint sets represented by trees with parent pointers only. Show that the additional work to perform another m `find` operations is $O(n+m)$, where n is the number elements, no matter which elements are being found. (The analysis with `find` and `union` is harder.)

3. In an undirected graph, an edge, `e`, connects `e.head` to `e.tail`. A graph, G', is a subgraph of G if every vertex of G' us a vertex of G, and every edge of G' is an edge of G. A tree is a connected subgraph with no cycles. A spanning tree is a subgraph containing all the vertices of G that is a tree. A spanning forest is a collection of subgraphs, one spanning tree for each connected component. The following algorithm finds connected components:

```
for ( x in V ) { makeset(x); }
for ( e in E ) {
   if ( ( x = find(e.head)) != ( y = find(e.tail)) )  union(x,y); }
```

a. Add a line or two to make a list of edges that forms a spanning tree if G is connected and a spanning forest otherwise.

b. Write code to make this spanning tree into a tree sense we used before. That is, there should be a root, parent pointers, and child lists.