Midterm #2

Instructions: This is a closed book, closed notes exam. There are **7** problems. Do your best to communicate your ideas *clearly* and *succinctly*. Show all work on these pages.

Good luck!

NAME:

On problem	you got	out of
1		10
2		12
3		12
4		14
5		16
6		16
7		20
Σ		100

1 Rooted trees [10 points]

In lecture and in the text there was described a binary tree representation for an ordered forest of rooted trees.

Consider the following ordered forest of rooted trees, F:

Using the representation alluded to, draw the binary tree, T, which represents F.

2 Quicksort [12 points]

Use the method of the **quicksort** algorithm to decompose the problem of sorting the following 10 numbers into the two smaller sorting problems. Don't completely sort the numbers; stop when you have found the two subproblems, clearly identifying them.

Pivot using the **first** element. **Partition** using the procedure described in lecture. **Clearly show your work.**

3 Heaps **[12 points]**

A. Modify the following binary heap to show the one which results after you do a `delete_max()` on it.

B. Modify the following binary heap to show the one which results after you do an `insert(20)` into it.

4 Disjoint Sets [14 points]

A. Suppose you use the oriented forest described in class (and in your text) to implement the `union/find/makeset` operations for manipulating disjoint sets. Suppose you use **union by size** and **collapsing find**. Suppose you currently have a set representation which looks like this:

Draw the oriented forest which you'll have after performing the operation `Union(6,9)`.

B. Starting the forest which **I gave you** in part (A), draw the oriented forest which will result after you do a `Find(8)`.

C. Give the best running time bound you know for the amount of time to do m `makeset/union/find` operations, n of which are `makeset`

5 Binary Tree Traversals **[16 points]**

Consider the following binary tree, T:

A. Post-Order traversal.

List the order in which nodes are visited in a **post-order** traversal of T.

B. Shallow-First Traversal

Consider a new type of binary tree traversal method. Suppose each node keeps track of the depth of the subtree rooted at that node.[1] Consider the following binary tree traversal algorithm:

```
void shallow_first_traversal (node* r)
{
  if (r==NULL) return;
  if      (r->left  && !r->right) VISIT(r->left);
  else if (r->right && !r->left)  VISIT(r->right);
  else if (r->left.depth <= r->right.dept) {       // Line 4
     VISIT(r->left);
     VISIT(r->right);
  }
  else {
     VISIT(r->right);
     VISIT(r->left);
  }
  VISIT(r);
}
```

For the tree T below, indicate the order in which nodes are visited in a shallow-first traversal.

[1] Reminder— the depth of a leaf is defined as 1, and the depth of any non-leaf is defined as one more than the maximum depth of its children.

C. What is the $\Theta(\cdot)$ space requirement to do a shallow-first traversal of an n-node binary tree? Explain.

D. The algorithm for *deep-first traversal* is obtained by changing the `<=` on Line 4 to a `>=`. What is the $\Theta(\cdot)$ space requirement to do a deep-first traversal of an n-node binary tree? Explain.

6 Minimum Spanning Tree **[16 points]**

A. Think back to Kruskal's algorithm for finding a minimum spanning tree. Suppose you run this algorithm on the edge-weighted graph G pictured below. **Darken** the edges which will appear in the resulting minimum spanning tree.

B. Assume the graph given to you has n-nodes and m-edges. Suppose the edges weights are real numbers, to be sorted in the comparison model. What is the best running time you know how to achieve for Kruskal's algorithm?

C. Now suppose the edge weights of the n-node binary tree are guaranteed to be in the range $0..n^3$. Now what is the best running time you can achieve for Kruskal's algorithm? Explain your answer.

7 Generalized Lists [20 points]

Consider the following fragment of the declaration of a generalized list of `int`s:

```
enum node_types {LIST, ATOM};

class glist {

  struct node {
    node_types tag;         //this node is either a
    glist* list;            // LIST  or an
    int atom;               // ATOM.  The tag indicates which.
    node* next;             //next entry in the list
    //...
  };
  node* head;       //head of the list

public:
  //...
};
```

Assume that no header nodes are used—the representation of the empty list, `()`, is `head==NULL`.

A. Draw the internal representation of the glist `((3 4) 1 ())`.

B. Write a public member function `natoms` which returns the total number of atoms in the `glist` object.

C. Write a public member function `operator==` which returns 1 if the left and right operands are `glist` objects of identical contents.

D. I can not simply drop the `tag` field from my `glist` representation, using `list==0` to indicate that the node is an atom. Explain. Suggest a change in the representation of a `glist` which *would* allow me to drop the `tag` field.