Midterm Exam October 31

Your name:____________________________________

	9:30AM	**Table (A–H):**	_______
Identify your Team:	**1PM**	**Table (A–K):**	_______
	2:30PM	**Table (A–E, 1–13):**	_______

- This exam is **closed book** except for a 2-sided cribsheet. Total time is 90 minutes.
- Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem.
- In answering the following questions, you may use without proof any of the results from class or text.

DO NOT WRITE BELOW THIS LINE

Problem	Points	Grade	Grader
1	20		
2	20		
3	20		
4	20		
5	20		
6	20		
7	20		
8	20		
Total	160		

Problem 1 (**Recursive Data**) (**20 points**).
One way to determine if a string has matching brackets, that is, if it is in the set, RecMatch[1], is to start with 0 and read the string from left to right, adding 1 to the count for each left bracket and subtracting 1 from the count for each right bracket. For example, here are the counts for two sample strings:

$$\begin{array}{ccccccccccccc} & [&] &] & [& [& [& [& [&] &] &] &] \\ 0 & 1 & 0 & -1 & 0 & 1 & 2 & 3 & 4 & 3 & 2 & 1 & 0 \end{array}$$

$$\begin{array}{ccccccccccc} & [& [& [&] &] & [&] &] & [&] \\ 0 & 1 & 2 & 3 & 2 & 1 & 2 & 1 & 0 & 1 & 0 \end{array}$$

A string has a *good count* if its running count never goes negative and ends with 0. So the second string above has a good count, but the first one does not because its count went negative at the third step. The empty string λ has a good count of 0. Let

$$\text{GoodCount} ::= \{s \in \{\,],[\,\}^* \mid s \text{ has a good count}\}.$$

One way to prove that RecMatch $=$ GoodCount is to show that each includes the other.

(a) What inductive hypothesis would you use to prove RecMatch $\subseteq$ GoodCount by structural induction?

(b) Use your inductive hypothesis from part (a) to prove RecMatch $\subseteq$ GoodCount by structural induction.

(c) The other inclusion, GoodCount $\subseteq$ RecMatch, can be proved by strong induction. What induction hypothesis would you use for this? No proof is needed.

[1] RecMatch is defined recursively: **Base case:** $\lambda \in$ RecMatch. **Constructor case:** If $s, t \in$ RecMatch, then $[\, s\,]\, t \in$ RecMatch.

Problem 2 (**Infinite Cardinality**) (**20 points**).
Let $\{1, 2, 3\}^\omega$ be the set of infinite sequences containing only the numbers 1, 2, and 3. For example, some sequences of this kind are:

$$(1, 1, 1, 1...),$$
$$(2, 2, 2, 2...),$$
$$(3, 2, 1, 3...).$$

Prove that $\{1, 2, 3\}^\omega$ is uncountable.

Hint: One approach is to define a surjective function from $\{1, 2, 3\}^\omega$ to the power set $\text{pow}(\mathbb{N})$.

Problem 3 (Diagonal Argument / Set Theory) (20 points).

(a) Explain why the union of two disjoint, countably infinite sets is countable, by **explicitly** constructing a bijection with $\mathbb{N}$.

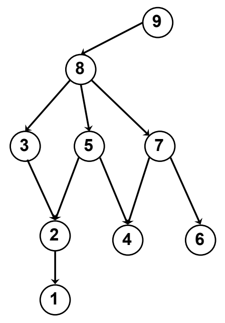
Let $\{0,1\}^*$ be the set of finite binary sequences, $\{0,1\}^\omega$ be the set of infinite binary sequences, and F be the set of sequences in $\{0,1\}^\omega$ that contain only a **f**inite number of occurrences of `1`'s.

(b) Describe a simple surjective function from $\{0,1\}^*$ to F.

(c) The set $\overline{F} ::= \{0,1\}^\omega - F$ consists of all the infinite binary sequences with *infinitely* many `1`'s. Use the previous problem parts to prove that $\overline{F}$ is uncountable.

Hint: We know that $\{0,1\}^*$ is countable and $\{0,1\}^\omega$ is not. *Hint:* You may assume $0, 1^*$ is countable and $0, 1^\omega$ is not, since these were proved in the book.

Problem 4 (Digraphs & Scheduling) (20 points).
The following DAG describes the prerequisites among tasks $\{1, \ldots, 9\}$.



(a) If each task takes unit time to complete, what is the minimum parallel time to complete all the tasks? Briefly explain.

(b) What is the minimum parallel time if no more than two tasks can be completed in parallel? Briefly explain.

Problem 5 (**Partial Order & Equivalence**) (**20 points**).
Say that vertices u, v in a digraph G are *mutually connected* and write

$$u \overset{*}{\longleftrightarrow} v,$$

when there is a path from u to v and also a path from v to u.

(a) Prove that $\overset{*}{\longleftrightarrow}$ is an equivalence relation on $V(G)$.

(b) The blocks of the equivalence relation $\overset{*}{\longleftrightarrow}$ are called the *strongly connected components* of G. Define a relation $\leadsto$ on the strongly connected components of G by the rule

$$C \leadsto D \quad \text{IFF there is a path from some vertex in } C \text{ to some vertex in } D.$$

Prove that $\leadsto$ is a weak partial order on the strongly connected components.

Problem 6 (**Search Tree, Structural Induction**) (**20 points**). **(a)** Edit the labels in this size 15 tree T so it becomes a search tree for the set of labels [1..15].

```
             T
            / \
           /   \
          /     \
         /       \
       A           B
      / \         / \
     /   \       /   \
    /     \     /     \
   C       D   E       F
  / \     / \         / \
 /   \   /   \       /   \
G     H I     J     K     L
             / \
            M   N
```

(b) For any recursive tree and set of labels, there is only one way to assign labels to make the tree a search tree. More precisely, let num : RecTr $\to \mathbb{R}$ be a labelling function on the recursive binary trees, and suppose T is a search tree under this labelling. Suppose that num_{alt} is another labelling and that T is also a search tree under num_{alt} for the *same* set of labels. Prove by structural induction on the definition of search tree that

$$\text{num}(S) = \text{num}_{\text{alt}}(S) \qquad \text{(same)}$$

for all subtrees $S \in \text{Subtrs}(T)$.

Reminder:
Definition. The Search trees $T \in$ BBTr are defined recursively as follows:

Base case: ($T \in$ Leaves). T is a Search tree.

Constructor case: ($T \in$ Branching). If left(T) and right(T) are both Search trees, and

$$\max(\text{left}(T)) < \text{num}(T) < \min(\text{right}(T)),$$

then T is a Search tree.

Problem 7 (**Search Tree, Structural Induction**) (**20 points**). **(a)** Edit the labels in this size 11 tree T so it becomes a search tree for the set of labels [1..11].

```
        T
       / \
      /   \
     A     B
    / \   / \
   /   \ E   F
  C     D
 / \   / \
G   H I   J
```

Reminder:
Definition. The Search trees $T \in$ BBTr are defined recursively as follows:

Base case: ($T \in$ Leaves). T is a Search tree.

Constructor case: ($T \in$ Branching). If left(T) and right(T) are both Search trees, and

$$\max(\text{left}(T)) < \text{num}(T) < \min(\text{right}(T)),$$

then T is a Search tree.

(b) Let T be a search tree whose labels are the integers in the interval $[1..n]$ where $n = \text{size}(T)$. Prove by structural induction on the definition of search tree that the leaves of T are precisely the subtrees whose label is an odd number:

$$\text{leaves}(T) = \{S \in \text{Subtrs}(T) \mid \text{num}(S) \text{ is odd}\}. \qquad \text{(odd-label)}$$

You may use the fact that the size of a search tree is an odd number (see Problem **??**). You may also assume that if T is a search tree under some labelling, then it remains a search tree when the same constant $r \in \mathbb{R}$ is added to each label.

(Continued on next page)

(c) Let T be a search tree of size n with labels $[1..n]$. Deleting the value 1 and inserting $n + 1$ yields the updated set of labels $[2..(n + 1)]$. A search tree for these updated values can be obtained by adding 1 to each of the labels in T. Let U be any search tree that matches this description.[2] Explain why T and U have no leaves in common and therefore have no shared subtree.

Problem 8 (Bipartite Matching) (20 points).
In a variant of the magic card trick decribed the text, an Assistant will have the audience choose four cards, and the Assistant then reveals three of them—in an order he chooses—to the Magician. The Magician then announces what the fourth card will be.

(a) Show that the Magician could not pull off this trick with a deck larger than 27 cards.

(b) Show that, in principle, the Magician could pull off the Card Trick with a deck of exactly 27 cards. (You do not need to describe the actual method.)

[2] More precisely, U is any search tree with labels $[2..(n + 1)]$ that is isomorphic to T.