Exam Time: 1h & 15m. Each problem is worth 10 points. 50 pts total. Choose any 5 problems.

1. **CF Languages.**
   
   1. (8pts) Construct a PDA that accepts \( \{a^n b^m | 0 \leq 2n \leq m \leq 3n \} \);
   
   2. (2pts) Test the PDA on \( x = a^2 b^5 \).

   **Solution.** You need to construct a PDA that, non-deterministically, pushes either two \( a \)s or three \( a \)s whenever it encounters an \( a \) in the input, and pops one \( a \) whenever it encounters a \( b \) in the input.

   ![Diagram of PDA](image)

   The test should be trivial.
2. **CF Languages.** For the language \( L = \{a^i b^j c^k \mid i < j \text{ or } k < j \} \), determine whether it is context-free and prove your answer.

**Solution 1.** This language is Context-Free. You can prove it by either producing a recognizing PDA or a generating grammar. We will set up a grammar. We can explain the construction of the grammar through several steps.

1. Any string with just \( b \) s is in the language: \( S \rightarrow bB; \quad B \rightarrow \varepsilon | bB \).
2. Any string with just \( a \) and \( b \) or \( b \) and \( c \) is in the language: \( S \rightarrow aAbB|bBcC; \quad A \rightarrow \varepsilon | aA; \quad C \rightarrow \varepsilon | cC \).
3. Any string with fewer \( a \) s than \( b \) s (and any number of \( c \) s) or fewer \( c \) s than \( b \) s (and any number of \( a \) s) is in the language: \( S \rightarrow aDbC|AbEc; \quad D \rightarrow aDb|bB; \quad E \rightarrow bEc|bB \).

**Solution 2.** Think of this as the union of two languages: one with an arbitrary number of \( c \) s and fewer \( a \) s than \( b \) s, and the other with an arbitrary number of \( a \) s and fewer \( c \) s than \( b \) s: \( S_1 \rightarrow A_0C; \quad S_2 \rightarrow AC_b; \quad A \rightarrow aA|\varepsilon; \quad C \rightarrow cC|\varepsilon; \quad A_0 \rightarrow aA_0|bB; \quad B \rightarrow bB|bB; \quad C_b \rightarrow bC_b|bB; \quad S \rightarrow S_1|S_2 \).

3. **Turing Machines.**

   a) (4 pts) Define *Deterministic Turing Machine* as given by your textbook.
   b) (6 pts) Construct a DTM that decides \( \{ww \mid w \in \{0, 1\}^*\} \)

**Solution.**

a. See p. 160. In particular, you need to make sure you specify what the transition function \( \delta \) takes as input and produces as output. The ”informal description” earlier in the page does not adequately specify this transition.

b. The string given is just a string over the alphabet, not a string of the form \( BwBwB \). The major problem is to determine where the midpoint is (where the B ”should” be and isn’t). One solution introduces an extended alphabet \( \{0, 1, 0R, 1R, 0L, 1L\} \). We start with \( Bx_1 \ldots x_nB \), where \( n \geq 0 \). A reasonable verbal description (this may be hard in itself) would have been enough; several submissions had ”states and labeled arrows”. Here is a description based on the transition function - all missing entries lead to rejection.

Note that the first several states (to \( q_6 \)) are devoted to marking the first and second halves of the string - and rejecting if the string has odd length. At the end of the process, all the left side characters are marked \( L \), while the right side ones are marked \( R \). The remaining states support checking that the left half and right half match.
4. Turing Machines. (8 pts) Construct a multi-tape DTM to accept the language 
\( \{a^n b^n c^n \mid n \geq 0 \} \). (2 pts) What can you say about the time-complexity of your solution as compared to a one-tape Turing Machine?

Solution.

1. To be somewhat strict, i.e. enforcing the rule that the input tape is read-only; that all computation takes place on the working tapes, and that the output occurs on the output tape, we assume an initial configuration that has the input \( w \in \{a, b, c\}^* \) in tape 1:

\[
\begin{array}{ccccccc}
B & w_1 & w_2 & w_3 & \ldots & w_{k-1} & w_k & B \\
\end{array}
\]

where \( w_j \in \{a, b, c\} \) for \( j = 0, \ldots, k \). To simplify matters a little, we assume doubly infinite tapes (half-infinite ones would require moving to the left end of the input and then copy); we also indicate the characters written and the head movements as strings, rather than vectors, to save space. The first set of actions involves copying, and we care only about what is under the head that tracks the input tape:

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>B</th>
<th>( 0 )</th>
<th>( 1 )</th>
<th>( 0^R )</th>
<th>( 1^R )</th>
<th>( 0^L )</th>
<th>( 1^L )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s )</td>
<td>( (q_1, B, L) )</td>
<td>( (q_2, 0^R, L) )</td>
<td>( (q_2, 1^R, L) )</td>
<td>( (q_6, 0^L, R) )</td>
<td>( (q_6, 1^L, R) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( (h, B, R) )</td>
<td>( (q_3, 0, L) )</td>
<td>( (q_3, 1, L) )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( (q_4, B, R) )</td>
<td>( (q_3, 0, L) )</td>
<td>( (q_3, 1, L) )</td>
<td>( (q_4, 0^L, R) )</td>
<td>( (q_4, 1^L, R) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( (q_5, 0^L, R) )</td>
<td>( (q_5, 1^L, R) )</td>
<td>( (q_1, 0^R, L) )</td>
<td>( (q_1, 1^R, L) )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( q_4 )</td>
<td>( (q_7, B, L) )</td>
<td>( (q_8, 0, L) )</td>
<td>( (q_8, 1, L) )</td>
<td>( (q_9, 0^R, L) )</td>
<td>( (q_9, 1^R, L) )</td>
<td>( (q_{10}, 0, R) )</td>
<td>( (q_{10}, 1, R) )</td>
</tr>
<tr>
<td>( q_5 )</td>
<td>( (q_{10}, 0, R) )</td>
<td>( (q_{10}, 1, R) )</td>
<td>( (q_{11}, 0^R, R) )</td>
<td>( (q_{11}, 1^R, R) )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( q_6 )</td>
<td>( (q_{11}, 0^R, R) )</td>
<td>( (q_{11}, 1^R, R) )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

From this point on, all the characters under heads 2, 3 and 4 **must** be \( a, b \) and \( c \), or \( B, B, B \). Anything else leads to rejection: \( (h, \_\_\_0, SSSSR) \).
where the underline denotes whatever character was under the head on tape 1.

2. The complexity of this algorithm is clearly linear ($3n$ steps to copy, $n$ steps to check, 1 step to write 0 or 1 on tape 4): $O(n)$. Note that all blanks on the tables (if they represent possible configurations) can be replaced by $\delta(q_t, _____) \to (h, ____0, SSSSR)$. For the complexity of a single tape machine, we have to do a little more work. Here is an algorithm (no table, though):

   a. Replace each $c$ (if you find any at the right end of the input) by $c$ ($n$ steps - length of the whole string, at most). Also return to rightmost blank upon finding the rightmost $b$. If there are no $c$s, then a blank will lead to accept; anything else to failure. If there are no $b$s before $a$s, fail.

   b. Replace the first $c$ by a $c$, find the rightmost $b$ and replace it by $b$. Repeat this until you have exhausted the $c$s. If you find anything other than a $b$ (or $b$) while doing this, fail. After you have turned all rightmost $c$s back to $c$s: this part of the algorithm is $O(n^2) - n$ passes over strings of $n$ characters.

   c. Move to the rightmost $b$, change it to a $b$; move left through the $b$s until you find an $a$. If you find anything other than an $a$, fail. Keep going back and forth until all the $b$s are changed back to $b$s, and the corresponding $c$s have been changed to $c$s.

   d. Left of the leftmost $a$ you must find $B$; fail otherwise. The second pass is also $O(n^2)$. The last pass - checking that there is nothing left over - is $O(n)$. Total: $O(n^2)+O(n^2)+O(n) = O(n^3)$. I’ll grant that the class $O(n^2)$ is contained in $O(n^3)$, so that every $O(n^2)$ function is automatically $O(n^3)$, but I was expecting a least upper bound, more or less.

5. **Primitive Recursive Functions.** Define a function $\text{floor}(m, n)$, which matches the ”usual” $\text{floor}(\frac{m}{n})$ function, and show that it is primitive recursive.

   **Hint:** recall that $\text{floor}(\frac{m}{n})$ returns the largest integer $i$ such that $i \leq m/n$ or the smallest integer $i$ such that $(i + 1) \cdot n > m$. Furthermore, you may want to recall that we have not defined *Rational Numbers*. You will have to remember to quote some results that you will need.

   **Solution.** The function
   
   $$f(m, n) \equiv \min_{i \leq m}[(i + 1) \cdot n > m]$$

   is ”clearly” primitive recursive:
   
   1. sum is p.r.; product is p.r.; > is p.r.; bounded minimization is p.r.

   2. The only problem is that, if $n = 0$, the min does not exist, since the inequality cannot be satisfied for any value of $i$, let alone values of $i$ bounded above by $m$. So we have to decide what to do about it. We can fix the problem with:

   $$\text{if } ((\exists i)_{i \leq m}[(i + 1) \cdot n > m]) \text{ then } (\min_{i \leq m}[(i + 1) \cdot n > m]) \text{ else } 0.$$  

   Also: bounded existential quantification is p.r., if-then-else is p.r. and the zero function is p.r..

   On the other hand, this opens us up to the criticism that we have redefined floor.
6. **R.E. and R. sets.**

1 (5 pts) State the Projection Theorem.

2 (5 pts) Use the Projection theorem to prove that the union of two recursively enumerable sets is recursively enumerable.

**Solution.** See p. 233 for the Projection Theorem and p.235 for the union.

7. **Reducibility.**

1 (3 pts) Define *many-to-one reducibility*.

2 (7 pts) Assume that \( A \cup B = \{0,1\}^* \) and \( A \cap B \neq \emptyset \). Show that if \( A \) and \( B \) are recursively enumerable, then \( A \leq_m A \cap B \).

**Solution.** See p. 246 for the definition.

2. Let \( x_0 \in A \cap B \). That \( x_0 \) exists follows from the hypothesis that \( A \cap B \neq \emptyset \); that you can construct it follows from the fact that \( A \cup B = \{0,1\}^* \) and both \( A \) and \( B \) are r.e. You can enumerate the strings of \( \{0,1\}^* \) and simulate both TMs for one instruction on the first string, two instructions on the first and second; three on the first three strings, etc. Eventually you will find a string for which both TMs terminate within the number of instructions you are using. For each \( x \in A \cup B \) at least one of the TMs will accept. Depending on which one terminates first, define \( f \) as: \( x \in A \rightarrow f(x) = x_0 \), \( x \in B \rightarrow f(x) = x \). You need to show that this function meets all the conditions (recursive and ....).