1) Rewrite $F(a, b, c, d) = \sum (0, 3, 4, 8, 10, 11)$ in fully simplified product-of-sums form.

2) Consider the following page reference string for a virtual memory system in which physical memory has exactly 3 frames:

\begin{verbatim}
2, 4, 6, 4, 2, 7, 4, 3, 7, 2, 3
\end{verbatim}

For each of the following page replacement algorithms, show which references will cause page faults and show the contents of the 3 frames at the time of each fault. Assume that the frames are initially empty. You do not need to show the first 3 faults that are caused by demand paging.

a) Least Recently Used
b) Second Chance

3) There are 3 standard goals to the 2-process mutual exclusion problem.
Goal 1: Mutual exclusion is guaranteed
Goal 2: Deadlock cannot occur.
Goal 3: Indefinite postponement cannot occur.

Attempted Solution: common variable: lock (initially false)
Assume the existence of an atomic (non-interruptible) test_and_set function that both returns the value of its boolean argument and sets the argument to true.

\begin{verbatim}
Process 1
while (true) {
    while (test_and_set(lock));
    Critical section;
    lock = false;
    Noncritical section;
}

Process 2
while (true) {
    while (test_and_set(lock));
    Critical section;
    lock = false;
    Noncritical section;
}
\end{verbatim}

For the above solution,
a) Select one goal that is not satisfied and provide an execution sequence that violates the goal.
b) Select one goal that is satisfied and give a brief explanation that justifies why the goal is met for all possible execution sequences.
Choose any 2 of the 3 problems.

1) Given a possibly empty binary tree, write a function that returns the number of nodes in the tree that have a right child, but no left child. The prototype for your function is
   
   int RightNoLeft(TreeNode *ptr).

   Global variables may not be used. No additional functions may be defined. Declare all data structures.

2) Given an array of n nonzero real numbers a[0]…a[n-1], write a function to partition the array (not sort) so that all its negative elements come before all its positive elements. Your algorithm should have O(n) time complexity. The function prototype is
   
   void negpospartition(float a[], int n).

3) Count the precise number of "fundamental/basic operations" executed in the following code. Your answer should be a function of \( n \geq 1 \) in closed form. Note that “closed form” means that you must resolve all \( \Sigma \)'s and \( \cdots \)'s. An asymptotic answer (such as one that uses big-oh, big-theta, etc.) is not acceptable.

   ```c
   for(int k = 1; k < n; k++) {
       Perform 1 fundamental/basic operation;
       for (int j = k; j <= n; j++)
           Perform 1 fundamental/basic operation;
   //endfor j
};//endfor k
   ```
1. Give regular expressions describing each of the following languages over \( \Sigma = \{0, 1\} \):

   a. \( \{ w : w \text{ contains the substring 101} \} \)
   b. \( \{ w : w \text{ contains at least three 0's} \} \)
   c. \( \{ w : w \text{ contains at most three 1's} \} \)
   d. \( \{ w : |w| \geq 3 \} \)
   e. \( \{ w : |w| \leq 3 \} \)
   f. \( \{ w : w \text{ starts and ends with different symbols} \} \)
   g. \( \{ w : \text{every odd position of } w \text{ is 0} \} \)
   h. \( \{ w : w \text{ does not contain exactly two 1's} \} \)
   i. \( \{ w : \text{every 0 in } w \text{ is followed by two 1's} \} \)
   j. \( \{ w : w \text{ starts with the substring 011 and contains the substring 110} \} \)

   Each question is worth 2 points. No partial credit will be given.

2. Give the state diagram for a Turing machine that decides the following language over \( \Sigma = \{0, 1\} \):

   \( L = \{ w : w \text{ contains an even number of occurrences of the substring 011 and } |w| \text{ is odd} \} \)

   Use the following notation to label each of your machine’s transitions:

   \[
   \begin{array}{c}
   \text{(read } \alpha, \text{ write } \beta, \text{ move left)}
   \\
   q_i \xrightarrow{\alpha \beta, L} q_j
   \\
   \text{(read } \alpha, \text{ write } \beta, \text{ move right)}
   \\
   q_i \xrightarrow{\alpha \beta, R} q_j
   \end{array}
   \]

3. A coloring of a graph is an assignment of colors to its nodes so that no two adjacent vertices have the same colors.

   Let \( \text{COLOR} = \{ G, k : G = (V, E) \text{ is a graph that can be colored by } k \text{ colors} \} \).

   Provide a polynomial verifier to prove that \( \text{COLOR} \in \text{NP} \).