(1) Discuss the relevance of studying algorithm analysis in the context of computer science. Describe how algorithm analysis can improve software quality. (10 pts.)
(2) a. Mention two of the six computational problems as classified by problem requirements. Provide simple examples of the two you supplied. (5 pts.)

b. Explain the difference between an analytically hard problem and a computationally hard problem. Cite examples if you can. (5 pts.)
(3) a. Define what is probably meant by an algorithm designer when s/he says "the algorithm is $O(n)$." (5 pts.)

b. Show that $f(n)=1000000n + 50000$ is $O(n)$. (10 pts.)
Consider the following recursive Ada function to reverse a string:

```ada
function REVERSE_OF( THE_STRING : STRING ) return STRING is
begin
  if THE_STRING'LENGTH <= 1 then
    return THE_STRING;
  else
    return THE_STRING( THE_STRING'LAST ) &
    REVERSE_OF( THE_STRING ( THE_STRING'FIRST..THE_STRING'LAST - 1 ) )
  end if;
end REVERSE_OF;
```

[Essentially, the algorithm implemented by the function above works this way: if the string has length less than or equal to one, then it is returned unchanged; if the length is greater than one, the answer is achieved by concatenating the last character in the string with the result of reversing the substring consisting of the first through the second to the last characters in the string.]

a. Explain why the following recurrence equation expresses the number of comparisons (\(<=')s\) and concatenations (\('&')s\) in `REVERSE_OF` as a function of the string length \(n\) (\(i.e., n=THE\_STRING'LENGTH\)) (5 pts.)

\[ C(n) = C(n-1) + \Theta(1) \]

b. Solve the recurrence given in (a) above. (10 pts.)
(5) The following Ada function implements binary search as given in the textbook on p. 105. We assume that the list type, the item type, and all other type declarations have been made elsewhere.

```adala
function BINARY_SEARCH ( LIST : LIST_TYPE ;
    LOWER , UPPER : INTEGER_TYPE ;
    X : ITEM_TYPE ) return INTEGER is
begin
    -- look for X in LIST( LOWER .. UPPER );
    -- report its position if found, else report 0
    MID : INTEGER := ( LOWER + UPPER ) / 2 ;
    if LOWER = UPPER then
        if X = LIST( LOWER ) then
            return LOWER;
        else
            return 0;
        end if;
    else
        MID := ( LOWER + UPPER ) / 2 ;
        if X > LIST( MID ) then
            return BINARY_SEARCH ( LIST , MID+1 , UPPER , X ) ;
        else
            return BINARY_SEARCH ( LIST , LOWER , MID-1 , X ) ;
        end if;
    end if;
end BINARY_SEARCH ;
```

a. Trace the sequence of elements in LIST (as given below) that are compared to X by BINARY_SEARCH if X=31. (5 pts.)

<table>
<thead>
<tr>
<th>LIST</th>
<th>3</th>
<th>9</th>
<th>13</th>
<th>14</th>
<th>17</th>
<th>19</th>
<th>22</th>
<th>28</th>
<th>29</th>
<th>30</th>
<th>31</th>
<th>33</th>
<th>35</th>
<th>37</th>
<th>48</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

b. If there are n items in LIST, how many of these items represent the worst case performance of BINARY_SEARCH? That is, if X were equal to any of these items, BINARY_SEARCH would require the greatest number of comparisons. [Hint: Recall that binary search has a worst case performance of approximately \( \log n \).] (10 pts.)
(6) Consider again the Towers of Hanoi problem. You were asked in Exercise Set 2 to compute the average number of moves necessary to solve the problem starting from an arbitrary legal arrangement if every legal arrangement of disks distributed on the three pegs is equally likely. Assume that C is the target peg and A the source peg.

a. Explain why the recurrence below expresses \( L(n) \), the total number of possible legal arrangements of \( n \) disks: (5 pts.)

\[
L(n) = 3 \quad \text{if } n=1 \\
L(n) = 3L(n-1) \quad \text{if } n>1
\]

b. Show that \( L(n)=3^n \) solves the recurrence in (a). (5 pts.)
c. To compute $A(n)$, the average number of moves necessary to solve the Towers of Hanoi problem starting from an arbitrary legal arrangement with $n$ disks, we will set up a weighted recurrence involving $A(n-1)$, the average number of moves to solve the problem with $n-1$ disks. We know that $A(1)=2/3$ (since 2 out of 3 times, the single disk may not be in the target peg C). With equal probability, the largest disk could be found in either peg A or B (non-target pegs). So in two out of three cases, we need to move this largest disk from where it is (either A or B) to where it should be (on C). Before we could do this though, we need to move the $n-1$ smaller disks to the peg where the largest disk is not on. Say the largest disk is on peg A. Then we could move the $n-1$ smaller disks onto peg B. After this, we could move the largest disk to peg C and then solve the Towers of Hanoi problem with the $n-1$ disks now on peg B (i.e., move them from there to peg C). On the average, moving the $n-1$ disks would take $A(n-1)$ moves; moving the largest disk (if it is necessary) consumes another; and then, finally, moving the $n-1$ disks from where they are onto peg C involves $2^{n-1}$ moves.

With the above description, explain the recurrence for $A(n)$ below:

$$A(n) = \begin{cases} 
2/3 & \text{if } n=1 \\
(2/3)[A(n-1)+1+2^{n-1}] + (1/3)A(n-1) & \text{if } n>1 
\end{cases}$$
d. Verify that $A(n) = (2/3)(2^n - 1) + (2/3)n$ solves the recurrence in (c). (10 pts.)