CS 260 Review 1

1. Write and solve recurrence equations for the time complexity of the following algorithm.

```
function F ( n: integer ) : integer;
        begin
                if n <= 1 then
                        return (2)
                else
                        return (F(n-1) * F(n-1))
        end; { F }
```

   What does function *F* compute? Can you modify it in such a way that it computes the result just with n calls?

2. Two recursive algorithms solving the same computational task are given. Time performance of these algorithms is describes by recurrence equations

   $T(1) = 1$
   $T(n) = 4T(n/2) + c_1 n,$

   $R(1) = 1$
   $R(n) = 3R(n/2) + c_2 n.$

   Which algorithm has a better asymptotic performance? Justify your answer.

3. Write down a detailed pseudo-code implementing the procedure of deleting all positions of an input list, which come before the first occurrence of a given element x. The first occurrence of x should be kept in the list. In your implementation you are only allowed to use the basic operations of the List ADT, i.e. FIRST(L), END(L), RETRIEVE(p,L), LOCATE(x,L), NEXT(p,L), PREVIOUS(p,L), INSERT(x,p,L), DELETE(p,L), MAKENULL(L).

4. What is the computational complexity of the operation PRINTLIST expressed in terms of the size of the input? Assume that the input list is represented as a singly-linked list and that operations FIRST, NEXT and RETRIEVE take constant time. Your estimate should be tight, i.e. the time spent on the execution of the operation PRINTLIST should be both "big oh" and "big omega" of your function. Provide a justification of your answer.

```
procedure PRINTLIST( L: LIST );
        { PRINTLIST prints the elements of L in the order of occurrence }
        var
                p: position;
```

```
        begin
                p:=FIRST(L);
                while p<>END(L) do begin
                        print(RETRIEVE(p,L));
                        p:=NEXT(p,L)
                end
        end; { PRINTLIST }

function END ( L: LIST ): position;
        var
                q: position;
        begin
                q:=L;
                while q↑.next <> nil do
                        q:= q↑.next;
                return (q)
        end; { END }
```

5. Find a formula for the number of calls occurring during the execution of the recursive function *F*(n) provided below.

```
function F( n: integer) : integer;
        begin
                if n <= 1 then
                        return (1)
                else
                        return (F(n-2)+F(n-1))
        end; { F }
```

   What is the rate of growth of this number with respect to n?.

6. Remove recursion from the following program:

```
function comb ( n, m: integer ): integer;
        begin
                if ( n = 1 ) or ( m = 0 ) or ( m = n ) then
                        return (1)
                else
                        return (comb(n-1,m) + comb(n-1,m-1))
        end; { comb }
```

7. Suppose that we have lists containing pre-order and post-order listings of the nodes a tree. Describe an algorithm, which operates on these lists and marks leaves with letter L and internal nodes with letter I. Your algorithm may not access the tree itself, only the lists containing the nodes.