Prolog assignment 4 Shreyas Marwah 2310110604

Answer 1

```
% move(+N, +Source, +Target, +Auxiliary)
% Moves N disks from Source to Target using Auxiliary as helper.
move(0, _, _, _) :-
    !. % Base case: no disks to move, do nothing.
move(N, Source, Target, Auxiliary) :-
    N > 0,
    M is N - 1,
    % Move top N-1 disks from Source to Auxiliary
    move(M, Source, Auxiliary, Target),
    % Move the largest disk from Source to Target
    write('Move disk from '), write(Source), write(' to '),
write(Target), nl,
    % Move the N-1 disks from Auxiliary to Target
    move(M, Auxiliary, Target, Source).
```

The recursive algorithm of tower of Hanoi solves the problem by:

1. Moving n−1disks from x to z using y (recursion).
2. Moving the largest disk directly from x to y (constant time).
3. Moving n−1disks from z to y using x (recursion)

Thus, the number of moves for n disks is T(n):

T(n) = 2 * T(n-1) + 1

When we keep expanding the T(n-k) term, we will get a series of 2s in multiplication, i.e. 2^n. This means that the time complexity is: **O(2^n)**

Answer 2

(A) Multiplication as repeadted addition

```
% multiply(+X, +Y, -Z)
% X * Y = Z, where X, Y, and Z are natural numbers
multiply(0, _, 0).  % Base case: 0 *x = 0.
multiply(s(X), Y, Z) :-
    multiply(X, Y, Temp),
    plus(Y, Temp, Z).
```

(B) Exponentiation using repeated multiplication

```
% assuming that the above implementation of multiplication is
already defined:
% power(+Base, +Exponent, -Result) % Base^Exponent = Result, where
Base, Exponent, and Result are natural numbers power(_, 0, s(0)). %
Base case: Any number raised to 0 is 1 (s(0)). power(Base,
s(Exponent), Result) :- power(Base, Exponent, Temp), % Recursively
```

calculate Base^(Exponent-1) multiply(Base, Temp, Result). % Multiply Base with the result.

Answer 3

Part (A)

```
% A binary tree is either an empty tree or a tree with a root and
two subtrees.
binary_tree(nil).  % Base case: an empty tree.
binary_tree(tree(_, Left, Right)) :-
    binary_tree(Left),  % left subtree.
    binary_tree(Right). % right subtree.
```

Part (B)

Prerorder

```
% preorder(+Tree, -List)
% Traverses the binary tree in preorder and produces a list of
elements.
preorder(nil, []).  % Base case: empty tree results in an empty
list.
preorder(tree(Element, Left, Right), [Element|List]) :-
    preorder(Left, LeftList),     % Recursively traverse the left
subtree.
    preorder(Right, RightList),   % Recursively traverse the right
subtree.
    append(LeftList, RightList, List).  % Combine results.
```

In-order

```
% inorder(+Tree, -List)
% Traverses the binary tree in in-order and produces a list of
elements.
inorder(nil, []).  % Base case: empty tree results in an empty list.
inorder(tree(Element, Left, Right), List) :-
    inorder(Left, LeftList),      % Recursively traverse the left
subtree.
    inorder(Right, RightList),    % Recursively traverse the right
subtree.
    append(LeftList, [Element|RightList], List).  % Combine results.
```

POSTodrer

```
% postorder(+Tree, -List)
```

Prolog assignment 4 Shreyas Marwah 2310110604

```
% Traverses the binary tree in post-order and produces a list of
elements.
postorder(nil, []).  % Base case: empty tree results in an empty
list.
postorder(tree(Element, Left, Right), List) :-
    postorder(Left, LeftList),     % Recursively traverse the left
subtree.
    postorder(Right, RightList),   % Recursively traverse the right
subtree.
    append(LeftList, RightList, Temp),  % Combine left and right
results.
    append(Temp, [Element], List).      % Add the current node's
element.
```