1. Count the number of lists in a deep list.

E.g: *?- count_lists([[1,5,2,4],[1,[4,2],[5]],[4,[7]],8,[11]],R).*
R = 8.

2. Double the odd numbers and square the even.

E.g: *?- numbers([2,5,3,1,1,5,4,2,6],R).*
R = [4,10,6,2,2,10,16,4,36].

3. Convert a number to binary (the powers of 2 grow from right to left).

E.g: *?- to_binary(5,R1),to_binary(8,R2),to_binary(11,R3).*
R1 = [1,0,1], R2 = [1,0,0,0], R3 = [1,0,1,1].

4. Replace all the occurrences of x in a difference list with the sequence y x y.

E.g: *?- replace_all(2,[1,2,3,4,2,1,2,2,3],[2,3],8,R).*
*R = [1,**8,2,8**,3,4,**8,2,8**,1,**8,2,8**].*

5. Delete the occurrences of x on even positions (the position numbering starts with 1).

E.g: *?- delete_pos_even([1,2,3,4,2,3,3,2,5],2,R).*
R = [1,3,4,2,3,3,5].

6. Compute the divisors of a natural number.

E.g: *?- divisor(15,R1), divisor(2,R2), divisor(1,R3), divisor(0,R4),divisor(-6,R5).*
R1 = [1,3,5,15], R2 = [1,2], R3 = [1], R4 = alot, R5 = [1,2,3,6] (same for 6).

7. Reverse a natural number.

E.g: *?- reverse(15,R1), reverse(121235124,R2).*
R1 = 51, R2 = 421542121.

8. Delete each k-th element from the end of the list.

E.g: ?- *delete_kth_end([1,2,3,4,5,6,7,8,9,10],3,R)*
R = [1,3,4,6,7,9,10].

9. Separate the even elements on odd positions from the rest (the position numbering starts at 1).

E.g: *?- separate([1,2,2,3,4,5,6,6,12,44,8,5,5,10,5],Even,Rest).*
Even = [2,4,6,12,8], Rest = [1,2,3,6,44,5,5,10,5].

10. Binary incomplete tree. Collect odd nodes with 1 child in an incomplete list.

E.g: *tree(t(26,t(14,t(2,_,_),t(15,_,_)),t(50,t(35,t(29,_,_),_),t(51,_,t(58,_,_)))))*.
?- tree(X), collect_odd_from_1child(X,R).
R = [35, 51|_].

11. Ternary incomplete tree. Collect the keys between X and Y (closed interval) in a difference list.

E.g: *tree(t(2,t(8,_,_,_),t(3,_,_,t(4,_,_,_)),t(5,t(7,_,_,_),t(6,_,_,_),t(1,_,_,t(9,_,_,_)))))*.
?- tree(T), collect_between(T,2,7,R,[18]).
R = [2,3,4,5,6,7,1,18].

12. Binary Tree. Collect even keys from leaves in a difference list.

E.g: *tree(t(5,t(10,t(7,nil,nil),t(10,t(4,nil,nil),t(3,nil,t(2,nil,nil)))),t(16,nil,nil)))*.
?- tree(T), collect_even_from_leaf(T,R.[1]).
R = [4,2,16,1].

13. Replace the min element from a ternary incomplete tree with the root.

E.g: *tree(t(2,t(8,_,_,_),t(3,_,_,t(1,_,_,_)),t(5,t(7,_,_,_),t(6,_,_,_),t(1,_,_,t(9,_,_,_)))))*.
?- tree(T), replace_min(T,R).
R = t(2,t(8,_,_,_),t(3,_,_,t(2,_,_,_)),t(5,t(7,_,_,_),t(6,_,_,_),t(2,_,_,t(9,_,_,_))))

14. Collect all the nodes at odd depth from a binary incomplete tree (the root has depth 0).

E.g: *tree(t(26,t(14,t(2,_,_),t(15,_,_)),t(50,t(35,t(29,_,_),_),t(51,_,t(58,_,_))))).*
?- tree(X), collect_all_odd_depth(X,R).
R = [14,50,29,58].

15. Flatten only the elements at depth X from a deep list.

E.g: *?- flatten_only_depth([[1,5,2,4],[1,[**4,2**],[**5**,[6,7,8]],[4,[7]],8,[11]],3,R).*
R = [4,2,5,7].

16. Delete duplicate elements that are on an odd position in a list (the position numbering starts at 1).

E.g: *?- remove_dup_on_odd_pos([**1**,2,**3**,1,**3**,3,**3**,9,**10**,6,**10**,8,7,3],R).*
R = [2,1,3,9,6,8,7,3].

17. Determine the node/s having the median value in a ternary incomplete tree.

E.g: *tree(t(2,t(8,_,_,_),t(3,_,_,t(1,_,_,_)),t(5,t(7,_,_,_),t(5,_,_,_),t(1,_,_,t(9,_,_,_))))).*
?- tree(T), median(T,R).
R = [t(5,t(7,_,_,_),t(5,_,_,_),t(1,_,_,t(9,_,_,_))), t(5,_,_,_)].

18. Replace each node with its height in a binary incomplete tree (a leaf has height 0).

E.g: *tree(t(2,t(4,t(5,_,_),t(7,_,_)),t(3,t(0,t(4,_,_),_),t(8,_,t(5,_,_))))).*
?- tree(T), height_each(T,R).
R = tree(t(3,t(1,t(0,_,_),t(0,_,_)),t(2,t(1,t(0,_,_),_),t(1,_,t(0,_,_))))).

19. Replace each constant depth sequence in a deep list with its length.

E.g: *?- len_con_depth([[1,2,3],[2],[2,[2,3,1],5],3,1],R).*
R = [[3],[1],[1,[3],1],2].

20. Decode a list encoded with RLE.

E.g: *?- rle_decode([[a,4], [b,1] ,[c,2], [a,2], [d,1] , [e,4]],R).*
R = [a,a,a,a,b,c,c,a,a,d,e,e,e,e].

21. Encode a list with RLE.

E.g: *?- rle_encode([a,a,a,a,b,c,c,a,a,d,e,e,e,e], R).*
R = [[a,4], [b,1] ,[c,2], [a,2], [d,1] , [e,4]].

22. Compute the indegree and the outdegree for each node in a graph using the dynamic predicate info(Node, OutDegree, InDegree).

E.g: *edge(1,2). edge(2,1). edge(1,4). edge(1,3). edge(3,2).*
=> info(1,3,1). info(2,1,2). info(3,1,1). info(4,0,1).

Toolbox:
- arithmetic operations
- list operations (complete,deep,difference, incomplete)
- 1 sorting algorithm
- tree operations (complete, incomplete)
- graphs and side effects