```
use_module(libary(clpfd)).
transpose(Ms, Ts) :-
        (   Ms = [] -> Ts = []
        ;   Ms = [F|_],
            transpose(F, Ms, Ts)
        ).

transpose([], _, []).
transpose([_|Rs], Ms, [Ts|Tss]) :-
        lists_firsts_rests(Ms, Ts, Ms1),
        transpose(Rs, Ms1, Tss).

lists_firsts_rests([], [], []).
lists_firsts_rests([[F|Os]|Rest], [F|Fs], [Os|Oss]) :-
        lists_firsts_rests(Rest, Fs, Oss).

/* The above is not written by me, I take no credit for it,
it is from the SWI-Prolog implementation:
https://github.com/lamby/pkg-swi-prolog/blob/master/library/clp/clpfd.pl */



%finds the row which all valid must be permutation of
base_row(0,[]).
base_row(X,[H|T]):-
    H #=# X,
    X1 #= X-1,
    base_row(X1, T),!.


row(X,R,Master):-
    length(R,X),
    fd_domain(R,Master),
    fd_all_different(R).

by_col(_,[],_).
by_col(N,[H|Sol],Master):-
    row(N,H,Master),
    by_col(N,Sol,Master).

by_row(N,Sol,Master):-
    length(Sol,N),
    by_col(N,Sol,Master).


label([]).
label([H|Sol]):-
    fd_labeling(H),
    label(Sol).

tower(N,T,C):-
    C = counts(Top,Bottom,Left,Right),
    base_row(N,Master),
    !,
    by_row(N,T,Master),
    transpose(T,SolTrans),
    by_row(N,SolTrans,Master),
    label(T),
    naive_view(SolTrans,[Top,Bottom]),
    naive_view(T,[Left,Right]),
    label([Top,Bottom,Left,Right]).
```

```
naive_view([],[[],[]]).
naive_view([H|Sol], [[Side1|Side1Tail],[Side2|Side2Tail]]):-
    append([], [Head1|Tail1],H),
    can_see(Side1,Head1, Tail1),
    reverse(H,RevH),
    append([], [Head2|Tail2],RevH),
    can_see(Side2,Head2,Tail2),
    naive_view(Sol,[Side1Tail,Side2Tail]).


can_see(1,_,[]).
can_see(X,Cur_high, [H|Tail]):-
    H #> Cur_high,
    X #=# X1+1,
    can_see(X1, H, Tail).
can_see(X, Cur_high, [H|Tail]):-
    H #=< Cur_high,
    can_see(X, Cur_high, Tail).



%---------- plain tower --------
plain_base_row(_,[]).
plain_base_row(Z,[H|T]):-
    H is Z,
    Z1 is Z + 1,
    plain_base_row(Z1, T),!.


plain_row([], _).
plain_row([H|Tail],Master):-
    select(H,Master,New),
    plain_row(Tail, New).


plain_col_valid([], _).
plain_col_valid([H|TranSol], Master):-
    plain_row(H,Master),
    plain_col_valid(TranSol,Master).


plain_by_col(_, _, [],_).
plain_by_col(N, Beg, [H|Sol], Master):-
    length(H,N),
    plain_row(H,Master),
    append(Beg,[H],Curr),
    transpose(Curr,CurrCol),
    plain_col_valid(CurrCol,Master),
    plain_by_col(N, Curr,Sol,Master).

plain_by_row(N, Sol,Master):-
    length(Sol,N),
    !,
    plain_by_col(N, 0, Sol,Master).

plain_tower(N,T,C):-
    C = counts(Top,Bottom,Left,Right),
    length(Master,N),
    plain_base_row(1,Master),!,
    length(T,N),
    !,
    plain_by_col(N, [], T,Master),
```

```
    plain_view_row(T,Left,Right),
    transpose(T,TranSol),
    plain_view_row(TranSol,Top,Bottom).



plain_view_row([],[],[]).
plain_view_row([H|Rest],[Left|LeftTail],[Right|RightTail]):-
    plain_count(0, True1, H),
    sum_list(True1,Left),
    reverse(H,RevH),
    plain_count(0,True2,RevH),
    sum_list(True2,Right),
    plain_view_row(Rest,LeftTail,RightTail).

plain_count(_,[],[]).
plain_count(CurrMax, [True|OtherTrue], [H|Rest]):-
    CurrMax < H,
    True is 1,
    plain_count(H, OtherTrue, Rest).
plain_count(CurrMax, [True|OtherTrue], [H|Rest]):-
    CurrMax >= H,
    True is 0,
    plain_count(CurrMax, OtherTrue,Rest).


%-------- speedup --------

speedup(R):-
    statistics(cpu_time,[_,_]),
    findall(0,plain_tower(4,T,C),_),
    statistics(cpu_time,[_,Plain]),
    findall(0,tower(4,T,C),_),
    statistics(cpu_time,[_,Finite]),
    R is Plain/Finite.


%-------- ambiguous

ambiguous(N, C, T1, T2):-
    tower(N,T1,C),
    tower(N,T2,C),
    T1\=T2.
```