```
% permutation helper function
perm([], []).
perm(List, [First|Perm]) :- select(First, List, Rest), perm(Rest, Perm).

% build range 1 - N helper function
range(1, [1]).
range(N, [N|L]) :- N > 1, X is N - 1, range(X, L).

% https://stackoverflow.com/questions/4280986/how-to-transpose-a-matrix-in-prolog
transpose([], []).
transpose([F|Fs], Ts) :-
    transpose(F, [F|Fs], 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).

% rowchecking - 1 through N should appear exactly once
rowcheck(N, X) :- length(X, N), range(N, A), perm(A, X).
allrows([], _).
allrows([X|L], N) :- rowcheck(N, X), allrows(L, N).

% column checking
allcols(T, N) :- transpose(T, A), allrows(A, N).

% count the number of towers seen from a row
% first|tail, count

towerhelper([], _, Z, Z).
towerhelper([X|L], Y, Z, C) :- (X > Y -> A = X, B is Z + 1; A = Y, B = Z),
towerhelper(L, A, B, C).

towercount([X|L], C) :- towerhelper(L, X, 1, C).

% to properly towercount a row: rowcheck(N, X), towercount(X, C).

% helper function to make ensure the visible tower count for each row matches the
row
% list of view count, tower rows, length
dircheck([], [], _).
dircheck([C|L], [X|Y], N) :- towercount(X, C), rowcheck(N, X), dircheck(L, Y, N).

% helper function to reverse the rows of board
revboard(X, Y) :- maplist(reverse, X, Y).
```

```
% N > 0
ncheck(N) :- N > 0.

% check rows and check columns for desired structure
tcheck(T, N) :- length(T, N), allrows(T, N), allcols(T, N).

% compare the rows to their corresponding counts, reverse and transpose where needed
% direction, board, N
ccheck(C, X, N) :- length(C, N), tcheck(X, N), dircheck(C, X, N).

% ntower/d will use fd predicates, the other predicates will be used for
plain_ntower/3

fd_rowcheck(N, X) :- length(X, N), fd_domain(X, 1, N), fd_all_different(X),
    fd_labeling(X, [variable_method(first_fail),
variable_method(most_constrained)]).
fd_allrows(T, N) :- maplist(fd_rowcheck(N), T).
fd_allcols(T, N) :- transpose(T, A), fd_allrows(A, N).

fd_tcheck(T, N) :- length(T, N), fd_allrows(T, N), fd_allcols(T, N).

mini_dircheck(N, C, X) :- fd_rowcheck(N, X), towercount(X, C).
fd_dircheck(C, X, N) :- maplist(mini_dircheck(N), C, X).

fd_ccheck(C, X, N) :- length(C, N), fd_tcheck(X, N), fd_dircheck(C, X, N).

ntower(0, [], counts([], [], [], [])).

ntower(N, T, counts(Top, Bot, Left, Right)) :- fd_tcheck(T, N), ncheck(N),
    fd_ccheck(Left, T, N), revboard(T, A),
    fd_ccheck(Right, A, N), transpose(T, B),
    fd_ccheck(Top, B, N), revboard(B, D),
    fd_ccheck(Bot, D, N).

% base case
plain_ntower(0, [], counts([], [], [], [])).

plain_ntower(N, T, counts(Top, Bot, Left, Right)) :- tcheck(T, N), ncheck(N),
    ccheck(Left, T, N), revboard(T, A),
    ccheck(Right, A, N), transpose(T, B),
    ccheck(Top, B, N), revboard(B, D),
    ccheck(Bot, D, N).

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

speedup(Ratio) :-
    N = 4,
    C = counts([2,3,1,2],[2,2,3,1],[3,1,2,2],[2,3,3,1]),

    statistics(cpu_time, [Start1|_]),
    ntower(N, _, C),
    statistics(cpu_time, [End1|_]),
    Time1 is End1 - Start1,

    statistics(cpu_time, [Start2|_]),
    plain_ntower(N, _, C),
    statistics(cpu_time, [End2|_]),
    Time2 is End2 - Start2,

    Ratio is Time2 / Time1.
```