```
% Auxiliary to reorder arguments to length for maplist
length_(N, L) :- length(L, N).

% Apply constraints on rows/columns
fds_domain_check(T, N) :-
                 maplist(fd_domain_(1, N), T),
                 maplist(fd_all_different, T).

% Auxiliary to reorder arguments to fd_domain for maplist
fd_domain_(Min, Max, L) :- fd_domain(L, Min, Max).

fds_check_all_counts(T, Tr, Left, Top, Right, Bottom) :-
                            fds_check_counts(T, Left),
                            fds_check_counts(Tr, Top),
                            fds_check_reverse_counts(T, Right),
                            fds_check_reverse_counts(Tr, Bottom).

% Finite Domain Solver version of the plain_tower/3 code; no permutations
necessary
fds_check_counts([], []).
fds_check_counts([HT|TT], [HC|TC]) :-
                      fds_tower_count(HT, HC, 0, 0),
                      fds_check_counts(TT, TC).

fds_check_reverse_counts([], []).
fds_check_reverse_counts([HT|TT], [HC|TC]) :-
                      reverse(HT, RT),
                      fds_tower_count(RT, HC, 0, 0),
                      fds_check_reverse_counts(TT, TC).

% Finite domain solver arithmetic constraints on operations
fds_tower_count([], Count, Count, _).
fds_tower_count([H|T], Count, Counter, Highest) :-
                                H #< Highest,
                                fds_tower_count(T, Count, Counter,
Highest).
fds_tower_count([H|T], Count, Counter, Highest) :-
                                H #> Highest,
                                C #= Counter+1,
                                fds_tower_count(T, Count, C, H).

tower(N, T, C) :-
      % Check row/column lengths
      length(T, N),
      maplist(length_(N), T),
      % Apply constraints to rows/colums
        fds_domain_check(T, N),
      transpose(T, Tr),
      fds_domain_check(Tr, N),
      % Check count list lengths
      counts(Top, Bottom, Left, Right) = C,
      length(Top, N),
      length(Bottom, N),
      length(Left, N),
      length(Right, N),
      % Check all counts
      fds_check_all_counts(T, Tr, Left, Top, Right, Bottom),
      % Label solution; why doesn't it work in fd_domain_check -> explore
this
      maplist(fd_labeling, T).
```

```
% Check number of elements is correct; indeed is a square matrix
row_length([], _).
row_length([H|T], N) :- length(H, N), row_length(T, N).

% Transpose row lists to column lists; old SWI clpfd implementation
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).

% Check matrix is valid for counts
check_all_counts(T, Tr, N, L, Left, Top, Right, Bottom) :-
                                check_counts(T, Left, N, L),
                                check_counts(Tr, Top, N, L),
                                check_reverse_counts(T, Right, N, L),
                                check_reverse_counts(Tr, Bottom, N, L).

% Counts for left and top side; by default already in order
check_counts([], [], _, _).
check_counts([HT|TT], [HC|TC], N, L) :-
                      permutation(L, HT),
                      visible(HT, HC, N),
                      check_counts(TT, TC, N, L).

% Counts for right and bottom side; need to be reversed
check_reverse_counts([], [], _, _).
check_reverse_counts([HT|TT], [HC|TC], N, L) :-
                                reverse(HT, RT),
                                permutation(L, RT),
                                visible(RT, HC, N),
                                check_reverse_counts(TT, TC, N, L).

% Check how many towers are visible
% Somehow, removing the special cases N = 1 or N makes it faster?
%visible([H|_], 1, N) :- H = N.
%visible(Towers, N, N) :- consecutive(Towers, N, 1).
visible(Towers, Count, _) :- tower_count(Towers, Count, 0, 0).

%consecutive([], N, N).
%consecutive([H|T], N, Count) :- H = Count,
%                     C is Count+1,
%                     consecutive(T, N, C).

tower_count([], Count, Count, _).
tower_count([H|T], Count, Counter, Highest) :-
                      H < Highest,
                      tower_count(T, Count, Counter, Highest).
tower_count([H|T], Count, Counter, Highest) :-
                      H > Highest,
                      C is Counter+1,
                      tower_count(T, Count, C, H).

plain_tower(N, T, C) :-
```

```
      % Check row and column lengths
    length(T, N),
    row_length(T, N),
      % Break up C structure
      counts(Top, Bottom, Left, Right) = C,
      % Check count lengths
      length(Top, N),
      length(Bottom, N),
      length(Left, N),
      length(Right, N),
      % Transpose matrix
      transpose(T, Tr),
      % Get simple list for permutations
      findall(X, between(1, N, X), L),
      % Check all counts
      check_all_counts(T, Tr, N, L, Left, Top, Right, Bottom).

% CPU time performance ratio; don't care about start times, only stop
times
speedup(FPR) :-
      statistics(cpu_time, _),
      tower(5, _, counts([2,3,5,1,2], [2,2,1,4,3], [2,1,4,2,2],
[2,3,1,4,3])),
      statistics(cpu_time, [_, T_STOP]),
      plain_tower(5, _, counts([2,3,5,1,2], [2,2,1,4,3], [2,1,4,2,2],
[2,3,1,4,3])),
      statistics(cpu_time, [_, PT_STOP]),
      FPR is PT_STOP / T_STOP.

% Get puzzle with C counts and T1, T2 distinct solutions
ambiguous(N, C, T1, T2) :-
      tower(N, T1, C),
      tower(N, T2, C),
      T1 \= T2.
```