```c
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */

struct TreeNode* createNode(int val) {
    struct TreeNode* new;

    new = (struct TreeNode*)malloc(sizeof(struct TreeNode));
    new->val = val;
    new->left = NULL;
    new->right = NULL;

    return new;
}

struct TreeNode** _generateTrees(int start, int count, int* returnSize) {

    struct TreeNode** result;
    struct TreeNode** left_result;
    struct TreeNode** right_result;
    int left_return_size;
    int right_return_size;
    int end;
    int index;
    int left_index;
    int right_index;


    *returnSize = 0;
    result = NULL;

    if(count <= 0)
    {
        *returnSize = 1;
        return result;
    }

    end = start + count;

    for(index = start; index < end; index++)
    {

        left_result = _generateTrees(start, index - start, &left_return_size);
        right_result = _generateTrees(index+1, end - (index+1), &right_return_size);

        if(result == NULL)
        {
            result = (struct TreeNode**)malloc(sizeof(struct TreeNode*) * (left_return_size) * (right_return_size) ) ;
        }else
        {
            result = (struct TreeNode**)realloc(result, sizeof(struct TreeNode*) * (*returnSize + (left_return_size) * (right_return_size)) ) ;
        }

        for(left_index = 0; left_index < left_return_size; left_index++)
        {
            for(right_index = 0; right_index < right_return_size; right_index++)
            {
                result[*returnSize] = createNode(index);
                result[*returnSize]->left = (left_result == NULL) ? NULL : left_result[left_index];
                result[*returnSize]->right = (right_result == NULL) ? NULL : right_result[right_index];
                (*returnSize)++;
            }
        }
    }

    return result;
}

struct TreeNode** generateTrees(int n, int* returnSize) {

     return _generateTrees(1, n, returnSize);
}
```