```c
/**
 * Return an array of arrays of size *returnSize.
 * The sizes of the arrays are returned as *returnColumnSizes array.
 * Note: Both returned array and *columnSizes array must be malloced, assume caller calls free().
 */
#define ALLOC_LENGTH                (100)
#define MAX_STACK_LENGTH            (100)
int alloc_length = ALLOC_LENGTH;
int **result;
void _combinationSum2(int* candidates, int candidatesSize, int target, int* returnSize, int** returnColumnSizes, int* stack, int stack_ptr, int pos){
    int index;

    for(index = pos; index < candidatesSize; index++)
    {
        if (index > pos && candidates[index] == candidates[index - 1]) continue;
        stack[stack_ptr+1] = candidates[index];
        if(target == candidates[index])
        {
            //printf("stack_ptr:%d, *returnSize:%d\n", stack_ptr, *returnSize);
            stack_ptr++;
            result[*returnSize] = (int*)malloc(sizeof(int)*(stack_ptr+1));
            memcpy(result[*returnSize], stack, sizeof(int)*(stack_ptr+1));
            (*returnColumnSizes)[*returnSize] = stack_ptr+1;
            (*returnSize)++;
            if( (*returnSize % ALLOC_LENGTH) == 0 )
            {
                //printf("Realloc *returnSize:%d\n", *returnSize);
                alloc_length+=ALLOC_LENGTH;
                result = (int**)realloc(result, sizeof(int*)*alloc_length);
                *returnColumnSizes = (int*)realloc(*returnColumnSizes, sizeof(int)*alloc_length);
            }
            stack_ptr--;
        }else if(target > candidates[index])
        {
            //printf("target:%d, index:%d, stack_ptr:%d, can:%d\n", target, index, stack_ptr, candidates[index]);
            _combinationSum2(candidates, candidatesSize, target - candidates[index], returnSize, returnColumnSizes, stack, stack_ptr+1, index+1);
        }

    }



}

void swap(int* a, int* b)
{
    int tmp;

    tmp = *a;
    *a = *b;
    *b = tmp;
}

int partition(int* nums, int l, int r)
{
   int base;
   int pivot;
   int index;

   pivot = nums[r];
   base = l - 1;

   for(index = l; index < r; index++)
   {
       if(nums[index] < pivot)
       {
           base++;
           swap(&nums[base], &nums[index]);
       }
   }

    base++;
    swap(&nums[base], &nums[r]);

    return base;
}

void quickSort(int* nums, int l, int r)
{
    if(l<r)
    {
        int pivot_pos;

        pivot_pos = partition(nums, l, r);
        quickSort(nums, l, pivot_pos - 1);
        quickSort(nums, pivot_pos + 1, r);
    }
}

int** combinationSum2(int* candidates, int candidatesSize, int target, int* returnSize, int** returnColumnSizes) {
    int stack[MAX_STACK_LENGTH];
    int stack_ptr;


    quickSort(candidates, 0, candidatesSize-1);
    *returnSize = 0;
    result = (int**)malloc(sizeof(int*)*alloc_length);
    *returnColumnSizes = (int*)malloc(sizeof(int)*alloc_length);
    stack_ptr = -1;
    _combinationSum2(candidates, candidatesSize, target, returnSize, returnColumnSizes, stack, stack_ptr, 0);

    if (*returnSize == 0)
    {
        free(result);
        free(*returnColumnSizes);
        *returnColumnSizes = NULL;
        result = NULL;
    }

    return result;
}
```