```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().
 */

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 index;
    int pivot;


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

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

    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);
    }
}

void _permuteUnique(int* nums, int numsSize, int* returnSize, int** returnColumnSizes, int** result, int* stack, int stack_ptr)
{
    int index;
    int tmp;

    for(index = 0; index < numsSize; index++)
    {
        if(nums[index] != INT_MIN)
        {
            stack[stack_ptr+1] = nums[index];
            if( (index == 0) || (nums[index -1] != nums[index]) )
            {
                if( (stack_ptr+1) == (numsSize-1) )
                {
                    result[*returnSize] = (int*)malloc(sizeof(int)*numsSize);
                    memcpy(result[*returnSize], stack, sizeof(int)*numsSize);
                    (*returnColumnSizes)[*returnSize] = numsSize;
                    (*returnSize)++;
                }else
                {
                    tmp = nums[index];
                    nums[index] = INT_MIN;
                    _permuteUnique(nums, numsSize, returnSize, returnColumnSizes, result, stack, stack_ptr+1);
                    nums[index] = tmp;
                }
            }
        }
    }
}

int** permuteUnique(int* nums, int numsSize, int* returnSize, int** returnColumnSizes) {

    int** result;
    int*  valid;
    int stack[8];
    int alloc_length;
    int index;

    alloc_length = 1;
    *returnSize = 0;

    for(index = numsSize; index > 0; index--)
    {
       alloc_length *= index;
    }

    result = (int**)malloc(sizeof(int*)*alloc_length);
    *returnColumnSizes = (int*)malloc(sizeof(int)*alloc_length);
    quickSort(nums, 0, numsSize-1);


    _permuteUnique(nums, numsSize, returnSize, returnColumnSizes, result, stack, -1);

    return result;
}
```