```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)

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

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


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

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

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

    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** fourSum(int* nums, int numsSize, int target, int* returnSize, int** returnColumnSizes){

    int index;
    int second;
    int third;
    int fourth;
    int** result;
    int alloc_length;
    int result_index;
    int64_t sum;

    alloc_length = ALLOC_LENGTH;
    result = (int**)malloc(sizeof(int*)*alloc_length);
    *returnColumnSizes = (int*)malloc(sizeof(int)*alloc_length);
    result_index = 0;

    quickSort(nums, 0 , numsSize-1);

    for(index = 0; index < numsSize; index++)
    {
        if( (index > 0) && nums[index] == nums[index-1])
        {
            continue;
        }

        for(fourth = index+1; fourth < numsSize; fourth++)
        {

            if( (fourth != index+1) &&
                (fourth > 1) &&
                (nums[fourth] == nums[fourth-1])
              )
            {
                continue;
            }

            second = fourth + 1;
            third = numsSize - 1;

            while(second < third)
            {
```

```
                sum = (int64_t)nums[index] + nums[second] + nums[third] + nums[fourth];

                if(sum == target)
                {

                    result[result_index] = (int*)malloc(sizeof(int)*4);

                    result[result_index][0] = nums[index];
                    result[result_index][1] = nums[second];
                    result[result_index][2] = nums[third];
                    result[result_index][3] = nums[fourth];

                    (*returnColumnSizes)[result_index] = 4;
                    result_index++;

                    if( (result_index % ALLOC_LENGTH) == 0 )
                    {
                        alloc_length += ALLOC_LENGTH;
                        result = (int**)realloc(result, sizeof(int*)*alloc_length);
                        *returnColumnSizes = (int*)realloc(*returnColumnSizes, sizeof(int)*alloc_length);
                    }


                    second++;
                    third--;

                    while((second < third) && nums[second] == nums[second-1]) second++;
                    while((second < third) && nums[third] == nums[third+1]) third--;
                }else if(sum > target)
                {
                    third--;
                }else
                {
                    second++;
                }
            }
        }

    }

    *returnSize = result_index;

    return result;

}
```