```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 threeSumClosest(int* nums, int numsSize, int target){

    int index;
    int second;
    int third;
    int sum;
    int near;


    near = INT_MIN;

    quickSort(nums, 0 , numsSize-1);

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

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

        while(second < third)
        {

            sum = nums[index] + nums[second] + nums[third];

            if(sum == target)
            {
                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++;
            }

            if( (near == INT_MIN) || abs(near - target) > abs(sum - target))
            {
                near = sum;
            }
        }

    }


    return near;

}
```