```
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
#define SOLUTION_1          1
#define SOLUTION_2          0


#define ALLOC_LENGTH    (2)

struct map
{
    uint8_t count;
    int index;
};

#if(SOLUTION_1)

int* twoSum(int* nums, int numsSize, int target, int* returnSize){

    int* result;
    struct map* map;
    int f_index;
    int min;
    int max;
    int tmp;

    result = (int*)malloc(sizeof(int) * ALLOC_LENGTH);
    *returnSize = ALLOC_LENGTH;
    min = INT_MAX;
    max = INT_MIN;

    for(f_index = 0; f_index < numsSize; f_index++)
    {
        if(nums[f_index] < min)
        {
            min = nums[f_index];
        }

        if(nums[f_index] > max)
        {
            max = nums[f_index];
        }
    }

    map = (struct map*)calloc( (max-min+1), sizeof(struct map));


    for(f_index = 0; f_index < numsSize; f_index++)
    {
        tmp = target - nums[f_index];

        if( ( min <= tmp ) &&( tmp <= max ) && map[tmp - min].count > 0 )
        {
            if(f_index < map[tmp - min].index)
            {
                result[0] = f_index;
                result[1] =  map[tmp - min].index;
            }else
            {
                result[0] =  map[tmp - min].index;
                result[1] = f_index;
            }
        }
```

```
        map[nums[f_index]-min].count++;
        map[nums[f_index]-min].index = f_index;
    }


    return result;
}

#elif(SOLUTION_2)

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 binarySearch(int* nums, int value, int l, int r)
{
    int m;

    while(l<=r)
    {
        m = ( l + (r-l) ) / 2;

        if(nums[m] == value)
```

```
        {
            return m;
        }else if(nums[m] > value)
        {
            r =  m-1;
        }else if(nums[m] < value)
        {
            l = m+1;
        }
    }


    return -1;
}

int* twoSum(int* nums, int numsSize, int target, int* returnSize){

    int* result;
    int f_index;
    int s_index;
    int c_num;
    int search_result;
    int result_index;

    result = (int*)malloc(sizeof(int) * ALLOC_LENGTH);


    for(f_index = 0; f_index < numsSize; f_index++)
    {
        for(s_index = f_index+1; s_index < numsSize; s_index++)
        {
            if(target == ( nums[f_index] + nums[s_index] ) )
            {
                result[0] = f_index;
                result[1] = s_index;
                *returnSize = 2;
                return result;
            }
        }
    }

    *returnSize = 0;
    return result;
}
#endif
```