1.Two Sum

> Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to* `target`. You may assume that each input would have **exactly one solution**,and you may not use the same element twice. You can return the answer in any order.

My Anwser

Difficulty	Status	Runtime	Distribution(%)	Memory	Distribution(%)
Easy	AC	215ms	18.37	6.4MB	62.58

```
int* twoSum(int* nums, int numsSize, int target, int* returnSize){
    int i, j;
    int *result = (int *)malloc(sizeof(int) * 2);

    *returnSize = 0;
    for (i = 0; i < numsSize - 1; i++) {
        for (j = i + 1; j < numsSize; j++) {
            if (nums[i] + nums[j] == target) {
                result[0] = i;
                result[1] = j;
                *returnSize = 2;
                break;
            }
        }
    }
    return result;
}
```

Algorithm

Time complexity	Space complexity
$O(n^2)$	$O(1)$

The brute force approach is simple. Loop through each element *x* and find if there is another value that equals to *target - x*.
But if it's your first time to solve the problem,you may need to know that the returned array must be malloced.

Another Answer

> | Difficulty | Status | Runtime | Distribution(%) | Memory | Distribution(%) |
> |---|---|---|---|---|---|
> | Easy | AC | 11ms | 94.88 | 7.9MB | 5.40 |

```
typedef struct {
    int key;
    int val;
    UT_hash_handle hh;
} HashTable;

HashTable *g_hashTable;

HashTable *HashFind(int key) {
    HashTable *hashTable;
    HASH_FIND_INT(g_hashTable, &key, hashTable);
    return hashTable;
}

void HashInsert(int key, int val) {
    HashTable *hashTable = HashFind(key);
    if (hashTable == NULL) {
        hashTable = (HashTable *)malloc(sizeof(HashTable));
        hashTable->key = key;
        hashTable->val = val;
        HASH_ADD_INT(g_hashTable, key, hashTable);
    }
}

int* twoSum(int* nums, int numsSize, int target, int* returnSize){
    int i;
    HashTable *hashTable;
    int *result = (int *)malloc(sizeof(int) * 2);
    g_hashTable = NULL;
    for (i = 0; i < numsSize; i++) {
        hashTable = HashFind(target - nums[i]);
```

```
        if (hashTable != NULL) {
            result[0] = hashTable->val;
            result[1] = i;
            *returnSize = 2;
            return result;
        } else {
            HashInsert(nums[i], i);
        }
    }
    *returnSize = 0;
    return result;
}
```

Algorithm

> | Time complexity | Space complexity |
> | --- | --- |
> | $O(n)$ | $O(n)$ |

To improve our runtime complexity,we need a more efficient way to check if the complement exists in the array. The best way to maintain a mapping of each element in the array to its index is a hash table.