

找出一个数组中和为 n 的两个数

数组是递增的有序数组。

方法一：

- 双循环。
- 第一层循环拿到数组的一个元素，第二层循环和第一层循环拿到的元素相加如果和 n 相等，则找到了退出循环。
- 由于用到了双循环，时间复杂度是 $O(n^2)$

方法二：

- 双指针，加一层循环。
- 定义两个变量作为指针，第一个指针指向数组第一个元素，第二个指针指向数组最后一个元素。
- 判断以上两个数相加和 n 关系。如果大于 n，证明两数之和要减小，由于第一个指针是第一个数不能再减小，只能减少第二个指针，于是将第二个指针的位置向前移动；如果小于 n，于是将第一个指针的位置向后移动；如果相等，则找到了，推出循环。

```
// 双循环
// 时间复杂度  $O(n^2)$ 
function twoArraySum1(arr: number[], n: number): number[] {
    const len: number = arr.length;
    let res: number[] = [];

    if (!len) {
        return res;
    }
    for (let i = 0; i < len - 1; i++) {
        let isSearched = false;
        for (let j = i + 1; j < len; j++) {
            if (i + j === n) {
                res = [i, j];
                isSearched = true;
                break;
            }
        }
        if (isSearched) {
            break;
        }
    }

    return res;
}

// 功能测试
let arr1 = [];
for (let i = 0; i < 10 * 10000; i++) {
    arr1.push(i);
}
console.time("双循环耗时");
```

```
twoArraySum1(arr1, 105000); // 544.86ms
console.timeEnd("双循环耗时");
```

```
// 双指针加单循环
// 时间复杂度 O(n)
function twoArraySum2(arr: number[], n: number): number[] {
  const len: number = arr.length;
  let res: number[] = [];

  if (!len) {
    return res;
  }

  let startIndex = 0; // 头 (指针位置, 索引)
  let endIndex = len - 1; // 尾 (指针位置, 索引)

  while(startIndex < endIndex) {
    let start = arr[startIndex];
    let end = arr[endIndex];
    let sum = start + end;
    if (sum < n) {
      startIndex++;
    } else if (sum === n) {
      res = [start, end];
      break;
    } else {
      endIndex--;
    }
  }

  return res;
}

// 功能测试
let arr2 = [];
for (let i = 0; i < 10 * 10000; i++) {
  arr2.push(i);
}
console.time("双指针加单循环耗时");
twoArraySum2(arr2, 105000); // 0.34 ms
console.timeEnd("双指针加单循环耗时");
```

从上面耗时可以看出，**双循环**明显大于**双指针加单循环**的耗时，而且数组长度越大，表现越明显。