二分法 Binary Search

使用条件

- 排序数组 (30-40%是二分)
- 当面试官要求你找一个比 O(n) 更小的时间复杂度算法的时候(99%)
- 找到数组中的一个分割位置，使得左半部分满足某个条件，右半部分不满足(100%)
- 找到一个最大/最小的值使得某个条件被满足(90%)

复杂度

1. 时间复杂度：O(logn)
2. 空间复杂度：O(1)

例题

代码模板

```
    int binarySearch(int[] nums, int target) {
        // corner case 处理
        if (nums == null || nums.length == 0) {
            return -1;
        }

        int start = 0, end = nums.length - 1;

        // 要点1: start + 1 < end
        while (start + 1 < end) {

            // 要点2: start + (end - start) / 2
            int mid = start + (end - start) / 2;
            // 要点3: =, <, > 分开讨论，mid 不 +1 也不 -1
            if (nums[mid] == target) {
                return mid;
            } else if (nums[mid] < target) {
                start = mid;
            } else {
                end = mid;
            }
        }

        // 要点4: 循环结束后，单独处理start和end
        if (nums[start] == target) {
            return start;
        }
        if (nums[end] == target) {
            return end;
        }
        return -1;
```

```
    }
```

双指针 Two Pointer

使用条件

复杂度

1. 时间复杂度
    - 时间复杂度与最内层循环主体执行的次数有关
    - 与有多少重循环有关
2. 空间复杂度 O(1)
    - 只需要分配两个指针的额外内存

例题

- LintCode 1879. 两数之和VII(同向双指针)
- LintCode1712.和相同的二元子数组(相向双指针)
- LintCode627. 最长回文串 (背向双指针)
- LintCode 64: 合并有序数组

代码模板

相向双指针(patition in quicksort)

```
public void patition(int[] A, int start, int end) {
        if (start >= end) {
            return;
        }
        int left = start, right = end;
        // key point 1: pivot is the value, not the index
        int pivot = A[(start + end) / 2];
        // key point 2: every time you compare left & right, it should be
        // left <= right not left < right
        while (left <= right) {
            while (left <= right && A[left] < pivot) {
                left++;
            }
            while (left <= right && A[right] > pivot) {
                right--;
            }
            if (left <= right) {
                int temp = A[left];
                A[left] = A[right];
                A[right] = temp;
                left++;
                right--;
            }
        }
}
```

背向双指针

```
left = position;
right = position +1;
while(left >=0 && right < length){
    if (可以停下来了) {
        break;
    }
    left--;
    right++;
}
```

同向双指针

```
int j = 0;
for(int i = 0; i < n; i++){
    // 不满足则循环到满足搭配为止
    while (j < n && i 到 j之间不满足条件){
        j += 1;
    }
    if (i 到 j之间满足条件){
        处理i，j这次搭配
    }
}
```

合并双指针

```
    ArrayList<Integer> merge(ArrayList<Integer> list1, ArrayList<Integer> list2)
{
        // 需要 new 一个新的 list，而不是在 list1 或者 list2 上直接改动
        ArrayList<Integer> newList = new ArrayList<Integer>();

        int i = 0, j = 0;
        while (i < list1.size() && j < list2.size()) {
            if (list1.get(i) < list2.get(j)) {
                newList.add(list1.get(i));
                i++;
            } else {
                newList.add(list2.get(j));
                j++;
            }
        }

        // 合并上下的数到 newList 里
        // 无需用 if (i < list1.size())，直接 while 即可
        while (i < list1.size()) {
            newList.add(list1.get(i));
            i++;
        }
        while (j < list2.size()) {
            newList.add(list2.get(j);
            j++;
        }

        return newList;
    }
```

二叉树分治 Binary Tree Divide & Conquer

使用条件

复杂度

例题

代码模板

二叉搜索树非递归 BST Iterator

使用条件

复杂度

例题

代码模板

宽度优先搜索

使用条件

复杂度

例题

代码模板

深度优先搜索 DFS

使用条件

复杂度

例题

代码模板

动态规划 Dynamic Programming

使用条件

复杂度

例题

代码模板

堆 Heap

使用条件

复杂度

例题

代码模板

并查集 Union Find

使用条件

复杂度

例题

代码模板

字典树 Trie

使用条件

复杂度

例题

代码模板