



查看时间复杂度方法：
直接看这个函数或这段代码它根据n的不同情况它会运行多少次

时间复杂度

最常见7中时间复杂度

O为函数，函数以极限方式
进行处理，故会忽略常数

$O(1)$: constant complexity 常数线性复杂度

$O(\log n)$: Logarithmic Complexity 对数时间复杂度

$O(n)$: Linear Complexity 线性时间复杂度

$O(n^2)$: N square Complexity平方时间复杂度

$O(n^3)$: N cubic Complexity 平方时间复杂度

$O(2^n)$: Exponential Growth 指数时间复杂度

$O(n!)$: Factorial 阶乘时间复杂度

Master Theorem(主定理)

Algorithm	Recurrence relationship	Run time	Comment
二分查找 (Binary search) (本身有序)	$T(n) = T(n/2) + O(1)$	$O(\log n)$	Apply Master theorem case $c = \log_b a$ where $a=1, b=2, c=0, k=0$
二叉树遍历 (Binary tree traversal)	$T(n) = 2T(n/2) + O(1)$	$O(n)$	Apply Master theorem case $c < \log_b a$ where $a=2, b=2, c=0$
二维矩阵二分查找 (有序)	$T(n) = 2T(n/2) + O(1)$	$O(n)$	Apply the Akra-Bazzi theorem for $p=1$ and $g(u) = \log(u)$ to get $\theta(2n \cdot \log n)$
归并排序(Merge sort)	$T(n) = 2T(n/2) + O(n)$	$O(n \log n)$	Apply Master theorem case $c = \log_b a$ where $a=2, b=2, c=1, k=0$

三数之和问题
接雨水问题

两重遍历数组模板：
//实现i 和j 两个下标对数组的遍历，同时保证i和j不会重复
for(int i =0; i<a.size()-1;i++){
 for(int j=i+1;j<a.size();j++){
 // do something
 }
}

双指针左右夹逼模板：
for(int left=0, int right=array.size()-1;left<right;){
 int min = array[left]<array[right]?array[left++]:array[right--]
 // do something
}

一维数组常用解法：
1. 一位数组坐标变换： i, j交换
2. 左右夹逼，向中间进行收敛

为何找重复子问题：计算机只能识别if else,while,for,reursion

算法无思路时处理方式(基本思想)：
1. 暴力？
2. 基本情况
3. 泛化：找最近重复子问题

数组_链表_跳表

数据结构加速核心思想：
升维+空间换时间

数组

□□□□□□□□□□□□□□

链表

跳表

特点：
1. 基于链表实现，前提链表里面元素有序，对标的是平衡树(AVL Tree)
 插入/删除/搜索 都是 $O(\log n)$ 的数据结构
2. 优势： 原理简单，容易实现，方便扩展，效率更高，常用于热门项目替代平衡树
3. 跳表替代平衡二叉树，技术更新换代，
 由于跳表（19989）技术出现晚于平衡二叉树(196×年)， 所以一些较老的数据结构使用二叉树，
 而一些较新的项目使用跳表， 特别是一些元素个数不多的情况下使用跳表。如：Redis， LevelDB等数据

prepend	$O(1)$
append	$O(1)$
lookup	$O(1)$
insert	$O(n)$
delete	$O(n)$

prepend	$O(1)$
append	$O(1)$
lookup	$O(n)$
insert	$O(1)$
delete	$O(1)$

链表翻转模板：
ListNode* reverse (ListNode*head,ListNode*tail)
{
 // 左闭右开
 ListNode* prv=null;
 ListNode* temp=null;
 while(head!=tail){
 temp=head->next;
 head->next=prv;
 prv=head;
 head=temp;
 }
 return prv;
}

工程应用： LRU





