# 20 滑动窗口的最大值

## 题目描述：

给定数组，和滑动窗口的大小，请找出所有滑动窗口内的最大值。

例如，数组{2,3,4,2,6,2,5,1}，滑窗大小为3，那么结果为{4,4,6,6,6,5}。

## 思路：

使用双向队列（deque），用来保存有可能是滑动窗口最大值的数组的下标。在存入一个数字的下标之前，首先要判断队列里已有数字是否小于待存入的数字。如果已有的数字小于待存入的数字，那么这些数字已经不可能是滑动窗口的最大值，因此需要在队列尾部删除（pop\_back）。同时，如果队列头部的数字已经从窗口里滑出，那么滑出的数字也需要从队列的头部删除（pop\_front）。

## 复杂度分析：

## 代码解析：

//size代表滑动窗的大小为多少

vector<int> maxInWindows(const vector<int>& num, unsigned int size)

{

vector<int> result;

if(size < 1 || num.size() < size)

{

return result;

}

//deque为双端队列

deque<int> qmax;

for(int i = 0; i < num.size(); ++i)

{

//判断队列首元素中存储的为滑动窗口内最大值

while(!qmax.empty() && num[qmax.back()] <= num[i])

{

qmax.pop\_back();

}

qmax.push\_back(i);

//当队列头存储的单元格位置不在滑动窗口内时将其弹出

if(qmax.front() == i-size)

{

qmax.pop\_front();

}

//从第size个数进行存储，即从滑动窗大小位置时进行存储

if(i >= size - 1)

{

result.push\_back(num[qmax.front()]);

}

}

return result;

}

# 21旋转数组的最小数字

## 题目描述：

数组｛3，4，5，1，2｝为{1,2,3,4,5}的一个旋转，该数组的最小元素为1。

## 思路：

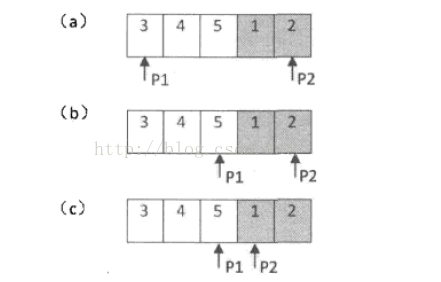
我们注意到旋转之后的数组实际上可以划分为两个排序的子数组，而且前面的子数组的元素都是大于或者等于后面子数组的元素。我们还注意到最小的元素刚好是这两个子数组的分界线。在排序的数组中我们可以利用二分查找来实现O（logn)的查找。本题给出的数组在一定程度上是排序的，因此我们可以试着用二分查找的思路来寻找这个最小的元素。

### 思考点一：

1.我们先把第一个指针指向第0个元素，把第二个指针指向第4个元素，如图所示。位于两个指针中间（在数组的下标是2）的数字是5，它大于第一个指针指向的数字。因此中间数字5一定位于第一个递增字数组中，并且最小的数字一定位于它的后面。因此我们可以移动第一个指针让它指向数组的中间。

2.此时位于这两个指针中间的数字为1，它小于第二个指针指向的数字。因此这个中间数字为1一定位于第二个递增子数组中，并且最小的数字一定位于它的前面或者它自己就是最小的数字。因此我们可以移动第二个指针指向两个指针中间的元素即下标为3的元素。

3.此时两个指针的距离为1，表明第一个指针已经指向了第一个递增子数组的末尾，而第二个指针指向第二个递增子数组的开头。第二个子数组的第一个数字就是最小的数字，因此第二个指针指向的数字就是我们查找的结果



### 思考点二：

**重点考虑：当left元素 mid元素 right元素 均相等时如何进行判断**

数组｛1，0，1，1，1｝和数组｛1，1，1，0，1｝都可以堪称递增排序数组｛0，1，1，1，1｝的旋转，图2分别画出它们由最小数字分隔开的两个子数组。



这两种情况中，第一个指针和第二个指针指向的数字都是1，并且两个指针中间的数字也是1，这3个数字相同。在第一种情况中，中间数字（下标为2)位于后面是子数组；在第二种情况中，中间数字（下标为2）位于前面的子数组中。因此，当两个指针指向的数字及它们中间的数字三者相同的时候，我们无法判断中间的数字是位于前面的子数组中还是后面的子数组中国，也无法移动两个指针来缩小查找的范围。此时，采用顺序查找的方法。

# 22斐波那契数列

## 题目描述：

写一个函数，输入n，求斐波那契数列的第n项。

## 思路：

经典的动态规划基础题。

动态规划分为两类：

1. 最优子结构

2. 公共子问题

斐波那契数列这题属于公共子问题。利用公式：f(n) = f(n-1) + f(n-2) 容易发现 f(n-1) 和 f(n-2) 求解过程中有大量重复的计算，如 f(n-3) 既会出现在f(n-1)的求解分支中，也会出现在 f(n-2) 的求解分支中。这就是公共的子问题。

对于解决公共子问题也有2种途径：

1. 备忘录法：增加缓存，如果缓存中已有解则直接调用，不再单独求解。

2. 自底向上构建：“真正的动态规划”，备忘录法显然增加了空间复杂度，对于此题可以从f(0) = 0 f(1) = 1 出发逐步求解出 f(n)，而不是从f(n) 向下索解。

1.递归求解 f(n) = f(n-1) + f(n-2)

时间复杂度：T(n) = T(n-1) + T(n-2) + O(1) = O(2^n)

空间复杂度：递归深度O(n)

2.备忘录法 f(n) = f(n-1) + f(n-2)

时间复杂度：T(n) = T(n-1) + O(1) = O(n) (f(n-2) 已经在求f(n-1)时计算出)

空间复杂度：递归深度O(n)，缓存O(n)，总O(n)

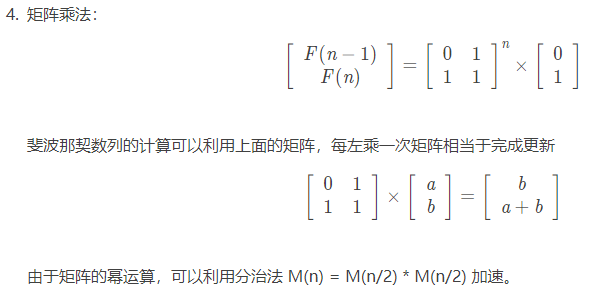
1. 自底向上，迭代求解。

时间复杂度：O(n)

空间复杂度：O(1)

4.矩阵乘法：

时间复杂度：T(n) = T(n/2) + O(1) = O(logn)

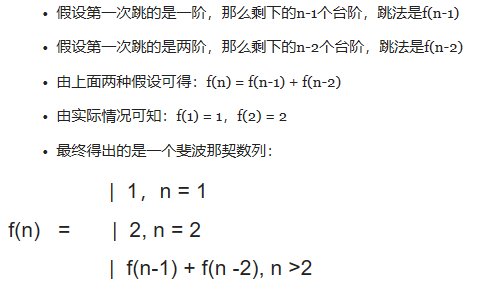
空间复杂度： 递归计算矩阵幂的深度O(logn)

# 23跳台阶

## 题目描述：

一只青蛙一次可以跳上1级台阶，也可以跳上2级。求该青蛙跳上一个n级的台阶总共有多少种跳法。

## 思路：

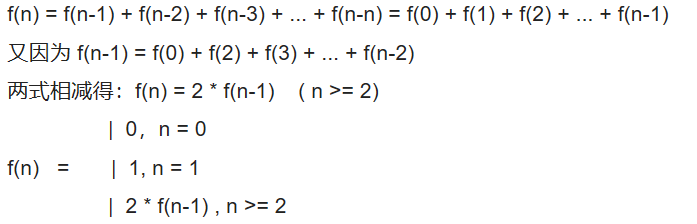


# 24变态跳台阶

## 题目描述：

一只青蛙一次可以跳上1、2。。。n级台阶。求该青蛙跳上一个n级的台阶总共有多少种跳法。

## 思路：



# 25矩形覆盖

与22 23同理均为斐波那契数列的变形应用

# 26二进制数中1的个数

## 题目描述：

输入一个整数，输出该数二进制表示中1的个数。其中负数用补码表示。

## 思路：

int NumberOf1(int n)

{

int count = 0;

//计算的为1的个数

while (n)

{

count++;

//采用与运算

n = (n - 1) & n;

}

return count;

}

# 27数值整数次方

## 题目描述：

给定一个double类型的浮点数base和int类型的整数exponent。求base的exponent次方

## 思路：

首先思考平时数字的计算次方的方法，对于一个数字的0次方，值是1，一个数字的1次方，是它本身。所以先把这两个特殊的编写出来。其次思考当正数的2次方，是本身乘以本身，其后到3次方，是在2次方的基础上乘以本身，所以想到了递归的方式，对于负数，只是每次乘以数字分之一。

# 129查找单链表倒数第k个位置上的结点

## 题目描述：

给定一个double类型的浮点数base和int类型的整数exponent。求base的exponent次方

## 思路：

# 30反转链表

## 题目描述：

给定一个double类型的浮点数base和int类型的整数exponent。求base的exponent次方

## 思路：

# 31合并两个排序的链表

## 题目描述：

给定一个double类型的浮点数base和int类型的整数exponent。求base的exponent次方

## 思路：