Implement [pow(*x*, *n*)](http://www.cplusplus.com/reference/valarray/pow/).

**Example 1:**

**Input:** 2.00000, 10

**Output:** 1024.00000

**Example 2:**

**Input:** 2.10000, 3

**Output:** 9.26100

题目求一个数的n次方。

我们拿到题目的时候就会想到一个一个乘呗，但是这种做法虽然简单暴力，但是时间复杂度很高，不推荐。

我们回顾一下快速排序的思想，每次都取一半，然后变成 两个一半，然后在把每个一半分区再取一半。我们通过递归可以很轻松的做到取半操作。那么重点是取半完之后需要做什么计算呢？快速排序方法，取半之后的操作的排序，我们这里需要做的操作是把取半的区域里的所有数据相乘。这里注意，你取半后所做的操作一定是写在递归的下边的。还有最后的重中之重就是写出口。当没有返回值的时候，我们要写的是出口，当有返回值的时候我们要写的是终点。出口的意思是，当我们递归递归再递归到一定位置的时候就不会进行递归，递归完毕然后执行下面需要对递归出来的数据进行处理的操作。终点的意思也一样，不过终点需要给出一个递归停止时候的值，也就是最后一个递归的返回值。快速排序，递归的条件是low<high，当low=high的时候，说明当前的分区只有一个数字，一个数字不用排序，不用再次向下递归，结束本次的递归。我们这道题呢，需要的是有返回值的递归，我们需要考虑每次要取半，取到什么时候呢？取到终点的时候返回什么值呢？我们最终需要的是每个区的所有数据的乘积，我们考虑，当n=0的时候我们就不用向下再分了，因为也没法分了，当这段没有数字他们的乘积为多少呢？不是0，是1，因为1在乘法里相当于没有啊。所以最后我们考虑到终点是0，返回1.0；因为这里每个分区里面的数据都是一样的，所以不用做处理两个子分区的操作，只需要处理一个分区，然后把两个分区相乘就可以了。

下面是代码

public static double mypow(double x,double n){

int N = (int)n;

if(N<0){ //当n小于0的时候

x = 1/x;// 我们需要把数字取反

N = Math.abs(N);//n取绝对值

}

return quickpow(x,N);

}

public static double quickpow(double x,int n){

if(n==0){ //递归终点

return 1.0;

}

double half = quickpow(x,n/2);

//长度每次分半，直到长度为1的时候，1/2=0的时候停止

//返回的时候half=1.0

//然后开始向前递归

//有了基础的half，考虑当前的分半是否整分

if(n%2==0)//整分直接平放

return half\*half;

else//没整分，因为是向下取整，每次奇数都会余出来一个单个数据

//所以在原来的基础上多乘以一个x

return half\*half\*x;

}