二维数组：

二维数组的初始化方式：

int[][] m = {{1,2,3},  
        {2,3,4}};

int m[][];  
m = new int[4][4];

使用这种方法，初始化出的第二维的长度都是相同的，如果需要初始化第二维长度不一样的二维数组，则可以使用如下的格式：  
               int n[][];  
               n = new int[2][]; //只初始化第一维的长度  
               //分别初始化后续的元素  
               n[0] = new int[4];  
               n[1] = new int[3];

             这里的语法就体现了数组的数组概念，在初始化第一维的长度时，其实就是把数组n看成了一个一维数组，初始化其长度为2，则数组n中包含的2个元素分别是n[0]和n[1]，而这两个元素分别是一个一维数组。后面使用一维数组动态初始化的语法分别初始化n[0]和n[1]。

一、含义

      递归算法是一种直接或间接地调用自身的算法。在计算机编写程序中，递归算法对解决一大类问题是十分有效的，它往往使算法的描述简洁而且易于理解。

用循环打印：

1 \* 1 = 1

1 \* 2 = 2 2 \* 2 = 4

1 \* 3 = 3 2 \* 3 = 6 3 \* 3 = 9

1 \* 4 = 4 2 \* 4 = 8 3 \* 4 = 12 4 \* 4 = 16

1 \* 5 = 5 2 \* 5 = 10 3 \* 5 = 15 4 \* 5 = 20 5 \* 5 = 25

1 \* 6 = 6 2 \* 6 = 12 3 \* 6 = 18 4 \* 6 = 24 5 \* 6 = 30 6 \* 6 = 36

1 \* 7 = 7 2 \* 7 = 14 3 \* 7 = 21 4 \* 7 = 28 5 \* 7 = 35 6 \* 7 = 42 7 \* 7 = 49

1 \* 8 = 8 2 \* 8 = 16 3 \* 8 = 24 4 \* 8 = 32 5 \* 8 = 40 6 \* 8 = 48 7 \* 8 = 56 8 \* 8 = 64

1 \* 9 = 9 2 \* 9 = 18 3 \* 9 = 27 4 \* 9 = 36 5 \* 9 = 45 6 \* 9 = 54 7 \* 9 = 63 8 \* 9 = 72 9 \* 9 = 81

public geyongming{

public static void main(String[] args){

for(int i = 1; i < 10; i++){

for(int j = 1; j <= i; j++){

System.out.print(j+" "+i+" = "+i\*j);

}

System.out.println();

}

}

}

递归方法写法：

步骤：1.用条件控制，规定末状态，即控制递归停止，以避免死循环。

2.写入递归方法体

3.递归调用方法并规定步幅

在主方法中直接调用递归方法

打印：

1 \* 1 = 1

1 \* 2 = 2 2 \* 2 = 4

1 \* 3 = 3 2 \* 3 = 6 3 \* 3 = 9

1 \* 4 = 4 2 \* 4 = 8 3 \* 4 = 12 4 \* 4 = 16

**public** **static** **void** printMultiTable(**int** i){

**if**(i==1){

System.***out***.println("1\*1=1");

}**else**{

*printMultiTable*(i-1);

**for**(**int** j = 1;j<=i;j++){

System.***out***.print(j + "\*" + i + "=" + j \* i + " ");

}

System.***out***.println();

}

}

用递归打印n！

**public** **static** **int** factorial(**int** n){

**if**(n==0){

**return** 1;

}**else**{

**return** n\**factorial*(n-1);

}

}

递归与循环的区别：

递归代码可读性更强，更实用和容易变化

循环的运行时间和内存使用都优于递归

采用分治策略的排序算法：

快速排序:

**public** **static** **void** quickSort(**int** a[]) {

*sort*(a, 0, a.length - 1);

}

**public** **static** **void** sort(**int** a[], **int** low, **int** hight) {

**int** i, j, index;

**if** (low > hight) {

**return**;

}//检验是否排序完成

i = low;

j = hight;

index = a[i];

**while** (i < j) {

**while** (i < j && a[j] >= index)

j--;

**if** (i < j)

a[i++] = a[j];

**while** (i < j && a[i] < index)

i++;

**if** (i < j)

a[j--] = a[i];

}

a[i] = index;

*sort*(a, low, i - 1);

*sort*(a, i + 1, hight);

}

归并排序:

*mergeSort*(numbers,0,numbers.length-1);

*printList*(numbers);

**public** **static** **void** merge(**int**[] a, **int** low, **int** mid, **int** high) {

**int**[] temp = **new** **int**[high - low + 1];

**int** i = low;// 左指针

**int** j = mid + 1;// 右指针

**int** k = 0;

// 把较小的数先移到新数组中

**while** (i <= mid && j <= high) {

**if** (a[i] < a[j]) {

temp[k] = a[i];

k++;

i++;

} **else** {

temp[k] = a[j];

k++;

j++;

}

}

// 把左边剩余的数移入数组

**while** (i <= mid) {

temp[k] = a[i];

k++;

i++;

}

// 把右边边剩余的数移入数组

**while** (j <= high) {

temp[k] = a[j];

k++;

j++;

}

// 把新数组中的数覆盖nums数组

**for** (**int** k2 = 0; k2 < temp.length; k2++) {

a[k2 + low] = temp[k2];

}

}

**public** **static** **void** mergeSort(**int**[] a, **int** low, **int** high) {

**int** mid = (low + high) / 2;

**if** (low < high) {

// 左边

*mergeSort*(a, low, mid);

// 右边

*mergeSort*(a, mid + 1, high);

// 左右归并

*merge*(a, low, mid, high);

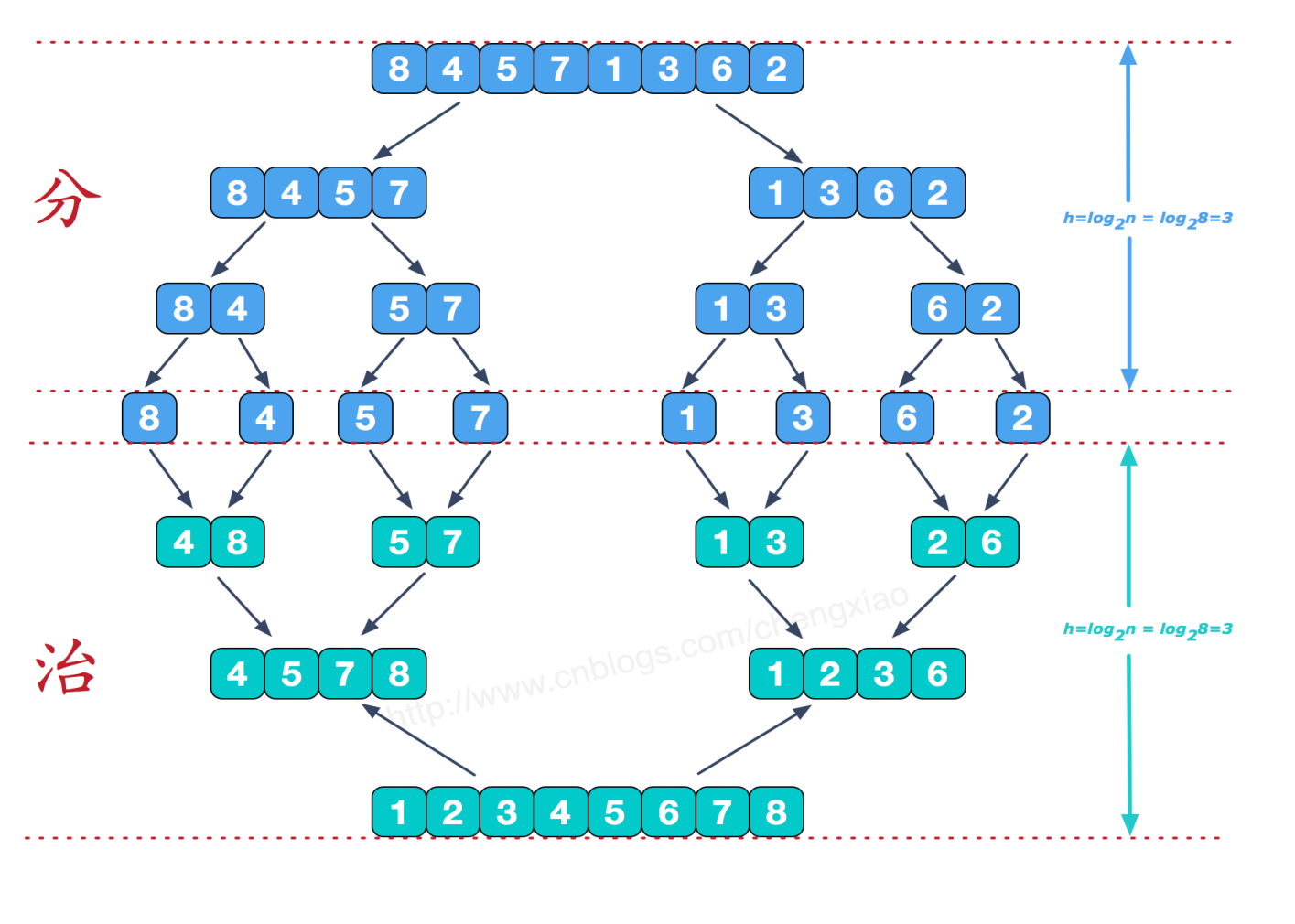
}

}

# **基本思想**

　　归并排序（MERGE-SORT）是利用**归并**的思想实现的排序方法，该算法采用经典的**分治**（divide-and-conquer）策略（分治法将问题**分**(divide)成一些小的问题然后递归求解，而**治(conquer)**的阶段则将分的阶段得到的各答案"修补"在一起，即分而治之)。

**分而治之**



 　　可以看到这种结构很像一棵完全二叉树，本文的归并排序我们采用递归去实现（也可采用迭代的方式去实现）。**分**阶段可以理解为就是递归拆分子序列的过程，递归深度为log2n。

# **合并相邻有序子序列**

　　再来看看**治**阶段，我们需要将两个已经有序的子序列合并成一个有序序列，比如上图中的最后一次合并，要将[4,5,7,8]和[1,2,3,6]两个已经有序的子序列，合并为最终序列[1,2,3,4,5,6,7,8]，来看下实现步骤。

