QuickSort

# DivideConquer分治法：

**快速排序**使用**分治法（Divide and conquer）策略**来把一个**串行（list）**分为两个子串行（sub-lists）。

# 快速排序：

## 空间复杂度：

在平均状况下，排序 n 个项目要**Ο(n log n)次**比较。在最坏状况下则需要**Ο(n2)**次比较，但这种状况并不常见。事实上，快速排序通常明显比其他Ο(n log n) 算法更快，因为它的内部循环（inner loop）可以在大部分的架构上很有效率地被实现出来。

## 时间复杂度：

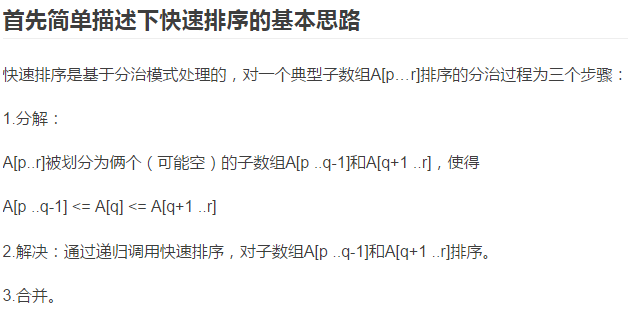
# 算法步骤：

## 从数列中挑出一个元素，称为 “基准”（pivot），

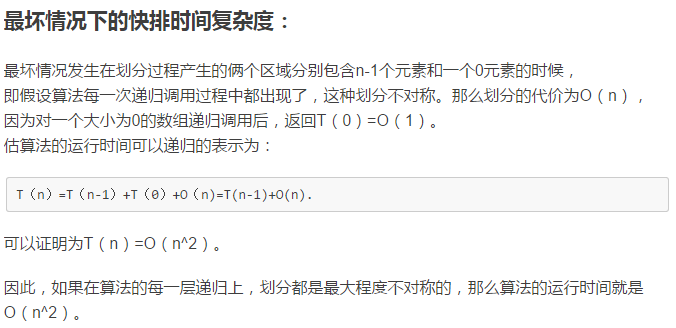
## 重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（相同的数可以到任一边）。在这个分区退出之后，该基准就处于数列的中间位置。这个称为分区（partition）操作。

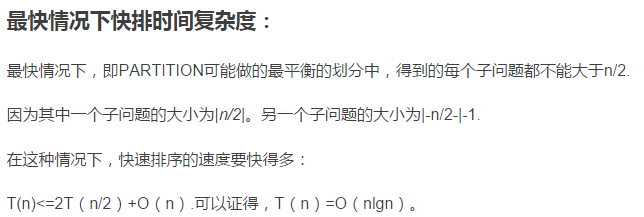
## 递归地（recursive）把小于基准值元素的子数列和大于基准值元素的子数列排序。

# 快速排序的基本思路：



# 算法复杂度：





# C语言优秀代码：

#include <stdio.h>

**//找出基准值正确的索引值，并保证索引左边的小于基准值，索引右边的大于基准值**

**int partition(int \*arr,int low,int high)**

{

int pivot=arr[high];

int i=low-1;

int j,tmp;

for(j=low;j<high;++j)

if(arr[j]<pivot){

tmp=arr[++i];

arr[i]=arr[j];

arr[j]=tmp;

}

**//将大于或等于基准值的索引最小的数据与基准值交换**。

**tmp=arr[i+1];**

**arr[i+1] =arr[high];**

**arr[high]=tmp;**

return i+1;

}

**//采用递归实现**

**void quick\_sort(int \*arr,int low,int high)**

{

if(low<high){

int mid=**partition(arr,low,high);**

quick\_sort(arr,low,mid-1);

quick\_sort(arr,mid+1,high);

}

}

//test

int main()

{

int arr[10]={1,4,6,2,5,8,7,6,9,12};

quick\_sort(arr,0,9);

int i;

for(i=0;i<10;++i)

printf("%d ",arr[i]);

}

# 转换成java代码

**public** **class** QuickSort {

**public** **static** **void** main(String[] args) {

**int**[] a = { 33, 32, 4, 63, 1, 87, 2, 10 };

// 默认以最后一个数为基准值

**for** (**int** i : a) {

System.***out***.print(i + " ");

}

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

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

**for** (**int** i : a) {

System.***out***.print(i + " ");

}

}

**private** **static** **void** **quickSort**(**int**[] a, **int** lowIndex, **int** highIndex) {

**if** (lowIndex < highIndex) {

**int** indexOfPivot = *partition*(a, lowIndex, highIndex);

*quickSort*(a, lowIndex, indexOfPivot - 1);

*quickSort*(a, indexOfPivot + 1, highIndex);

}

}

**private** **static** **int** partition(**int**[] a, **int** minindex, **int** maxindex) {

**int** pivot = a[maxindex];

**int** i = minindex - 1;

**int** j, tmp;

**for** (j = minindex; j < maxindex; j++) {

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

tmp = a[++i];

a[i] = a[j];

a[j] = tmp;

}

**for** (**int** ii : a) {

System.***out***.print(ii + " ");

}

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

}

**int** indexOfPivot = i + 1;

tmp = a[indexOfPivot];

a[indexOfPivot] = a[maxindex];

a[maxindex] = tmp;

**for** (**int** ii : a) {

System.***out***.print(ii + " ");

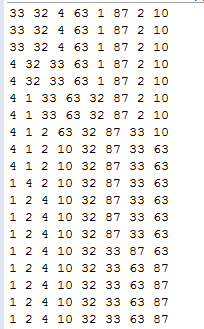
}

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

**return** indexOfPivot;

}

}



递归调用quickSort方法：

**最重要的（核心方法）方法就是partition方法**。

**partition方法**完成的主要功能：

默认最后一个数是基准值，用i记录**小于基准值的个数**，j作为循环索引，从索引0开始到length-2，如果值小于基准值，就将该值与++i值互换位置，如果大于等于基准值，则不操作，注意此时i也不变。

***说白了：就是把大于基准值的数往后赶。***

就这样不断地分成两份，递归下去，到**low=high**时，即只有数组只有1个数时，就退出。

# 掌握的目标：

**能够利用C语言或Java语言，迅速的写出这个快速排序程序。**

**理解记忆技巧：**

## 传入参数：数组名arr、最小索引low、最大索引high。quickSort方法和partition方法的参数都是这三个。不过quickSort没有返回值,partition方法需要返回分界的索引值。

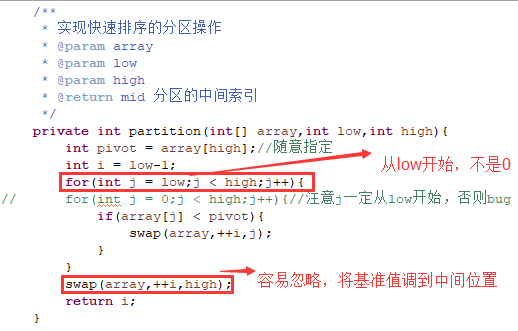
## quickSort方法中就是递归调用。

# 自己实现的Java代码

## 由三个函数完成

核心函数：快速分区partition函数：int partition(int[] array,int low,int high)。

对**数组array**的索引**[low,high]区间**排序，返回分区后的中间索引mid。



## 递归函数：重点找到递归终止条件



## 工具函数：swap

