# 快速排序算法



目录 [隐藏]





## 基本思想

[编辑本段]

快速排序（Quicksort）是对冒泡排序的一种改进。由C. A. R. Hoare在1962年提出。它的基本思想是：通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

## 算法过程

[编辑本段]

设要排序的数组是A[0]……A[N-1]，首先任意选取一个数据（通常选用第一个数据）作为关键数据，然后将所有比它小的数都放到它前面，所有比它大的数都放到它后面，这个过程称为一趟快速排序。一趟快速排序的算法是：

1）设置两个变量I、J，排序开始的时候：I=1，J=N-1；

2）以第一个数组元素作为关键数据，赋值给X，即 X=A[0]；

3）从J开始向前搜索，即由后开始向前搜索（J=J-1），找到第一个小于X的值，让该值与X交换；

4）从I开始向后搜索，即由前开始向后搜索（I=I+1），找到第一个大于X的值，让该值与X交换；

5）重复第3、4步，直到 I=J；

例如：待排序的数组A的值分别是：（初始关键数据：X=49）

A[0] 、 A[1]、 A[2]、 A[3]、 A[4]、 A[5]、 A[6]：

49 38 65 97 76 13 27

进行第一次交换后： 27 38 65 97 76 13 49

( 按照算法的第三步从后面开始找)

进行第二次交换后： 27 38 49 97 76 13 65

( 按照算法的第四步从前面开始找>X的值，65>49,两者交换，此时：I=3 )

进行第三次交换后： 27 38 13 97 76 49 65

( 按照算法的第五步将又一次执行算法的第三步从后开始找

进行第四次交换后： 27 38 13 49 76 97 65

( 按照算法的第四步从前面开始找大于X的值，97>49,两者交换，此时：J=4 )

此时再执行第三步的时候就发现I=J，从而结束一躺快速排序，那么经过一趟快速排序之后的结果是：27 38 13 49 76 97 65，即所以大于49的数全部在49的后面，所以小于49的数全部在49的前面。

快速排序就是递归调用此过程——在以49为中点分割这个数据序列，分别对前面一部分和后面一部分进行类似的快速排序，从而完成全部数据序列的快速排序，最后把此数据序列变成一个有序的序列，根据这种思想对于上述数组A的快速排序的全过程如图6所示：

初始状态 {49 38 65 97 76 13 27}

进行一次快速排序之后划分为 {27 38 13} 49 {76 97 65}

分别对前后两部分进行快速排序 {27 38 13} 经第三步和第四步交换后变成 {13 27 38} 完成排序。

{76 97 65} 经第三步和第四步交换后变成 {65 76 97} 完成排序。

## 变种算法 [编辑本段]

快速排序(Quicksort)有三个值得一提的变种算法，这里进行一些简要介绍：

平衡快排（Balanced quicksort）：每次尽可能地选择一个能够代表中值的元素作为关键数据，然后遵循普通快排的原则进行比较、替换和递归。 外部快排（External quicksort）：与普通快排不同的是，关键数据是一段buffer，首先将之前和之后的M/2个元素读入buffer并对该buffer中的这些元素进行排序，然后从被排序数组的开头（或者结尾）读入下一个元素，假如这个元素小于buffer中最小的元素，把它写到最开头的空位上；假如这个元素大于buffer中最大的元素，则写到最后的空位上；否则把buffer中最大或者最小的元素写入数组，并把这个元素放在buffer里。保持最大值低于这些关键数据，最小值高于这些关键数据，从而避免对已经有序的中间的数据进行重排。完成后，数组的中间空位必然空出，把这个buffer写入数组中间空位。然后递归地对外部更小的部分，循环地对其他部分进行排序。 **三路基数快排（Three-way radix quicksort，也称作multikey quicksort、multi-key quicksort）**：结合了基数排序（radix sort，如一般的字符串比较排序就是基数排序）和快排的特点，是字符串排序中比较高效的算法。该算法被排序数组的元素具有一个特点，即multikey，如一个字符串，每个字母可以看作是一个key。算法每次在被排序数组中任意选择一个元素作为关键数据，首先仅考虑这个元素的第一个key（字母），然后把其他元素通过key的比较分成小于、等于、大于关键数据的三个部分。然后递归地基于这一个key位置对“小于”和“大于”部分进行排序，基于下一个key对“等于”部分进行排序。

## Pascal的qsort [编辑本段]

free Pascal自带的qsort标程:

```
program quicksort;
const
{$ifndef MACOS}
max = 100000;
{$else}
max = 1000; {Actually it works with 100000 also, but that might }
{lead problems occacionally.}
{$endif}
type
tlist = array[1..max] of longint;
var
data : tlist;
procedure qsort(var a : tlist);
procedure sort(l,r: longint);
var
i,j,x,y: longint;
begin
i:=l;
j:=r;
x:=a[(l+r) div 2];
repeat
while a[i]<x do
inc(i);
while x<a[j] do
dec(j);
if not(i>j) then
begin
y:=a;
a:=a[j];
a[j]:=y;
```

```
inc(i);
j:=j-1;
end;
until i>j;
if l<j then
sort(l,j);
if i<r then
sort(i,r);
end;
begin
sort(1,max);
end;
var
i : longint;
begin
write('Creating ',Max,' random numbers between 1 and 500000');
randomize;
for i:=1 to max do
data:=random(500000);
writeln;
writeln('Sorting...');
qsort(data);
writeln;
for i:=1 to max do
begin
write(data:7);
if (i mod 10)=0 then
writeln;
end;
end.
#include <iostream>
using namespace std;
void run(int* pData,int left,int right)
{
int i,j;
int middle,iTemp;
i = left;
j = right;
//middle = pData[(left+right)/2]; //求中间值
middle = pData[(rand() % (right+1))]; //最好取随机数
do{
while( ( pData[i] < middle) && (i < right))//从左扫描大于中值的数
i++;
while((pData[j]>middle) && (j>left))//从右扫描大于中值的数
j--;
if(i<=j)//找到了一对值
{
```

```
//交换
iTemp = pData[j];
pData[j] = pData[i];
pData[i] = iTemp;
i++;
j--;
}
}while(i<=j);//如果两边扫描的下标交错，就停止（完成一次）
//当左边部分有值(left<j)，递归左半边
if(left<j)
run(pData,left,j);
//当右边部分有值(right>i)，递归右半边
if(right>i)
run(pData,i,right);
}
void QuickSort(int* pData,int Count)
{
run(pData,0,Count-1);
}
void main()
{
int data[] = {10,9,8,7,6,5,4};
QuickSort(data,7);
for (int i=0;i<7;i++)
cout << data[i] <<" ";
cout<<endl;
}
```

简洁实用的标程：

```
program qsort;
var
i,n : longint;
a : array[1..100000] of longint;
procedure qsort(i,j : longint);
var
i1,j1 : longint;
t : longint;
mid : longint;
begin
i1 := i;
j1 := j;
mid := a[(i+j) shr 1] ;
while i <= j do
begin
while a[i] < mid do inc(i);
while a[j] > mid do dec(j);
if i <= j then
begin
```

```
t := a[i];
a[i] := a[j];
a[j] := t;
inc(i);
dec(j);
end;
end;
if i1 < j then qsort(i1,j);
if i < j1 then qsort(i,j1);
end; //qsort
begin
readln(n);
for i := 1 to n do read(a[i]);
qsort(1,n);
for i := 1 to n do write(a[i],' ');
end.
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import java.util.Random;
public class QuickSort {
public static void main(String[] args) {
QuickSort q = new QuickSort();
Map<Integer, Integer> map = new HashMap<Integer, Integer>();
int i = 0;
Random rand = new Random();
while(map.size() < 8){
//int num = ((int)(Math.random()*100))%100;
int num = (int)(rand.nextDouble()*8)+1;
map.put(num, num);
}
Object[] m = map.values().toArray();
q.quickSort(m,0, m.length-1);
}
/**
* 快速排序主循环体，主要利用一个可变数组list
* 把作为参考的基准数先放到list中，然后跟数组一段区间内的数
* 比较，大的放到list末尾，小的就放到这个基准数的左边
* @param i 数组区间的起始坐标
* @param j 数组区间的终止坐标
*/
public void quickSort(Object[] m,int i, int j){
int t = m[i].hashCode(); //t 记录一个用来做参考的数
ArrayList<Integer> temp = new ArrayList<Integer>();
temp.add(t);
int length = j + 1 - i; //记录i----j总共是多少个数字
int start = i++; //记录i的起始坐标
```

```
int left = 0; //给temp里面填数字时候的坐标,也就是小于t的数值的索引
int right = length -1;
/**
* 判断temp数组当前的左边left和右边坐标right是否重叠
* 如果重叠就表示区间内所有数据都按一定的顺序放到temp中
*/
while(left < right){
if(t < m[j].hashCode()){
//将此数字加到末尾
temp.add(m[j].hashCode()) ;
right--;
}else{
//在temp中放到t的前面
temp.add(left, m[j].hashCode());
left++;
}
if(left == right){
break;
}
if(t < m[i].hashCode()){
temp.add(m[i].hashCode());
right--;
}else{
temp.add(left, m[i].hashCode());
left++;
}
i += 1;
j -= 1;
}
//到此时temp里面的顺序就排成了左边都小于t，右边都大于t
for(int k = 0; k < temp.size(); k++){
m[start + k] = temp.get(k);
//out(m);
}
//以t为分界线再分别对左边和右边进行划分，这样逐级进行最终将分成若干对最小的两个数值比较
//t的左边在原数组里的范围
if(left > 1){
quickSort(m,start, start + left -1);
//out(m);
}
//t的右边在原数组里的范围
if(left + 1 < length -1){
quickSort(m,start + left + 1, start + length -1);
//out(m);
}
}
public void out(Object[] m){
```

```
for(int i = 0; i < m.length; i++){
System.out.print(m[i] + " ");
}
System.out.println();
}
}
```

### c++中的QSORT [编辑本段]

```
template<typename BidirectionalIterator, typename Compare>
void quick_sort(BidirectionalIterator first, BidirectionalIterator last, Compare cmp)
{
if (first != last)
{
typedef typename iterator_traits<BidirectionalIterator>::value_type value_type;
value_type tmp = *left;
BidirectionalIterator left = first;
BidirectionalIterator right = last;
while (left != right)
{
while (left != right && cmp(tmp, *right))
{
right--;
}
*left = *right;
while (left != right && cmp(*left, tmp))
{
left++;
}
*right = *left;
}
*left = tmp;
quick_sort(first, left--, cmp);
quick_sort(left++, last, cmp);
}
}
template<typename BidirectionalIterator>
void quick_sort(BidirectionalIterator first, BidirectionalIterator last)
{
quick_sort(first, last, std::less_equal< typename iterator_traits<BidirectionalIterator>::value_type >() );
}
```

### VB中的Qsort [编辑本段]

```
'快速排序算法，对字符串数组进行排序
Private Sub quicksort(ByRef arrValue() As String, ByVal intLx As Integer, ByVal intRx As Integer)
```

```
'arrValue()是待排的数组，intLx,intRx为左右边界
Dim strValue As String
Dim I As Integer
Dim j As Integer
Dim intLoop As Integer
I = intLx
j = intRx
Do
While arrValue(I) <= arrValue(j) And I < j: I = I + 1: Wend
If I < j Then
strValue = arrValue(I)
arrValue(I) = arrValue(j)
arrValue(j) = strValue
End If
While arrValue(I) <= arrValue(j) And I < j: j = j - 1: Wend
If I < j Then
strValue = arrValue(I)
arrValue(I) = arrValue(j)
arrValue(j) = strValue

End If
Loop Until I = j
I = I - 1: j = j + 1

If I > intLx Then
Call quicksort(arrValue, intLx, I)
End If
If j < intRx Then
Call quicksort(arrValue, j, intRx)
End If
End Sub
Private Sub Form_Load()
Dim arr(8) As String
arr(0) = "r"
arr(1) = "e"
arr(2) = "a"
arr(3) = "n"
arr(4) = "b"
arr(5) = "u"
arr(6) = "c"
arr(7) = "o"
arr(8) = "f"
Call quicksort(arr, 0, UBound(arr))
End Sub
```

**Java中的Qsort** [编辑本段]

```
public class QuickSort {
/**
* 快速排序
*/
public static void main(String[] args) {
Random random=new Random();
int[] pData=new int[10];
for(int i=0;i<pData.length;i++){ //随机生成10个排序数
Integer a =random.nextInt(100);
pData[i]= a;
System.out.print(pData[i]+" ");
}
System.out.println();
int left=0;
int right=pData.length-1;
Sort(pData,left,right);
for(int i=0;i<pData.length;i++){
System.out.print(pData[i]+" ");
}
System.out.println();
}
public static int[] Sort(int[] pData, int left, int right){
int middle,strTemp;
int i = left;
int j = right;
middle = pData[(left+right)/2];
do{
while((pData[i]<middle) && (i<right))
i++;
while((pData[j]>=middle) && (j>left))
j--;
if(i<=j){
strTemp = pData[i];
pData[i] = pData[j];
pData[j] = strTemp;
i++;
j--;
}
for(int k=0;k<pData.length;k++){
System.out.print(pData[k]+" ");
}
System.out.println();
}while(i<j);//如果两边扫描的下标交错，完成一次排序
if(left<j)
Sort(pData,left,j); //递归调用
if(right>i)
```

```
Sort(pData,i,right); //递归调用
return pData;
}
}
C#中的Qsort
static void qsort(int[] a, int left, int right)
{
int l, r, pivot, temp;
l = left;
r = right;
pivot = a[(l + r) / 2];
while (l < r)
{
while (a[l] < pivot) ++l;
while (a[r] > pivot) --r;
if (l >= r) break;
temp = a[l];
a[l] = a[r];
a[r] = temp;
if (a[l] != pivot) ++l;
if (a[r] != pivot) --r;
}
if (l == r) ++l;
if (left < r) qsort(a, left, l - 1);
if (l < right) qsort(a, r + 1, right);
}
```



**扩展阅读：**

1.http://www.cnblogs.com/clive/archive/2009/08/13/three_variants_of_quicksort.html

**相关词条：** [我来完善]

冒泡法 起泡法

**开放分类：**

计算机，算法

**合作编辑者：** 更多

manxian001、solazm、beakore、lighting_cui、JohnLocker、mornsunrain、yancong008、_风过无痕、alps_goal、cheng_ming