### **一、快速排序思想**

快速排序的思想，是找出一个中轴（pivot），之后进行左右递归进行排序，关于递归快速排序，C程序算法如下。

void quick\_sort(int \*arr,int left,int right){  
   if(left>right) return;  
   int pivot=getPivot();  
   quick\_sort(arr,left,pivot-1);  
   quick\_sort(arr,pivot+1,right);  
}

### **二、划分思想**

关于划分，不同的划分决定快排的效率，下面以lomuto划分和hoare划分来进行讲述思路

##### **1.lomuto划分**

思想：lomuto划分主要进行一重循环的遍历，如果比left侧小，则进行交换。然后继续进行寻找中轴。最后交换偏移的数和最左侧数，C程序代码如下。

/\*\*lomuto划分\*/  
int lomuto\_partition(int \*arr,int l,int r){  
   int p=arr[l];  
   int s=l;  
   for(int i=l+1;i<=r;i++)  
       if(arr[i]<p) {  
           s++;  
           int tmp=arr[i];  
           arr[i]=arr[s];  
           arr[s]=tmp;  
       }  
   int tmp=arr[l];  
   arr[l]=arr[s];  
   arr[s]=tmp;  
   return s;  
}

##### **2.hoare划分**

思想：hoare划分思想是先从右侧向左进行寻找，再从左向右进行寻找，如果左边比右边大，则左右进行交换。外侧还有一个嵌套循环，循环终止标志是一重遍历，这种寻找的好处就是，在一次遍历后能基本有序，减少递归的时候产生的比较次数。这也是经典快排中所使用的方法

/\*\*hoare划分\*/  
int hoare\_partition(int \*a,int l, int r) {  
   int p = a[l];  
   int i = l-1;  
   int j = r+1 ;  
   while (1) {  
       do {  
           j--;  
       }while(a[j]>p);  
       do {  
           i++;  
       }while(a[i] < p);  
       if (i < j) {  
           int temp = a[i];  
           a[i] = a[j];  
           a[j] = temp;  
       }else  
           return j;  
   }  
}

##### **3.经典快排的划分改进**

经典快排实际对hoare划分进行了少许改进，这个temp变量不需要每次找到左右不相等就立即交换，而是，暂时存放，先右边向左找，将左边放在右边，再左边向右找，把右边放左边，最后把初始temp变量放在左值。这样比hoare划分减少少许移动变量次数。

/\*\*经典快排\*/  
int classic\_quick\_sort(int \*arr,int left,int right){  
   int tmp=arr[left];  
   while(left<right){  
       while(left<right&&arr[right]>=tmp) right--;  
       arr[left]=arr[right];  
       while(left<right&&arr[left]<=tmp) left++;  
       arr[right]=arr[left];  
   }  
   arr[left]=tmp;  
   return left;  
}

##### **4.源码补充(Java源码)**

在Java或者C#源码中，Arrays.sort由多个排序算法构成，比如，数据量不大，使用双轴快排(dualPivotQuickSort)，数量巨大，使用归并排序(merge sort)，中间的先检测下数据是否基本有序等特征，再使用相应的排序算法。

双轴快排思想：总体思路是找出2个轴心。

我仅仅把选轴的部分进行挑出来进行将述，首先选定2个轴，L轴和R轴，使用i保存左值，j保存右值。首先从左向右边找，如果比pivot1大，进行左值和偏移的自增，并且交换左值和偏移。如果在pivot1和pivot2之间，就直接继续循环。循环中，如果比pivot大，那么从右往左边找，如果值比pivot2大，那么进行跳出到OUT\_LOOP的位置，如果在pivot1和pivot2之间，与pivot2交换，如果比pivot2小，交换j和左值，左值和右值。

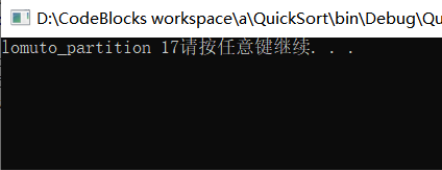
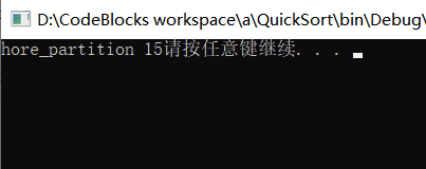
dualPivot(int[] A,int L,int R){  
   int pivot1 = A[L];  
   int pivot2 = A[R];  
   int i = L;  
   int k = L+1;  
   int j = R;  
   OUT\_LOOP:  
   while(k < j){  
       if(A[k] < pivot1){  
           i++;  
           Swap(A, i, k);  
           k++;  
       }else  
       if(A[k] >= pivot1 && A[k] <= pivot2){  
           k++;  
       }else{  
           while(A[--j] > pivot2){  
               if(j <= k){  
                   break OUT\_LOOP;  
               }  
           }  
           if(A[j] >= pivot1 && A[j] <= pivot2){  
               Swap(A, k, j);  
               k++;  
           }else{  
               i++;  
               Swap(A, j, k);  
               Swap(A, i, k);  
               k++;  
           }  
       }  
   }  
   Swap(A, L, i);  
   Swap(A, R, j);  
   }  
}

### **三、测试用例**

关于测试，我使用C程序的<time.h>中的clock函数进行测试。测试代码如下，数据量100'000：

int main()  
{  
   int data[100000];  
   srand((int)time(0));  
   for(int i=0;i<100000;i++){  
       data[i]=rand();  
   }  
   clock\_t start,end;  
   start=clock();  
   quick\_sort(data,0,sizeof(data)/sizeof(int)-1);  
   end=clock();  
   printf("hore\_partition %ld\n",(end-start));  
   system("pause");  
   return 0;  
}

我们进行测试10次左右，发现结果如下图所示：



结论：10w个数据进行排序，使用hore划分排序约14~15ms。使用lomuto划分排序约17~18ms左右，经典快排和lomuto的时间几乎一致。