推荐大家使用`qsort`而非手写快排的那点事

如有对本文档有任何疑问，请联系`1706zcy@buaa.edu.cn`

0.开头语

我们在学习C语言的时候，总会和大家讲到`qsort`函数，这个时候总有些对指针不熟悉，讨厌`qsort`调用的同学，会直接去上网查找手写快速排序的做法。当然，这也不是不可以，但是很多时候，依旧会吃大亏。

在这里，我们深入浅出，给大家简单介绍一下快速排序的原理，以及为什么**推荐大家直接调用`qsort`，而不是**手写快速排序。

1.`qsort`的标准调用形式

以下以给`int`数组排序作为规范实例:

```
#include<stdio.h>
#include<stdlib.h>
int a[200010], n;
int cmp(const void* p1, const void* p2) {
    int* a = (int*)p1, * b = (int*)p2;
    if (*a < *b)return -1;//a要排在b前面时，返回负数值
    else if (*a > *b) return 1;//a要排在b后面时，返回正数值
    else return 0;//不需要变动时，返回0
}
int main() {
    int i;
    scanf("%d", &n);
    for (i = 1; i <= n; i++) scanf("%d", &a[i]);
    qsort(a + 1, n, sizeof(a[0]), cmp);
    for (i = 1; i <= n; i++)printf("%d ", a[i]);
    return 0;
}
```

2.快速排序的原理

快速排序之所以叫做快速排序，是因为在冒泡和选择排序对长度为 n 的数组排序时间复杂度 $O(n^2)$ 时，快速排序**的平均速度**为 $O(nlogn)$

- 当 n 为 500000 时，**冒泡排序需要大概 13 秒才能完成运算**
- **此时快速排序却可以轻松在 1 秒之内完成**

以下内容摘取并更改自邓俊辉《数据结构》

> 快速排序由英国计算机科学家、1980年图灵奖得主C. A. R. Hoare爵士于1960年发明
>
> ### 轴点

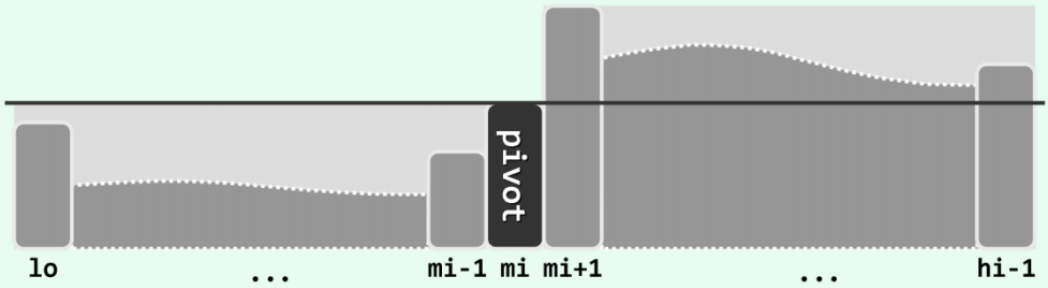


图12.1 序列的轴点（这里用高度来表示各元素的大小）

关于轴点的性质：

- 轴点的元素，**在数组有序之后，依旧会排在这里**
- 轴点左方的元素**一定都会比自己小**
- 轴点右方的元素**一定都会比自己大**

由此，快速排序的伪代码即可得到

```
template <typename T> //向量快速排序
void Vector<T>::quickSort ( Rank lo, Rank hi ) { //0 <= lo < hi <= size
    if ( hi - lo < 2 ) return; //单元素匙间自然有序，否则...
    Rank mi = partition ( lo, hi - 1 ); //在[lo, hi - 1]内极造轴点
    quickSort ( lo, mi ); //对前缀递弼排序
    quickSort ( mi + 1, hi ); //对后缀递弼排序
}
```

需要注意，上述伪代码是参照`C++`语法完成的。

轴点与划分过程

可见，算法的主体框架为循环迭代；主循环的内部，通过两轮迭代交替地移动lo和hi。

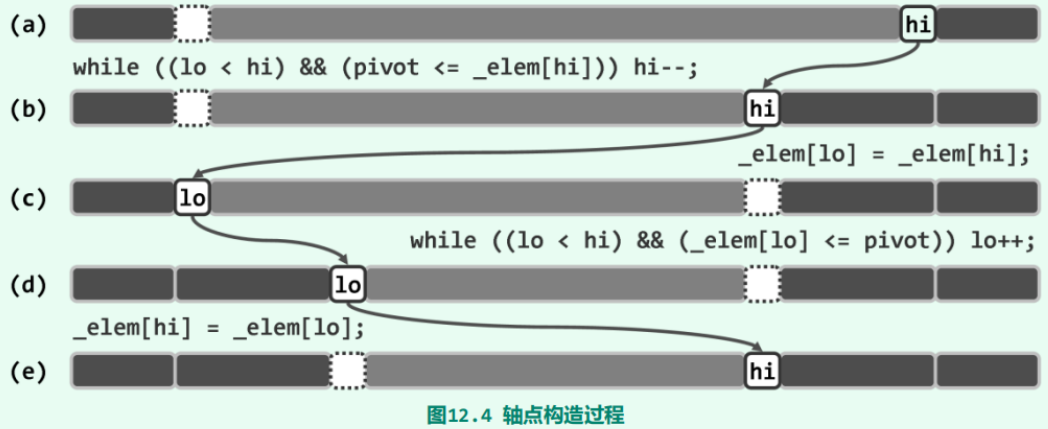


图12.4 轴点构造过程

平均复杂度运算

具体过程太长，这个东西**不需要信息类与航类的大一同学了解，只需要知道结果即可**

$$T(n) = O(2\ln 2\log_2 n) = O(1.386\log_2 n)$$

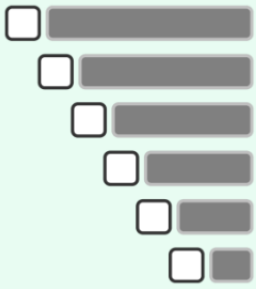
正因为其良好的**平均**性能，才会收到人们的青睐，并被集成到`Linux`和`STL`等环境当中

3.来讲一讲为什么随便在网上copy的快速排序代码不靠谱

这个原因有很多，比如说网上的代码不一定能经过大量的评测，或者胡写一通也有可能...

但是最关键的问题在于，大多数的代码没有办法处理退化的最差情况。

以下内容摘取并更改自邓俊辉《数据结构》

> ■ 重复元素
>
> 
>
> 图12.6 partition()算法的退化情况，也是最坏情况
>
> 考查所有（或几乎所有）元素均重复的退化情况。对照代码12.2不难发现，partition()算法的版本A对此类输入的处理完全等效于此前所举的最坏情况。事实上对于此类向量，主循环内部前一子循环的条件中“pivot <= _elem[hi]”形同虚设，故该子循环将持续执行，直至“lo < hi”不再满足。当然，在此之后另一内循环及主循环也将随即结束。
>
> 如图12.6所示，如此划分的结果必然是以最左端元素为轴点，原向量被分为极不对称的两个子向量。更糟糕的是，这一最坏情况还可能持续发生，从而使整个算法过程等效地退化为线性递归，递归深度为$O(n)$，导致总体运行时间高达$O(n^2)$。

如果上面一大串看不懂，**只要知道以下结论就行**

网上摘抄的很多代码，在最极端的情况下，最终表现和冒泡、选择是差不多的

其运算量都是 $O(n^2)$

4.致那些实在是特别讨厌调用`qsort`的学生

如何正确地使用快速排序并且逃过极端情况的制裁?

具体的原理概括为几句话

- 极端情况基本出现在**大量雷同数据的情况。**
- 如果 n 个数完全相同，那网上一般能找到的快排代码一定无法逃避 $O(n^2)$ 的命运。
- 既然如此，就只针对这种情况进行优化就行了。
- 针对轴点划分方式优化，可以**强制将这种极端情况转换为最快的情况**，强制修正为最快的 $O(nlogn)$
- 再加上随机数进行进一步优化。

于是就有如下代码：

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>
#define maxn 200010
//快速排序 DUP版
void swap(int* a, int* b) {
    (*a) ^= (*b), (*b) ^= (*a), (*a) ^= (*b);
}
int a[maxn];
int n;
//从[lo,hi]当中随机选取对应的轴点mi,并且根据DUP策略调整
/*****************************************
DUP针对LUG的优化主要是以下几点
优化处理多关键码雷同的退化情况
出现相等的时候,lo和hi交替移动
从LGU的勤于拓展懒于交换
变成懒于拓展勤于交换
会在交换上耗费时间,并加剧排序的不稳定性
*****************************************/
```

```
int partition_DUP(int lo, int hi) {
    //任意选取一个元素与a[lo]交换
    int tmp = lo + rand() % (hi - lo + 1);
    if(tmp != lo)swap(&a[lo], &a[tmp]);
    //以首元素作为候选节点 等价于随机选取轴点+腾出空间
    int pivot = a[lo];
    while (lo < hi) {
        while (lo < hi) {
            //大于的前提下拓展P区
            if (a[hi] > pivot)
                hi--;
            //否则将他划归到D区并停止循环
            else { a[lo++] = a[hi]; break; }
        }
        while (lo < hi) {
            //小于的前提下拓展D区
            if (a[lo] < pivot)
                lo++;
            //否则将他划归到P区并停止循环
            else { a[hi--] = a[lo]; break; }
        }
    }
    //出来的时候 lo==hi U区的唯一元素就是培养完成的节点该在的位置了
    a[lo] = pivot;
    return lo;
}
void quickSort(int lo, int hi) {
    //退化情况:单元素的区间必然有序 直接返回
    if (hi - lo < 2)return;
    //外部接口是左闭右开的,但是划分需要传入的是闭区间
    //划分的元素保证排序前和排序后所处位置一直,解决2个子问题之后直接停机
    int mi = partition_DUP(lo, hi - 1);
    quickSort(lo, mi);
    quickSort(mi + 1, hi);
}
int main() {
    srand((int)time(NULL));
    while (scanf("%d", &n) != EOF) {
        for (int i = 0; i < n; ++i)
            scanf("%d", a + i);
        quickSort(0, n);
        for (int i = 0; i < n; ++i)
            printf("%d ", a[i]);
        puts("");
    }
}
```

需要注意的是，这种做法的性能**和上面的`qsort`函数几乎一致。**

准确的说，`qsort`基本上也是基于这种方法实现的

不讲分析，**只讲结论**

这种快速排序写法可以可靠稳定地在 $O(nlogn)$ 的时间内解决问题，不会超时

只有几万亿分之一的概率会退化为 $O(n^2)$，但是这属于快速排序本身的特性，不影响使用

有没有其他优雅且快速的排序方法?

有，会用就行，**但是码量都不算小，考场上写不完后果自负**

归并排序

```
#include<stdio.h>
typedef long long ll;
int a[1000010], b[1000010];
int n, i;
int sum;
ll result;
void merge(int lo, int hi) {
    if (hi > lo) {
        int mi = (hi + lo) >> 1;
        merge(lo, mi), merge(mi + 1, hi);
        int bmark = lo, lomark = lo, himark = mi + 1;
        while (lomark <= mi || himark <= hi) {
            if ((himark > hi || a[lomark] <= a[himark]) && lomark <= mi)
                b[bmark++] = a[lomark++];
            else
                b[bmark++] = a[himark++], result += mi - lomark + 1;
        }
        for (int i = lo; i <= hi; ++i) a[i] = b[i];
    }
}
int main() {
    while (scanf("%d", &n) != EOF) {
        for (i = 0; i < n; ++i)scanf("%d", &a[i]);
        merge(0, n - 1);
        for (i = 0; i < n; ++i)printf("%d ", a[i]);
        putchar('\n');
    }
}
```

堆排序

```
#include<stdio.h>
#define maxn 200010
int n;
int a[maxn];
void swap(int* a, int* b) {
    (*a) ^= (*b), (*b) ^= (*a), (*a) ^= (*b);
}

//堆排序 按照大根堆进行调整
void perlocate_down(int lo, int hi) {
    //下滤操作,堆区间为[lo,hi] lo为堆的根
    int dad = lo;
    int son = (dad << 1) + 1;
    while (son <= hi) {
```

```
        if (son + 1 <= hi && a[son] < a[son + 1])
            son++;//如果存在双分支,选择其中较大者进行下滤
        if (a[dad] > a[son])break;//此时如果堆序性正确,则整个堆都正确,直接停机
        else {
            swap(&a[dad], &a[son]);
            dad = son, son = (dad << 1) + 1;
            //调整,并且根据下滤的节点继续往下看
        }
    }
}
void heap_sort(int lo, int hi) {
    //数组的[lo,hi)当做堆进行排序
    //step1: floyd建堆 自下而上地子堆下滤 复杂度O(n)
    for (int i = ((hi - lo) >> 1) - 1; i >= lo; --i)
        perlocate_down(i, hi - lo - 1);
    //step2: 建堆之后,每次将a[0](即整个堆的最大值)挪到最后 剩余元素做堆下滤调整
    //最快最慢平均复杂度均为O(nlogn) 是就地排序 不稳定排序
    for (int i = (hi - lo - 1); i > 0; --i) {
        swap(&a[0], &a[i]);
        perlocate_down(lo, i - 1);
    }
}
int main() {
    while (scanf("%d", &n) != EOF) {
        for (int i = 0; i < n; ++i)
            scanf("%d", a + i);
        heap_sort(0, n);
        for (int i = 0; i < n; ++i)
            printf("%d ", a[i]);
        puts("");
    }
}
```

希尔排序

```
#include<stdio.h>
#define maxn 200010
int n;
int a[maxn];
//希尔排序
void shellSort(int lo, int hi) {
    int len = hi - lo;
    int h = 1;//增量
    while (h < len / 3)h = 3 * h + 1;//增量之间保持尽可能的互质
    while (h >= 1) {
        for (int i = lo + h; i < hi; ++i) {
            //间隔增量h做插入排序 排序之后整个序列就是h-有序的
            //如果逆序对最大距离不超过k的话,单趟插入排序复杂度为O(kn)
            for (int j = i; j >= h && a[j] < a[j - h]; j -= h) {
                int temp = a[j];a[j] = a[j - h]; a[j - h] = temp;
            }
        }
        h /= 3;
    }
}
int main() {
    while (scanf("%d", &n) != EOF) {
```

```
        for (int i = 0; i < n; ++i)
            scanf("%d", a + i);
        shellSort(0, n);
        for (int i = 0; i < n; ++i)
            printf("%d ", a[i]);
        puts("");
    }
}
```

还有基数排序、计数排序等做法，此处不再赘述。

如果以后不写码，大概就和十大排序无缘了；要写的话，自己去了解自己去写吧。

5.致助教——如何制裁那些在网上随便复制粘贴快排的学生?

一般来说，网上大部分的代码**都处理不了极端的情况。**

所以我们要出数据就十分简单，比如说：

```
100000
500 500 500 500...(10万个500)
```

这样就完事了，保证让那些不懂快排原理精髓，还不愿意自学`qsort`的人无法AC。

什么？你说他会用上面的那些做法过了？人家认真学了那么多，你还制裁他干嘛？

6.节后语

总之就是求求大家...

老老实实学`qsort`,写`qsort`吧!