快排算法及其优化

姓名：陈鸿绪　学号：PB21000224　日期：10.18

算法思想：

1. 普通快排：选取固定基准，该基准在数组右侧，通过调用 partition_1 函数，而后递归调用自己完成普通快排。
2. 随机基准：随机时间种子，partition_2 函数随机获得基准，将基准移动到数组右侧再进行操作，而后调用自身对两个子数列进行递归。
3. 取中基准：partition_3 选择的是数组开头、中间、结尾三个数字，取出中间大小的数作为基准，划分为两个子数组进行递归操作。
4. 几乎有序情况：将数组在被递归划分为小数组时，当子数组的大小小于一定值的时候直接进行插入排序然后再返回。
5. 聚集元素：数组中存在相同元素，对于基准存在相同值的时候，将全部与基准相同的值移动到右侧，再进行交换元素操作，最后对移动基准元素和与之相同的元素到正确位置上，再进行子数列的递归操作。
6. 对如上优化有选择进行组合优化出更好的排序算法。本人优化的最好算法是：取中基准+几乎有序情况+聚集元素三者结合。

算法核心代码：

1. 普通快排：

```
int partition_1(int *a,int p,int r){
    if(p==r) return p;
    int i=p-1;
    for(int *tp=a+p;tp<a+r;++tp){
        if(*tp<*(a+r)){
            swap_1(a+(++i),tp);
        }
    }
```

```
        swap_1(a+r,a+(++i));
        return i;
}//固定基准

void qsort_1(int *a,int p,int r){
        if(p>=r) return;
        int q=partition_1(a,p,r);
        qsort_1(a,p,q-1);
        qsort_1(a,q+1,r);
}//固定基准
```

2. 随机基准：

```
int partition_2(int *a,int p,int r){
        if(p==r) return p;
        int s=rand()%(r-p+1)+p;//选取随机元素
        int i=p-1;
        swap_1(a+s,a+r);
        for(int *tp=a+p;tp<a+r;++tp){
                if(*tp<*(a+r)){
                        swap_1(a+(++i),tp);
                }
        }
        swap_1(a+r,a+(++i));
        return i;
}//随机基准

void qsort_2(int *a,int p,int r){
        if(p>=r) return;
        int q=partition_2(a,p,r);
        qsort_2(a,p,q-1);
        qsort_2(a,q+1,r);
}//随机基准
```

3. 取中基准：

```
int partition_3(int *a,int p,int r){
        int s=mid_three(a,p,r,(r+p)>>1);//三个数进行中间数选取
        int i=p-1;
        swap_1(a+s,a+r);
        for(int *tp=a+p;tp<a+r;++tp){
                if(*tp<*(a+r)){
```

```
            swap_1(a+(++i),tp);
        }
    }
    swap_1(a+r,a+(++i));
    return i;
}//取中基准
void qsort_3(int *a,int p,int r){
    if(p>=r) return;
    int q=partition_3(a,p,r);
    qsort_3(a,p,q-1);
    qsort_3(a,q+1,r);
}//取中基准
```

4. 插排优化：

```
void insert_sort(int *a,int p,int r){
    for(int i=p+1;i<=r;++i){
        for(int *tp=a+i-1;tp>=a+p;--tp){
            if(*(tp+1)<*tp) swap_1(tp+1,tp);
            else break;
        }
    }
}//几乎有序情况，普通插入排序
void qsort_4(int *a,int p,int r,int k){
    if(r-p+1<=k){
        insert_sort(a,p,r);
        return;
    }
    int q=partition_3(a,p,r);//同时使用了三数取中间
    qsort_4(a,p,q-1,k);
    qsort_4(a,q+1,r,k);
}//几乎有序情况+三数取中
```

5. 聚集元素：

```
void partition_4(int *a,int p,int r,int &left,int &right){
    int *ip=a+p-1,*jp=a+p,*tp=a+r-1;
    for(;jp<=tp;jp++){
        if(*jp<a[r]){
            swap_1(++ip,jp);
        }
        else if(*jp==a[r]){
            swap_1(tp--,jp--);
```

```
            }
        }//交换元素同时将与基准相同的元素移动到右边
        left=ip-a;
        for(tp=tp+1;tp<=a+r;tp++){
            swap_1(tp,++ip);
        }
        right=ip-a+1;
    }//选做
    void qsort_5(int *a,int p,int r){
        if(p>=r) return;
        int left,right;
        partition_4(a,p,r,left,right);
        qsort_5(a,p,left);
        qsort_5(a,right,r);
    }//选做
```

6. 聚集元素+插排优化+取中基准：

```
void partition_5(int *a,int p,int r,int *left,int *right){
    int s=mid_three(a,p,r,(r+p)>>1);//三数取中
    swap_1(a+s,a+r);
    int *ip=a+p-1,*jp=ip+1,*tp=a+r-1;
    for(;jp<=tp;++jp){
        if(*jp<*(a+r)){
            swap_1(++ip,jp);
        }
        else if(*jp==*(a+r)){
            swap_1(tp--,jp--);
        }
    }//聚集元素处理
    *left=ip-a;
    for(tp=tp+1;tp<=a+r;++tp){
        swap_1(tp,++ip);
    }
    *right=ip-a+1;
}//聚集+三数取中+插排优化
void qsort_6(int *a,int p,int r,int k){
    if(r-p+1<=k){
        insert_sort(a,p,r);
        return;
    }
    int left,right;
    partition_5(a,p,r,&left,&right);
```

```
        qsort_6(a,p,left,k);
        qsort_6(a,right,r,k);
    }//聚集+三数取中+插排优化
```

实验结果分析:

以下均为循环一百次各类快速排序算法的对比(注:以下单位均为 ms)

1. Stdlib 中的 qsort 函数:

2. 固定基准的快速排序:

3. 随机基准的快速排序:

4. 三数取中的快速排序:

5. 进行插入排序和三数取中优化:

6. 进行聚集基准元素优化:

7. 聚集+三数取中+插排优化

只运行了一次的插入排序算法:

8. 插入排序

对于以上不同算法不同时刻运行时间会有一些起伏波动，但是具体的大体情况与上图展示相同，可见在本电脑配置和编译器情况下，对于给定该数据集合，算法性能大致满足以下：

算法 7>算法 1>算法 5>算法 2>算法 4>算法 6>算法 3>>算法 8

可见对于应用了三个优化的算法 7 是最优的，对于给定的这个数据，性能优于 stdib 自带 qsort 函数，而随机取基准的性能的提升在该数据中并未得到体现，对于普通插入排序只运行一次的速度都明显慢于以上快速排序的 100 次. 可以得出算法 1-7 的时间复杂度量级相同，插入排序时间复杂度明显大于其余 7 个.