两种比较领先的排序思维对比

罗瑶光手稿

今天我有阅读了一种这几年比较流行的快速排序思想，基于中轴分离进行左右分配的数列递归排序法。地址：github 上的 dongxingrong。 我仔细的研究了下，发现有几个地方可取，于是进行评价：

```
public int getMiddle(Integer[] list, int low, int high) {
        int tmp = list[low];    //数组的第一个作为中轴
        while (low < high) {
            while (low < high && list[high] >= tmp) {
                high--;
            }
            list[low] = list[high];     //比中轴小的记录移到低端
            while (low < high && list[low] <= tmp) {
                low++;
            }
            list[high] = list[low]; //比中轴大的记录移到高端
        }
        list[low] = tmp;          //中轴记录到尾
        return low;               //返回中轴的位置
    }
```

我思考了很久，在同频算子减少的方向中，直接减少

temp 的 swap 变量是一种先进的思想。但是利弊交错，我进行了和我的 4 代小高峰过滤快排做了详细对比：

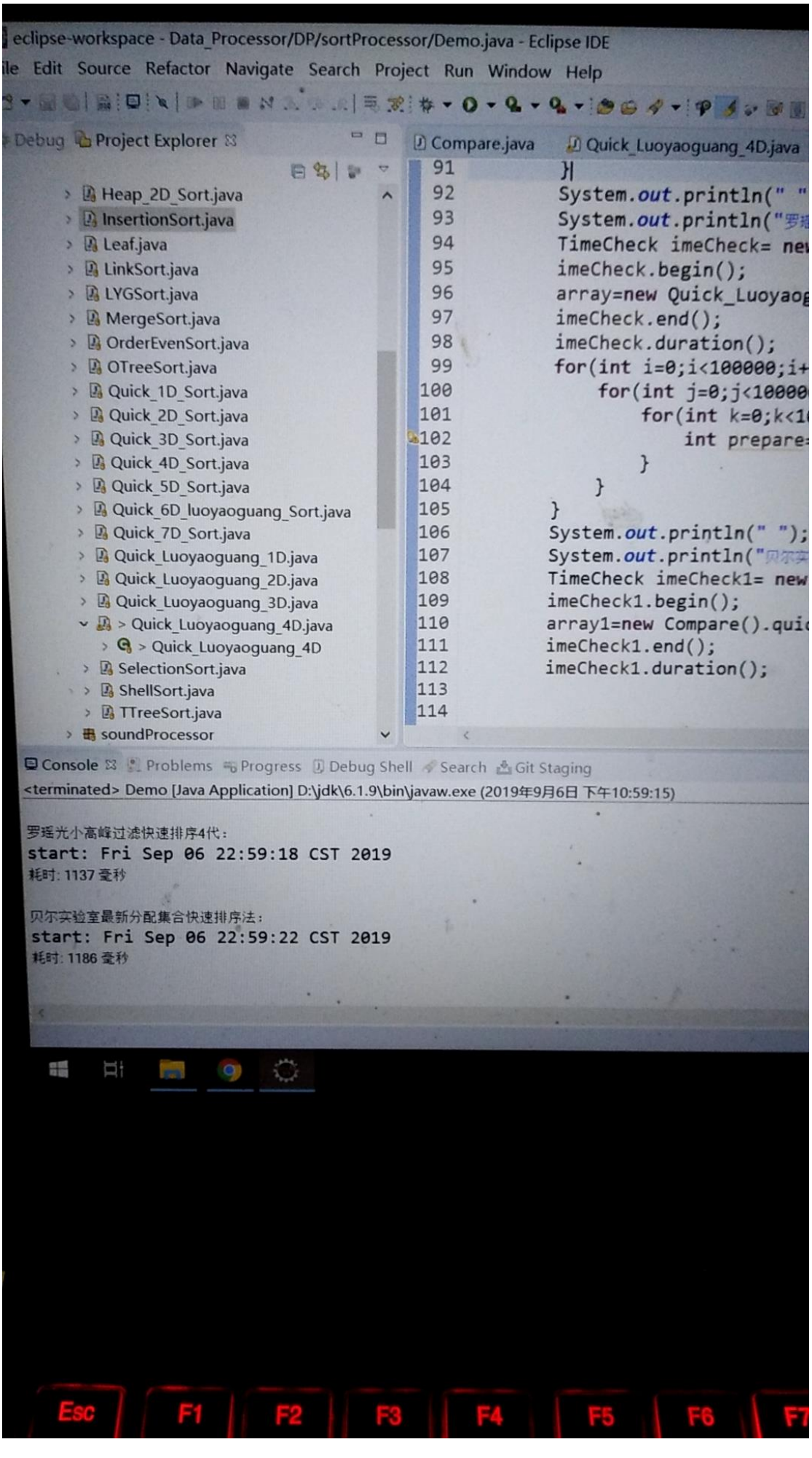
1000 万数列排序	瑶光高峰过滤	xingrong 中轴分配
算子减少	有	有
条件减少	有	有
离散效率	高	中
条件过滤	有	无
高频降解	无	有
计算性能	高	高
代码缩进	有	有
平均高峰先排耗时	1137 毫秒	1186 毫秒
平均分配先排耗时	1072 毫秒	1178 毫秒

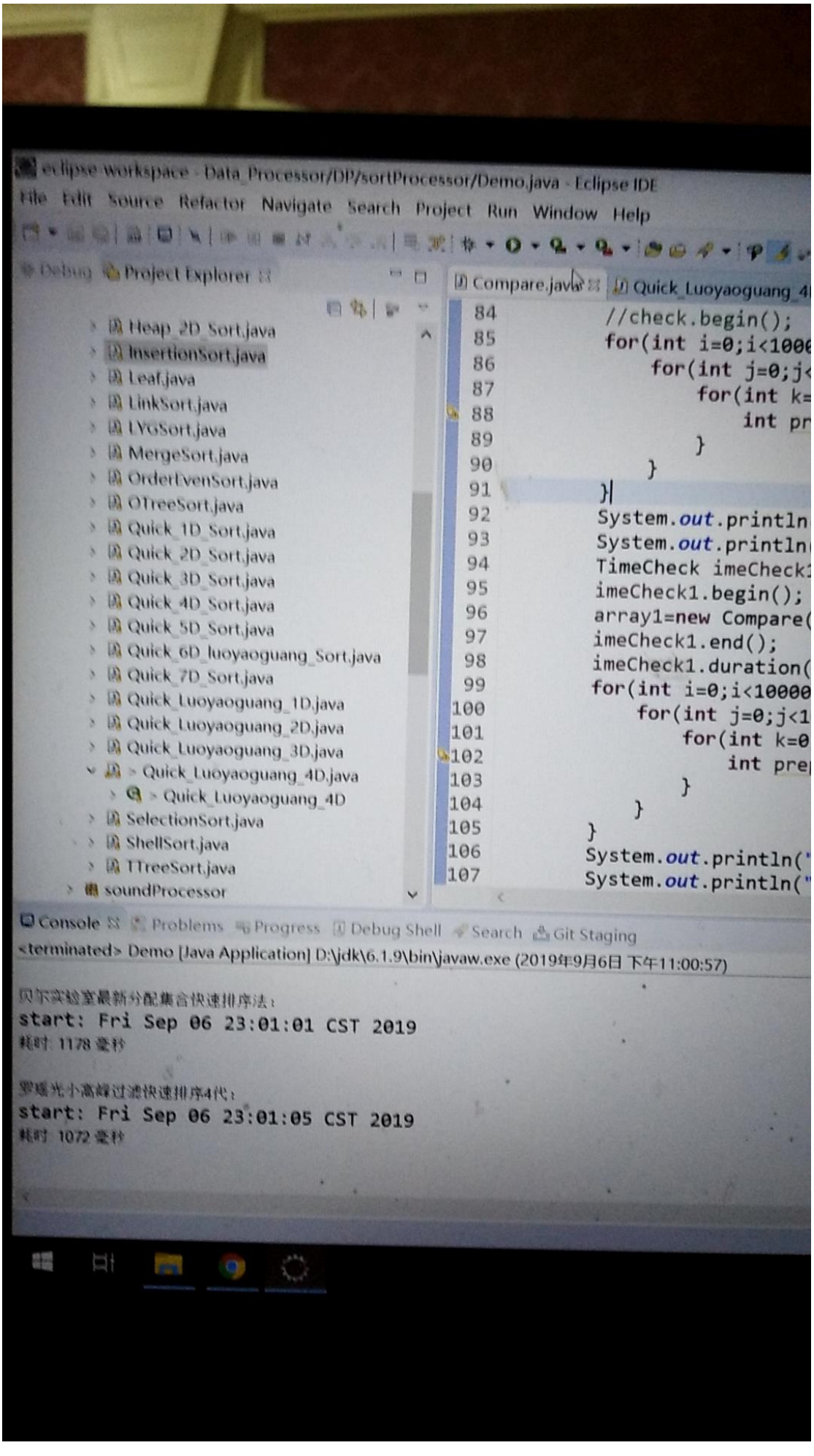
注解：这里的函数效率是指单个函数运行的计算消耗，包括堆栈消耗，内存阻塞，计算耗时等因素。

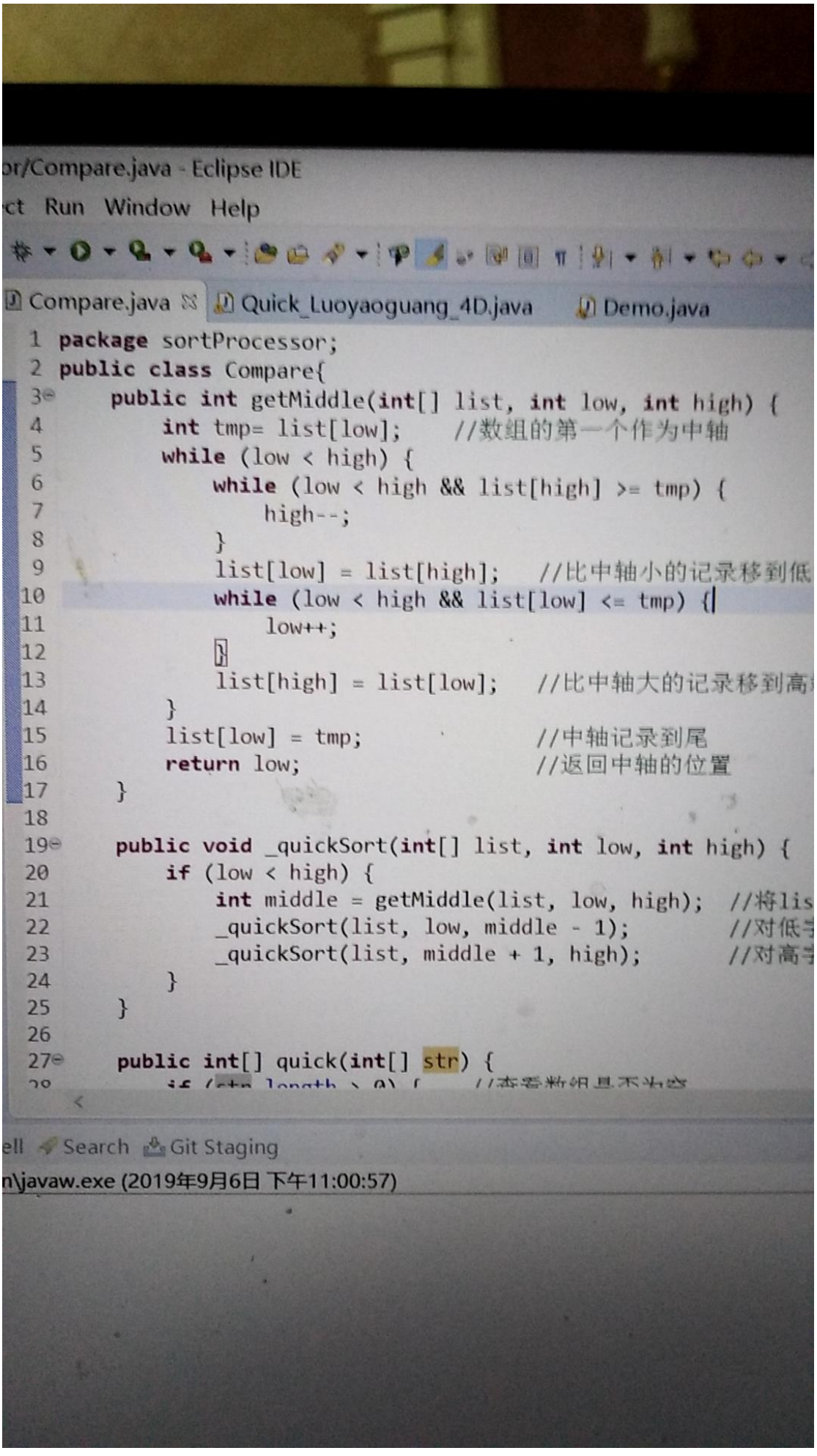
计算性能指的是数组的最大长度，单位消耗的时间，内存大小。

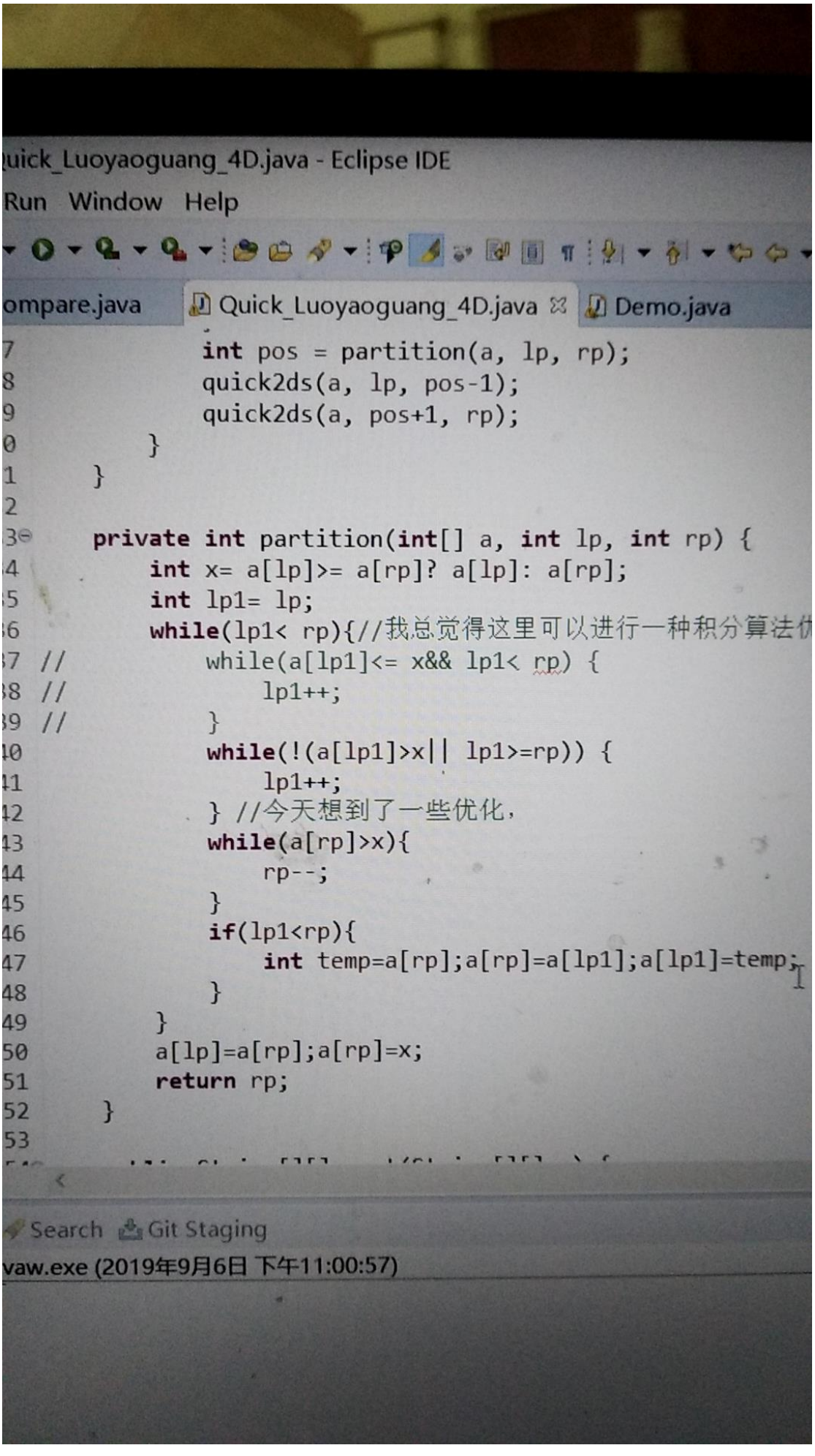
比较可以发现，因为递归的小高峰的存在，中轴心分配所需要的位移变换成为了必要执行功能，增大了开销，这种开销可以最大限度的达到 balanced 递归逻辑的层数，但是对于指数方的多维算子条件缺少了过滤层，所以，性能耗损大。这种排序思想是完美的，为追求完美，丢失了一些东西，比较可惜。但是这种思想需要进行解析，在很多算法应用领域有宝贵的价值。

下面的 4 张图是堆 1000 万个随机数进行排序的比较耗时测试。


eclipse-workspace - Data_Processor/DP/sortProcessor/Demo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Project Explorer
Heap_2D_Sort.java
InsertionSort.java
Leaf.java
LinkSort.java
LYGSort.java
MergeSort.java
OrderEvenSort.java
OTreeSort.java
Quick_1D_Sort.java
Quick_2D_Sort.java
Quick_3D_Sort.java
Quick_4D_Sort.java
Quick_5D_Sort.java
Quick_6D_luoyaoguang_Sort.java
Quick_7D_Sort.java
Quick_Luoyaoguang_1D.java
Quick_Luoyaoguang_2D.java
Quick_Luoyaoguang_3D.java
> Quick_Luoyaoguang_4D.java
> Quick_Luoyaoguang_4D
SelectionSort.java
ShellSort.java
TTreeSort.java
soundProcessor
Compare.java
Quick_Luoyaoguang_4D.java
91 }
92 System.out.println(" "
93 System.out.println("罗
94 TimeCheck imeCheck= ne
95 imeCheck.begin();
96 array=new Quick_Luoyaog
97 imeCheck.end();
98 imeCheck.duration();
99 for(int i=0;i<100000;i+
100 for(int j=0;j<10000
101 for(int k=0;k<1
102 int prepare=
103 }
104 }
105 }
106 System.out.println(" ");
107 System.out.println("贝尔实
108 TimeCheck imeCheck1= new
109 imeCheck1.begin();
110 array1=new Compare().quic
111 imeCheck1.end();
112 imeCheck1.duration();
113
114
Console
Problems
Progress
Debug Shell
Search
Git Staging
<terminated> Demo [Java Application] D:\jdk\6.1.9\bin\javaw.exe (2019年9月6日 下午10:59:15)
罗瑶光小高峰过滤快速排序4代：
start: Fri Sep 06 22:59:18 CST 2019
耗时: 1137 毫秒
贝尔实验室最新分配集合快速排序法：
start: Fri Sep 06 22:59:22 CST 2019
耗时: 1186 毫秒
Esc
F1
F2
F3
F4
F5
F6
F7


eclipse-workspace - Data_Processor/DP/sortProcessor/Demo.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Debug
Project Explorer
Heap_2D_Sort.java
InsertionSort.java
Leaf.java
LinkSort.java
LYGSort.java
MergeSort.java
OrderEvenSort.java
OTreeSort.java
Quick_1D_Sort.java
Quick_2D_Sort.java
Quick_3D_Sort.java
Quick_4D_Sort.java
Quick_5D_Sort.java
Quick_6D_luoyaoguang_Sort.java
Quick_7D_Sort.java
Quick_Luoyaoguang_1D.java
Quick_Luoyaoguang_2D.java
Quick_Luoyaoguang_3D.java
Quick_Luoyaoguang_4D.java
Quick_Luoyaoguang_4D
SelectionSort.java
ShellSort.java
TTreeSort.java
soundProcessor
Compare.java
Quick_Luoyaoguang_4
84 //check.begin();
85 for(int i=0;i<100
86 for(int j=0;j<
87 for(int k=
88 int pr
89 }
90 }
91 }
92 System.out.println
93 System.out.println
94 TimeCheck imeCheck
95 imeCheck1.begin();
96 array1=new Compare(
97 imeCheck1.end();
98 imeCheck1.duration(
99 for(int i=0;i<10000
100 for(int j=0;j<1
101 for(int k=0
102 int pre
103 }
104 }
105 }
106 System.out.println("
107 System.out.println("
Console
Problems
Progress
Debug Shell
Search
Git Staging
<terminated> Demo [Java Application] D:\jdk\6.1.9\bin\javaw.exe (2019年9月6日 下午11:00:57)
贝尔实验室最新分配集合快速排序法：
start: Fri Sep 06 23:01:01 CST 2019
耗时 1178 毫秒
罗盛光小高峰过滤快速排序4代：
start: Fri Sep 06 23:01:05 CST 2019
耗时 1072 毫秒


or/Compare.java - Eclipse IDE
ct Run Window Help
Compare.java
Quick_Luoyaoguang_4D.java
Demo.java
1 package sortProcessor;
2 public class Compare{
3 public int getMiddle(int[] list, int low, int high) {
4 int tmp= list[low]; //数组的第一个作为中轴
5 while (low < high) {
6 while (low < high && list[high] >= tmp) {
7 high--;
8 }
9 list[low] = list[high]; //比中轴小的记录移到低
10 while (low < high && list[low] <= tmp) {
11 low++;
12 }
13 list[high] = list[low]; //比中轴大的记录移到高
14 }
15 list[low] = tmp; //中轴记录到尾
16 return low; //返回中轴的位置
17 }
18
19 public void _quickSort(int[] list, int low, int high) {
20 if (low < high) {
21 int middle = getMiddle(list, low, high); //将lis
22 _quickSort(list, low, middle - 1); //对低
23 _quickSort(list, middle + 1, high); //对高
24 }
25 }
26
27 public int[] quick(int[] str) {
Search
Git Staging
n\javaw.exe (2019年9月6日 下午11:00:57)


uick_Luoyaoguang_4D.java - Eclipse IDE
Run Window Help
ompare.java
Quick_Luoyaoguang_4D.java
Demo.java
int pos = partition(a, lp, rp);
quick2ds(a, lp, pos-1);
quick2ds(a, pos+1, rp);
}
}
private int partition(int[] a, int lp, int rp) {
int x= a[lp]>= a[rp]? a[lp]: a[rp];
int lp1= lp;
while(lp1< rp){//我总觉得这里可以进行一种积分算法优
// while(a[lp1]<= x&& lp1< rp) {
// lp1++;
// }
while(!(a[lp1]>x|| lp1>=rp)) {
lp1++;
} //今天想到了一些优化，
while(a[rp]>x){
rp--;
}
if(lp1<rp){
int temp=a[rp];a[rp]=a[lp1];a[lp1]=temp;
}
}
a[lp]=a[rp];a[rp]=x;
return rp;
}
Search
Git Staging
vaw.exe (2019年9月6日 下午11:00:57)