**2024秋-《算法设计与分析》**

**排序算法分析实验报告**

|  |
| --- |
| 2024年11月10日 |

目录

[一、 实验要求 1](#_Toc9947)

[二、 实验报告 1](#_Toc7906)

[2.1实验设计 1](#_Toc5677)

[2.2 实验结果分析 2](#_Toc17369)

[2.2.1 比较几个经典排序算法在不同输入的表现 2](#_Toc11034)

[2.2.2 快速排序中枢点选取优化 4](#_Toc25525)

[2.2.3 三种快速排序比较 5](#_Toc28986)

[2.2.4 \*三种快速排序性能对比 6](#_Toc23384)

[三、 结论 7](#_Toc31464)

# 实验要求

对几个经典的排序算法进行分析，理解算法在不同输入时的表现，深入剖析算法优缺点及其根源。具体要求如下：

1. 以给出的插入排序算法为例，实现几种排序算法，至少要实现冒泡排序、快速排序、归并排序、希尔排序算法；
2. 在排序算法中插桩，记录关键操作次数（如比较次数、移动次数等）；
3. 固定n随机产生多组测试样本，统计算法的平均运行时间和关键操作次数，改变n的规模重复多次实验，并对结果进行统计；
4. 改变数组规模，对不同规模问题下各算法的结果进行统计并绘制图表，与理论值进行对照分析；
5. 优化快速排序的中枢点选取，对优化前后的性能进行分析；
6. 对快速排序的三种实现进行性能比较。

# 实验报告

## **2.1实验设计**

1. 实现各种经典排序算法：插入排序、冒泡排序、希尔排序、归并排序和快速排序。在排序算法中插桩，记录关键操作次数，对不同规模下各算法的结果进行统计并绘制图表，与理论值进行对照分析。
2. 对于理论运行时间的计算，采用如下方法：以时间复杂度为O(nlogn)的算法为例，假定（可任意取）时，理论运行时间与实际运行时间相等为，则

即。

1. 写了一个data\_gen程序以生成规定规模的随机数据文件，每个文件中包含20组不同的规模相同的数据，减小测量时运行时间和关键操作次数的偶然误差并且各算法使用的是相同的数据。

## 2.2 实验结果分析

### 2.2.1 比较几个经典排序算法在不同输入的表现

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Time/ms  n | 1000 | 5000 | 1e4 | 2e4 | 3e4 | 5e4 | 1e5 | 2e5 | 5e5 | 1e6 | 2e6 | 5e6 |
| InsertSort | 1.35 | 36.95 | 196.7 | 872 | 2464.4 | 6383.8 | 23744.5 |  |  |  |  |  |
| BubbleSort | 3.55 | 97.1 | 501.55 | 2380.15 | 6652.6 | 18522.1 | 66369.4 |  |  |  |  |  |
| ShellSort | 0.15 | 1.2 | 3.4 | 8.15 | 15.75 | 30.15 | 64.15 | 135.55 | 342.55 | 847.4 | 2063.7 | 5460.5 |
| MergeSort | 0.25 | 1.75 | 5.4 | 13.4 | 24.6 | 43.65 | 87.55 | 167.15 | 383.6 | 852.2 | 1921.65 | 4747.9 |
| QuickSort | 0.05 | 0.35 | 1.35 | 3.5 | 6.95 | 12.95 | 25.45 | 54.05 | 125.2 | 297.5 | 792.4 | 2559.2 |

表 1：运行时间比较

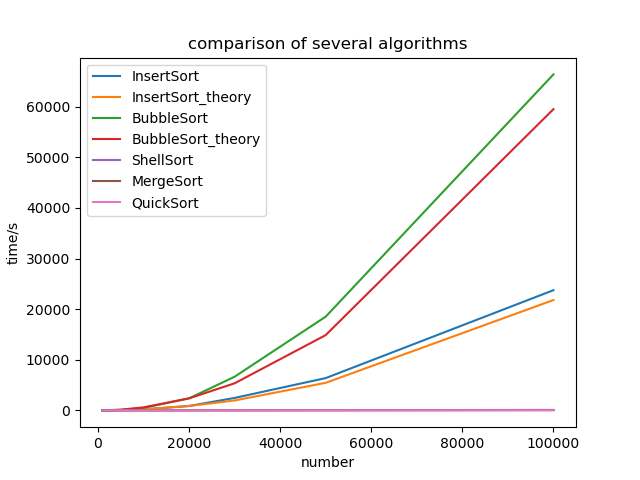
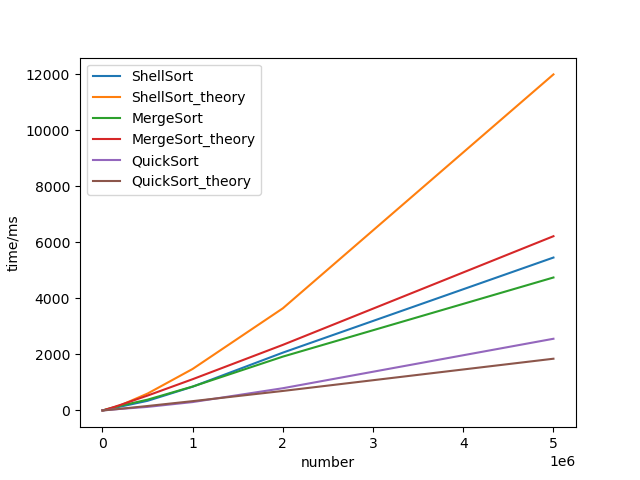
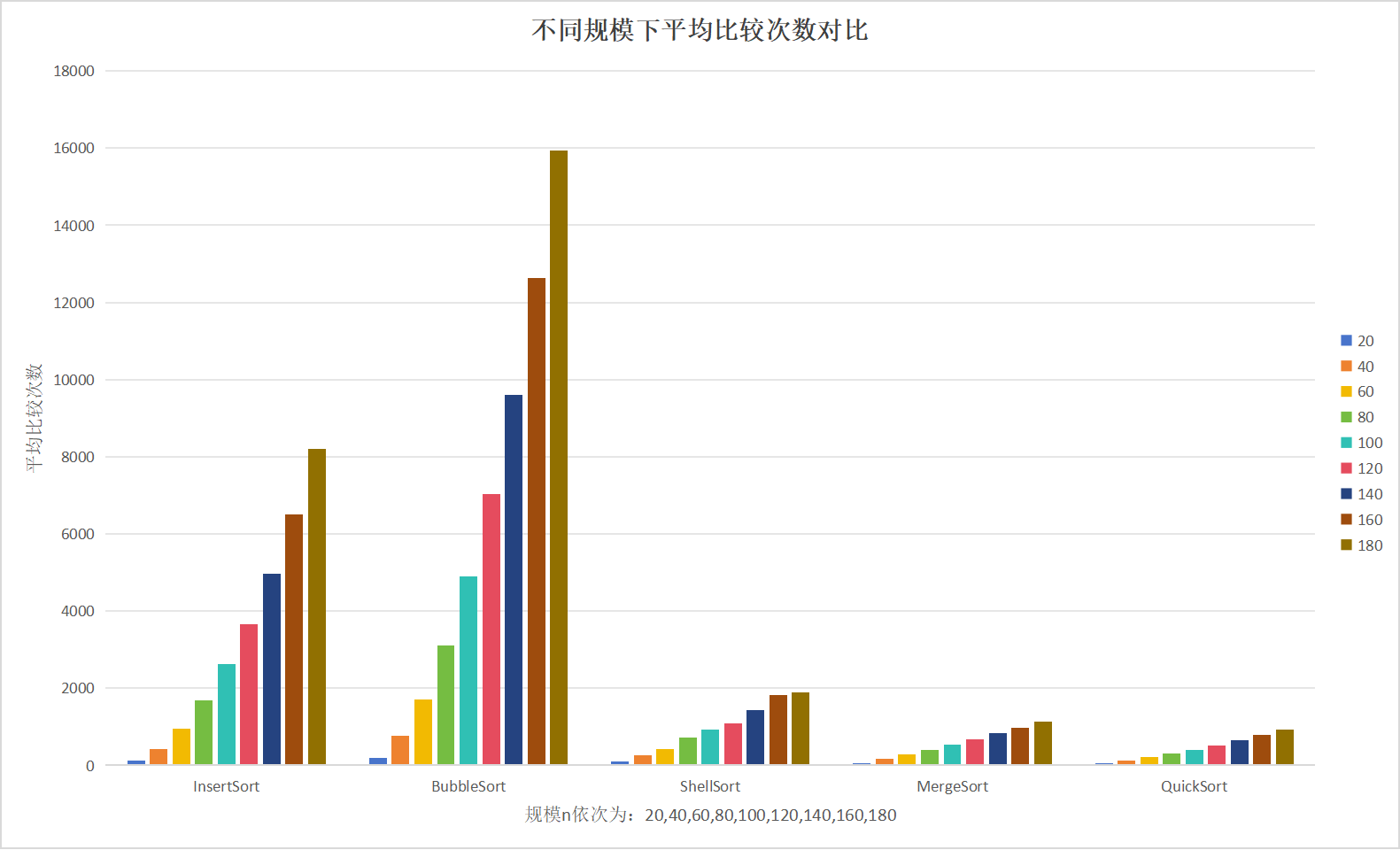
****

图 1：运行时间比较及与理论比较

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Op  n | 20 | | 40 | | 60 | | 80 | | 100 | | 120 | | 140 | | | 160 | | | 180 | |
| InsertSort | 113 | 94 | 430 | 391 | 941 | 882 | 1680 | 1601 | 2628 | 2592 | 3647 | 3528 | 4958 | 4819 | 6512 | | 6353 | 8211 | | 8032 |
| BubbleSort | 183 | 94 | 758 | 391 | 1715 | 882 | 3105 | 1601 | 4901 | 2592 | 7036 | 3528 | 9613 | 4819 | 12640 | | 6353 | 15946 | | 8032 |
| ShellSort | 97 | 35 | 262 | 100 | 421 | 177 | 711 | 309 | 926 | 423 | 1078 | 474 | 1426 | 583 | 1830 | | 868 | 1897 | | 813 |
| MergeSort | 64 |  | 165 |  | 281 |  | 407 |  | 541 |  | 679 |  | 825 |  | 974 | |  | 1126 | |  |
| QuickSort | 44 | 26 | 121 | 69 | 206 | 119 | 314 | 169 | 396 | 231 | 512 | 288 | 657 | 344 | 796 | | 413 | 919 | | 472 |

表 2：关键操作次数比较：第一列为比较次数，第二列为移动次数



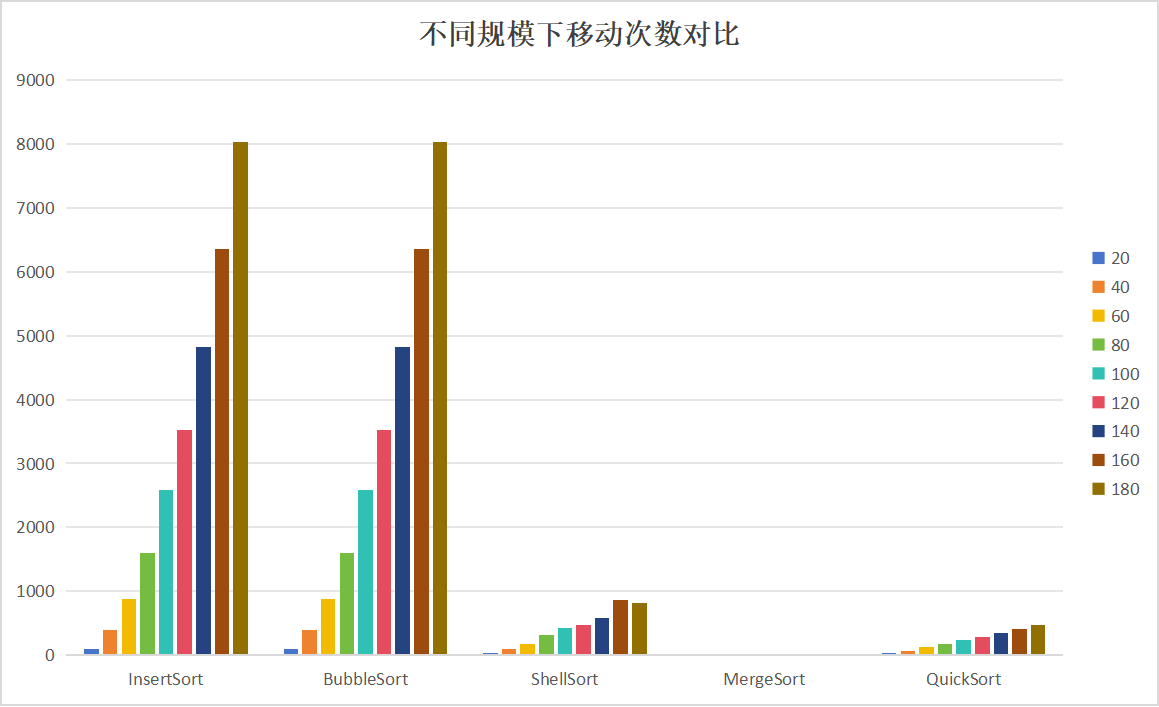


图 2：关键操作次数比较

分析：

1. 理论平均时间复杂度及关键操作数

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithms | 平均时间复杂度 | 比较次数 | 移动（交换）次数 |
| BubbleSort | O(n^2) | O(n^2) | O(n^2) |
| InsertSort | O(n^2) | O(n^2/4) | O(n^2/4) |
| ShellSort | O(n^1.3), 介于O(n)和O(n^2)之间 | ? | ? |
| MergeSort | O(nlogn) | O(nlogn) | 0 |
| QuickSort | O(nlogn) | O(nlogn) | ? |

表 3:理论分析（？表示未知）

1. 由表1和图1可以看出，实现的五种排序算法在不同规模下的运行时间符合理论结果。在规模相同时，不同算法的运行时间对比与理论分析也符合。插入排序和冒泡排序的运行时间远大于希尔排序、归并排序和快速排序，增长速度也快得多。
2. 由表2和图2，插入排序和冒泡排序的比较次数和移动次数符合n^2的增长速度，同时可以看到插入排序的比较次数少于冒泡排序，这可能是在规模相同时，插入排序的运行时间少于冒泡排序的原因。而希尔排序、归并排序、快速排序关键操作次数比较少，增长速度也缓得多。
3. 关于希尔排序和归并排序。希尔排序的平均时间复杂度为O(n^1.3)，归并排序的平均时间复杂度为O(nlogn)，在n<4000时，n^1.3<nlogn，但是根据实际运行结果，在n>=2e6时，希尔排序的运行时间才大于归并排序。原因可能是：随机生成的数据更利好希尔排序，使得希尔排序的平均时间复杂度小于n^1.3；归并排序的归并操作耗费了比较多的时间。这使得在规模相同时，归并排序的关键操作次数少于希尔排序，运行时间却大于希尔排序。

### 2.2.2快速排序中枢点选取优化

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Mode | 1000 | 5000 | 1e4 | 2e4 | 3e4 | 5e4 | 1e5 | 2e5 | 5e5 | 1e6 | 2e6 | 5e6 |
| first | 0.05 | 0.35 | 1.35 | 3.5 | 6.95 | 12.95 | 25.45 | 54.05 | 125.2 | 297.5 | 792.4 | 2559.2 |
| random | 0 | 0 | 0.9 | 2.65 | 7.5 | 14.85 | 27.45 | 52.4 | 126.8 | 317.55 | 828.85 | 2487.8 |
| Mid3 | 0 | 0.95 | 2.55 | 5 | 8.35 | 14.85 | 29.3 | 59.15 | 136.5 | 314.65 | 741.95 | 2301.15 |

表 4:运行时间比较（单位：ms），0指时间小与1E-6s

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Mode | 1000 | | 5000 | | 1e4 | | 2e4 | | 3e4 | | 5e4 | | 1e5 | | 2e5 | | 5e5 | | 1e6 | | 2e6 | | 5e6 | |
| first | 0.07 | 0.04 | 0.70 | 0.37 | 2.87 | 1.49 | 8.33 | 4.34 | 19.51 | 10.22 | 40.41 | 21.24 | 81.26 | 42.70 | 166.5 | 86.42 | 385.1 | 184.9 | 982.9 | 402.9 | 2734 | 862.3 | 9941 | 1889 |
| random | 0.007 | 0.004 | 0.05 | 0.02 | 0.10 | 0.05 | 0.22 | 0.12 | 0.35 | 0.18 | 0.62 | 0.32 | 1.32 | 0.70 | 3.06 | 1.49 | 9.78 | 3.88 | 27.47 | 7.86 | 85.24 | 15.84 | 441.0 | 39.8 |
| Mid3 | 0.006 | 0.004 | 0.04 | 0.03 | 0.09 | 0.06 | 0.19 | 0.12 | 0.30 | 0.19 | 0.56 | 0.33 | 1.23 | 0.72 | 2.84 | 1.51 | 9.34 | 3.93 | 26.62 | 7.95 | 83.53 | 16.02 | 438.1 | 40.2 |

表 5:关键操作次数比较（单位：1000000）

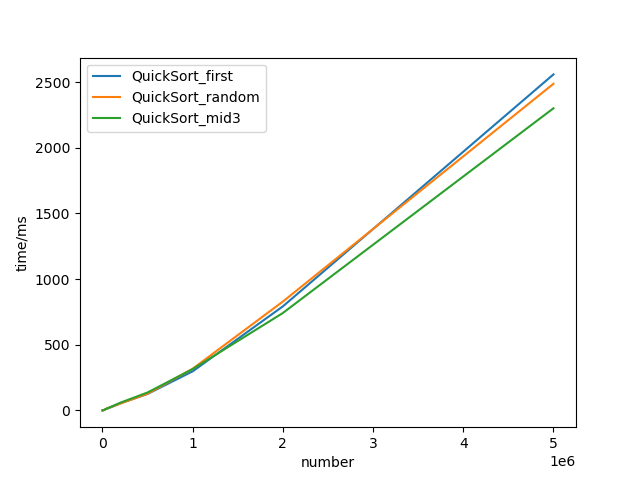


图 3:运行时间曲线

分析：

由表4表5和图3，可以看到基元的选取对算法的运行是有一定影响的。first表示基元为队列第一个元素，random表示基元为队列中随机选取的一个元素，mid3表示基元为队首、队中、队尾三个数的中位数。使用random和mid3的方法选取基元，大大减少了排序的比较次数和移动次数，运行时间在一定程度上也能降低。效果不太明显的原因可能为：random和mid3操作耗费了一定时间。

**2.2.3三种快速排序比较**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithms | 1000 | 5000 | 1e4 | 2e4 | 3e4 | 5e4 | 1e5 | 2e5 | 5e5 | 1e6 | 2e6 | 5e6 |
| 空穴 | 0.05 | 0.35 | 1.35 | 3.5 | 6.95 | 12.95 | 25.45 | 54.05 | 125.2 | 297.5 | 792.4 | 2559.2 |
| book | 0 | 0.8 | 0.8 | 3.2 | 6.35 | 17.25 | 35.3 | 74.2 | 188.1 | 494.55 | 1387.7 | 5450.85 |
| Hoare | 0 | 0 | 0.75 | 3.1 | 8.55 | 14 | 27.6 | 55.7 | 128.3 | 276.4 | 584 | 1381.9 |

表 6：运行时间比较（单位：ms）

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| algorithms | 1000 | | 5000 | | 1e4 | | 2e4 | | 3e4 | | 5e4 | | 1e5 | | 2e5 | | 5e5 | | 1e6 | | 2e6 | | 5e6 | |
| 空穴 | 0.07 | 0.04 | 0.70 | 0.37 | 2.87 | 1.49 | 8.33 | 4.34 | 19.51 | 10.22 | 40.41 | 21.24 | 81.26 | 42.70 | 166.5 | 86.42 | 385.1 | 184.9 | 982.9 | 402.9 | 2734 | 862.3 | 9941 | 1889 |
| book | 0.01 | 0.006 | 0.07 | 0.04 | 0.2 | 0.09 | 0..34 | 0.19 | 0.53 | 0.29 | 0.94 | 0.54 | 2.03 | 1.16 | 4.46 | 2.72 | 13.62 | 9.22 | 35.11 | 26.09 | 101.1 | 82.7 | 482.2 | 436.8 |
| hoare | 0.01 | 0.002 | 0.08 | 0.01 | 0.17 | 0.03 | 0.36 | 0.07 | 0.56 | 0.11 | 0.98 | 0.19 | 2.08 | 0.41 | 4.39 | 0.89 | 11.80 | 2.48 | 24.18 | 5.43 | 51.00 | 11.77 | 135.0 | 32.6 |

表 7：关键操作次数比较（单位：1000000）

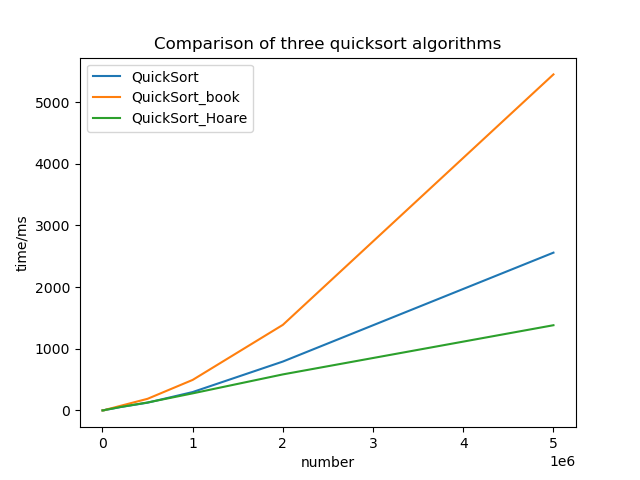


图 4：运行时间比较

分析：

由表6、表7和图4，book表示实现的是教材上的快速排序算法，Hoare表示实现的是Hoare提出的算法。可以看到book和Hoare大大降低了关键操作次数，尤其是Hoare，且Hoare运行时间较空穴有了降低。但是book的运行时间增加了，这可能是因为：book的排序顺序是从后往前，缓存命中率比较低，会带来很大的时间开销。

**2.2.4 \*三种快速排序性能对比**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithms | 1000 | 2000 | 3000 | 4000 | 5000 | 6000 | 7000 | 8000 | 9000 | 10000 |
| quicksort | 0 | 0.1 | 0.5 | 0.7 | 1.5 | 2.2 | 2.5 | 4.1 | 4.95 | 6.4 |
| blockquicksort | 1.75 | 11.9 | 16.3 | 32.05 | 49.3 | 68.1 | 92.2 | 119.55 | 150.2 | 180.8 |
| dualpivotquicksort | 0.75 | 0.75 | 0.8 | 1 | 1.35 | 2.25 | 3.1 | 3.6 | 5.1 | 6.4 |

表 8:运行时间比较

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithms | 1000 | | 2000 | | | 3000 | | | 4000 | | | 5000 | | | 6000 | | | 7000 | | | 8000 | | | 9000 | | | 10000 | | |
| quicksort | 0.72 | 0.38 | | 1.65 | 0.84 | | 2.69 | 1.35 | | 3.70 | 1.88 | | 4.54 | 2.44 | | 5.65 | 2.99 | | 6.83 | 3.56 | | 7.92 | 4.15 | | 8.98 | 4.74 | | 10.6 | 5.30 |
| blockquicksort | 49.6 | 0.44 | | 487 | 1.93 | | 160 | 3.54 | | 723 | 5.48 | | 778 | 7.88 | | 865 | 9.71 | | 1051 | 12.3 | | 1164 | 14.4 | | 1432 | 16.9 | | 1391 | 19.5 |
| dualpivotquicksort | 0.63 | 0.25 | | 1.46 | 0.58 | | 2.36 | 0.88 | | 3.26 | 1.20 | | 4.20 | 1.53 | | 5.11 | 1.88 | | 6.09 | 2.19 | | 7.09 | 2.61 | | 8.17 | 2.95 | | 9.20 | 3.24 |

表 9：关键操作次数比较（单位为：10000）

当n=100000000时：



图 5：运行实际截图

分析：

可以看到实现的dualpivotQuickSort降低了比较次数和移动次数，从而降低了运行时间，但在数据规模比较大时才比较明显。而实现的blockQuickSort（规定blocksize=1024），性能反而降低了，原因可能是：合并排序好的块需要耗费一定的时间。

1. **结论**

本次实验比较了几种经典排序算法在同一规模下的运行时间和关键操作次数，最后得到性能比较：快速排序>归并排序>希尔排序>插入排序>冒泡排序。然后对快速排序进行了更深入的实验：中枢点选取对性能的影响；排序顺序的影响等。最后还实现了blockQuickSort和dualpivotQuickSort以优化基础的快速排序。