**2024秋-《算法设计与分析》**

**分治法算法分析实验报告**

|  |
| --- |
| 2024年10月26日 |

**目 录**

[一、 实验要求 1](#_Toc17179)

[二、 实验报告 1](#_Toc23336)

[1.1 实验原理 1](#_Toc11989)

[1.2 实验目的 3](#_Toc20415)

[1.3 实验设计 3](#_Toc2784)

[1.4 实验结果与分析 4](#_Toc27346)

# 实验要求

对几种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. 实现常见排序算法，至少要实现插入排序、冒泡排序、快速排序、归并排序、shell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. 以待排序文件的行数n为输入规模，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

## 实验原理

1. **五种排序算法的算法原理**

在本实验中，我们实现插入排序、冒泡排序、快速排序、归并排序、shell排序算法五种算法，五种算法原理如下:

插入排序是一种简单直观的排序算法。它的工作原理是通过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。插入排序在实现上，通常使用in-place排序（即只需用到O(1)的额外空间的排序）。该算法的平均算法时间复杂度为：

冒泡排序的排序过程基于重复地遍历要排序的数组，一次比较两个元素，如果它们的顺序错误就把它们交换过来。每次循环会讲最大的元素置于数组最后端，直到将数组排至有序，即在某次循环上没有发生任何交换。该算法的平均算法时间复杂度为：

快速排序是一种基于分治思想的排序算法，对于一个特定长度的数组，快速排序选取该数组中的一个元素作为基元，将该数组中所有小于基元的元素置于基元的左侧，大于基元的元素置于基元的右侧。然后，以基元为分界线，将该数组分割成规模小于当前数组的两个数组，再递归的对这两个小数组使用快速排序。该算法的平均时间复杂度为：

归并排序也是一种基于分治思想的排序算法，对于一个特定长度的数组，归并排序将数组均分为两个小数组，并分别进行归并排序，两个小数组在归并排序后已有序，接下来通过双指针等方法将两个有序的小数组进行合并，得到一个有序的当前数组，即完成归并排序。该算法的平均时间复杂度为：

Shell排序是一种使用了插入排序算法的算法，Shell排序算法会先选定一个增量数组，遍历增量数组中每一个元素，根据当前增量，将原数组分成若干长度相近的子序列，对于该子序列，进行插入排序。最后，直到遍历到当前增量为1时，对子序列的插入排序就会使得原数组有序。该算法的平均时间复杂度为：

上述五种排序算法的最好，最坏以及平均时间复杂度如下表：

|  |  |  |  |
| --- | --- | --- | --- |
| 算法 | 平均时间复杂度 | 最好时间复杂度 | 最坏时间复杂度 |
| 插入排序 |  |  |  |
| 冒泡排序 |  |  |  |
| 快速排序 |  |  |  |
| 归并排序 |  |  |  |
| Shell排序 |  |  |  |

1. **三种不同的基元选择方式**

最简单的基元选择方式就是选择当前数组的最后一个或者第一个元素作为基元；第二种基元选择方式是通过生成随机下标的方式选择数组对应位置上的元素作为基元；第三种基元选择方式是选择最后三个元素中的中位数为基元。

1. **三种不同的快速排序实现方式**

空穴法算法步骤如下：

1. 选择一个基准元素（通常选择第一个元素）。
2. 设置两个指针，一个指向数组的起始位置，另一个指向数组的末尾。
3. 移动右指针，直到找到一个小于基准元素的值，然后将该值赋给左指针所指的位置。
4. 移动左指针，直到找到一个大于基准元素的值，然后将该值赋给右指针所指的位置。
5. 重复上述步骤，直到左指针和右指针相遇。
6. 将基准元素放到左指针所指的位置，此时基准元素左侧的所有元素都小于它，右侧的所有元素都大于它。

Hoare分区的快速排序算法步骤如下：

1. 选择一个基准元素（通常选择第一个元素）。
2. 设置两个指针，一个指向数组的起始位置，另一个指向数组的末尾。
3. 同时移动两个指针，左指针向右移动直到找到一个大于或等于基准元素的值，右指针向左移动直到找到一个小于或等于基准元素的值。
4. 交换两个指针所指的元素。
5. 重复上述步骤，直到左指针和右指针相遇。
6. 交换基准元素和左指针所指的元素，此时基准元素左侧的所有元素都小于或等于它，右侧的所有元素都大于或等于它。

Lomuto分区的快速排序算法步骤如下：

1. 选择一个基准元素(通常为第一个元素)。
2. 设置两个指针prev(指向最左边)和pcur(指向最左边右一个位置)。
3. 当pcur小于基准元素时，prev右移一位，与pcur交换；否则pcur一直往右移动
4. 当pcur超出右边界后，交换基准元素与prev位置上元素

## 实验目的

本次实验的目的主要有以下：

1. 实现插入排序，冒泡排序，快速排序，归并排序，Shell五种排序算法，并记录运行时间和关键操作次数（比较次数，交换次数）。
2. 对于不同规模下的五种算法的时间和关键操作次数，进行统计，绘图和比较分析。
3. 对于快速排序的基元选择进行优化，进行优化前后性能比较分析。
4. 对于快速排序的三种不同实现方法，进行性能比较分析。

## 实验设计

我们在本次实验中，用C++语言来实现五种不同的排序算法和三种不同基元选择下的快速排序以及三种不同方式实现的快速排序的算法，再使用Python进行绘图分析。

在Algorithm.cpp中，实现了七个排序算法以及子函数，功能和基本原理如下表：

|  |  |  |
| --- | --- | --- |
| 函数名 | 功能 | 基本原理 |
| InsertSort | 插入排序 | 遍历比较，逐个插入 |
| BubbleSort | 冒泡排序 | 遍历比较，逐个冒泡 |
| Partition | 基于基元分割 | 逐个比较，进行交换 |
| QuickSort | 快速排序 | 递归调用自身 |
| Merge | 合并有序的两个数组 | 双指针合并 |
| MergeSort | 归并排序 | 递归调用自身 |
| ShellSort | 希尔排序 | 依据增量分组插入排序 |

在QuickSortAlgorithm.cpp中，实现了三种基元选择的排序算法和三种方式实现的快速排序算法，功能和基本原理如下表:

|  |  |  |
| --- | --- | --- |
| 函数名 | 功能 | 基本原理 |
| Partition | 基于基元分割 | 选固定位置的基元进行分割 |
| QuickSort | 普通的快速排序 | 递归调用自身 |
| PartitionRand | 基于随机基元的分割 | 随机生成基元下标进行分割 |
| QuickSortRand | 随机基元的快速排序 | 递归调用自身 |
| PartitionThree | 基于中位数基元的分割 | 选择固定三个元素，选择中位数作为基元分割 |
| QuickSortThree | 基于中位数基元的快速排序 | 递归调用自身 |
| QuickSortKongXue | 基于空穴的快速排序 | 空穴法快速排序 |
| QuickSortHoare | 最原始的快速排序 | Hoare分割下的快速排序 |
| QuickSortQianHou | 前后双指针的快速排序 | 前后双指针进行快速排序 |

在FiveAlgorithmTest.cpp和QuickSortTest.cpp中分别统计五种排序算法和6种快速排序算法在不同的数据规模下的耗时，交换次数和比较次数，用Python绘制图像。

## 实验结果与分析

###### 五种排序算法的结果对比

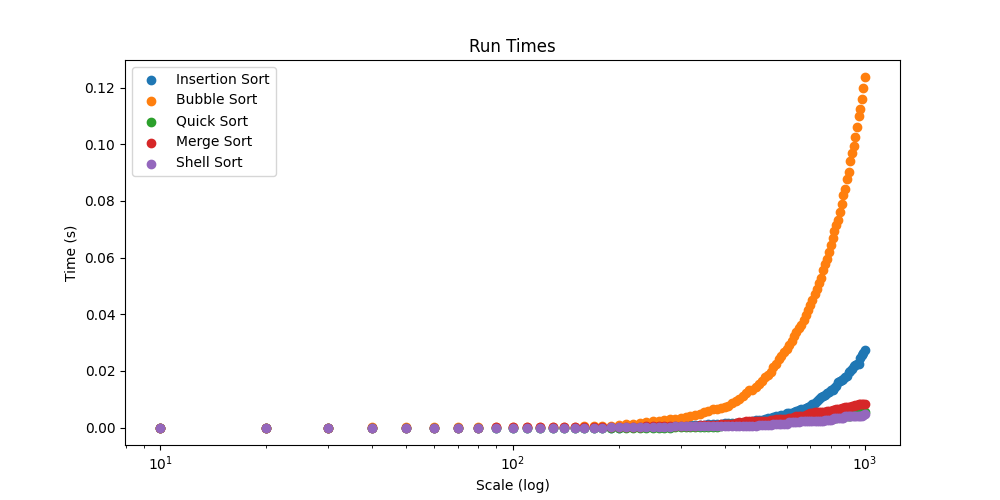


图 1 五种算法时间对比图

从上图可以看出，冒泡排序的耗时远大于其他四种算法，其次是插入排序算法，而快速排序算法和归并算法以及希尔排序在1e3的规模下基本上有相同的耗时，这也符合冒泡排序和插入排序平均时间复杂度为，远高于归并排序和快速排序的平均时间复杂度，而在1e3规模下，希尔排序的的时间复杂度和相差不大。为了更好地对比在更大规模下快速排序，归并排序和希尔排序的耗时，画了以下只有这三种算法的时间对比图：

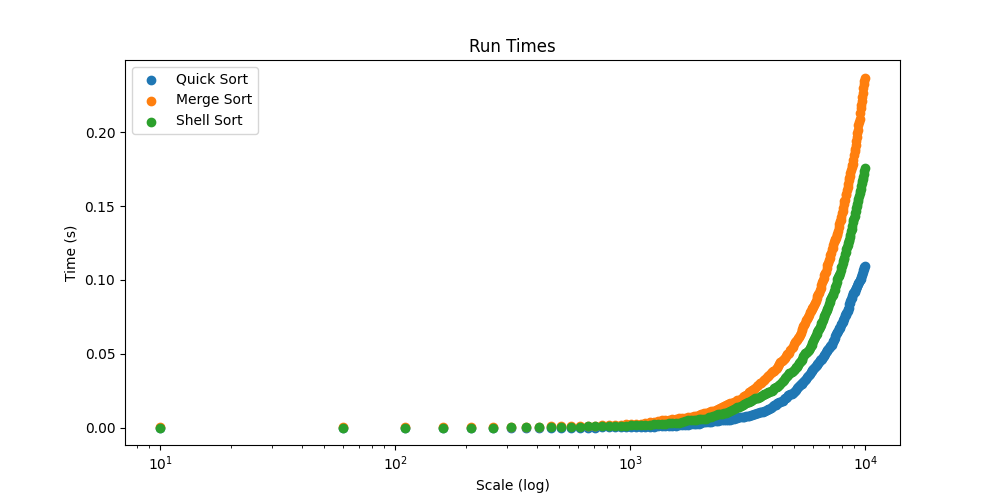
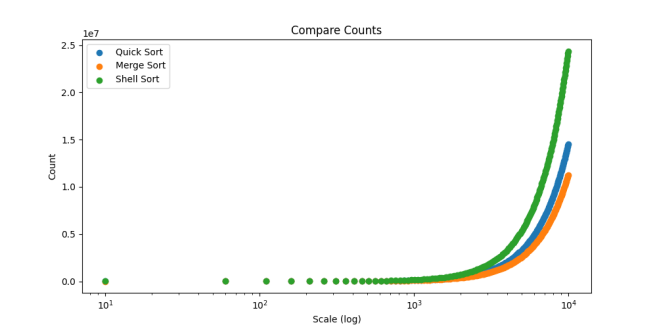
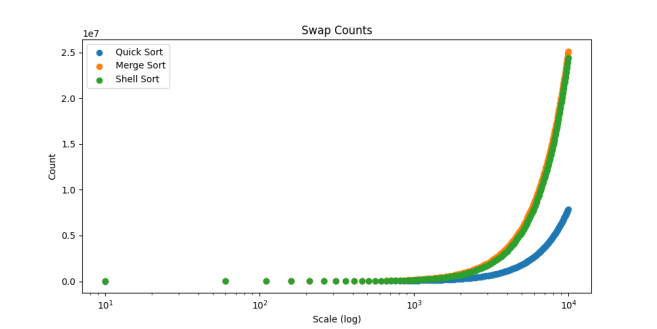
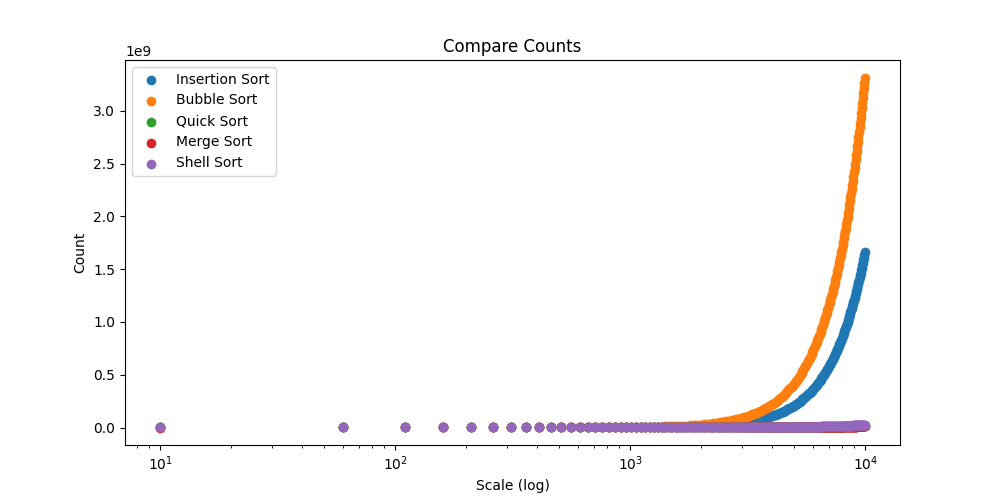
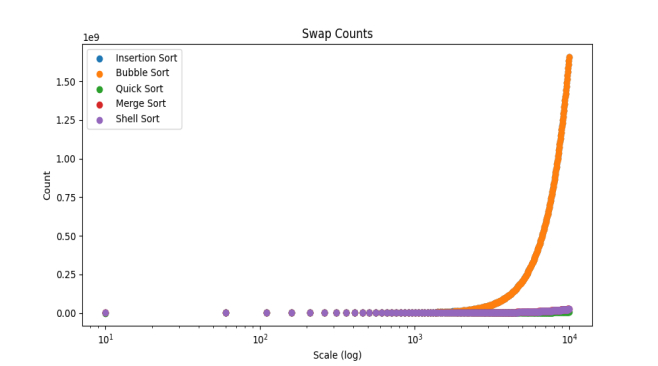


图 2 三种算法的时间对比图

从图中我们可以看到，随着数字规模的增大，最快的是快速排序，其次是归并排序，最慢的是希尔排序，这也与我们对于时间复杂度的分析相吻合，的平均时间复杂度是小于的平均时间复杂度的。



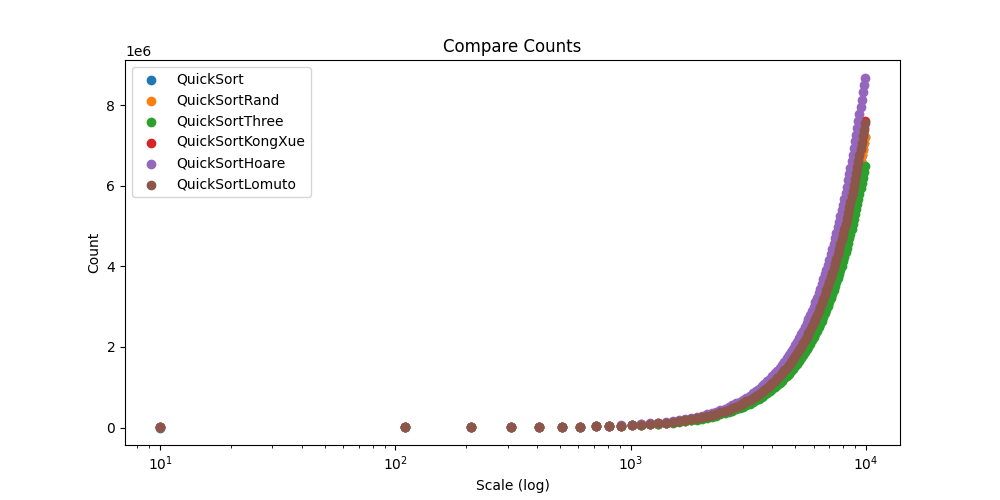
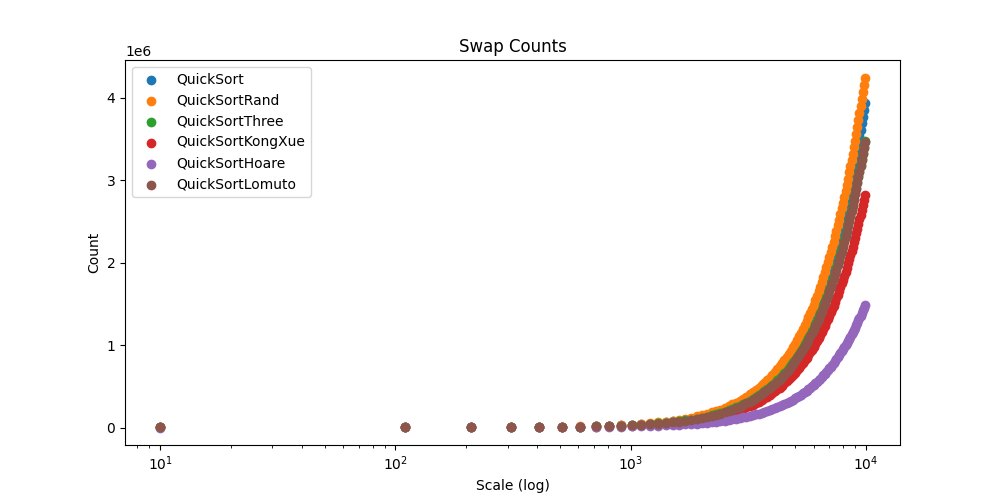
从上述的交换次数和比较次数的示意图我们可以看出，插入排序和冒泡排序的比较和交换次数都远超其他三种算法，而在相同规模下，快排的交换次数优于归并排序和希尔排序，而比较次数和归并排序相近，所以是最快的算法；归并排序的交换次数与希尔排序相近但是比较次数小于希尔排序，所以比希尔排序快；而希尔排序无论是比较次数还是交换次数，都是比较大，所以最慢。

###### 三种基元选择方法及三种快速排序的对比

###### D:/MyGame/Desktop/Liuguodong/算法设计与分析/code_template-sorting/run_timesQuick.pngrun_timesQuick

图 3 6种快速排序的对比

从上图可以看出，在1e4的数据规模内，这六种不同的快速排序耗时基本相同。以下是算法的比较次数和交换次数对比图:



从上方对比图我们可以看出，和耗时的情况相同，在该数据规模下，六种的交换次数和比较次数都比较相近，只有Hoare的交换次数相对多一些，但是没有明显影响算法的耗时。