**深 圳 大 学 实 验 报 告**

**课程名称： 算法设计与分析**

**实验名称： 实验1 排序算法性能分析**

**学院：计算机与软件学院 专业： 计算机科学与技术（创新班）**

**报告人： 李文俊 学号： 2023150001 班级： 高性能班**

**同组人： 无**

**指导教师： 杨烜**

**实验时间： 2025/3/7——2025/3/14**

**实验报告提交时间： 2025/3/18**

**教务处制**

**一．问题描述**

1.实现选择排序、冒泡排序、合并排序、快速排序、插入排序算法并进行效率分析和算法对比

2. TopK问题：现在有10亿的数据（每个数据四个字节），请快速挑选出最大的十个数，并在小规模数据上验证算法的正确性

**二．算法原理**

**（1）五个排序算法**

1.选择排序

1.1原理描述

每一趟从待排序的数据元素中选出**最小（或最大）的一个元素**，顺序放在**已排好序的数列的最后**，直到全部待排序的数据元素排完

将8，16，21，25，27，49作为降序选择排序的例子，如图1所示。

图示

AI 生成的内容可能不正确。

图1 选择排序图示

**实现细节：**

**外层循环**控制已排序区间的末尾位置i ，只需进行n-1次即可完成所有元素的排序。

**内层循环**在未排序区间 [i+1, n] 中寻找最小值索引 min\_idx，每次找到 min\_idx 后，仅进行一次交换操作（arr[i] 与 arr[min\_idx]），避免频繁交换带来的性能损耗

**减少交换次数**：记录最小值的索引 min\_idx，而非每次比较都交换。例如，若未排序区间的最小值已在正确位置（min\_idx == i），则无需交换

1.2 核心伪代码

**文本, 信件

AI 生成的内容可能不正确。**

图2 选择排序伪代码

1.3 复杂度分析

每一趟都需要遍历数组来找到最小元素，所以复杂度和原始序列的顺序无关，

所以，最好最坏和平均时间复杂度都是。

1.4 效率测试

使用随机数生成，生成了十万到五十万的数据集，每个数据量都进行了20次测试并取平均值。

以输入规模为10万的数据运行时间为基准点，计算输入规模为其他值的理论运行时间，

结果如表1所示。

表1 选择排序运行时间表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据量 | 100000 | 200000 | 300000 | 400000 | 500000 |
| 单次排序时间（ms） | 6170.62 | 25068.6 | 56081.8 | 100140 | 156547 |
| 理论时间（ms） | 6170.62 | 24682.48 | 55535.58 | 55535.58 | 154265.5 |
| 相对误差(%) | 0 | 1.564 | 0.983 | 1.428 | 1.4789 |

画出不同规模数据的理论运行时间曲线和实测效率曲线，如图3所示。



图3 选择排序曲线

理论运行时间曲线和实测效率曲线几乎重合，误差在合理范围内，符合O()曲线，经验分析和理论分析一致。

2.冒泡排序

2.1原理描述

在待排序的一组数中，将**相邻的两个数进行比较**，若前面的数比后面的数大就交换两数，否则不交换。如此下去，直至最终完成排序

**每一趟能够把待排序中最大的数交换到最后**

将8，16，21，25，27，49作为冒泡排序的例子，如图4所示。

图示

AI 生成的内容可能不正确。

图4 冒泡排序图示

**实现细节：双循环结构**

**外层循环：**控制冒泡轮数，每一轮将当前未排序部分的最大值“冒泡”到数组末尾。理论上需要 n-1 轮（n 为数组长度），但可通过标志位Flag检测：若某轮排序中未发生任何交换，说明数组已完全有序，可提前终止排序

**内层循环：**在未排序区间（除去已排序的末位元素）**[0, n-i-1]** 内遍历相邻元素，比较并交换逆序对。每轮结束后，未排序区间长度减 1。

**原地交换：**通过相邻元素两两比较，若逆序则直接交换，无需额外空间。

2.2 核心伪代码

**文本

AI 生成的内容可能不正确。**

图5 冒泡排序伪代码

2.3 复杂度分析

, 为最内层循环交换次数

**最好情况下**，数组已经有序的情况，，此时只需要遍历一次数据，没有交换发生，结束排序，**时间复杂度为O(n)**

**最坏情况下**，数组逆序，，，所以**时间复杂度为**

因此，**平均时间复杂度为**

2.4 效率测试

使用随机数生成，生成了十万到五十万的数据集，每个数据量都进行了20次测试并取平均值。

以输入规模为10万的数据运行时间为基准点，计算输入规模为其他值的理论运行时间，

结果如表2所示。

表2 冒泡排序运行时间表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据量 | 100000 | 200000 | 300000 | 400000 | 500000 |
| 单次排序时间（ms） | 23236.8 | 96038.6 | 217758 | 389013 | 609887 |
| 理论时间（ms） | 23236.8 | 92947.2 | 209131.2 | 371788.8 | 580920 |
| 相对误差(%) | 0 | 3.32 | 4.12 | 4.63 | 4.98 |

画出不同规模数据的理论运行时间曲线和实测效率曲线，如图6所示。



图6 冒泡排序曲线

理论运行时间曲线和实测效率曲线几乎重合，误差在合理范围内，符合O()曲线，经验分析和理论分析一致。

3.插入排序

3.1原理描述

将一个元素**插入**到**已经排好序的部分中**，从而得到一个新的有序序列。重复进行，直到所有元素都被插入到有序序列中

将1，2，3，4，4，9作为插入排序的例子，如图7所示。

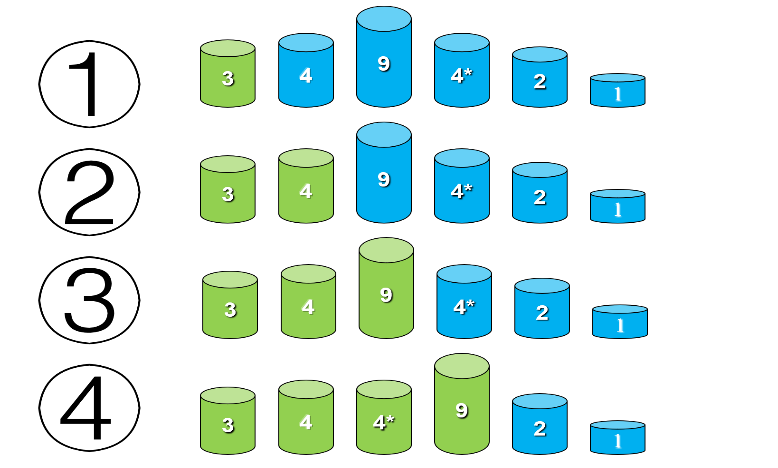


图7 插入排序图示

**实现细节：双循环结构**

**外层循环：**从 i = 1 到 n-1，表示当前待插入元素 arr[i]。

**内层循环：**从 j = i-1 向左遍历已排序区间，将 arr[i] 与已排序元素逐个比较，若 arr[j] > key（key 为 arr[i] 的备份值），则将 arr[j] 右移一位，直到找到合适插入位置。

**元素移动而非交换：**将 arr[i] 备份为 key，内层循环中直接移动元素（arr[j+1] = arr[j]），最后将 key 插入空位。与冒泡排序的多次交换相比，减少了赋值操作次数

3.2 核心伪代码

**文本

AI 生成的内容可能不正确。**

图8 插入排序伪代码

3.3 复杂度分析

, 为最内层循环交换次数

**最好情况下**，数组已经有序的情况，，此时只需要遍历一次数据，没有交换发生，结束排序，**时间复杂度为O(n)**

**最坏情况下**，每次插入元素都是最小值，，，所以**时间复杂度为**

**平均情况下，**，**时间复杂度同最坏情况，为**

3.4 效率测试

使用随机数生成，生成了十万到五十万的数据集，每个数据量都进行了20次测试并取平均值。

以输入规模为10万的数据运行时间为基准点，计算输入规模为其他值的理论运行时间，

结果如表1所示。

表3 插入排序运行时间表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据量 | 100000 | 200000 | 300000 | 400000 | 500000 |
| 单次排序时间（ms） | 5195.16 | 20989.3 | 47182.9 | 83884 | 130862 |
| 理论时间（ms） | 5195.16 | 20780.64 | 46756.44 | 83122.56 | 129879 |
| 相对误差(%) | 0 | 1.00 | 0.91 | 0.91 | 0.75 |

画出不同规模数据的理论运行时间曲线和实测效率曲线，如图9所示。



图9插入排序曲线

理论运行时间曲线和实测效率曲线几乎重合，误差在合理范围内，符合O()曲线，经验分析和理论分析一致。

4.归并排序

4.1原理描述

**分治法：**将待排序数组**不断分成两个子数组**，分别排序后**再合并**，最终得到一个有序数组

将8，16，21，25，25\*，49作为归并排序的例子，如图7所示。

形状, 圆圈

AI 生成的内容可能不正确。

图10 归并排序图示

**实现细节：分治递归结构**

**分：递归**将数组分为左右两半，直到子数组长度为1（递归终止条件），可表示为 **left>=right**

**治：合并**两个已排序的子数组。合并时通过双指针遍历左右子数组，按序填充到临时数组，最后将临时数组复制回原数组。

**避免频繁内存分配：**合并时使用预分配的全局临时数组（而非每次递归创建新数组），同时数组的拷贝根据合并序列长度对应拷贝（而非将数组完全拷贝），明确左右子数组的区间为 [left, mid] 和 [mid+1, right]，防止重叠或遗漏元素，并且减少内存分配开销

**临时数组的必要性：**合并两个子数组需额外空间存储中间结果，因此需 O(n) 的临时数组。若尝试原地合并，时间复杂度会退化为 O(n²)，故牺牲空间换时间效率。

4.2 核心伪代码

**图形用户界面, 文本, 应用程序

AI 生成的内容可能不正确。** 文本, 日程表

AI 生成的内容可能不正确。

图11 归并排序伪代码

4.3 复杂度分析

时间复杂度为O ()

4.4 效率测试

使用随机数生成，生成了十万到五十万的数据集，每个数据量都进行了20次测试并取平均值。

以输入规模为10万的数据运行时间为基准点，计算输入规模为其他值的理论运行时间，

结果如表4所示。

表4 归并排序运行时间表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据量 | 100000 | 200000 | 300000 | 400000 | 500000 |
| 单次排序时间（ms） | 15.8854 | 32.9415 | 50.8678 | 69.5914 | 88.3387 |
| 理论时间（ms） | 15.8854 | 33.68359 | 52.20376 | 71.19277 | 90.53042 |
| 相对误差(%) | 0 | 2.20 | 2.55 | 2.24 | 2.42 |

画出不同规模数据的理论运行时间曲线和实测效率曲线，如图12所示。



图12 归并排序曲线

理论运行时间曲线和实测效率曲线几乎重合，误差在合理范围内，符合O**(**)曲线，经验分析和理论分析一致。

5.快速排序

5.1原理描述

分治法：**选择一个基准元素**（通常选择第一个或最后一个元素），通过一趟排序**将数据分成两部分**：一部分的所有数据都比基准元素小，另一部分的所有数据都比基准元素大。然后**对这两部分数据进行快速排序**，直到整个序列有序

将8，16，21，25，25，49作为降序快速排序的例子，如图1所示。

手机屏幕截图

AI 生成的内容可能不正确。

图13 快速排序图示

**实现细节：分治递归结构**

**（1）基准元素选择策略：常见方法：**

首/尾元素：简单但易导致最坏时间复杂度（如数组已有序）。

随机选择：随机选取下标，避免最坏情况，需额外交换操作。

三数取中法：取首、中、尾三元素的中位数，平衡性能与实现复杂度。  
 **核心目标：尽量使分区后左右子数组长度接近，保证递归深度均衡**

**（2）分区操作**

基准固定：选择当前子数组的右端点 right 作为基准值 pivot = arr[right]。

指针初始化：i 指向“小元素区”的末尾（初始为 left-1）。

遍历与交换：遍历 j 从 left 到 right-1。若 arr[j] <= pivot，则 i 右移，交换 arr[i] 和 arr[j]。

基准归位：遍历结束后，将基准值 arr[right] 与 arr[i+1] 交换，最终基准下标为 i+1。

**结果：分区后满足 [left, i] ≤ pivot 且 [i+2, right] ≥ pivot，基准位于正确位置 i+1**

**（3）终止条件：**子数组长度为 0（left > right）或 1（left == right），无需排序

5.2 核心伪代码

**文本

AI 生成的内容可能不正确。**

图14 快速排序伪代码

5.3 复杂度分析

**最坏情况下**, 即待排序记录序列已经按其关键字排好序, 其**递归树成为单支树**, 时间复杂度达O()

**最好情况下**， 时间复杂度为**O**

**平均情况下，**基准元素能够将数组较为均匀地分成两部分（通过优化基准元素的选取）**时间复杂度为O**

5.4 效率测试

使用随机数生成，生成了十万到五十万的数据集，每个数据量都进行了20次测试并取平均值。

以输入规模为10万的数据运行时间为基准点，计算输入规模为其他值的理论运行时间，

结果如表5所示。

表5 快速排序运行时间表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据量 | 100000 | 200000 | 300000 | 400000 | 500000 |
| 单次排序时间（ms） | 9.71909 | 20.5056 | 31.7517 | 43.1331 | 55.1917 |
| 理论时间（ms） | 9.71909 | 20.60848 | 31.93958 | 43.55754 | 55.3888 |
| 相对误差(%) | 0 | 0.49 | 0.58 | 0.97 | 0.35 |

画出不同规模数据的理论运行时间曲线和实测效率曲线，如图15所示。



图15 快速排序曲线

理论运行时间曲线和实测效率曲线几乎重合，误差在合理范围内，符合O**(**)曲线，经验分析和理论分析一致。

（2）TopK算法

1. K次选择排序

1.1 算法原理

进行k轮的选择排序，将最大的k个数交换到最前面。

**实现细节：**只需将选择排序外层n-1次循环改为10次

1.2核心伪代码

文本, 信件

AI 生成的内容可能不正确。

图16 K次选择排序伪代码

1.3复杂度分析

每一趟都通过遍历数组来找到第k大元素，所以时间复杂度是

1.4 效率测试

使用随机数生成，生成了一千万到五千万的数据集，每个数据量都进行了20次测试并取平均值。

以输入规模为一千万的数据运行时间为基准点，计算输入规模为其他值的理论运行时间，

结果如表6所示。

表6 K次快速排序运行时间表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据量 | 10000000 | 20000000 | 30000000 | 40000000 | 50000000 |
| 选择排序时间(ms) | 136.399 | 273.517 | 414.098 | 545.697 | 694.876 |
| 选择排序理论（ms） | 136.399 | 272.798 | 409.197 | 545.596 | 681.995 |
| 相对误差(%) | 0 | 0.26 | 1.19 | 0.01 | 1.88 |

画出不同规模数据的理论运行时间曲线和实测效率曲线，如图16（2）所示



图16（2）

理论运行时间曲线和实测效率曲线几乎重合，误差在合理范围内，符合O**(**)曲线，经验分析和理论分析一致。

2. 堆调整

2.1 算法原理

1.构建大小为k的小根堆（最小的元素在堆顶）时间复杂度O(k)

2.遍历剩下n-k个元素，如果大于堆顶，替换堆顶元素，调整堆

时间复杂度O(nlogk)。

**实现细节：**

**堆的存储结构：**使用数组模拟完全二叉树，父子节点索引关系为：父节点 i → 左子节点 2i + 1，右子节点 2i + 2，子节点 i → 父节点 ⌊(i-1)/2⌋。

**建堆优化：**从**最后一个非叶子节点**（索引 K/2 - 1）开始，自底向上执行下沉操作，时间复杂度 O(K)。

**重复元素处理：**若新元素等于堆顶，跳过调整（避免无效操作）

2.2核心伪代码

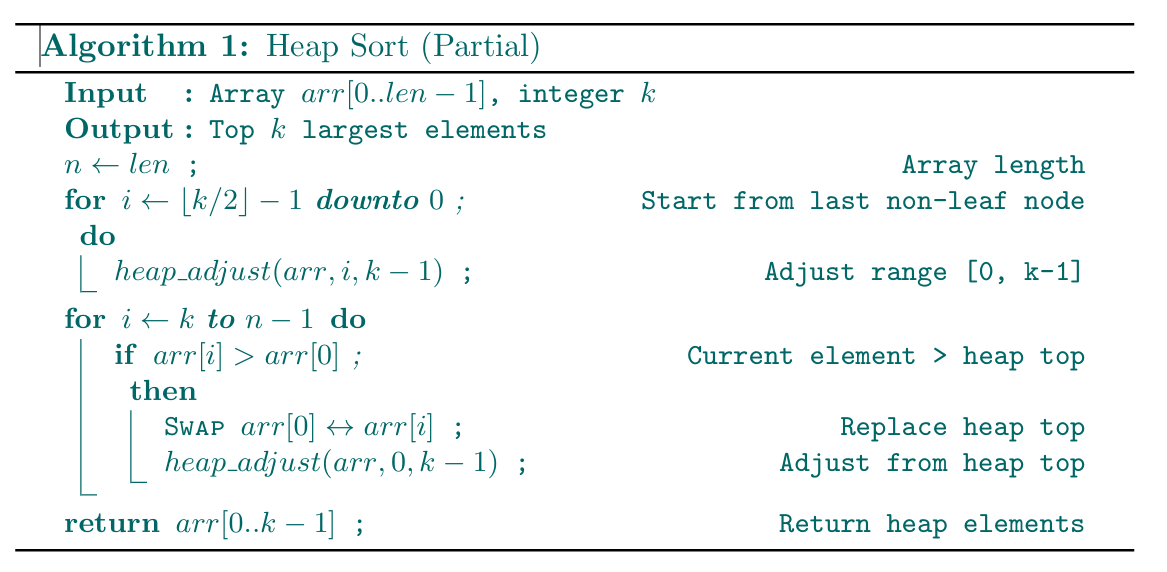


图17 堆排序伪代码

2.3复杂度分析

构建大小为k的小根堆，**时间复杂度为O(k)**，遍历n-k个元素，**调整堆的时间复杂度O(nlogk)**

所以**时间复杂度为O(nlogk)**

2.4 效率测试

使用随机数生成，生成了一千万到五千万的数据集，每个数据量都进行了20次测试并取平均值。

以输入规模为一千万的数据运行时间为基准点，计算输入规模为其他值的理论运行时间，

结果如表7所示。

表7 堆排序运行时间表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据量 | 10000000 | 20000000 | 30000000 | 40000000 | 50000000 |
| 堆调整时间（ms） | 12.2619 | 24.5156 | 36.8735 | 48.3009 | 62.1179 |
| 堆调整理论（ms） | 12.2619 | 24.5238 | 36.7857 | 49.0476 | 61.3095 |
| 相对误差(%) | 0 | 0.03 | 0.23 | 1.52 | 1.31 |

画出不同规模数据的理论运行时间曲线和实测效率曲线，如图18所示。



图18 堆调整曲线

理论运行时间曲线和实测效率曲线几乎重合，误差在合理范围内，符合O**(**)曲线，经验分析和理论分析一致。

3. 快速选择算法

3.1 算法原理

根据快速排序基准值的位置与 K 的关系，决定下一步处理哪一部分：

1.如果基准值的位置大于 K，则在左侧继续快排

2.如果基准值的位置小于 K，则在右侧继续查找，并调整 K 的值

**3.如果基准值的位置正好是第 K 个元素，则左侧就是前 K 大元素。**

**实现细节：**有选择的分治递归

**分治与分区：**基于快速排序的分区思想，但仅递归处理包含目标K的子数组，避免完全排序。

**基准选择：**随机选取基准元素（Pivot），将数组划分为左半部分（≤Pivot）和右半部分（≥Pivot）。根据K的位置决定递归左/右子数组，将TopK设置在左边在调整k时更方便，只需要根据基准位置和K位置比较即可（不用再计算长度）。

**终止条件：**当基准的索引恰好为 k时，直接返回，前0~k-1个元素便是TopK。

图示

AI 生成的内容可能不正确。

图19（1）快速选择图示

3.2核心伪代码

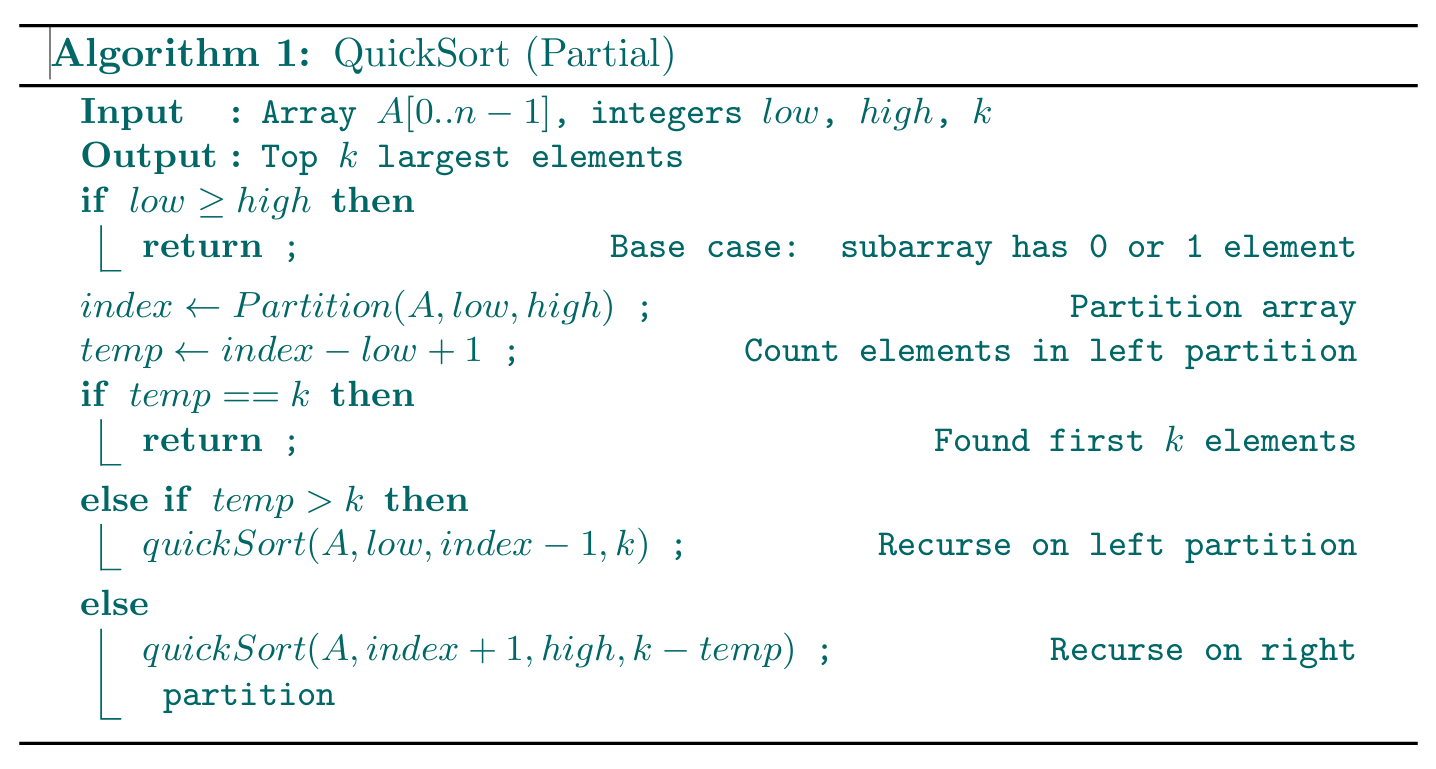


图19（2） 快速选择排序伪代码

3.3复杂度分析

**平均情况：**设每次划分后处理的子数组期望长度为 αn（0<α<1），则时间复杂度满足递推式：**T(n)=T(αn)+O(n),**

若每次划分后子数组长度期望为原数组的 1/2（即平均平衡划分），则递推式为： .

根据主定理（Master Theorem）或展开递推式：

最坏情况：每次划分选择的枢轴均为当前数组的最小或最大值，T(n)=T(n-1)+O(n)= O()

时间复杂度依赖划分的平衡性

3.4 效率测试

使用随机数生成，生成了一千万到五千万的数据集，每个数据量都进行了20次测试并取平均值。 结果如表8所示。

表8 快速选择算法运行时间表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据量 | 10000000 | 20000000 | 30000000 | 40000000 | 50000000 |
| 快速选择时间（ms） | 8.4125 | 16.9243 | 25.7421 | 34.12 | 42.797 |
| 快速选择理论（ms） | 8.4125 | 16.825 | 25.2375 | 33.65 | 42.0625 |
| 相对误差(%) | 0 | 0.59 | 1.99 | 1.39 | 1.74 |



图19（3）

**三．实验结果和分析**

1.五个排序算法结果比较和分析

**运行时间：冒泡>>选择>插入>>归并>快速**，如图20所示



图20

（1）O()算法对比

表9 O()算法运行时间表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据量 | 100000 | 200000 | 300000 | 400000 | 500000 |
| 选择排序时间(ms) | 6170.62 | 25068.6 | 56081.8 | 100140 | 156547 |
| 冒泡排序时间  （ms） | 23236.8 | 96038.6 | 217758 | 389013 | 609887 |
| 插入排序时间  （ms） | 5195.16 | 20989.3 | 47182.9 | 83884 | 130862 |



图21

**分析：运行时间：冒泡>选择≈插入**

**冒泡：**比较开销：n(n-1)/2 交换开销：n(n-1)/2

**选择**：比较开销：n(n-1)/2(固定) 交换开销：2\*(n-1)

**插入：**比较开销：n(n-1)/4 交换开销：n(n-1)/4.

**比较开销：**冒泡=选择>插入

**交换开销：**冒泡>插入>选择

**因此，冒泡效率最低**

插入排序和选择排序接近，虽然选择排序把交换次数降低到最低，但是比较次数大于插入排序,

**平均情况下，插入排序比选择排序更快。**

（2）O()算法对比

表10 O()算法运行时间表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据量 | 100000 | 200000 | 300000 | 400000 | 500000 |
| 快速排序时间(ms) | 9.71909 | 20.5056 | 31.7517 | 43.1331 | 55.1917 |
| **归并排序时间**  （ms） | 15.8854 | 32.9415 | 50.8678 | 69.5914 | 88.3387 |



图22

**分析：运行时间：快速<归并，主要原因：**

**1.** **常数因子更小**

虽然两者的渐进时间复杂度相同，但快速排序的单步操作更简单，常数因子更小：

* 快速排序 的核心是 分区操作（Partition），仅需简单的元素比较和交换。
* 归并排序 的核心是 合并操作（Merge），需要额外的内存分配、元素复制和指针移动。例如：

合并两个有序子数组时，需要将元素逐个复制到临时数组，再写回原数组，这种额外的数据移动会显著增加实际运行时间。

2. 数据移动量

* 快速排序 通过交换元素实现分区，仅移动不符合顺序的元素，且交换次数通常少于归并排序的复制次数。
* 归并排序 即使子数组已有序，仍需将所有元素复制到临时数组，再合并回原数组，导致 冗余的数据移动。

**结论：快速排序的常数因子更小、缓存局部性更好、数据移动更少，使其在大多数实际场景中优于归并排序。**但归并排序的稳定性（稳定排序）和严格*O*(*n*log*n*) 的时间复杂度，使其在特定场景（如链表排序、外部排序）中仍有优势

2.TopK问题结果比较和分析

**运行时间：快速选择<堆调整<选择排序**，如图23所示



图23

表11 TopK算法运行时间表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据量 | 10000000 | 20000000 | 30000000 | 40000000 | 50000000 |
| 选择排序时间(ms) | 136.399 | 273.517 | 414.098 | 545.697 | 694.876 |
| 堆调整时间  （ms） | 12.2619 | 24.5156 | 36.8735 | 48.3009 | 62.1179 |
| 快速选择时间  （ms） | 23.4125 | 48.2451 | 71.1241 | 94.6436 | 119.7457 |

**分析：**

**选择排序：**每次遍历都需访问整个数组，比较开销较大，对TopK进行排序，效率最低

**堆排序和随机排序只是找出TopK，而不排序**

**堆调整：仅需维护大小为 K 的堆，较高效**

**快速选择：平均情况：O(n)（通过优化每次基准值接近中间值）**

**快速选择<堆调整：**主要得益于其**基准值优化的实现**、缓存友好的内存访问模式，以及**均匀数据分布下的高效分区**，堆调整的理论时间复杂度优势被其常数因子和缓存劣势抵消

**当k越接近n时，快速排序会比堆排序更高效**

**四．经验总结**

一、五大算法实现和对比

1.时间复杂度的经验验证

选择排序、冒泡排序、插入排序的平均时间复杂度为O(n²)，其实际运行时间与输入规模n呈现明显的平方增长关系。

合并排序和快速排序的平均时间复杂度为O(n log n)，实际运行时间随n增长接近线性增长。

2.实际性能对比理论

**O(n²)算法**在n>1万时效率显著下降，例如冒泡排序在n=50万时耗时超过100秒，而快速排序仅需0.5秒

**快速排序**在随机数据场景下表现最优，因常数因子较小；**合并排序**频繁的拷贝和合并数组，实际耗时略高于快速排序

**插入排序**在小规模数据（n<1000）或部分有序数据中表现优于其他O(n²)算法

3.关键差异因素

**算法常数因子：**同为时间复杂度为O(nlogn),但合并排序的合并操作需频繁内存分配，而快速排序的原地分区减少内存开销，算法常数因子不同，运行时间也不同

**硬件缓存效应：**插入排序对局部性原理的利用优于选择排序，实测性能优于纯O(n²)理论预期

**递归和迭代差异：**快速排序的递归版本在n=1e6时因栈调用开销比迭代版本慢15%~20%。

堆排序的迭代实现（Heapify）避免了递归栈溢出风险，适合处理n≥1e9数据

二、TOP K问题的高效解法

1.算法的优化应用

朴素的堆排序和快速排序也能进行较快的排序，但**如果对针对特定的问题，对算法进行优化应用能发挥出算法更高效的性能**，比如只对K个小顶堆维护的堆调整算法和每次只对前K项所在位置进行快速排序的快速选择算法比一般的O(nlogn)排序有更显著的性能。

2.算法正确性验证

在**小规模数据**（如n=1kw-5kw）中，通过对比不同算法输出，验证结果的正确性。

**边界测试**：输入数据全为相同值或严格递增/递减时，算法仍能稳定输出正确TOP K。

三、算法选择的关键原则

1.输入规模与场景适配

**小规模数据（n<1k）**：优先选择插入排序或快速排序的优化实现。

**大规模随机数据**：快速排序（随机化枢轴+迭代实现）或堆排序（稳定排序）。

**数据部分有序**：插入排序效率接近O(n)，优于其他O(n²)算法

2.空间与时间的权衡

**空间充足**：归并排序的稳定性和可预测性更优。

**空间受限**：快速排序的原地分区或堆调整（仅需O(k)空间）更适用。

|  |
| --- |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  年 月 日 |
| 备注： |

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后10日内。