**深 圳 大 学 实 验 报 告**

**课程名称：­ 算法设计与分析**

**实验项目名称： 算法效率分析——排序与TopK问题**

**学院： 数学与统计学院**

**专业： 信息与计算科学（数学与计算机实验班）**

**指导教师： 刘刚**

**报告人： 王曦 学号： 2021192010 班级： 数计班**

**实验时间： 2023年02月27日**

**实验报告提交时间： 2023年02月27日**

**教务处制**

|  |  |
| --- | --- |
| **实验目的**   1. 掌握选择排序、冒泡排序、归并排序、快速排序、插入排序的算法原理. 2. 掌握不同排序算法的时间效率的经验分析方法, 验证理论分析与经验分析的一致性. | |
| **实验环境**   1. Windows 11操作系统. 2. VSCode IDE. 3. MinGW-w64编译器, GNU C++17, O2优化. | |
| **实验内容：**   1. 实现选择排序、冒泡排序、归并排序、快速排序、插入排序算法. 2. 以待排序数组大小n为输入规模, 固定n, 随机产生20组测试样本, 统计不同排序算法在20个样本上的平均运行时间. 3. 分别以n = 1000、n = 10000、n = 100,000、n = 1,000,000、n = 10,000,000(千万规模选做重复2的实验. 4. 画出不同排序算法在20个随机样本的平均运行时间与输入规模n的关系,如图1所示.     图1: 快速排序与冒泡排序的时间效率与输入规模的关系   1. 画出理论效率分析的曲线和实测的效率曲线, 注意: 由于实测效率是运行时间, 而理论效率是基本操作的执行次数, 两者需要进行对应关系调整. 调整思路: 以输入规模为10000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, 画出不同规模数据的理论运行时间曲线, 并与实测的效率曲线进行比较. 经验分析与理论分析是否一致? 如果不一致, 请解释存在的原因. 2. 实验数据可通过随机数产生并保存到文件中. 每次排序时直接从文本中读入数据. |
| **实验步骤：**  （用文字描述实验过程，并用截图辅助说明）   1. **产生随机数**   为保证后续实验中各排序算法所用的数据相同, 防止因随机的数据不同而导致运行时间有差异, 考虑将随机的数据生成到文件中, 后续实验从文件中读取输入数据.  因本次实验的算法与元素的值域无关, 且不涉及算法的稳定性, 故下面的输入数据的元素范围为1 ~ n. 因部分排序算法的性能受输入数据本身的影响, 在实验用到的20组数据中, 前3组分别是全为1的数据、1~n的升序序列、1~n的降序序列, 其余17组数据随机生成, 以反映排序算法的性能与输入数据的关系.   1. 编写附件中”排序实验代码\gen.cpp”的随机数据生成器. 2. 在”排序实验代码”目录下创建”test1000”、”test10000”、”test100000”、”test1000000”四个文件夹, 分别用于存放n = 1000、n = 10000、n = 100,000、n = 1,000,00对应的随机数据. 3. 如图2, 编译并运行上述随机数生成器, 输入n值以得到对应的随机数据.     图2: 随机数据的生成方法   1. 得到图3所示的随机数据.     图3: 得到的随机数据   1. **约定** 2. 下面规定要欲排序的序列a[]长度为n.   2. 以将序列非降序排列为例.  3. 下面的程序运行时间均不包括数据输入的时间.  4. 为修正理论运行时间和实际运行时间的关系, 下面以n = 10000时的实际运行为基准点, 用时间复杂度计算出其他n值对应的理论运行时间, 计算实际运行时间与理论运行时间的相对误差.   1. **选择排序** 2. 选择排序原理: 每次找出后缀a[i ... n]的最小值, 将其与a[i]交换. 3. 伪代码:     图4: 选择排序伪代码   1. C++代码见”排序实验代码\sortings.cpp”中的selectSorting()函数. 2. 算法实际运行时间与理论运行时间  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 数据规模n | 1000 | 10000 | 100,000 | 1,000,000 | | 实际运行时间(ms) | 2.250 | 161.850 | 16345.300 | 1646014.500 | | 理论运行时间(ms) | 1.619 | 161.850 | 16185.000 | 1618500.000 | | 误差 | 39.0% | 0.0% | 1.0% | 1.7% |   下面以计算n = 1000时的误差为例.  因插入排序的平均时间复杂度为, 以n = 10000时的实际运行时间为基准时,因  (1),  则n = 1000时的理论运行时间  (2),  进而误差  (3).  图象:    图5: 选择排序的运行时间与数据规模的关系   1. 分析: 选择排序的最坏时间复杂度为, 平均时间复杂度为. 因实验所用的数据大部分是随机生成的, 故程序的运行时间反映算法的平均时间复杂度. 观察到选择排序的运行时间与数据规模的关系图象呈抛物线. 由图象知: 实际运行时间与理论运行时间接近, 验证了选择排序的平均时间复杂度. n = 1000时实际运行时间与理论运行时间相差较大可能与n = 1000时数据较少、与数据规模更大的数据的随机化程度不同有关, 还可能与数据较少时前3组特殊数据的作用被放大有关. 2. **冒泡排序** 3. 冒泡排序原理: 每次检查相邻两元素, 若前面的元素与后面的元素满足排序条件, 则交换两相邻元素. 4. 伪代码:     图6: 冒泡排序伪代码   1. C++代码见”排序实验代码\sortings.cpp”中的bubbleSorting()函数. 2. 算法实际运行时间与理论运行时间  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 数据规模n | 1000 | 10000 | 100,000 | 1,000,000 | | 实际运行时间(ms) | 1.850 | 147.250 | 14681.150 | 1479862.500 | | 理论运行时间(ms) | 1.473 | 147.250 | 14725.000 | 1472500.000 | | 误差 | 25.6% | 0.0% | 0.3% | 0.5% |   图象:    图7: 冒泡排序的运行时间与数据规模的关系   1. 分析: 冒泡排序的最坏时间复杂度为, 平均时间复杂度为. 因实验所用的数据大部分是随机生成的, 故程序的运行时间反映算法的平均时间复杂度. 观察到冒泡排序的运行时间与数据规模的关系图象呈抛物线. 由图象知: 实际运行时间与理论运行时间接近, 验证了冒泡排序的平均时间复杂度. n = 1000时实际运行时间与理论运行时间相差较大可能与n = 1000时数据较少、与数据规模更大的数据的随机化程度不同有关, 还可能与数据较少时前3组特殊数据的作用被放大有关. 2. **归并排序** 3. 归并排序原理: 用分治的思想, 将序列分段排序后合并. 合并两有序序列的过程:从前往后枚举a[i]和b[j], 将最小值放入数组c[]中, 重复该过程直至a[]和b[]中有一个序列被枚举完, 此时将另一序列的剩余元素放入c[]中. 4. 伪代码:   QT@FS6]SP3(2425[7(8A$OP  图8: 归并排序伪代码   1. C++代码见”排序实验代码\sortings.cpp”中的mergeSorting()函数. 2. 算法实际运行时间与理论运行时间  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 数据规模n | 1000 | 10000 | 100,000 | 1,000,000 | | 实际运行时间(ms) | 0.150 | 2.250 | 19.500 | 211.150 | | 理论运行时间(ms) | 0.169 | 2.250 | 28.125 | 337.500 | | 误差 | -11.2% | 0.0% | -30.7% | -37.4% |   下面以计算n = 1000时的误差为例.  因插入排序的平均时间复杂度为, 以n = 10000时的实际运行时间为基准时,因  (4),  则n = 1000时的理论运行时间  (5),  进而误差  (6).  图象:    图9: 归并排序的运行时间与数据规模的关系   1. 分析: 归并排序的时间复杂度为, 与选择排序和冒泡排序相比, 运行时间上明显加快. 因实验所用的数据大部分是随机生成的, 故程序的运行时间反映算法的平均时间复杂度. 因数据规模n较小时, 对时间复杂度的贡献可视为常数, 故从图象上看, 归并排序的运行时间与数据规模几乎成线性关系. 因实验中的归并排序采用递归实现, 而理论上的时间复杂度未考虑函数调用带来的开销, 则理应实际运行时间高于理论运行时间. 但实验结果显示, 实际运行时间低于理论运行时间, 且随数据规模的增大, 两时间的差距越大, 这可能是因为编译器开启了O2优化后降低了函数递归调用的开销, 且优化了归并排序的汇编语言, 使其效率提高. 2. **快速排序** 3. 快速排序原理: 快速排序用分治的思想, 先取一个元素pivot作为比较标准, 将小于它的元素交换到它的左边, 将大于它的元素交换到它的右边, 再递归到左右两个序列进行快速排序. 4. 伪代码:   }4TK_U}LKY515U`XKPGW(AQ  XWC]})~8E(XOHLKY]0GLC%Y  图10: 快速排序伪代码   1. C++代码见”排序实验代码\sortings.cpp”中的quickSorting()函数. 2. 算法实际运行时间与理论运行时间  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 数据规模n | 1000 | 10000 | 100,000 | 1,000,000 | | 实际运行时间(ms) | 0.200 | 2.000 | 15.450 | 170.200 | | 理论运行时间(ms) | 0.150 | 2.000 | 25.000 | 300.000 | | 误差 | 33.3% | 0.0% | -38.2% | -43.3% |   图象:    图11: 快速排序的运行时间与数据规模的关系   1. 分析: 快速排序的最坏时间复杂度为, 平均时间复杂度为. 因实验所用的数据大部分是随机生成的, 几乎未达到快速排序的最坏情况, 且因数据规模n较小时, 对时间复杂度的贡献可视为常数, 故从图象上看, 归并排序的运行时间与数据规模几乎成线性关系, 验证了快速排序的平均时间复杂度. 从实际运行时间上看, 快速排序在随机数据上的整体性能高于与之相同平均时间复杂度的归并排序. 因实验中的快速排序采用递归实现, 而理论上的时间复杂度未考虑函数调用带来的开销, 则理应实际运行时间高于理论运行时间. 但实验结果显示, 实际运行时间低于理论运行时间, 且随数据规模的增大, 两时间的差距越大, 这可能是因为编译器开启了O2优化后降低了函数递归调用的开销, 且优化了快速排序的汇编语言, 使其效率提高. 2. **插入排序** 3. 插入排序原理: 将待排序的序列分为已排序和未排序两部分, 每次取未排序部分的第一个元素, 将其插入到已排序部分的正确位置. 4. 伪代码:     图12: 插入排序伪代码   1. C++代码见”排序实验代码\sortings.cpp”中的insertSorting()函数. 2. 算法实际运行时间与理论运行时间  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 数据规模n | 1000 | 10000 | 100,000 | 1,000,000 | | 实际运行时间(ms) | 2.900 | 225.000 | 22926.150 | 2281500.000 | | 理论运行时间(ms) | 2.250 | 225.000 | 22500.000 | 2250000.000 | | 误差 | 28.9% | 0.0% | 1.9% | 1.4% |   图象:    图13: 插入排序的运行时间与数据规模的关系   1. 分析: 插入排序的最坏时间复杂度为, 平均时间复杂度为. 因实验所用的数据大部分是随机生成的, 故程序的运行时间反映算法的平均时间复杂度. 观察到插入排序的运行时间与数据规模的关系图象呈抛物线. 由图象知: 实际运行时间与理论运行时间接近, 验证了插入排序的平均时间复杂度. n = 1000时实际运行时间与理论运行时间相差较大可能与n = 1000时数据较少、与数据规模更大的数据的随机化程度不同有关, 还可能与数据较少时前3组特殊数据的作用被放大有关. |
| **实验结果：**  （给出个人对结果的分析、结论）   1. 各种排序算法的实际运行时间与数据规模的关系     图14: 排序算法的实际运行时间与数据规模的关系   1. 不同排序算法的性能比较 2. 在数据规模较小时, 不同的排序方法效率差别不大, 且平均时间复杂度较低的算法可能因实现较为繁琐而导致时间复杂度的常数过大, 进而实际运行时间高于平均时间复杂度较高的算法. 3. 数据规模较大时, 平均时间复杂度的算法的实际运行时间明显低于平均时间复杂度和的算法. 4. 与归并排序相比, 快速排序的整体性能更优, 实际运行时间更短. 但归并排序的时间复杂度是稳定的, 而快速排序在特殊的输入数据下可能达到最坏的时间复杂度. 5. 在选择排序、冒泡排序和插入排序中, 冒泡排序的整体性能优于另外两个排序算法, 插入排序的整体性能最差. 这是因为插入排序在输入数据基本有序时效率较高, 而随机生成的数据大部分是无序的. 选择排序的最优时间复杂度和最坏时间复杂度都是, 运行时间介于冒泡排序和插入排序之间. 冒泡排序的最坏时间复杂度为, 最优时间复杂度为, 且与输入数据有关. 6. 事实上, 实验中的排序算法的实现仍有优化空间, 如减小非必要的比较次数、将递归实现改为非递归实现等.   **[思考题] Top K问题**  现有10亿个数据, 每个数据4个字节. 现需快速挑选出最大的10个数, 并在小规模数据上验证算法的正确性.   1. 思路: 2. 显然在大量数据中选出最大或最小的几个数不必将所有数据排序, 但可利用排序过程来实现. 因数据规模n = 10亿较大, 故考虑平均时间复杂度的算法. 因归并排序和快速排序都可能存在递归深度过大而导致效率低下的问题, 参考C++的algorithm库中std::sort()函数的优化方式, 考虑堆排序. 具体地, 对数据建立大根堆, 做10次弹出堆顶元素并调整堆的形态的操作即可选出最大的10个数. 建堆时间复杂度, 每次调整时间复杂度, 总时间复杂度. 3. 事实上, 无需将所有元素都存到堆中, 只需维护一个包含k = 10个元素的堆即可. 具体地, 将输入数据的前10个元素建堆. 对之后的每个元素, 若它比当前堆中的最小元素小, 则将堆的最小元素置为当前元素并调整堆的形态. 遍历完输入数据的所有元素后, 做10次弹出堆顶元素并调整堆的形态的操作即可选出最大的10个数. 建堆时间复杂度, 每次调整使时间复杂度, 最坏后面的个元素都需入堆(如序列升序的情况), 则总时间复杂度   .   1. 实现   见附件中”排序实验代码\heap.cpp”中的heapSorting()函数.   1. 小数据验证   数据采用gen.cpp生成的20组n = 1000的随机数, 将用堆选出的最大的10个数与将序列非升序排列后输出前10个数比较, 验证结果的正确性. 实验结果如下图所示:  图15: 用堆排序选出最大的10个数的实验结果   1. 10亿个数据的实际运行效率   采用mt19937, 随机生成10亿个数据, 同时采用上述方法选出前10大的元素, 实际运行时间为67105.000 ms. |
| **实验小结：**  （实验中出现问题的解决方法，实验心得体会等）  本次实验证明了选择排序、冒泡排序、快速排序、归并排序、插入排序的平均时间复杂度, 证明了插入排序与输入数据无关, 而选择排序、冒泡排序的时间复杂度与输入数据有关, 在较为随机的数据上易接近的最优时间复杂度.  在本次实验中理解到了平均时间复杂度、最优时间复杂度、最坏时间复杂度的区别,  体会到了输入数据的规模和本身的性质对排序算法效率的影响, 了解到了算法竞赛的数据中除随机数据外, 还需构造一些特殊的数据来将非正解卡到最坏时间复杂度的原因.  本次实验证明了快速排序名称的由来: 整体性能优于其它排序算法. 但它在特殊的输入数据下会达到的最坏时间复杂度, 这一点比稳定的时间复杂度的归并排序略显不足.  在思考题中了解到了在大量数据中取出最大或最小的几个元素时可用堆实现, 这样时间复杂度低, 且递归深度小, 开销小, 效率高. 若将思考题中的数据范围再扩大100倍, 则这些数据的空间会达到TB的级别, 此时无法将数据整体放入内存, 必须借助外存与内存的数据交换.  若TopK问题中数据的值域较小或可离散化到较小的值域, 则还可使用计数排序和桶排序解决. |
| **指导教师批阅意见：**  **成绩评定：**  **指导教师签字：**  年 月 日 |
| 备注： |