**深 圳 大 学 实 验 报 告**

**课程名称： 算法设计与实现**

**实验项目名称： 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**

**指导教师： 杨烜老师**

**报告人： 李若龙 学号：2018171028 班级： 计科02**

**实验时间： 2020/3/13**

**实验报告提交时间：**

**教务处制**

1. 问题描述：
2. 实现选择排序、冒泡排序、合并排序、快速排序、插入排序算法；

以待排序数组的大小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**；

分别以n=10000, n=20000, n=30000, n=40000, n=50000等等，重复2的实验，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

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。

1. 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。
2. 求解问题的算法原理描述

对于问题1，我们使用排序，排序算法的原理就是数据的交换，不同的排序算法，不同点在于元素交换的规则不同

冒泡：只要前比后大，交换

选择：选择最值元素交换到区间前面

插入：只要元素比key大就向后挪

归并：双指针，谁指的元素小就添加到答案，指针后移

快排：双指针，分别交换元素小于key，大于key的值

对于问题2，我们有三种方法，对应不同复杂度

1. 降序排序，取前10名
2. 顺序查找，找10次
3. 维护大小为10的小顶堆，遍历数组，如果堆大小小于10，任意元素均可入堆，如果堆大小为10，当前元素比堆顶元素大，堆顶弹出，当前元素入堆，如果当前元素比堆顶小，不管它

实验总体思路：

1.申请k个变量记录k种排序算法的运行时间

2.对不同的数据规模(10000 ~ 50000)，重复以下操作：

循环sample次，产生规模为batch的数据，复制k份，分别用k种排序算法，记录运行时间之和，

3.对结果取平均值，平均时间是总时间/sample

说明：

结果单位为秒，使用双精度浮点数存储

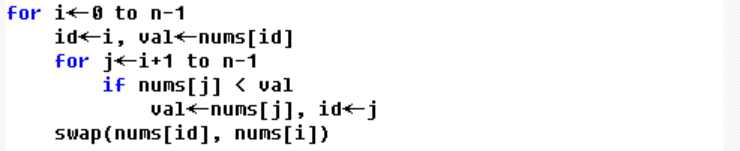
复制数据的时间并不会被记录，计时开始的标志是进入排序之前的语句

Sample取20， batch分别取10000， 20000， 30000， 40000， 50000

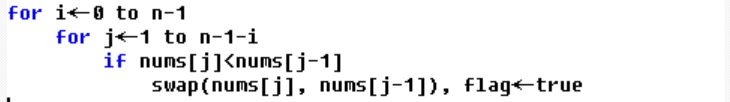
K取6，分别对应选择，插入，冒泡(2)，归并，快排

1. 算法实现的核心伪代码

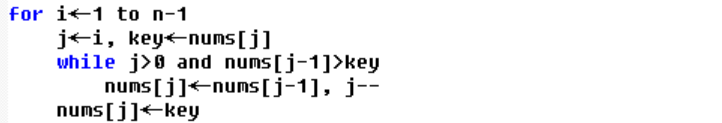
选择：



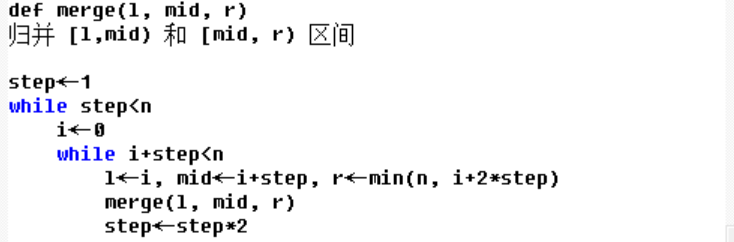
冒泡



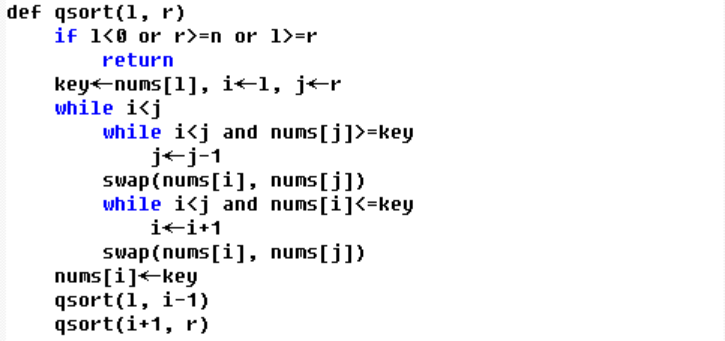
插入



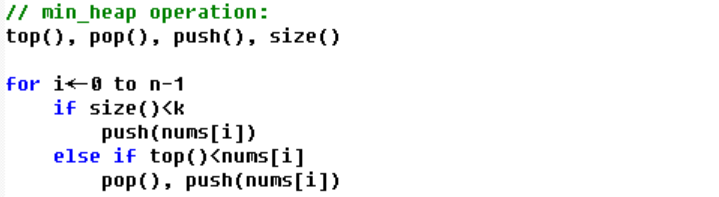
二路归并



快排（递归）



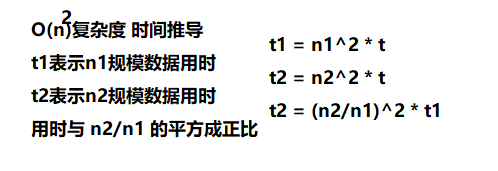
小顶堆 TopK：



1. 算法测试结果及效率分析

选择排序

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 选择 | 10000 | 20000 | 30000 | 40000 | 50000 |
| 实际 | 0.23925 | 1.00005 | 2.2248 | 3.98745 | 6.0222 |
| 理论 | 0.23925 | 0.957 | 2.15325 | 3.828 | 5.98125 |



**图像上：**

图像符合二次增长

**规模-时间变化：**

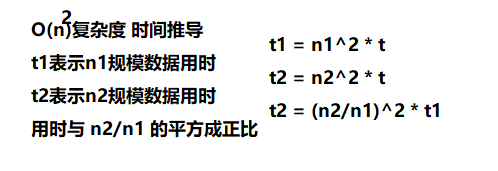
根据上面的时间推导，规模变为原理的n倍时，时间要变为原理的n^2 倍，基本符合

冒泡排序

实际1表示不带提前结束的冒泡排序

实际2表示带提前结束的冒泡排序

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 冒泡 | 10000 | 20000 | 30000 | 40000 | 50000 |
| 实际1 | 0.8191 | 3.3387 | 7.29745 | 12.2325 | 19.3906 |
| 实际2 | 0.7923 | 3.22215 | 7.0507 | 12.2555 | 19.4145 |
| 理论 | 0.8191 | 3.2764 | 7.3719 | 13.1056 | 20.4775 |



**图像上：**

图像符合二次增长

**规模-时间变化：**

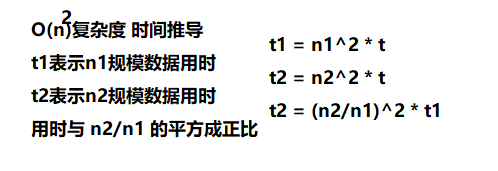
根据上面的时间推导，规模变为原理的n倍时，时间要变为原理的n^2 倍，基本符合

**差异解释：**

实际2曲线代表了带提前结束的冒泡排序，可以看到这个优化的作用微乎其微，但是能够微微优于一般情况，因为大量随机的数据，而且数据量巨大，大多数情况都不能提前结束

插入排序

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 插入 | 10000 | 20000 | 30000 | 40000 | 50000 |
| 实际 | 0.18335 | 0.70335 | 1.6581 | 2.8475 | 4.62105 |
| 理论 | 0.18335 | 0.7334 | 1.65015 | 2.9336 | 4.58375 |



**图像上：**

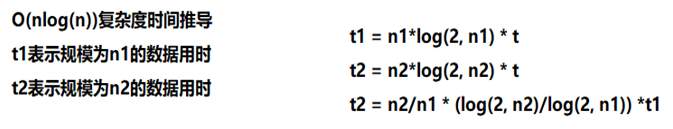
图像符合二次增长

**规模-时间变化：**

根据上面的时间推导，规模变为原理的n倍时，时间要变为原理的n^2 倍，基本符合

归并排序

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 二路归并 | 10000 | 20000 | 30000 | 40000 | 50000 |
| 实际 | 0.0076 | 0.01505 | 0.02315 | 0.03405 | 0.04105 |
| 理论 | 0.0076 | 0.016344 | 0.02552 | 0.034976 | 0.04464 |



图像上看，几乎是线性的，但其实是因为n的规模还不够大，体现不出对数函数的曲线特性

数据上看，规模扩大n2/n1倍，时间扩大为原来的n2/n1\*log(2, n2)/log(2, n1)倍，几乎符合

**为什么时间在估计时间附近波动？**

因为时间相比O(n^2)的算法，用时小很多，在10s级别的运行时间，一毫秒带来的误差微乎其微，但是在15ms的运行时间，1ms带来的误差就比较大了

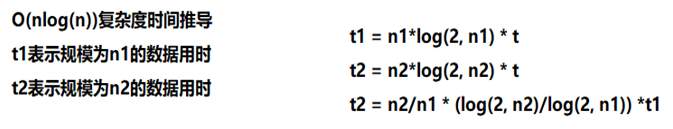
**为什么不使用递归实现？**

对于递归实现的归并，是对递归树的自顶向下的先序遍历

而非递归每次步长倍增，是对递归树的自底向上的层次遍历，结果是一样的，但是相比于递归，时间和空间的开销大大减少

大规模：归并排序

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 大规模归并 | 1000000 | 2000000 | 3000000 | 4000000 | 5000000 |
| 实际 | 0.85725 | 1.69705 | 2.59385 | 3.44425 | 4.403 |
| 理论 | 0.85725 | 1.800519 | 2.776256 | 3.773077285 | 4.785577 |

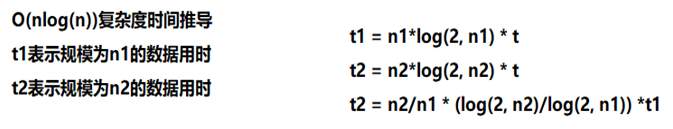


比预计快的原因：

使用非递归写法，大大节省递归的开销

快速排序

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 快速排序 | 10000 | 20000 | 30000 | 40000 | 50000 |
| 实际 | 0.0021 | 0.00435 | 0.00685 | 0.00985 | 0.01205 |
| 理论 | 0.0021 | 0.004516 | 0.007051 | 0.009664 | 0.012335 |



**图像上：**

图像上看，几乎是线性的，但其实是因为n的规模还不够大，体现不出对数函数的曲线特性

**规模-时间关系：**

数据上看，规模扩大n2/n1倍，时间扩大为原来的n2/n1\*log(2, n2)/log(2, n1)倍，几乎符合

大规模：快速排序

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 大规模：快排 | 1000000 | 2000000 | 3000000 | 4000000 | 5000000 |
| 实际 | 0.31695 | 0.66925 | 1.11865 | 1.613 | 2.1897 |
| 理论 | 0.31695 | 0.665704 | 1.026462 | 1.395015276 | 1.769365 |

时间略微超过预计的原因分析：

使用的是递归写法，递归造成额外的空间和时间开销

每次挪动元素的时候，使用的是swap，相比直接赋值， 整体算法的常数开销过大

**数组充满重复元素**，对快排效率有一定影响

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 快排：非递归 | 1000000 | 2000000 | 3000000 | 4000000 | 5000000 |
| 实际 | 0.46425 | 0.9824 | 1.5803 | 2.2476 | 2.9952 |
| 理论 | 0.46425 | 0.984401 | 1.525652 | 2.080605081 | 2.645747 |

可以看到换成非递归的实现，情况会好一点，但是仍然超过预计一点点，可能是因为数据量变大，重复元素变多导致的效率降低。。。

最坏情况下O(nlog(n))级别算法的表现：



最坏情况，即数组是降序的情况，可以看到快速排序时间复杂度退化到O(n^2)，因为每次操作只能确定一个元素的位置，而且进行了n-1次移动，除此之外，递归树的深度达到了O(n)，因为要递归调用n-1次

**无论我使用递归还是非递归的快速排序，数据规模超过十万都会照成栈空间的溢出导致程序exit，所以只能在1000-10000级别进行测量**

而插入排序，因为每次分的空间确定，即两边走分两半，所以不管是递归还是非递归的归并排序，空间复杂度都是O(log(n))，而且时间复杂度也可以较好的保持在O(nlog(n))，因为每趟操作与数据无关（而且因为时间太短，测量有较大的误差）

数据总览

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 10000 | 20000 | 30000 | 40000 | 50000 |
| 选择 | 0.23925 | 1.00005 | 2.2248 | 3.98745 | 6.0222 |
| 冒泡（无提前结束 | 0.8191 | 3.3387 | 7.29745 | 12.2325 | 19.3906 |
| 冒泡（提前结束 | 0.7923 | 3.22215 | 7.0507 | 12.2555 | 19.4145 |
| 插入 | 0.18335 | 0.70335 | 1.6581 | 2.8475 | 4.62105 |
| 二路归并 | 0.0076 | 0.01505 | 0.02315 | 0.03405 | 0.04105 |
| 快速排序 | 0.0021 | 0.00435 | 0.00685 | 0.00985 | 0.01205 |

分析：

可以看到复杂度为O(n^2)的算法明显随规模增大，时间成平方上升，而O(n log(n))的算法，消耗的时间非常的少，以至于在对比下形成一条“直线”

**冒泡排序同为O(n^2)，但是为什么和选择/插入排序相差那么大？**

1. 每一趟内循环，都能准确的确定一个元素的位置，但是冒泡的内循环，相比于选择/插入，需要不停的交换数据，而选择只交换最终的两个位置，插入则是数据的挪动（挪动数据一条语句，但是交换数据是三条）
2. 设置了flag表示要不要提前结束，每一次内循环都额外增加了为flag赋值的语句，消耗时间自然多了

**快速排序和归并排序同样作为O(n log(n))级别的算法，为什么快排这么快？**

快速排序的最坏情况取决于输入数据，如果key的值是最小或者最大值，交换的次数就会变成n，但是再大量的数据下，选取第一个元素作为key，往往是比较合理的，而且这个值往往比较靠近中间值

TopK问题：

使用规模为20，值遍历1-20的数组来测试正确性



方法1：

升序排序，然后输出前k个，复杂度O(nlog(n))

方法2：

连续找k次，每次找到之后做标记，复杂度O(nk)

方法3：

维护大小为k的小顶堆，遍历数组

1. 如果堆大小小于k则任意元素均可入堆
2. 如果堆大小为k且当前元素比堆顶元素大，堆顶弹出
3. 当前元素入堆，如果当前元素比堆顶小，不管它

最后堆中元素就是答案，复杂度O(nlog(k))

注：插入，删除元素的复杂度都是O(log(k))

数据：

同样采用20次，取平均, k=10，数据规模从一百万到五百万

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 方法/规模 | 1000000 | 2000000 | 3000000 | 4000000 | 5000000 |
| 方法1：排序 | 0.4736 | 0.85515 | 1.29845 | 1.71885 | 2.10665 |
| 方法2：顺序查找 | 0.14085 | 0.2763 | 0.3723 | 0.5667 | 0.65175 |
| 方法3：小顶堆 | 0.0254 | 0.0404 | 0.06735 | 0.08625 | 0.10585 |

分析：

可以看到排序是实打实的O(nlog(n))，速度也是最慢的

因为本题k的取值不变，所以方法2顺序查找的O(nk)无法体现，实际上是线性复杂度

小顶堆的速度快，因为只用遍历一次数组，而且因为k取值不变，O(nlog(k))无法体现，是线性复杂度

**同是线性复杂度，为什么方法2比方法3快这么多？**

**方法3只遍历一遍数组即可，而方法2是k遍**

1. 对求解这个问题的经验总结

如果需要面临大量的数据，相比于线性复杂度，对数级的复杂度是非常恐怖的，如果可能，尽量选择有对数复杂度的代码

如果一个算法需要频繁的获取最值，比如上面的TopK问题，或者是单源最短路**迪杰斯特拉**，往往可以用到堆去优化，因为存取一个值的复杂度是O(log(n))，而相比朴素算法的O(n)将会是极大的提升

在分析一个问题的解法的时候，第一步就是计算算法可能的复杂度，先把算法的复杂度确定下来，再找对应的算法

将问题一分为二，往往对应了对数的复杂度，在考虑问题的时候，优先考虑能否将问题分为两个子问题？如果能，那么很有可能我们的解就是有对数复杂度，比如快排，归并，都是将两个子问题的解合并

再比如快速幂 (fastPow)，也是同样的思路：

计算 x 的 n 次方，问题转化为

X的n/2次方 \* X的n/2次方 （n为偶数）

X的n/2次方 \* X的n/2次方 \* X （n为奇数）

|  |
| --- |
| 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。