算法分析第一次第二次作业

计算机二学位 s2206010114 马燕

1. 用伪代码描述直接插入排序算法，指出循环不变量，进而证明算法的正确性

直接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。

直接插入排序的伪代码描述：

```
InsertionSort(A)
    for j = 2 to A.length
        key = A[j]
        // 将 A[j] 插入到已排序的序列 A[1..j-1] 中
        i = j - 1
        while i > 0 and A[i] > key
            A[i + 1] = A[i]
            i = i - 1
        A[i + 1] = key
```

在直接插入排序中，定义循环不变量如下：

在每次 for 循环的迭代开始之前，子数组 A[1..j-1] 都是已经排序的。

证明：

初始化：在第一次迭代开始时，j = 2，此时子数组 A[1..1] 只包含一个元素，自然是已排序的。

保持：在每次迭代中，元素 A[j] 被取出作为 key，并通过 while 循环插入到正确的位置以保持数组的排序。插入 key 后，子数组 A[1..j] 便保持为已排序状态。每次迭代结束时，j 自增，但由于新的 key 插入后数组仍保持排序，循环不变量继续成立。

终止：在循环终止时，j = A.length + 1。此时，根据循环不变量，子数组 A[1..A.length] 是已排序的，这意味着整个数组已经排序完毕。

通过以上循环不变量的证明，可以看出直接插入排序算法是正确的。它每次将一个新元素插入到已排序的数组部分，最终得到一个完全排序的数组。

2. (数字华容道游戏中的不变性）

完成 https://people.csail.mit.edu/meyer/mcs.pdf

Page 164 Problem 5.38。

问题的目标是展示为什么从标准的初始位置无法通过奇偶性的概念达到目标位置

1. 列出表示和空格的坐标

从标准初始位置和目标位置,:

创建一个从 1 到 15 的数字列表，这些数字按照每个状态出现的顺序，忽略空白格。

确定两个状态中空白格的坐标。

标准初始位置

列表：1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15

空白格坐标:(4,4)

目标位置

列表：(15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1)

空白格坐标：(1, 4)

2. 计算奇偶性

状态S的奇偶性是由列表表示中逆序对的数量加上空白格所在行的行号决定的。如果总和是偶数，奇偶性为 0；如果是奇数，奇偶性为 1。

计算逆序对

计算所有的对(i, j)数量，其中 i < j，但位置 i 的数字大于位置 j 的数字。

标准初始位置

没有逆序对，列表是自然升序的。

空白格行的奇偶性：4（偶数），因此总奇偶性为 0。

目标位置

每对都是逆序对，因为列表完全是降序的。

15 个元素完全逆序的逆序对总数为（15*4）/2（奇数）

空白格行的奇偶性：1（奇数），因此总奇偶性为 0（105 + 1 = 106，偶数）。

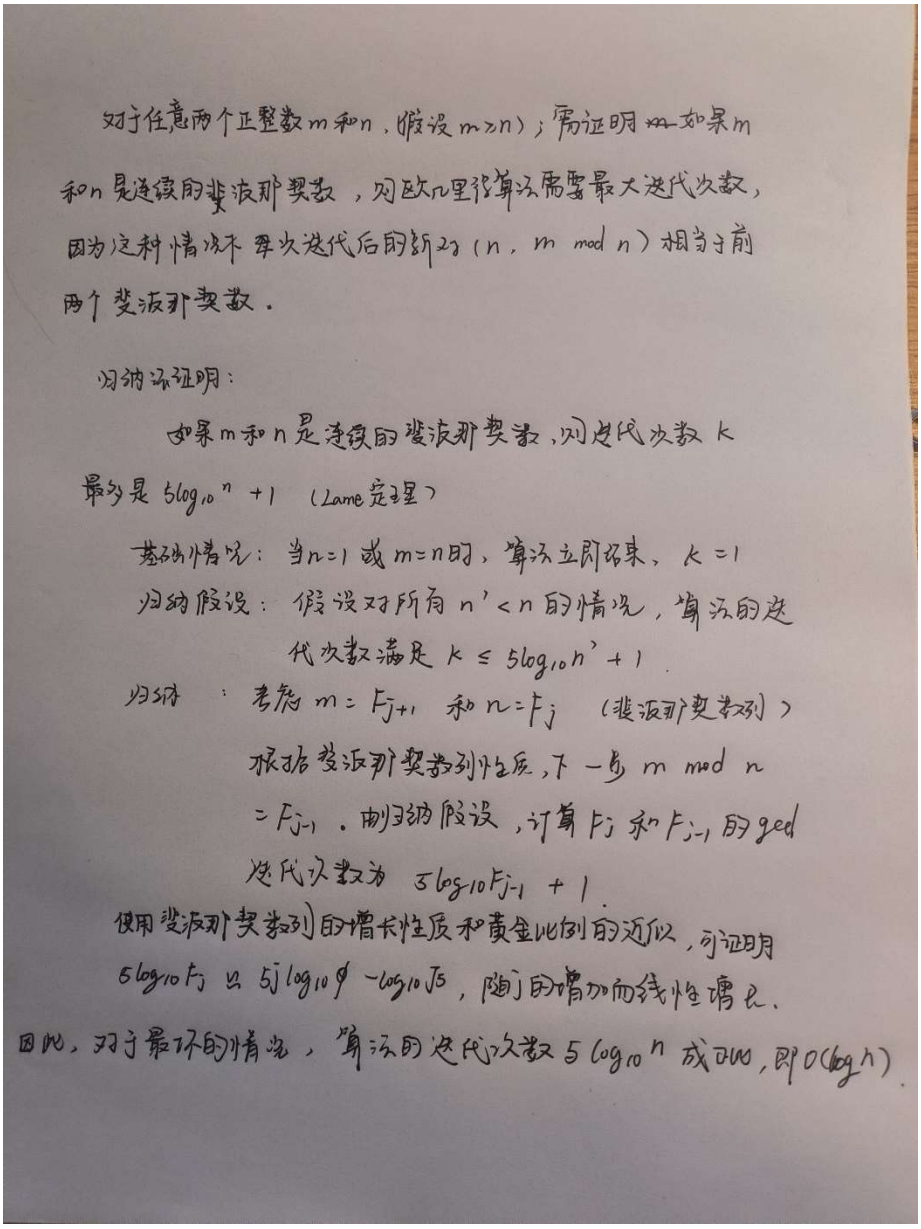
3. 比较奇偶性

两个状态计算出的奇偶性都出人意料地为 0。然而，通常在这类问题中，“不可能”的配置因为与起始配置的奇偶性不同而被认为确实无法实现。

4. 结论

两个状态的奇偶性相同

3. 给出欧几里得算法时间复杂性分析结果的严格证明（见 PPT 第 17 页）（提示：运用数学归纳法，题目有点难度）



4. 自行了解 Donald Knuth 的一些事迹

Donald Knuth，被广泛认为是现代计算机科学的先驱之一，对计算机科学领域做出了深远的影响。他最著名的成就可能是编撰了多卷本的《计算机程序设计艺术》(The Art of Computer Programming, TAOCP)，这是一系列详尽介绍算法和程序设计技术的书籍。

《计算机程序设计艺术》: Knuth 的这套书籍是计算机科学领域的经典之作，被誉为程序员的“圣经”。该系列书籍开始于 1968 年，至今仍未完成。这套书详细介绍了各种算法和数据结构，并通过精确的分析来展示这些算法的效率。

TeX 排版系统：在 1970 年代末，Knuth 开始开发 TeX 排版系统，这是为了改进学术论文的排版质量。TeX 系统及其后来的扩展版本 LaTeX，现在被广泛用于科学出版物的制作，特别是数学、计算机科学和工程学领域的文档。

第二次作业：

1.课上讲的渐近分析是让我们抓住算法时间复杂度中的“主要矛盾”。我们知道，主要矛盾和次要矛盾是辩证统一的。请从算法的实际应用角度谈一谈这里的辩证统一。

在算法的实际应用中，辩证统一体现在对主要矛盾和次要矛盾的平衡和权衡上。主要矛盾通常指的是算法的时间复杂度，即算法执行所需时间的主要因素，而次要矛盾则是指算法的空间复杂度或者其他资源消耗情况。

辩证统一的关键在于权衡主次，以最优化算法的整体性能为目标。在实际应用中，我们往往需要考虑多个因素：

1. **时间与空间的平衡**：有时候，我们可以通过增加空间复杂度来减少时间复杂度，或者牺牲一些时间来节省空间。比如，在某些情况下，为了提高算法的执行速度，我们可能会选择使用更多的内存空间来存储中间结果，以减少计算时间。
2. **算法的适应性**：不同的应用场景可能对时间和空间的需求不同。一些应用对速度要求极高，而对内存消耗并不敏感；而另一些应用则可能更注重节省内存空间，即使需要稍微增加执行时间也在所不惜。
3. **实际资源限制**：在实际应用中，通常会有资源限制，比如硬件资源、运行环境等。在这种情况下，我们需要根据实际情况进行权衡，选择合适的算法和优化策略，以在有限的资源下获得最佳的性能。
4. **问题本身的特点**：问题的特点也会影响到算法的选择。有些问题可能更适合使用时间复杂度较低的算法，而另一些问题则可能更适合使用空间复杂度较低的算法。

因此，从算法的实际应用角度来看，辩证统一体现在对时间复杂度和空间复杂度之间的平衡和权衡上，以及对问题本身特点、资源限制和实际需求的综合考虑上。在实际应用中，我们需要根据具体情况综合考虑各种因素，选择合适的算法和优化策略，以获得最佳的性能和效果。

2.教材习题一中的 1.1，1.4, 1.6，1.13，1.17

1.1 设 A 是 n 个不等的数的数组,n>2.以比较作为基本运算，试给出一个 0(1)时间的算法找出 A 中一个既不是最大也不是最小的数.写出算法的伪码,说明该算法最坏情况下执行的比较次数.

1. 初始化变量 second_largest 和 second_smallest 为无穷大和负无穷大。
2. 对于数组 A 中的每个元素 a：
   如果 a 大于最大值，则将最大值更新为 a，并更新 second_largest 为最大值。
   如果 a 小于最小值，则将最小值更新为 a，并更新 second_smallest 为最小值。
   否则，如果 a 大于 second_largest，则更新 second_largest 为 a。
   同理，如果 a 小于 second_smallest，则更新 second_smallest 为 a。
3. 返回 second_largest 和 second_smallest 中较大的一个，即为所求。

伪代码如下：

```
Function findSecondLargestAndSmallest(A):
    max_value = -∞
    min_value = +∞
    second_largest = -∞
```

```
second_smallest = +∞

for each element a in A:
    if a > max_value:
        second_largest = max_value
        max_value = a
    else if a < min_value:
        second_smallest = min_value
        min_value = a
    else if a > second_largest:
        second_largest = a
    else if a < second_smallest:
        second_smallest = a

return max(second_largest, second_smallest)
```

在最坏情况下，该算法的比较次数为 $3(n-2)$3(n−2)，即数组中除了最大和最小元素外的其余元素都需要进行比较。因此，该算法在 $O(1)$$O$(1) 的时间复杂度内找出一个既不是最大也不是最小的数。

1.4 1.4 计算下述算法所执行的加法次数. 计算下述算法所执行的加法次数.
输入: n=2^t, t 为正整数
输出: k
1. k←0
2. while n≥ 1 do
3.for j←1 to n do
4. k←k+1
5. n←n/2
6. return k

算法中的两个循环会导致不同数量级的迭代次数。

首先，外部循环 **while n ≥ 1** 的迭代次数取决于 n 的值。在每次迭代中，n 会减半，直到 n < 1。因为 n 的初始值是 2^t ，t 是正整数，所以外部循环总共执行 t 次。

内部循环 **for j ← 1 to n** 的迭代次数取决于 n 的值。它会在每次外部循环迭代时执行，其迭代次数为 n。因此，内部循环总共执行的次数为：

$2^t+2^{t-1}+2^{t-2}+\cdots+2^1+2^0$

这是一个几何级数，其总和为 2^{t+1}-1

然后计算加法操作的总次数。在每次内部循环迭代中，有一个加法操作：**k ← k + 1**。所以，总的加法操作次数就是内部循环总共执行的次数，即 2^{t+1}-1

因此，算法执行的加法操作次数为 2^{t+1}-1

这个结果也可以通过分析得出。每次内部循环中，**k ← k + 1** 操作都执行了一次，而总共有 2^{t+1}-1 次内部循环。

1.6 阅读关于下述算法 A 的伪码,说明该算法求解的是什么问题,并计算该算法所做的乘法运算(*)和加法运算次数.

输入:数组 P[0,n]，实数 x

输出：y

1. **y←P[0]; porwer←1**
2. **for i ← to do n**
3. **power←power*x**
4. **y←y+ P[i]* porwer**
5. **return y**

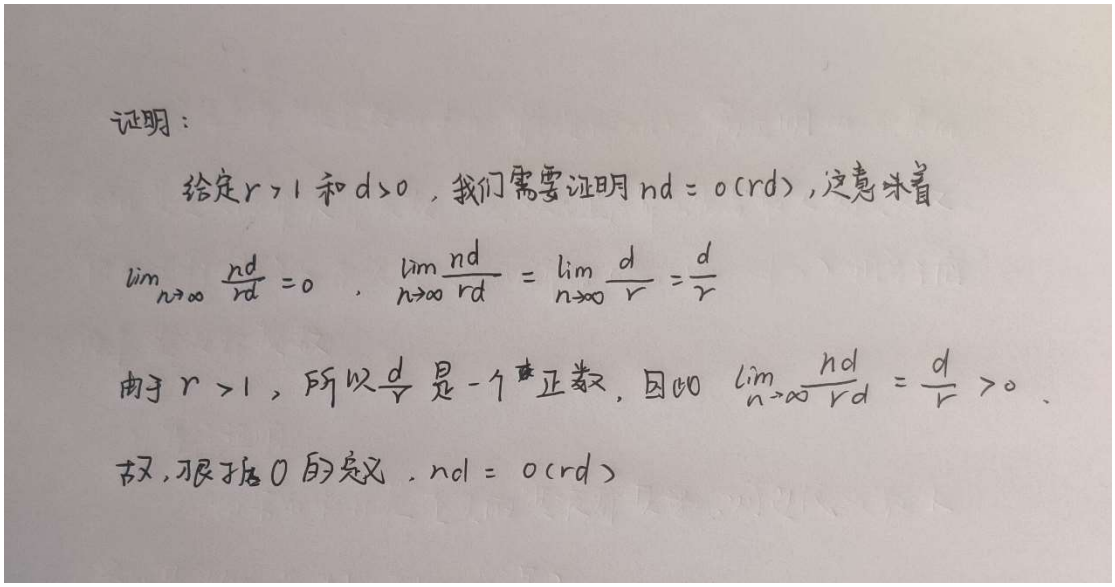
这个算法求解的是多项式求值的问题。给定一个多项式 $P(x)=P[0]+P[1]\cdot x+P[2]\cdot x^2+\cdots+P[n]\cdot x^n$ $P(x)=P[0]+P[1]\cdot x+P[2]\cdot x^2+\cdots+P[n]\cdot x^n$，以及一个实数 xx，算法 A 通过代入 x到多项式中求得对应的函数值 y。

计算该算法所做的乘法运算和加法运算次数如下：

乘法运算次数：$n+1$，因为第 3 行的操作 $power\cdot x$进行了 n 次，加上第 1 行的初始值 $power$一次，总共 $n+1$ 次乘法运算。

加法运算次数：$n+1$，因为第 4 行的操作 $y+P[i]\cdot power$进行了 n 次，加上第 1 行的初始值 y一次，总共 $n+1$ 次加法运算。

1.13 证明定理 1.5:对每个 r>1 和每个 d>0,有 $n^d=o(r^d)$.



1.17 对于下面每个函数 f(n),用日符号表示成 f(n)=@(g(n))的形式,其中 g(n)要尽可能简洁.比如 f(n)=n'+2n+3 可以写为 f(n)=@(n2). 然后,按照阶递增的顺序将这些函数进行排列.

(n- 2)!， 5log(n+ 100)^{10}， 2^{2n}， 0. 001n^4+ 3n^{3}十 1,

(lnn)^2，$\sqrt[3]{n}$+logn，3^n， log(n!)， log(n^{n+1})， $1+\frac{1}{2}\ldots+\frac{1}{n}$

1. $(n-2)!$：这个函数的阶数是 $O(n!)$，所以可以表示为 @($n!$)。
2. $5\log(n+100)^{10}$ 函数的阶数是 $O(\log n)$，所以可以表示为 @($\log n$)。
3. 2^{2n}：这个函数的阶数是 $O(2^n)$，所以可以表示为 @(2^n)。

4. $0.001n^4+3n^3+1$：这个函数的阶数是 $O(n^4)$，所以可以表示为 $@(n^4)$。
5. $(\ln n)^2$：这个函数的阶数是 $O((\ln n)^2)$，所以可以表示为 $@((\ln n)^2)$。
6. $\sqrt[3]{n}+\log n$：这个函数的阶数是 $O(\sqrt[3]{n})$，所以可以表示为 $@(\sqrt[3]{n})$。
7. 3^n：这个函数的阶数是 $O(3^n)$，所以可以表示为 $@(3^n)$。
8. $\log(n!)$：这个函数的阶数是 $O(n\log n)$，所以可以表示为 $@(n\log n)$。
9. $\log(n^n+1)$：这个函数的阶数是 $O(n\log n)$，所以可以表示为 $@(n\log n)$。
10. $1+\frac{1}{2}...+\frac{1}{n}$：这个函数的阶数是 $O(\log_2 n)$，所以可以表示为 $@(\log n)$。

按阶递增的顺序排列如下：

$@(1)<@(\log n)<@(\log n)<@({}_3 n)<@(n\log n)<@(n_4)<@(2_n)<@(3_n)<@(n!)$