1、算法的基本概念、性质

算法定义：一个定义良好的可计算过程，取1个或一组值作为输入，并产生出一个或一组值作为输出。即，算法就是一系列的计算步骤，用来将输入数据转换成输出结果。

问题：描述了期望的输入与输出关系，可以用通用语言来描述

问题实例：某一个问题的实例包含了求解该问题所需的输入

输入规模：算法实例的输入大小

程序并不都满足算法所要求的上述特征，如有穷性特征。算法代表了对特定问题的求解，而程序则是算法在计算机上的实现。

2、算法的五个重要特性

① 输入：一个算法具有零个或者多个取自指定集合的输入值

② 输出：对算法的每一次输入，算法具有一个或多个与输入值相联系的输出值

③ 有穷性：对算法的每一次输入，算法都必须在有限步骤（即有限时间）内结束

④ 确定性：算法的每一个指令步骤都是明确的，不会出现二义性

⑤ 可行性：算法的每一步都必须是可行的，也就是说，每一步都能够通过执行有限次数完成

3、算法时间复杂度的含义

算法的时间复杂度是一个关于代表算法输入值的字符串的长度的函数，他定性的描述了算法的运行时间。

4、插入排序

思想：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

时间复杂度：最好排好序的n，最坏逆序的西塔n平方。

对给定的具体算例正确排序：书p10

循环不变式：在每次for循环开始前，子数组由原来在其中的元素组成，但已经排好序。

证明循环不变式，证明三个式子：初始化&保持&终止。

5、伪代码

给出算法伪代码能够分析出算法的时间复杂度，能够用渐近记号表示：书p14

Σ（指令的一次执行时间\*一条指令执行次数）

6、渐进函数记号

O(g(n)) = { f(n): 存在正常量 c 和 n0, 使得对所有 n ≥ n0有0 ≤ f(n) ≤ cg(n) }

Ω(g(n)) = { f(n): 存在正常量 c 和 n0, 使得对所有 n ≥ n0，有0 ≤ cg(n) ≤ f(n) }

Θ(g(n)) = { f(n): 存在正常量 c1, c2, 和 n0, 使得对所有n ≥ n0, 有0 ≤ c1g(n) ≤ f(n) ≤ c2g(n) }

o(g(n)) = { f(n): 对任意正常量 c > 0，存在常量 n0> 0，使得对所有n ≥ n0，有0 ≤ f(n) < cg(n) }

ω(g(n)) = { f(n): 对任意正常量 c > 0，存在常量 n0> 0，使得对 所有 n ≥ n0，有0 ≤ cg(n) < f(n) }

7、限界函数的基本性质

传递性、自反性、对称性、转置对称性 书p51-52

8、计算和估算时间复杂度的一些定理

定理 3.2 多项式定理 若A(n)=am\*n^m+...+a1\*n+a0是一个 n 的 m 次项式，其中 ai为常量，i = 0，…，m，且 am> 0，则有 A(n) = Θ(n^m)

定理 3.3 用于估算复杂性（函数阶的大小）的定理

对于任意正实数 x 和 ε ，有下面的不等式：

1. 存在某个 n0，使得对于任何 n ≥ n0，有 (lgn)^x< (lgn)^(x+ε)。

2. 存在某个 n0，使得对于任何 n ≥ n0，有 n^x< n^(x+ε)。

3. 存在某个 n0，使得对于任何 n ≥ n0，有 (lgn)^x< n。

（任意正的多项式函数都比任意多对数函数增长得快）

4. 存在某个 n0，使得对于任何 n ≥ n0，有 n^x< 2^n。

（任意底大于1的指数函数比任意多项式函数增长得快）

5. 对任意实数 y，存在某个 n0，使得对于任何 n ≥ n0，有n^x\*(lgn)^y< n^(x+ε)。

定理 3.4 设 d(n)、e(n)、f(n) 和 g(n) 是将非负整数映射到非负实数的函数，则

1. 如果 d(n) 是 O(f(n))，那么对于任何常数 a>0，ad(n)是 O(f(n))；

2. 如果 d(n) 是 O(f(n))，e(n)是 O(g(n))，那么 d(n)+e(n) 是O(f(n)+g(n)) —— 加法法则；

3. 如果 d(n) 是 O(f(n))，e(n)是 O(g(n))，那么 d(n)e(n) 是O(f(n)g(n)) —— 乘法法则；

4. 对于任意固定的 x > 0 和 a > 1，n^x是 o(a^n) —— 任意底大于1的指数函数比任意多项式函数增长得快；

5. 对于任意固定的 x > 0，(lgn)^x是 O(lgn)；

6. 对于任意固定的常数 x > 0 和 y > 0，(lgn)^x 是 o(n^y) —— 任意正的多项式函数都比任意多对数函数增长得快。

O(1) < O(lgn) < O(n) < O(nlgn) < O(n^2) < O(n^3)

O(2^n) < O(n!) < O(n^n)

9、分治策略

分治法基本思想，问题规模较大而无法直接求解时，将原始问题分解为几个规模较小，但类似于原始问题的子问题，然后递归求解这些子问题，最后合并子问题的解得到原始问题的解。

遵循的三个步骤：

1）分解：将原始问题分解为若干个规模较小相互独立形似与愿问题一样的子问题，

2）解决：递归解决子问题，若足够小，否则递归

3）合并：将子问题的结果合并为愿问题的解

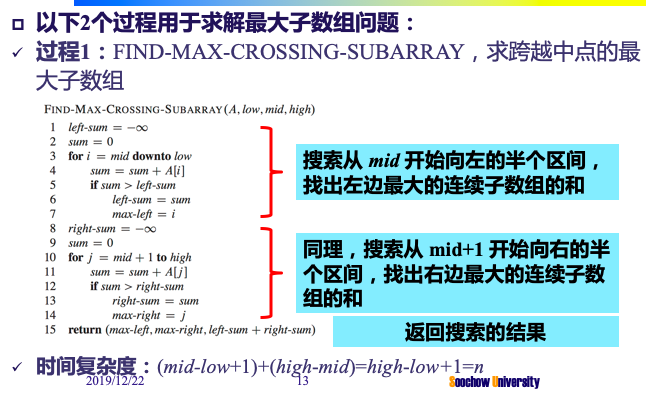
需要进一步分解递归合并求解时 递归情况， 不需要进一步求解 基本情况。

10、利用分治策略求解相关问题和算法

归并排序：p17-21

最大子数组问题：p38-42

strassen矩阵乘法：p45-47





strassen矩阵乘法：

朴素矩阵乘计算时间为西塔n3方。

基于分治策略：

strassen矩阵乘法：令递归树稍微不难么旺盛，只进行七次递归而不是八次。

11、递归式的化简

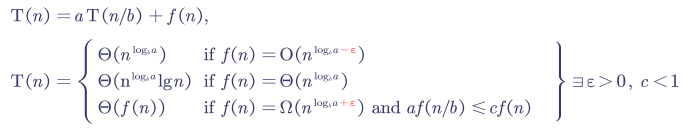
代入法：

猜测接的形式 & 数学归纳法找出常数。扩展边界条件。归纳假设不够强，可以去掉低阶项。注意归纳假设严格一致的性质。注意改变变量的方法。

递归树法：

每一个节点代表一个单一子问题的代价，子问题对应某次递归函数调用。生成好的猜测解。

主方法：



12、随机算法

如果一个算法的行为不仅由输入决定，而且也由一个随机数生成器产生的数值决定，则称这个算法是随机化的。

13、排序

堆排序：书p84-89

快速排序：书p95-99

计数排序：书p108-109

桶排序：书p112-113

14、动态规划

基本思想是分治思想和解决冗余

动态规划策略与分治策略类似，其基本思想也是将待求解的原问题分解成若干个子问题。但与分治策略不同的是，动态规划适用于子问题重叠的情况，即不同子问题具有公共的子子问题。动态规划算法对每个这样的子子问题只求解一次，将其解保存在一个表格中，再次碰到时无需重新计算，只需从表格中找到上次计算的结果加以调用即可，避免了不必要的计算工作。

步骤：

① 找出最优解的性质，并刻划其结构特征 ---》划分子问题

② 递归地定义最优解的值 ----》给出最优解的值的递归式

③ 按自底向上的方式计算出最优解的值

④ 由计算出的结果构造一个最优解

15、贪心算法

思想：分步骤实施，它在每一步仅作出当时看起来最佳的选择，即局部最优的选择，并寄希望这样的选择最终能导致全局最优解

条件：贪心选择性质和最优子结构性是两个关键要素

步骤：书p242

16、贪心策略 VS 动态规划

书p242-243

2.1-2

重写过程 INSERTION-SORT，使之按非升序(而不是非降序)排序。

INSERTION-SORT(A)

for j = 2 to A.length

key = A[j]

i = j - 1

while i > 0 and A[i] < key

A[i + 1] = A[i]

i = i - 1

A[i + 1] = key

2.1-4

考虑把两个 n位二进制整数加起来的问题，这两个整数分别存储在两个 n 元数组A 和B中。这两个整数的和应按二进制形式存储在一个(n+1)元数组 C中。请给出该问题的形式化描述，并写出伪代码。

Input: An array of booleans A=⟨a1​,a2​,…,an​⟩ and an array of booleans B=⟨b1​,b2​,…,bn​⟩, each representing an integer stored in binary format (each digit is a number, either 0 or 1, least-significant digit first) and each of length n.

Output: An array C=⟨c1​,c2​,…,cn+1​⟩ such that C′=A′+B′ where A′, B′ and C′ are the integers, represented by A, B and C.

ADD-BINARY(A, B)

carry = 0

for i = 1 to A.length

sum = A[i] + B[i] + carry

C[i] = sum % 2 // remainder

carry = sum / 2 // quotient

C[A.length + 1] = carry

return C

2.2-2

考虑排序存储在数组 A 中的n个数:首先找出A 中的最小元素并将其与A[1]中的元素进行交换。接着，找出A 中的次最小元素并将其与 A[2]中的元素进行交换。对 A 中前 n-1个元素按该方式继续。该算法称为选择算法，写出其伪代码。该算法维持的循环不变式是什么? 为什么它只需要对前 n-1个元素，而不是对所有n个元素运行?用θ记号给出选择排序的最好情况与最坏情况运行时间。

n = A.length

for i = 1 to n - 1

minIndex = i

for j = i + 1 to n

if A[j] < A[minIndex]

minIndex = j

swap(A[i], A[minIndex])

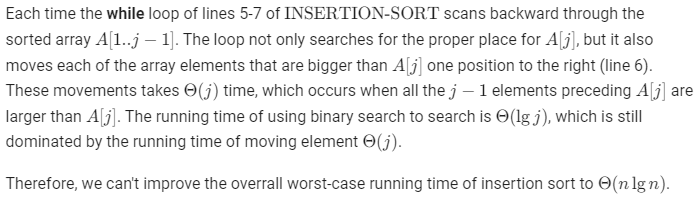
At the start of the loop in line 1, the subarray A[1..i−1] consists of the smallest i−1 elements in array A with sorted order.

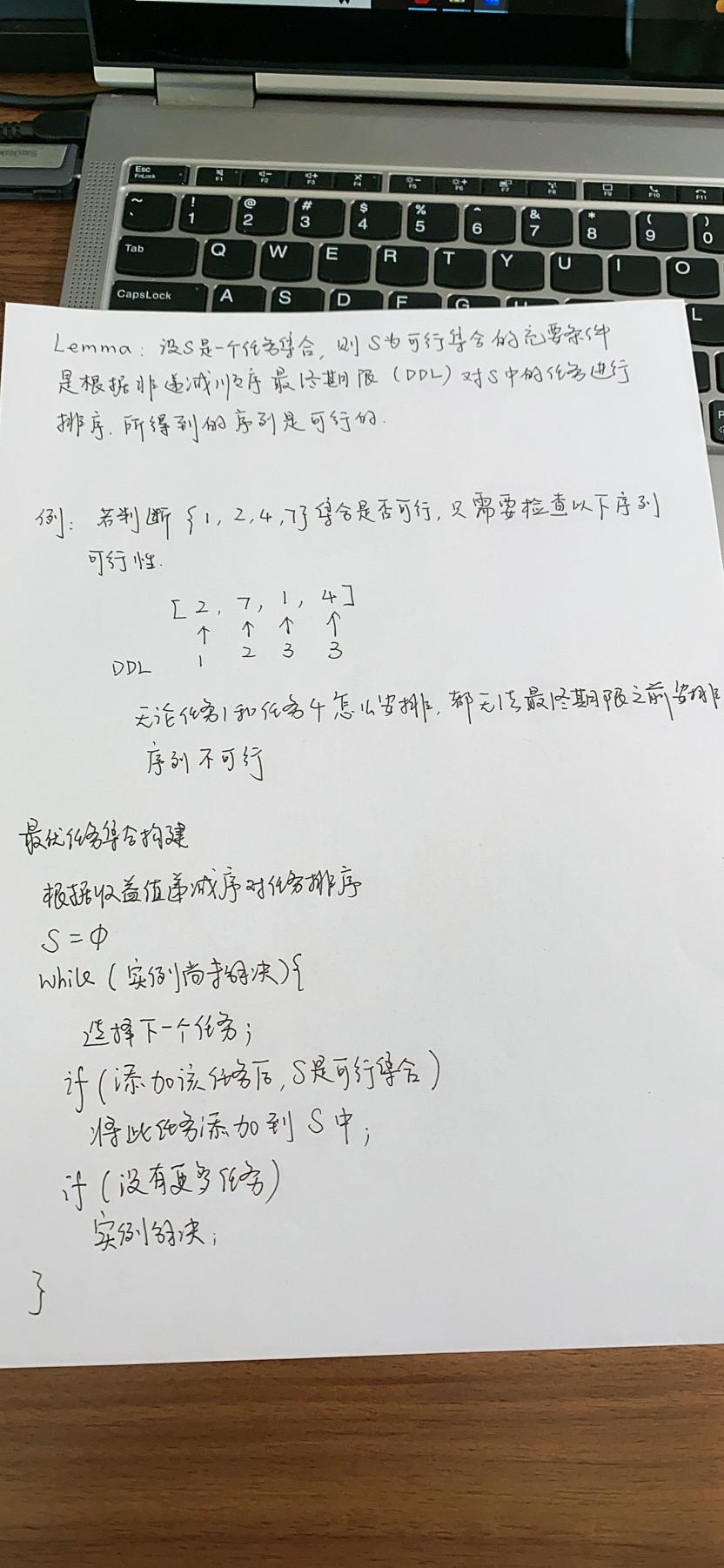
After n−1 iterations, the subarray A[1..n−1] consists of the smallest i−1 elements in array A with sorted order. Therefore, A[n] is already the largest element.

Θ(n2)

2.3-6

注意到 2.1 节中的过程INSERTION-SORT 的第 5~7 行的 while 循环采用一种线性查找来(反向)扫描已排好序的子数组 A[1..j-1]。我们可以使用二分查找来把插入排序的最坏情况总运行时间改进到Θ(nlgn)吗?





快速排序避免最坏情况

