MEng 组合内容报告

Min Huang

课程名称：CSE 551 - 算法基础

亚利桑那州立大学

综述

*这是 2025年春季学期选修的算法基础（CSE 551）课程的作品集报告。在这门课程中，我们必须完成 4个项目/作业，每个项目/作业都要求我们使用常规课堂上讲授的主题和概念。*

**1. 课程项目介绍**

课程作业主要包括四个部分。

第一部分是**复杂度分析、递归式求解和排序算法**作业

首先，通过证明多项式n²/8 - 10n - 4的复杂度为Θ(n²)，我们可以深入理解Θ符号的定义。

然后，使用循环方法和递归树方法来推导递归式的解。通过深入理解这两种方法，我们可以选择合适的方法来解决给定的递归公式。

对每一组函数 f(n) 和函数 g(n) 的增长率，判断它们之间的关系是大 O，大 Ω，还是 Θ，并提供简要说明。这项分析帮助我们深入了解函数的增长特性。

编写选择排序算法的伪代码，该算法用于选择最大元素。要求：对数组 A进行排序，找到最大元素并移至数组 B 的尾部，重复操作直至数组 A 为空。接着，计算伪代码中每条指令的执行次数，并分析算法在最好和最坏情况下的运行时间。

最后，使用高级编程语言实现该排序算法，并为数组 A编写测试用例。当程序运行时，可以打印不同输入数组大小下排序算法的执行时间。

第二部分是**医院-实习生稳定匹配**作业

使用 Gale-Shapley (GS) 算法的来解决医院与实习生之间的稳定匹配问题。

首先，定义医院和实习生的数量，同时定义各自的偏好列表。通过应用 GS 算法，模拟医院与实习生之间的匹配决策过程，实现稳定的配对。

然后，计算 GS 算法的时间复杂度，并证明算法生成的匹配结果是稳定的。接着，创建一组测试用例作为输入集，并手动计算 GS 算法的输出结果。

最后，使用高级编程语言实现该算法，并使用前面创建的测试用例作为输入，测试验证程序的输出结果。

第三部分是**最小生成树算法和最短旅行时间**作业

首先，通过分析图论中的最小生成树问题和最短路径问题，研究它们在边权变化时的稳定性。问题1设定 T为图 G的最小生成树，分析当图 G中所有边的权值被替换为平方后的权值时，T是否仍为新的图 G的最小生成树。问题2则探讨最短路径的稳定性，其中P 为图 G的最短路径，当图 G中所有边的权值被替换为平方后的权值时，P是否仍为新的图 G的最短路径。

接着，我们将设计一个网络，目标是降低生成树中最大边的权值，并通过理论证明探讨最小瓶颈生成树与最小生成树之间的关系。

最后，我们开发并实现一个基于动态旅行时间的最短路径问题的多项式时间算法，重点分析其时间复杂度。我们将使用归纳法证明该算法的正确性，并使用高级编程语言实现这一算法。在测试环节，我们会使用给定的测试用例对程序进行验证，并提交程序生成的输出结果。

第四部分是**分治算法和最近点对算法**作业

首先，我们需要优化大整数乘法的算法设计。使用分治技术，可以将两个较大的十进制整数拆分成 a+b\*i 和 c+d\*i的形式。接下来需要展示如何将表达式 (a+b\*i)(c+d\*i)转换为由a、b、c、d 和 i 组成的表达式，并确保使用尽可能少的乘法运算。最后，给出转换后的最终表达式。通过这个过程，我们可以深入理解分治算法的设计思想和优化方法。

接下来是最近点对算法的优化与实现。首先对课程幻灯片中给出的最近点对算法进行分析，确定算法正确性所需的最小 x 值。然后，使用高级编程语言实现该算法，确保采用最小 x 值，并在每个步骤中以所需的复杂度实现程序。最后，创建一个包含 16 个点的测试用例，以验证程序的正确性。 通过这一编程实践，我们可以加深对分治算法的理解，并培养算法设计、实现和测试的能力。

**2. 解决方法/方案说明**

**复杂度分析、递归式求解和排序算法**作业

(1)复杂度分析

用数学推导证明多项式 n²/8−10n−4 的复杂度为 Θ(n²)。根据 Θ 符号的定义，我们需要找到常数 c₁和 c₂以及 n₀，对于足够大的 n，c₁\*n² ≤ n²/8−10n−4 ≤ c₂\*n²公式成立。

对于右边的不等式，我们设定 c₂=1/8，证明对于足够大的 n，都有 n²/8−10n−4≤n²/8。考虑当 n足够大时，线性和常数项的影响可以忽略不计，因此不等式成立。

对于左边的不等式，通过假设 n≥n₀且 n₀=160，我们得出 c₁=1/16。通过消除较小项的影响，从而确保  n²/8−10n−4≥c₁\*n² 成立。显然，对于 n≥160，线性和常数项依旧可以被平方项的增长速率支配。

基于此推导方式，我们归纳出 n²/8−10n−4 的渐进紧界为 Θ(n²)。

(2)循环方法递归式求解

设每次递归n都会被除以3，可以得到T(n)=3T(n/3)+n。在下一步的递归中，我们进一步得出这一层的结果：T(n/3)=3T(n/9)+n/3。在第三步中，通过替代前面的结果，逐渐得到更加深入的递归调用。最终，通过临界分析，根据推导结果，我们可以得出时间复杂度是T(n)=Θ(nlog₃n)。

(3)递归树方法递归式求解

在分析递归关系 T(n)=3T(n/3)+n时，我们采用递归树的方法来解析其计算量。递归树的根节点的计算量为 nn，这一计算量在每层中被分解为三个规模逐渐减小的子问题。关键在于，递归树的深度可以通过每次分裂的次数确定，即 log₃n。在树的最底层，子问题已经缩减到常数规模。

为了更清晰地了解计算量分布，我们从递归树根部向底层分析：初始节点的计算量为 n，第一层的计算量也为 n，因为 3(n/3)=n。这一模式在每层都保持一致，即在第 k 层计算量为 3k(n/3k)=n。递归树有 log₃n层，每层的计算量固定为 n，于是整棵递归树的总计算量为 T(n)=n\*log₃n

基于上述分析，我们进一步结合递归关系来理解算法的复杂度。通过渐进符号的处理，我们最终确定该递归式的时间复杂度为 Θ(n\*log₃n)

(4)复杂度关系判断

我们对每一组函数 f(n) 和 g(n) 的复杂度进行了计算。通过比较这些函数的增长速率，确定它们适用的渐进符号。例如，使用大 O、大 Ω 和大 Θ 符号来表示相应的关系。

(5)编写最大元素选择排序算法的伪代码

我们使用 Java 编程语言实现排序算法的步骤如下：

初始化两个数组 A 和 B，其中 A 包含待排序的元素，B 用于存放排序后的结果。外层循环遍历 A 中的元素，依次寻找未排序部分的最大值。在每次循环中，内层循环对当前未排序的数组部分执行查找操作，找到最大元素的索引。找到最大元素后，将其放置到数组 B 的相应位置。然后，将数组 A 中该最大元素的位置设置为一个极小值，以避免在后续迭代中再次找到该元素。重复上述步骤，直到所有元素均已排序并存入数组 B 中。

(6)程序设计实现部分

使用 Java 编程语言进行开发，编写程序，根据用户输入的数组大小 n，生成 n 个范围在 0 到 999 之间的随机整数，并将这些整数存储在数组 A 中。

实现伪代码中的排序算法，对数组 A 中的随机整数进行排序，并将排序后的结果输出到控制台。测量排序算法的执行时间（不包括生成随机整数的时间），通过在程序中记录开始时间和结束时间来计算排序时间。

**医院-实习生稳定匹配**作业

时间复杂度分析：假设m 是医院数量，n 是实习生数量

while 循环在最坏情况下（即每个医院尝试与每个实习生匹配）执行 m\*n 次， 所以可能有 m\*n 个配对。所以，总时间复杂度为 O(mn)。

证明算法生成的匹配是稳定的:采取反证法，假设s-h是一个不稳定的配对，在”情况 1: 如果 h 从未向 s 提供过岗位”的情况下计算匹配的结果是稳定的，在”情况 2: 如果 h 向 s 提供过岗位”的情况下计算匹配的结果是稳定的，无论哪种情况， s-h 配对都是稳定的， 由于假设不成立， 因此证明算法生成的匹配结果是稳定的。

创建测试用例并手动执行算法:假设输入为2家医院，6名学生，随机生成医院对学生的偏好列表和学生对医院的偏好列表，并按照Gale-Shapley 稳定匹配算法的执行逻辑计算生成匹配结果.

使用Java语言编程参考伪代码编程实现算法，使用set集合定义学生偏好列表，使用Queue队列定义医院偏好列表，在while循环的条件判断当有未配对的学生 且 医院仍有空余席位时，进入循环体，当该医院仍有空余席位且有优先学生时继续处理聘用学生的逻辑，否则将当前医院重新加入到队列中，以便在下一次循环时再次处理该医院。

**最小生成树算法和最短旅行时间**作业

通过构造反例来证明边权重平方后，原来的最小生成树不再是新的最小生成树。假设一个由三个节点和三条边构成的简单图 G，边的代价分别为 1， 2， 3。原最小生成树 T 的总代价为 3，由代价为 1 和 2 的边组成。当我们将边的代价平方后变为 1， 4， 9，此时新的最小生成树会选择代价为 1 和 4 的边，形成总代价为 5。因此，原来的最小生成树不再是新的最小生成树。这个反例说明，当边权被平方时，边的相对代价顺序可能被改变，从而影响最小生成树的选择。

通过构造反例来证明边权重平方后，原来的最短路径不再是新的最短路径。假设有向图 G包含四个顶点 a，b，c，d，边的代价分别为 1， 3， 2， 1， 3 和 1。原最短路径是 a→d，总代价为 2。当我们将边的代价平方后变为 1， 9， 4， 1， 9 和 1，此时新的最短路径选择了路径 a→b→c→d，其总代价为 3。因此，原来的最短路径不再是新的最短路径。这个反例说明，边权平方操作可能改变边的相对代价次序，从而影响最短路径的选择。

朋友们研究了所有的旅行方案并绘制的一个有向图，设有向图中V为顶点数，E为边数，我们对于给定的图论算法进行时间复杂度分析。使用 O符号表示复杂度。其中边的数量 E将迭代 V−1次，因此 while 循环的时间复杂度为 O(V×E)，可以得出算法的总体时间复杂度为O(V \* E)；

使用数学归纳法证明图算法的正确性：首先验证当集合仅包含起始节点时，算法是正确的；然后假设算法在集合大小为 k时正确，再证明在添加一个新节点的情况下（即集合大小为 ( k+1）），算法仍保持正确。这样逐步验证，我们确保算法在所有情况下都能正确找到最短路径。

**分治算法和最近点对算法**作业

传统乘法公式通常需要进行四次乘法运算，分别计算 ac、bd、(a+b)\*(c+d)，以及展开得到结果。通过应用分治算法，可以通过结构化的计算步骤将乘法次数优化至三次。计算公式的简化：s = ac + bd + ((a + b)\*(c + d) - ac - bd)\*i。

计算步骤为：首先分别计算 ac 和 bd。接着，计算 (a + b) \* (c + d)。最后，通过组合以上结果，根据优化公式得到最终乘积。

我们证明并验证了优化后的公式在减少计算复杂度的同时依然保持准确性，使得运算更加高效。

在计算距离时，对于任意两点Si和Sj，距离满足

|i-j| >= x/|i-j|>x，且距离应当小于等于δ。当|i-j| >= 7或|i-j| > 7时，能保证距离满足该条件。因此，最小的x值为7。

我们使用分治策略实现最近点对算法。在 Python 中编写代码以分割点集，并在每个子集中递归地寻找最近点对。递归地计算最近距离，最后合并结果。

**3. 课程项目成果/结果叙述**

**复杂度分析、递归式求解和排序算法**作业

* 成功证明了公式 n²/8−10n−4=Θ(n²) 的成立，表明在渐近意义上 n²/8−10n−4 与 n² 是等价的
* 深入理解 Θ 表示法的定义及应用
* 使用循环方法和递归树方法推导递归式的解
* 完成排序算法的编码实现

**医院-实习生稳定匹配**作业

* 对Gale-Shapley算法的时间复杂度有更深的理解
* 证明了算法的稳定性，并且验证了其匹配结果的正确
* 设计并验证了一个具有实际代表性的测试用例，通过该实例展示了算法运行的过程和结果
* 使用Java编程语言实现了该算法，并对创建的用例进行了测试，从而验证了程序的正确性和结果的稳定性。通过编程实践，我们有效地巩固了对理论知识的理解，并提高了编程实现的能力

**最小生成树算法和最短旅行时间**作业

* 成功构造反例，证明了边权重平方后，原最小生成树和最短路径可能不再是最优解
* 证明了最小瓶颈树不一定是最小生成树
* 成功使用归纳法证明了每个最小生成树都是最小瓶颈树
* 使用归纳法成功证明了有向无环图最早到达时间算法的正确性

**分治算法和最近点对算法**作业

* 成功运用分治法降低了乘法运算的复杂程度
* 通过手工方式实现了最近点对算法
* 确定并验证了在给定条件下的最小x值为7
* 通过Python编程方式实现了最近点对算法
* 通过16个点的测试用例，验证了最近点对算法程序的功能正确性，程序输出了正确的最近点对及其距离。

4. 项目收获

* 单人项目。从项目需求理解、代码实现、结果获取和整理、报告的产生，经历全部过程。
* 通过本课程学习以及相关项目的实践，收获以下：
  + 深入理解了时间复杂度分析方法，尤其是Θ符号的应用。
  + 深入分析了算法的时间复杂度和稳定性，提高了算法分析能力。
  + 掌握了递归式求解的多种方法，包括循环法和递归树法。
  + 通过编程实践，加强了排序算法的设计和实现能力。
  + 学习并实现了Gale-Shapley稳定匹配算法，理解了其在实际问题中的应用。
  + 通过构造反例，加深了对最小生成树和最短路径在边权变化时的稳定性理解。
  + 学会了使用归纳法证明算法的正确性。
  + 学习了如何使用分治策略优化大整数乘法，提高了算法设计能力。
  + 实现了最近点对算法，加深了对分治思想在几何问题中应用的理解。

参考文献

[1] 算法(第4版)， [美]Robert，谢路云 译， 人民邮电出版社，2021。

[2] Algorithm Design，[美] Jon Kleinberg / [美] Éva Tardos  著，王海鹏 译， 出版社： 人民邮电出版社， 2021。

[3]算法导论（原书第3版）），(美)科尔曼 著，[于青林](https://book.jd.com/writer/%E4%BA%8E%E9%9D%92%E6%9E%97_1.html" \t "_blank) 译， 机械工业出版社， 2013。

[4] R.E.Tarjan，Data Structures and Network Algorithms，SIAM，1983年。