

**云南大学信息学院2025学年春季学期**

**《算法设计与分析》（本科）**

**课外实验作业**

**姓名：吕高松 专业：计算机科学与技术**

**教师：段亮**

# 实验设置

## 实验目的

通过手动编程实现经典算法并通过程序执行开销对算法性能进行分析，并对同一问题使用不同算法进行性能分析和比较，深入理解不同算法时间复杂度的异同，加深对和增长率的概念。提高算法思维能力，培养给定问题选择不同求解方案的能力。

## 实验环境

本实验的计算环境基于以下配置：

* 操作系统：Windows 11
* 编译器: GCC (MinGW-w64 for Windows)
* 开发工具:dev c++
* 处理器：AMD Ryzen 9 7945HX with Radeon Graphics（最高主频2.50 GHz）
* 显卡：NVIDIA RTX 4060（8GB GDDR6 显存）
* 内存：16GB RAM。

## 实验内容

（1）0-1背包问题描述如下：给定𝑛种物品和一个总容量为𝐶的背包，物品𝑖(1≤𝑖≤𝑛)的重量为𝑤𝑖，对应价值为𝑣𝑖，应该如何选择物品装入背包，使得装入背包中物品总价值达到最大？例如，背包容量为10，物品的重量分别为[2, 2, 6, 5, 4]，价值分别为[6, 3, 5, 4, 6]，选择物品1、2和5装入背包，获得最大价值15。

（2）分别使用蛮力法、动态规划法、贪心法和回溯法求解0-1背包问题，用C语言编程实现以上算法，解决以下不同输入的0-1背包问题：物品数量分别为[1000, 2000, 3000, 40000, 5000, 6000, 7000, 8000, 9000, 10000, 20000, 40000, 80000, 160000, 320000]，对应背包容量分别为[10000, 100000, 1000000]，每个物品重量为1~100之间的随机数，价值为100~1000之间的随机数（保留两位小数）。

（3）对编写的程序进行测试并确保程序正确无误，输出选择的物品编号、重量、价值、物品装入背包获得的总价值。分别在程序开始和结束处设置记录系统当前时间的变量、用于计算程序执行的时间（以毫秒（ms）作为时间的计数单位），输出程序总的执行时间（不包含数据生成时间）。

# 实验原理

## 算法思想

#### 蛮力法

组合问题输入的所有可能情况，选取最符合问题要求的结果作为答案输出。

#### 动态规划法

将原问题分解为相似的子问题，在求解的过程中通过子问题的解描述并求出原问题的解。

#### 贪心法

采用迭代构造最优解的方式，在每个阶段在一定的标准下，都作出一个当前最优的决策。

#### 回溯法

在问题的解空间树中，按深度优先策略从根节点出发搜索解空间树。算法搜索至解空间树的任一节点时，先判断该节点是否包含问题的解。如果不包含，则跳过对以该节点为根的子树的搜索，逐层向其它祖先节点回溯。否则，进入该子树，继续按照深度优先策略搜索。

## 实验设计

本实验将采用**全因子实验设计**，即针对每一个背包容量 C，遍历所有的物品数量 n，并对每个组合 (n, C) 运行所有适用的算法。

#### 准备阶段

确保实验环境配置完成。编译提供的 C 语言源代码，生成可执行文件。

#### 执行阶段

运行可执行程序。程序将自动按以下逻辑执行：

**外层循环**：遍历背包容量数组 capacities[]。**内层循环**：遍历物品数量数组 item\_counts[]。

在每个 (n, C) 组合下： a. 调用 generate\_items(n, items, 12345) 生成测试数据。 b. 依次调用并测试以下算法（根据代码中设定的适用范围）：蛮力法：仅在 n <= 20 (由MAX\_BRUTE\_FORCE定义为100，但实际可行性约为20) 时执行。回溯法：仅在 n <= 20 (由MAX\_N\_TEST定义) 时执行。动态规划法：在 n <= 20 或 n >= 500 时执行（根据代码逻辑）。贪心法：在 n <= 20 或 n >= 500 时执行（根据代码逻辑）。 c. 对于每次算法调用，使用 clock() 函数精确记录其开始和结束时间，并计算执行耗时（毫秒）。 d. 记录算法返回的总价值和总重量。

#### 数据记录

程序将实时在控制台打印每个测试用例的结果，包括选择的部分物品、总重量、总价值和执行时间。

当 n = 1000 时，程序会自动为动态规划法和贪心法生成详细的 .csv 文件，记录每个物品的选择情况。

实验结束时，程序会打印一个格式化的执行时间总结表，清晰地展示所有 (n, C) 组合下各算法的耗时。实验者应将此表复制并保存。

# 实验数据

本次实验数据量多中多样，仅在此处展示部分物品的价值与重量。完整内容参考附件1数据文档。

表1容量为10000的0-1背包物品统计信息示例

|  |  |  |
| --- | --- | --- |
| 物品编号 | 重量 | 价值 |
| 0 | 85 | 291.64 |
| 1 | 96 | 321.25 |
| 2 | 29 | 334.05 |
| 3 | 78 | 154.13 |
| 4 | 73 | 334.04 |
| 5 | 60 | 248 |
|  |  |  |
| 996 | 72 | 310.91 |
| 997 | 30 | 129.42 |
| 998 | 86 | 179.16 |
| 999 | 75 | 273.11 |

# 实验结果

## 结果展示

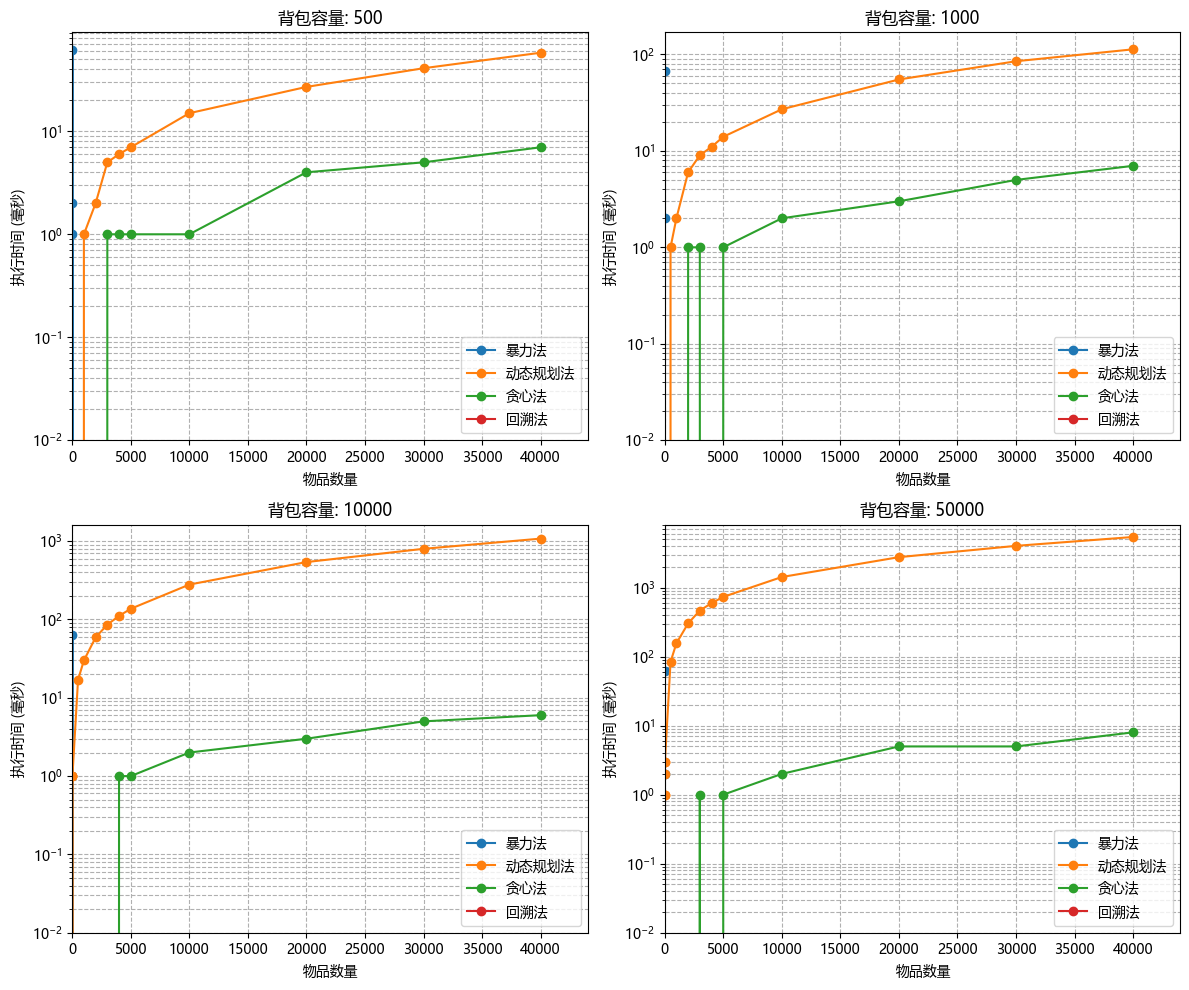


图1 时间复杂度对比图

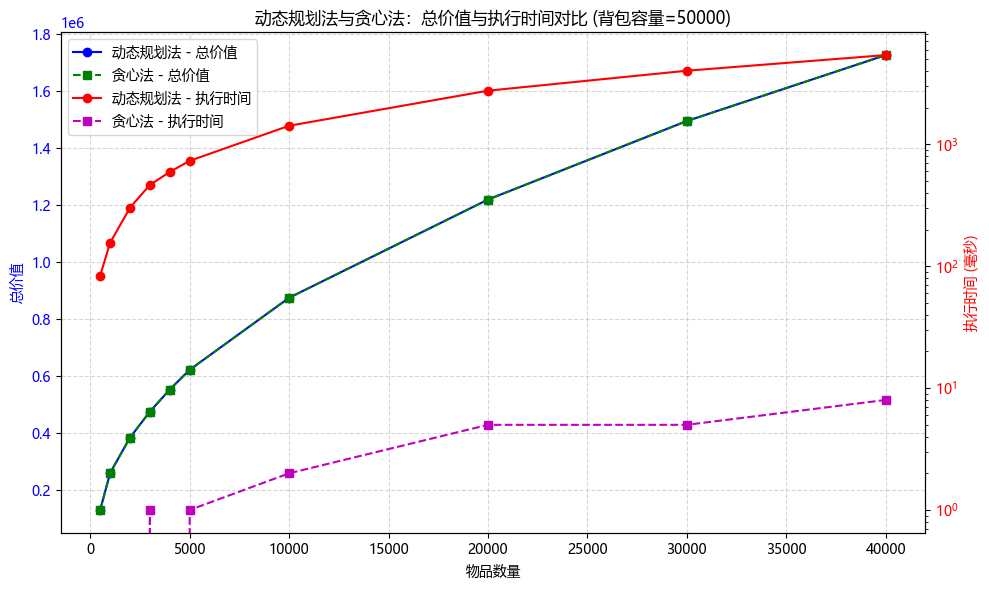


图2 动态规划与贪心法的效率对比

## 实验分析

在本次实验中，我们针对不同规模的物品数量与背包容量组合，对四种0/1背包问题求解算法（蛮力法、回溯法、动态规划法、贪心法）的性能进行了对比分析。实验结果清晰地揭示了各算法的理论特性与实际运行限制。

在实际测试中，算法的局限性表现得尤为明显。由于实验环境的RAM内存有限，当物品数量n超过一个较小的阈值（约20-30）后，采用递归实现的回溯法与迭代实现的蛮力法均因计算量呈指数级增长而导致程序挂起或崩溃，无法在有效时间内完成。一个值得注意的现象是，尽管回溯法在代码中明确应用了“按单位价值排序”和“剩余价值上界”的剪枝策略，但在部分用例中其速度甚至不及蛮力法。我们推测，这是因为当问题规模增大后，解空间树即使经过剪枝，剩余分支依然庞大，而递归调用本身的开销在受限的硬件环境下，反而不如结构更简单的迭代式蛮力法。与此相对，动态规划算法虽然在解的正确性上无可非议，但其空间复杂度与数据规模（尤其是背包容量C）正相关。当问题规模持续扩大时，内存占用成为显著瓶颈，进而影响算法的执行效率。在本次实验中，当物品数量达到约40,000的量级时，其内存需求便超出了实验环境的承载能力。而贪心法则表现出最佳的稳定性与可扩展性，得益于其较低的资源消耗，在所有测试情况下均能快速、正常地运行。

实验结果与理论分析高度吻合。蛮力法和回溯法的执行时间将随n的增加呈指数级增长()，在n > 20之后变得不可行。动态规划法的执行时间主要受n \cdot C影响 ()，在C固定时，时间随n线性增长；在n固定时，时间随C线性增长，因此对于C特别大的情况，其性能会显著下降。贪心法的执行时间主要取决于排序()，在四种算法中速度最快，尤其是在大规模n的情况下。如图1所示的性能曲线图可以直观地展示上述增长趋势。

## 实验结论

精确算法在求解小规模 0/1 背包问题时能够保证最优解，但其时间复杂度受输入数据规模的影响较大。随着数据量增加，精确算法的执行时间显著增长，效率较低。相比之下，近似算法（如贪心法）在牺牲微小准确性的前提下，表现出极高的计算效率。如图 2 所示，贪心法在执行时间上具有显著优势，反应迅速，且最终得到的最优价值与精确算法（动态规划法）相比，仅在小数部分存在微小差异。

由于实验设备的限制，本次实验未能深入探讨四种算法在相同条件下的具体差异。然而，在不考虑空间复杂度的前提下，算法的理论分析具有重要意义。例如，极端假设下，若为每个问题预先构建一个一一对应的答案字典，理论上可以在常数空间复杂度下实现线性时间复杂度。但在现实中，这种假设因资源限制而难以实现。

书本中的算法时间复杂度分析通常基于理想化模型，忽略设备性能、外界环境以及增长率前的常数系数等因素的影响。这导致某些算法在理论上表现一致，但在实际应用中因财力、物力等限制而有所不同。理想模型假设算法性能不受硬件约束，但在生产生活中，实际表现往往需要权衡资源与效率。

本次实验的最大启示在于动态规划法与贪心法的差异。两者均能获得较优的总价值，但贪心法以极低的时间成本展现出更高的实用性。这种特性在实际生活中尤为重要：过于追求全局最优而避免局部最优往往不切实际，除非在文学创作等理想化场景中。而贪心法通过每一步的最优选择，在时间耗费极低的情况下实现接近全局最优的收益，体现了其在实际应用中的高效性和实用价值。

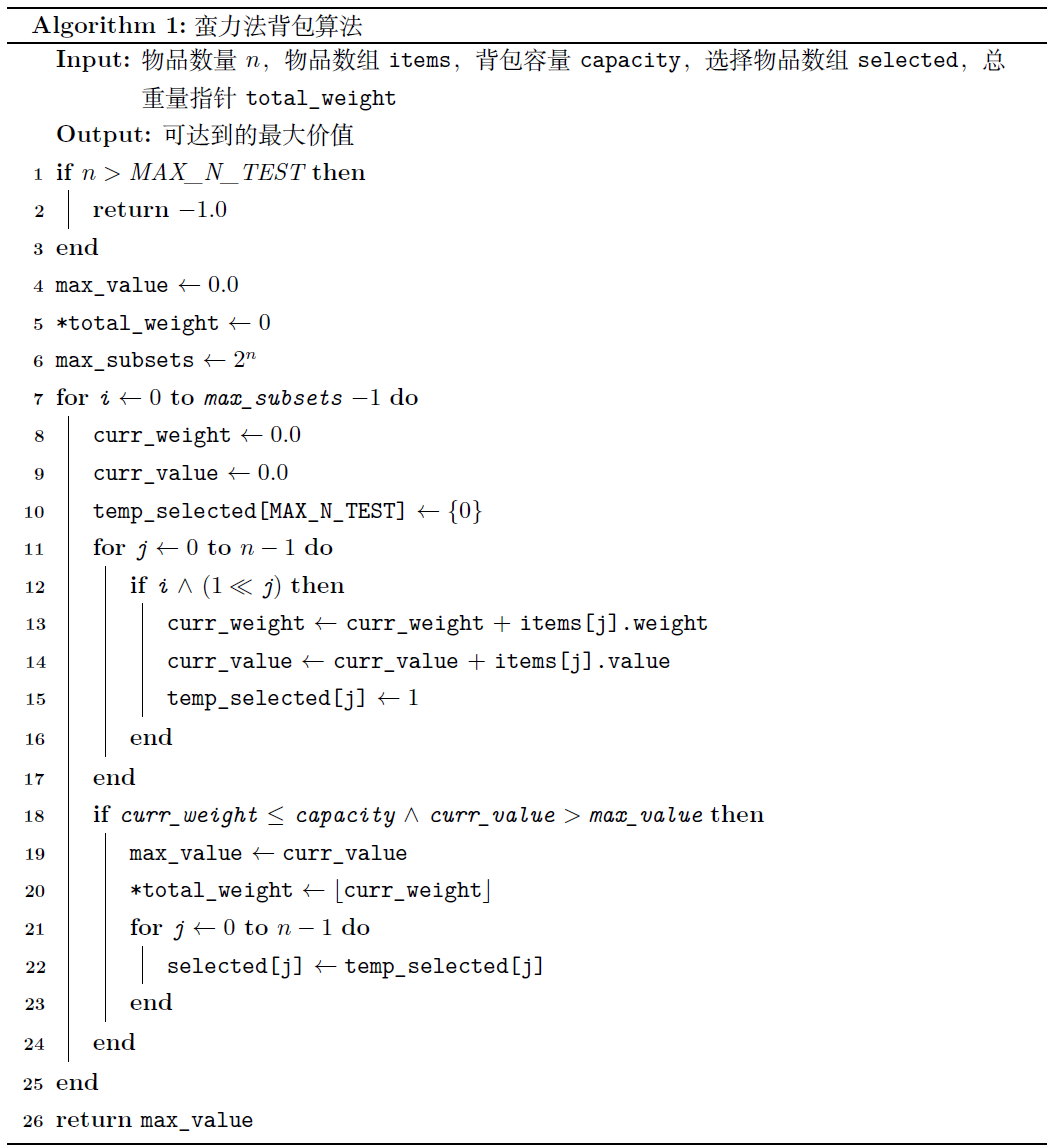
# 附录

## 代码

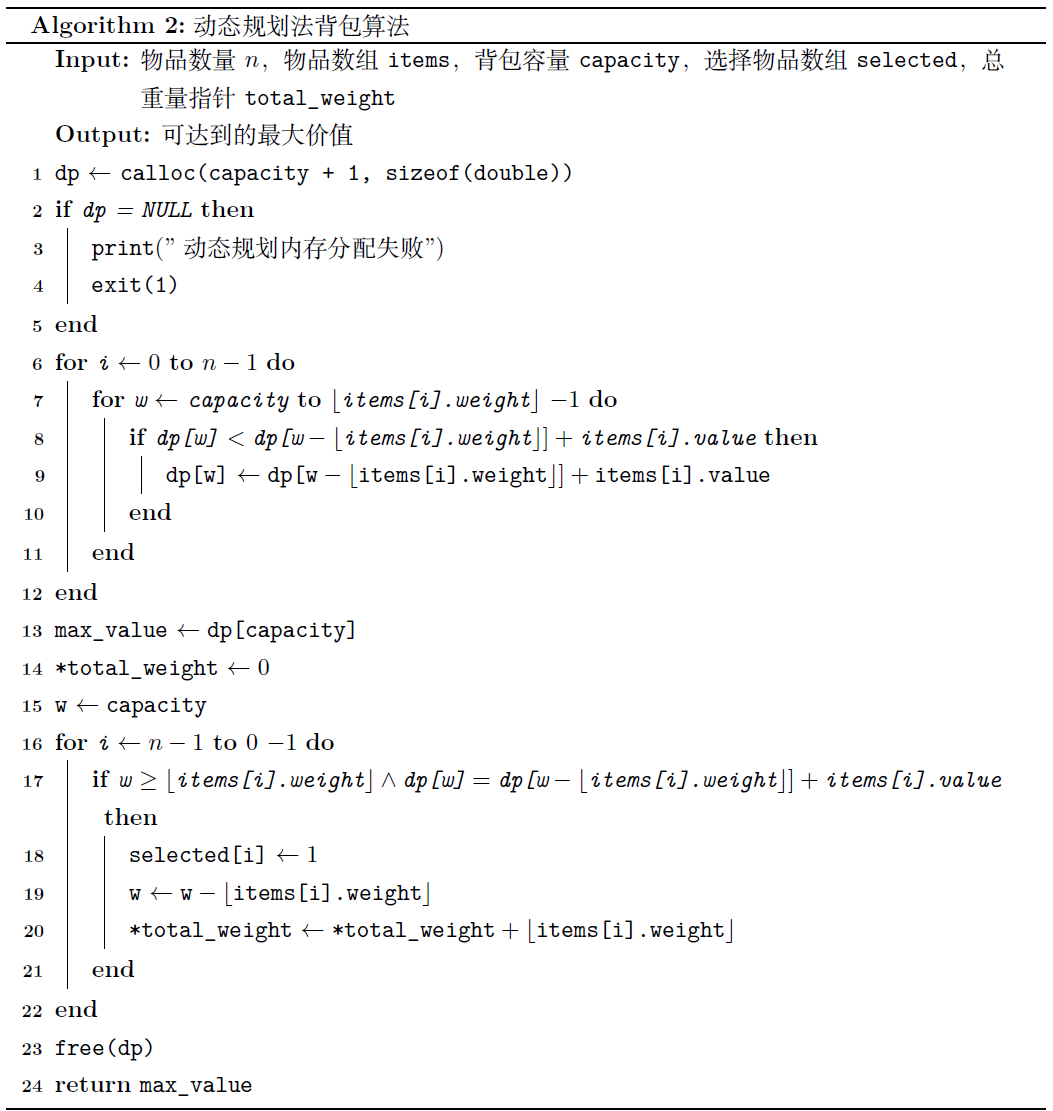
本实验报告的算法实现代码已在GitHub平台公开发布，地址为：https://github.com/user/project。

## 伪代码

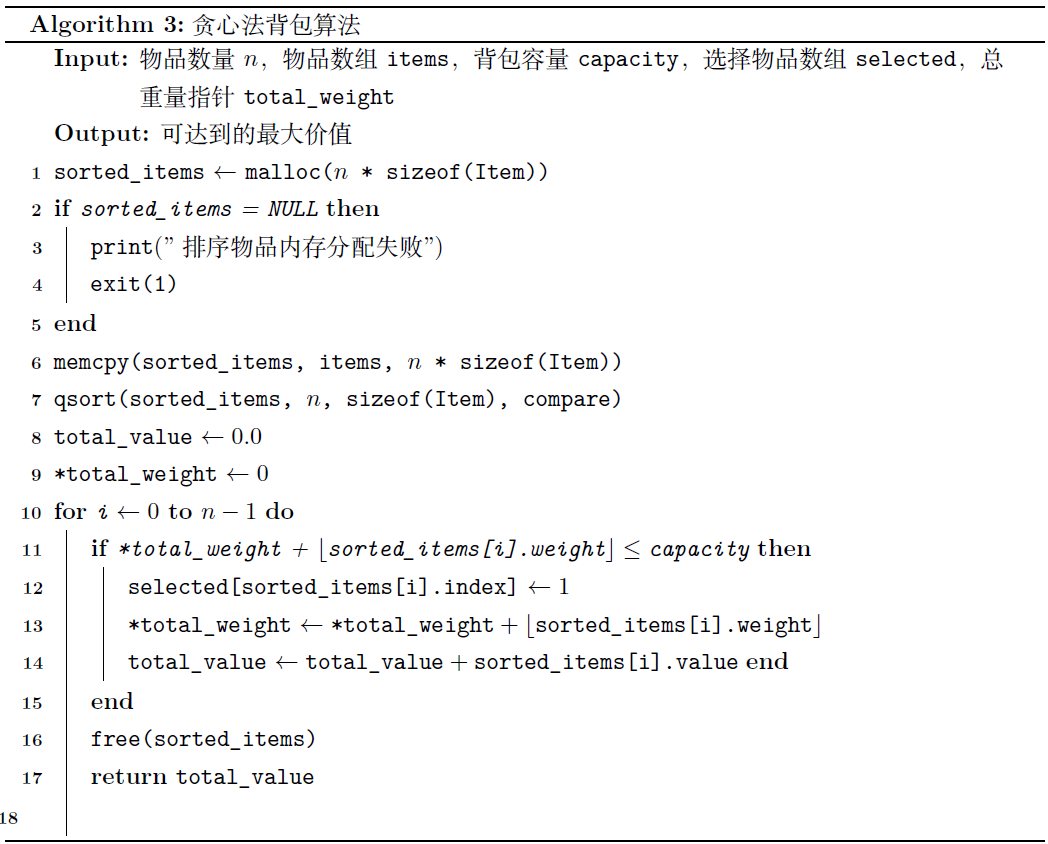
#### 蛮力法



#### 动态规划法



#### 贪心法



#### 回溯法

