

**《算法设计与分析》实验报告**

**题目：SubsetSumMadeSimple算法实现**

|  |  |  |  |
| --- | --- | --- | --- |
| **姓名** | **学号** | **任务分工** | **成绩** |
| **石润林** | **3022244244** | **全部** |  |

SubsetSumMadeSimple算法实现

**摘要**:本文致力于介绍和实现arXiv:1807.08248v1 [cs.DS] 22 Jul 2018论文中提出的用于求解子集和问题（Subset Sum Problem）的算法，名为“SubsetSumMadeSimple”。子集和问题是计算复杂性理论和算法设计领域的一个经典问题，其核心在于确定一个给定整数集合中是否存在一个子集，其元素之和正好等于特定的目标值。传统解法主要包括暴力法、动态规划法和回溯法。暴力法通过枚举所有可能的子集组合来找出满足条件的子集，其时间复杂度为O()，其中n是集合中元素的数量。动态规划法则采用一种自底向上的策略，通过构建一个表格来存储中间结果，以避免重复计算，其时间复杂度为O(nu)，其中u是目标和。回溯法则是一种优化的暴力搜索，它在搜索过程中剪枝，避免不必要的计算，但最坏情况下的时间复杂度仍然接近O()。而SubsetSumMadeSimple算法利用FFT和D&C将时间复杂度降低至O()，显著提高了数据量较大时的问题求解速度。

**关键词**: 子集和问题，时间复杂度优化，FFT快速傅里叶变换，分治

# 实验目的

1. 子集和问题的背景及意义

子集和问题（Subset Sum Problem）是计算理论和算法设计中的一个基础问题，具有重要的理论和实际意义。该问题定义为：给定一个整数集合和一个目标整数，确定是否存在集合的一个子集，使得其元素之和等于目标整数。这是一个NP完全问题，对计算复杂性理论具有重要价值。

子集和问题在多个领域都有广泛应用，如，加密学——用于评估密码系统的安全性；资源分配——在工业和商业领域，用于优化资源分配；计划和调度——项目管理中，确保任务按时完成且资源有效利用。

理论上，子集和问题帮助我们理解P类和NP类问题之间的关系。研究这个问题的高效算法，可以推动算法理论的发展，特别是在处理NP完全问题方面。

尽管有其理论重要性和广泛的实际应用，子集和问题的求解面临巨大挑战。由于其NP完全性质，为大规模数据集找到有效且高效的算法非常困难，这使得设计新算法或改进现有算法成为计算科学领域的研究热点。

1. 说明课题的任务与目的

阅读论文arXiv:1807.08248v1 [cs.DS] 22 Jul 2018，理解论文中提出的用于求解子集和问题（Subset Sum Problem）的SubsetSumMadeSimple算法，并正确的实现它，而后进行相关的时间复杂度分析。

1. 说明解决方案主要思路

认真阅读论文并理解其思路，按照所给的伪代码即可实现其SubsetSumMadeSimple算法。主要难点在于理解算法实现的证明部分的理论推导。

# 2.实验设计流程

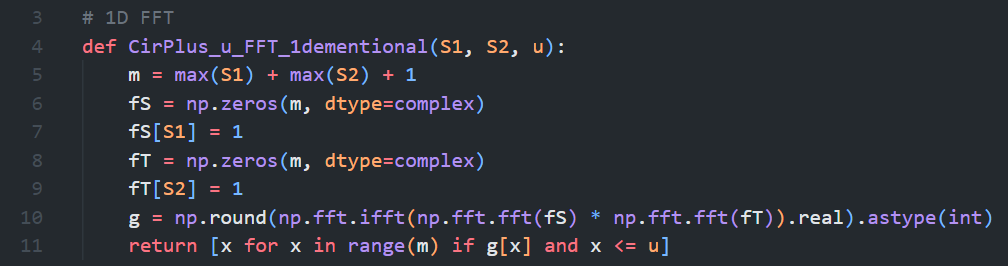
1. 总体架构

阅读论文并理解算法——>实现算法——>运行以测试正确性并进行时间复杂度分析

1. 关键技术1

利用一维FFT计算X⊕u Y = { x + y | x ∈ X, y ∈ Y } ∩ [u].

将集合元素求和问题映射到多项式乘法上，即利用一维FFT实现两个集和中元素的求和，并取小于等于u（理论上为向上取整，但实际SubsetSum问题中u即为整数）的和：将集合映射到多项式上，集合中元素值对应次项系数为1，其余项系数为0，进行FFT进行求解。



具体的，使用快速傅里叶变换（FFT）和逆快速傅里叶变换（IFFT）来高效地计算多项式乘积的系数的详细说明：

1. 多项式表示：fS和fT是两个由S1和S2映射而来的多项式，其中数组的元素表示多项式的系数。这里假设数组中的每个元素对应于多项式中的项，其中元素的值是该项的系数，而数组索引是该项的次数。

2. 快速傅里叶变换（FFT）：np.fft.fft(fS)和np.fft.fft(fT)分别对fS和fT执行快速傅里叶变换。FFT 将多项式从其系数表示转换为点值表示，即在不同复数根上的值。

3. 乘积计算：对这两个变换的结果进行逐元素乘法。这相当于在点值表示中计算两个多项式的乘积。

4. 逆快速傅里叶变换（IFFT）：np.fft.ifft(...)对乘积的点值表示执行逆变换，将其转换回系数表示。这一步得到了两个多项式乘积的系数。

5. 取实部和四舍五入：由于FFT和IFFT可能引入小的浮点数误差，结果取实部并四舍五入到最接近的整数，以确保系数是整数。

6. 类型转换：结果数组转换为整数类型，便于后续程序处理。

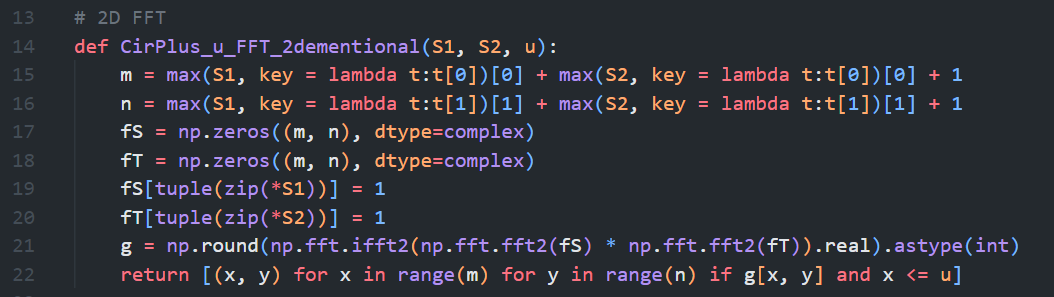
7. 取FFT结果中小于等于u的值计入列表以返回

时间复杂度为O(ulogu)（因为参数S1与S2中元素最大不超过u）。

1. 关键技术2

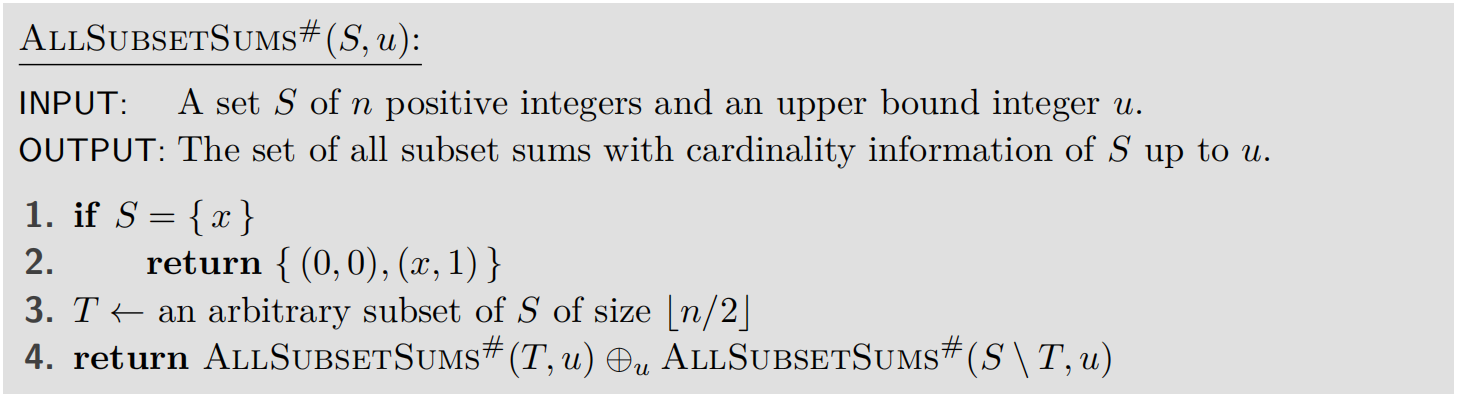
利用二维FFT计算X⊕u Y = { (x1+y1, x2+y2) | (x1, x2)∈X, (y1, y2)∈Y }∩[u]×N

与一维FFT同理，将(值，势)二元组集合映射到两个多项式上进行二维FFT运算及后续操作，时间复杂度为O(uvlog(uv))，u同上，v为S1、S2中势的最大值。

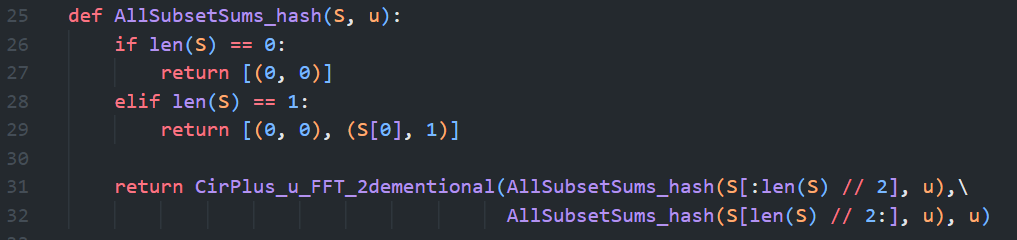


1. 关键技术3

计算(X) = {(∑Y, |Y|) | Y⊆X}∩[u]×N 其中[u] = {0, 1, 2, ..., }



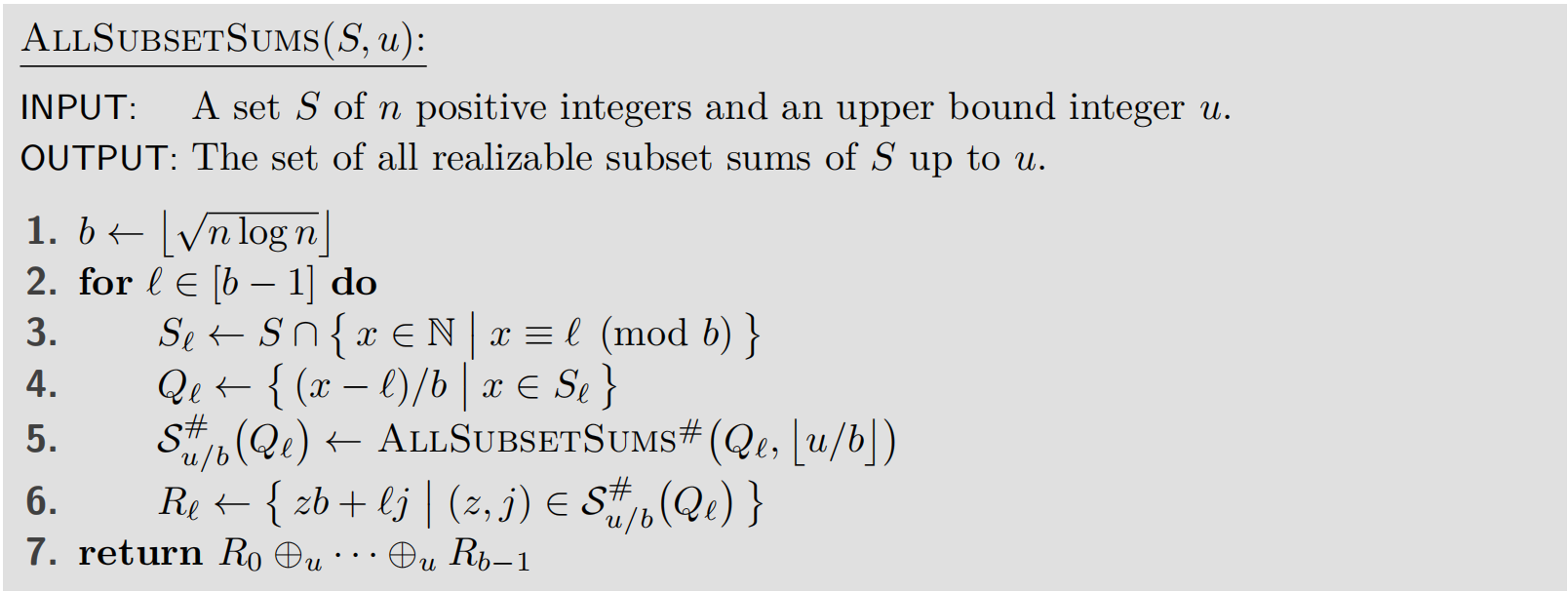
计算一个集合的全部子集的(和, 势)二元组，采用分治法解决，分治方法为等长二分，合并时调用二维FFT计算一次⊕u，划分的最小问题规模有空集、单一元素集两种情况。



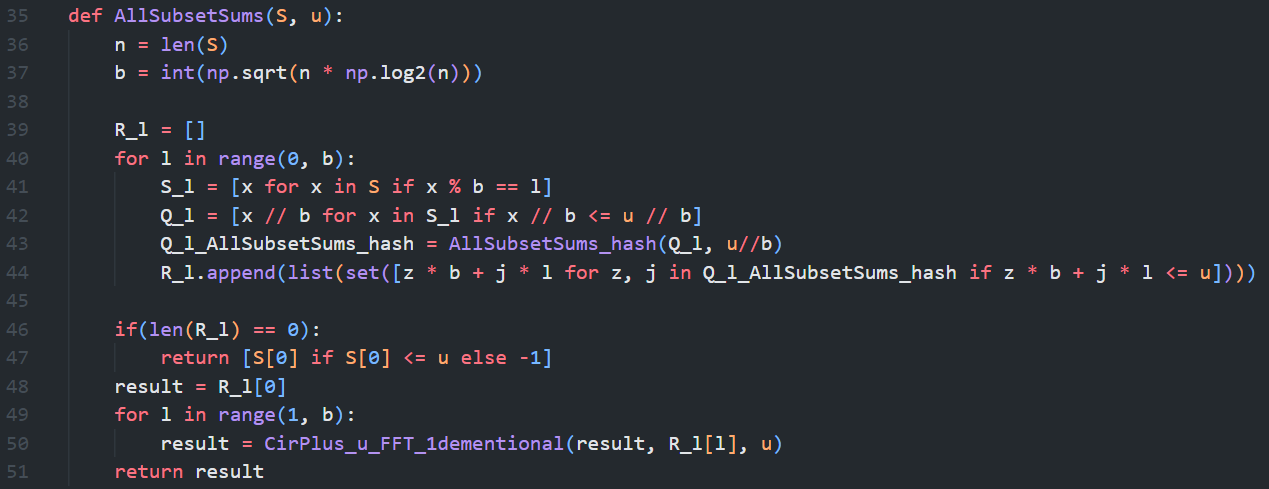
递归方程为：T(n) = 2T(n/2) + O(unlogu)，由主方法case2得时间复杂度为O(unlogulogn)

1. 关键技术4

计算(X) = {∑Y| Y⊆X}∩[u]

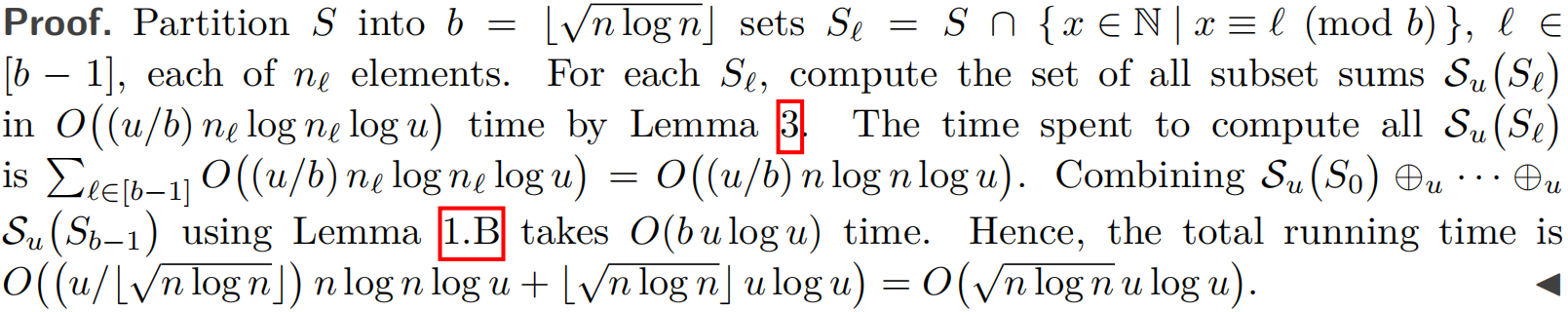


计算一个集合的全部子集的和：取b=，将当前集合划分为取模b的同余类后对中元素整除b，得到，对每个计算()，得到的()中二元组(z, j)对应的z\*b+j\*l即为相应的原集合的一个子集的和，这很显然，因为j为势，表示了(X)中z为j个模b同余l的元素的和，而z为j个整除的商的和，每个均差了一个余数l，故z\*b+j\*l组成的集合即为()；合并所有时调用一维FFT计算b-1次⊕u，即得到了(S)。由关键技术3结论，计算每个()的时间复杂度为(表示的势，总和为n)。



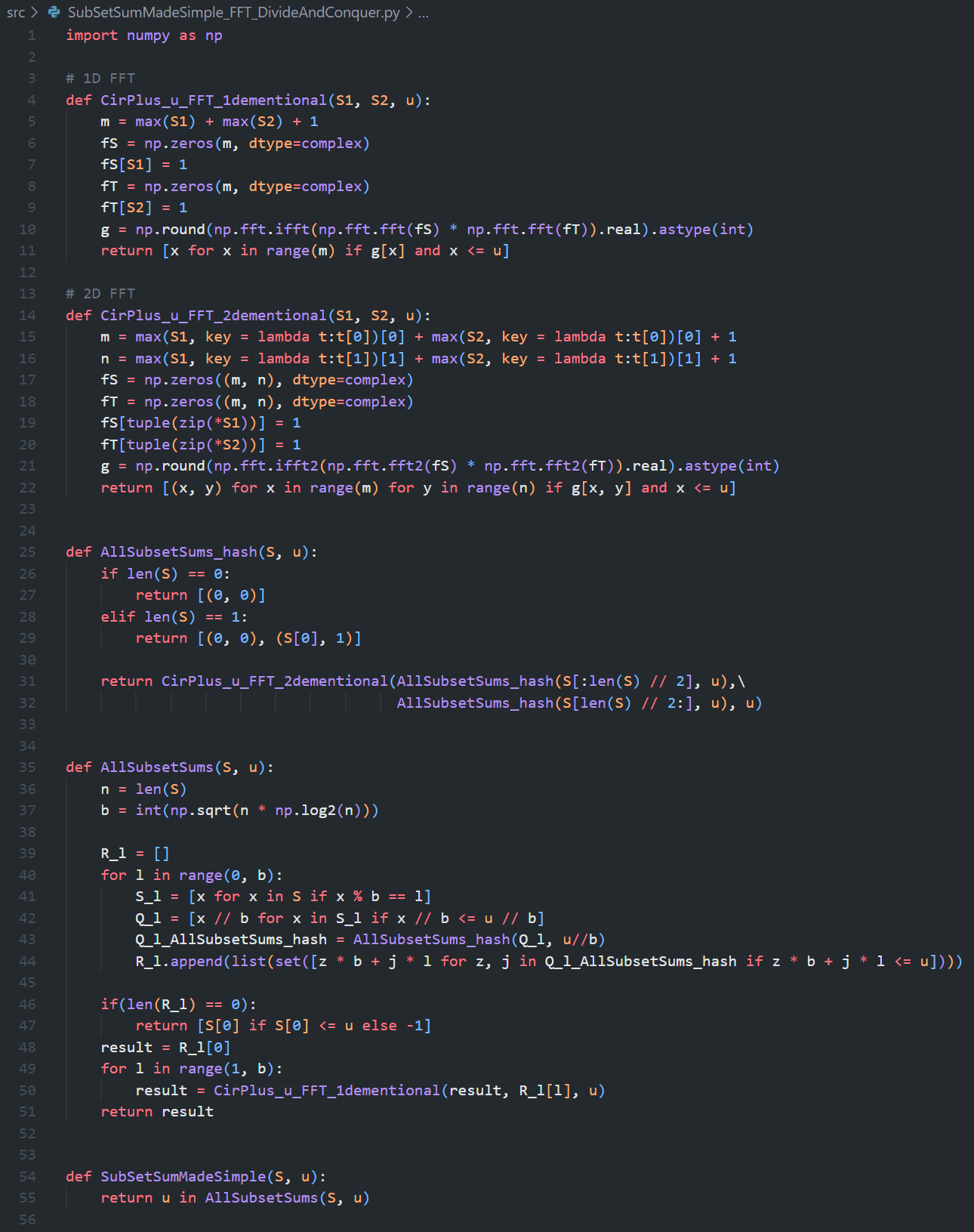
1. 关键技术5

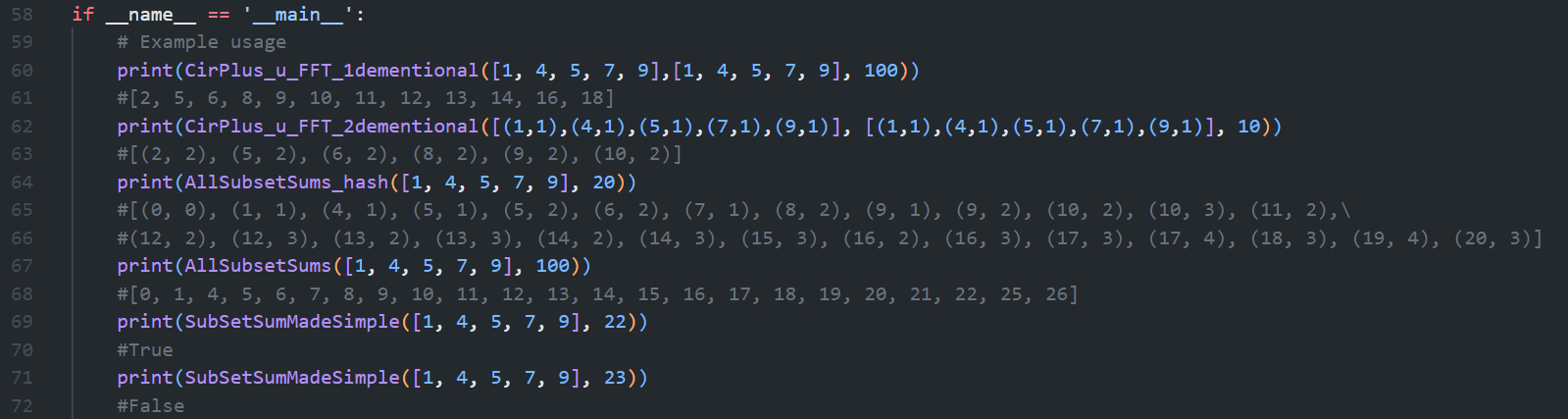
算法的理论时间复杂度分析



即计算每个()时间为(表示的势，总和为n)，计算全部b个()(l∈[b-1])的时间为，合并所有()时调用一维FFT计算b-1次⊕u的时间为，取，从而计算(S)的总体时间复杂度为：

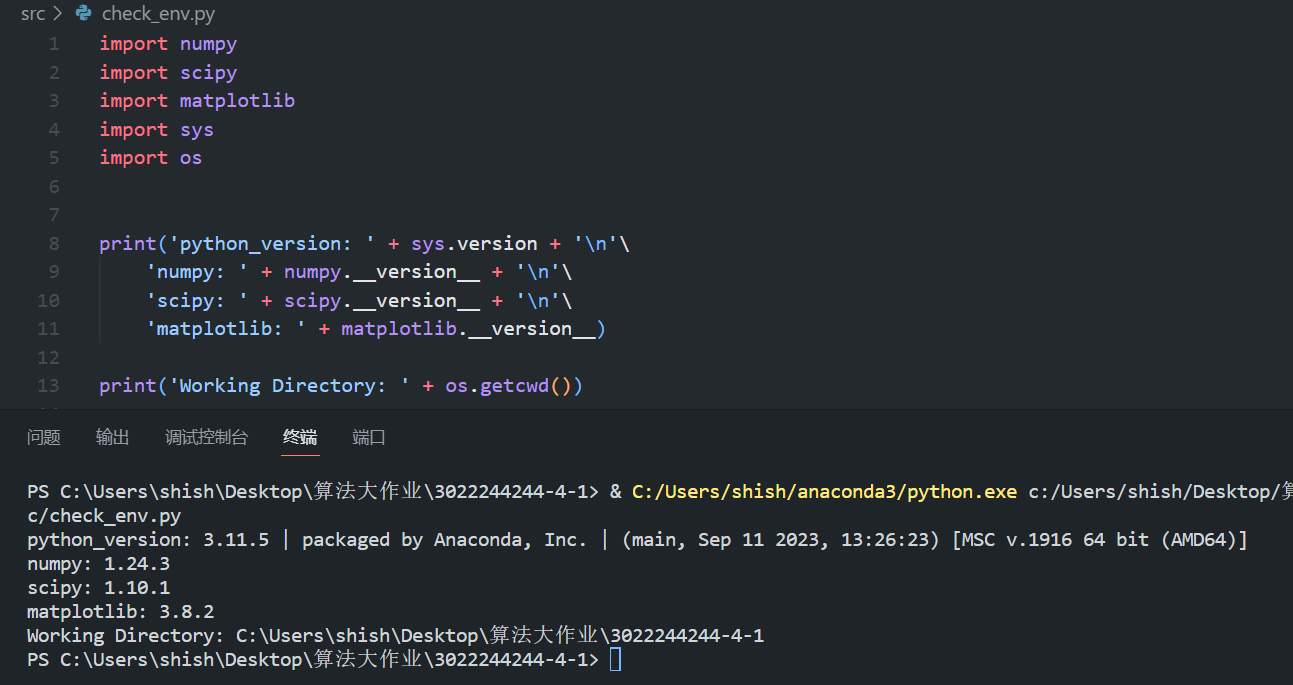
1. 关键技术6

由于SubsetSum问题可以约化到计算(S)的问题（计算(S)并判断解集中是否存在u即可）故可用上述算法解决SubsetSum问题。SubsetSumMadeSimple算法实现完整代码：



# 3.实验结果及复杂性分析

1. 实验环境设置



python\_version: 3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC v.1916 64 bit (AMD64)]

numpy: 1.24.3

scipy: 1.10.1

matplotlib: 3.8.2

工作路径需要设置为: \3022244244-4-1

1. 测试方法及性能评价指标

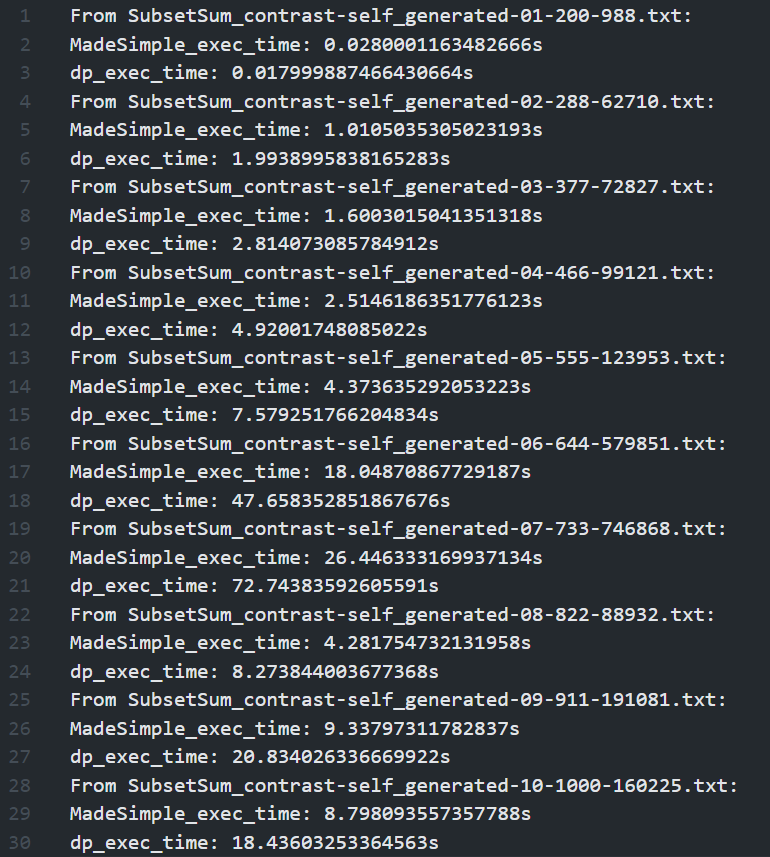
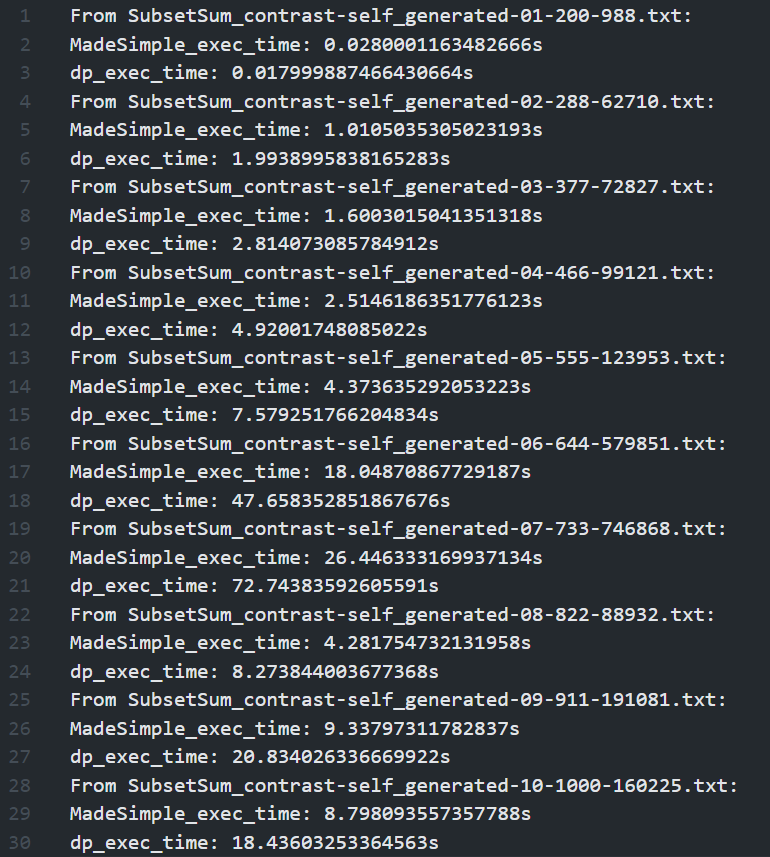
\*实验测试程序exec.py的工作路径需要设置为: \3022244244-4-1

算法正确性测试在上文的具体代码下就有使用实例，可以手动运行代码以进行测试，本实验中着重进行算法性能的测试，测试主要分为三部分

1. 相同输入数据用传统dp算法和SubsetSumMadeSimple算法求解的求解时间比较
2. SubsetSumMadeSimple算法时间复杂度的验证（线性回归）
3. SubsetSumMadeSimple算法分别关于n和u的时间复杂度的验证（非线性回归）

测试数据集均为自己生成，下面展开说明。

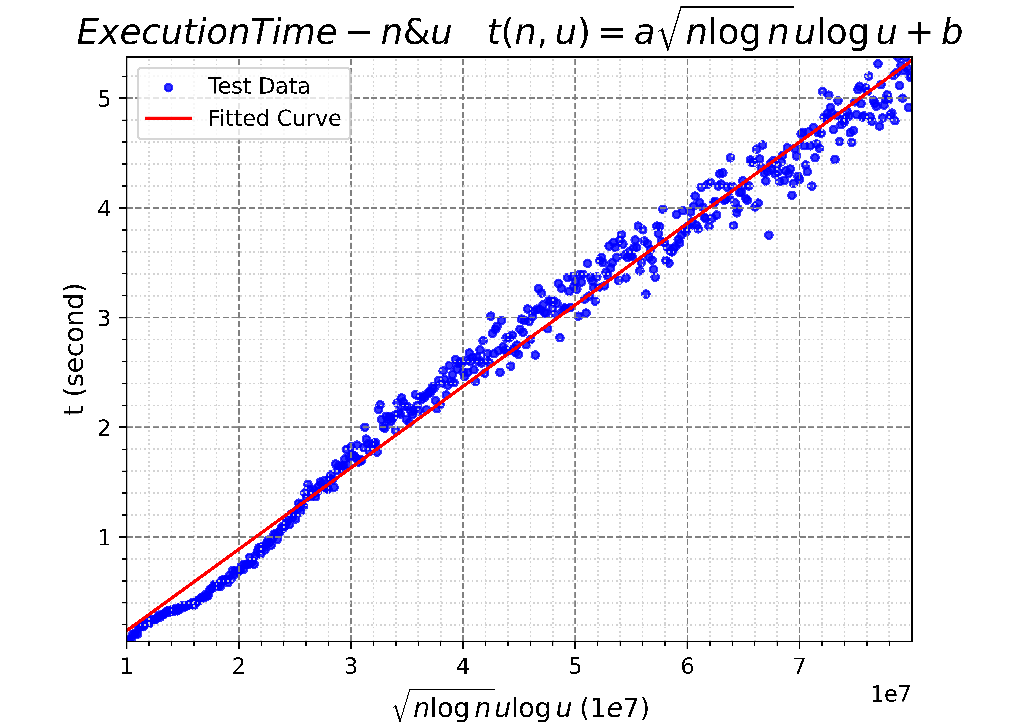
1.相同输入数据用传统dp算法和SubsetSumMadeSimple算法求解的求解时间比较：



SubsetSum\_contrast-self\_generated-x-n-u.txt为随机生成的第x个n个集合元素目标和为u的数据，可见数据量非常小（如图中第一个测试结果）时dp快于MadeSimple，因为MadeSimple需要一些函数调用等时间开销，而当数据量更大时，明显地看到MadeSimple的运行时间优于dp。

2.SubsetSumMadeSimple算法时间复杂度的验证（线性回归）

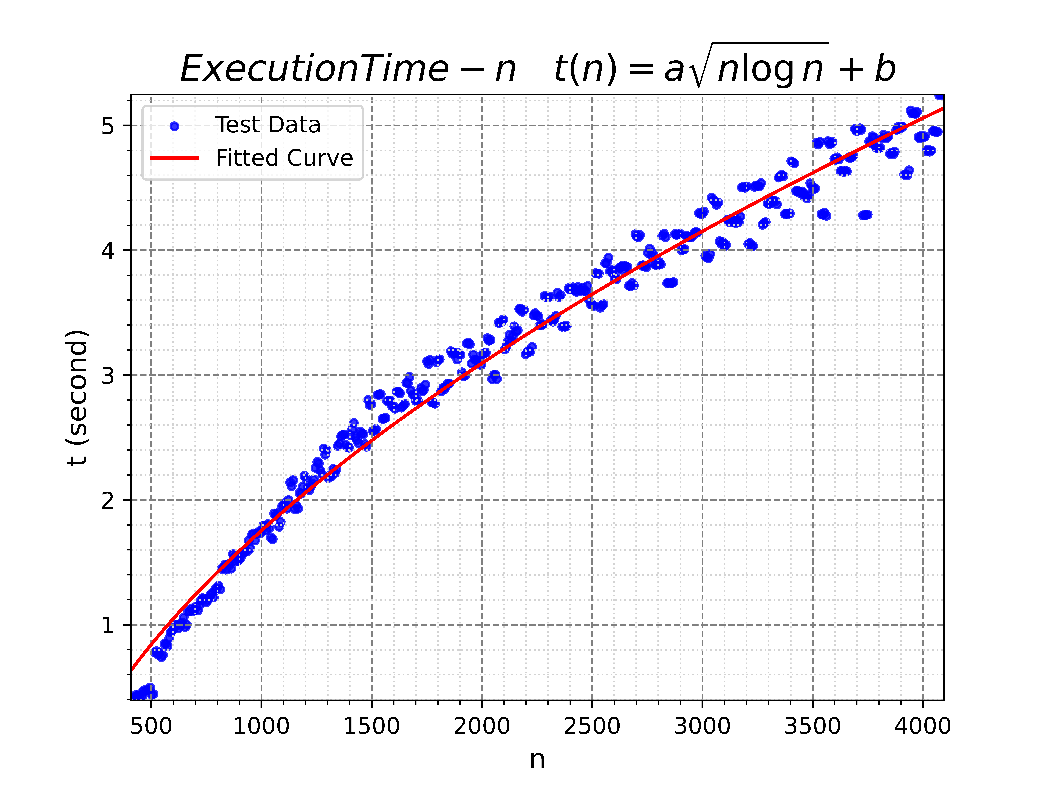
生成指定数量的的等间距的n、u数据量的测试数据，以为横坐标，运行时间为纵坐标绘图，表示出平面上的(, t)点及拟合的y=ax+b直线



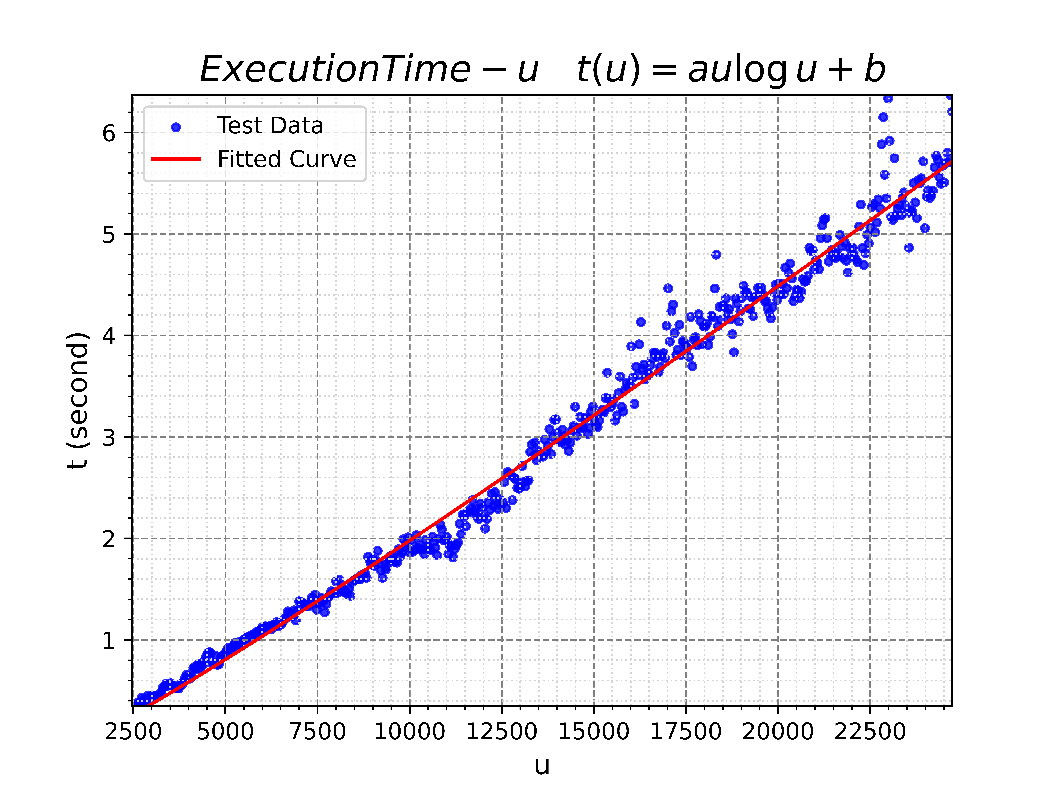
此图测试数据为512个，实际点以及拟合直线可见总体时间复杂度符合

3.SubsetSumMadeSimple算法分别关于n和u的时间复杂度的验证（非线性回归）

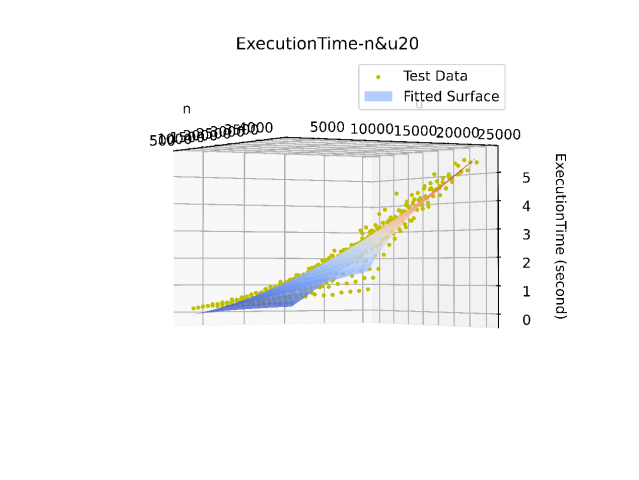
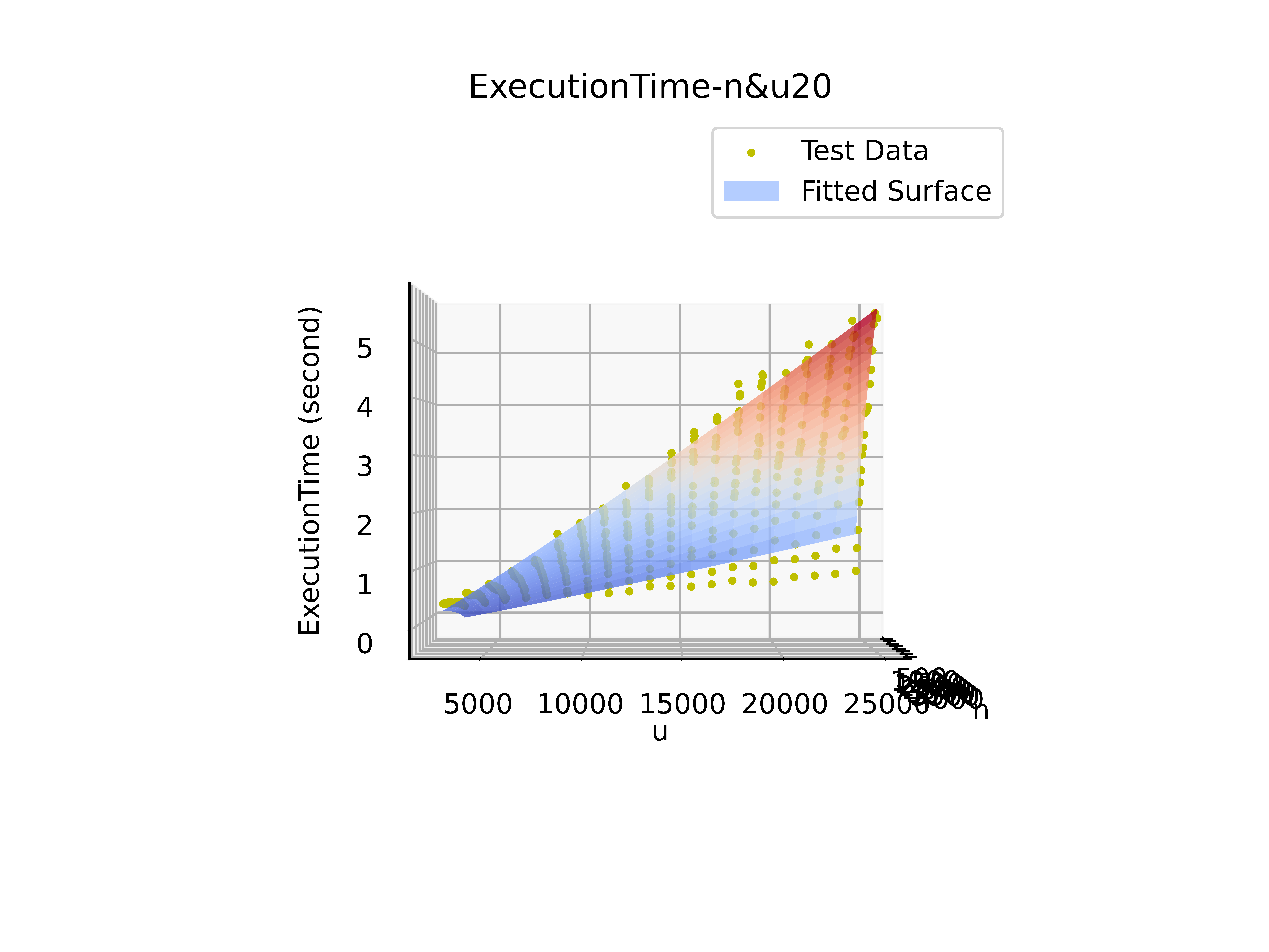
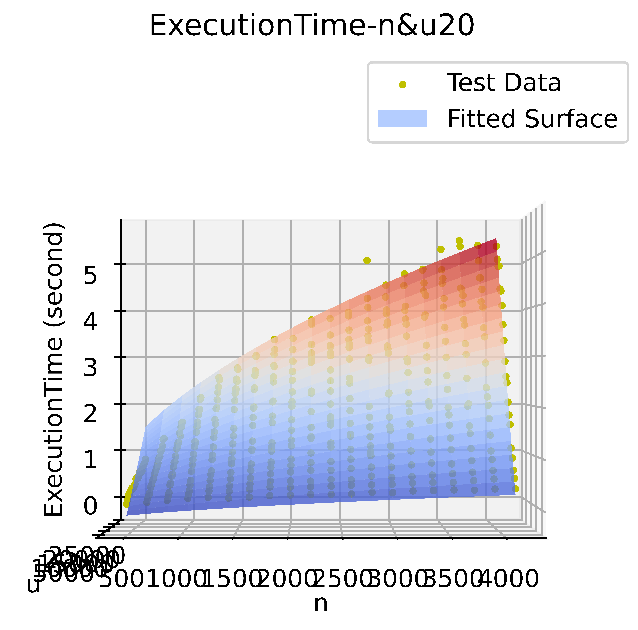
1)关于n（512个数据点）：

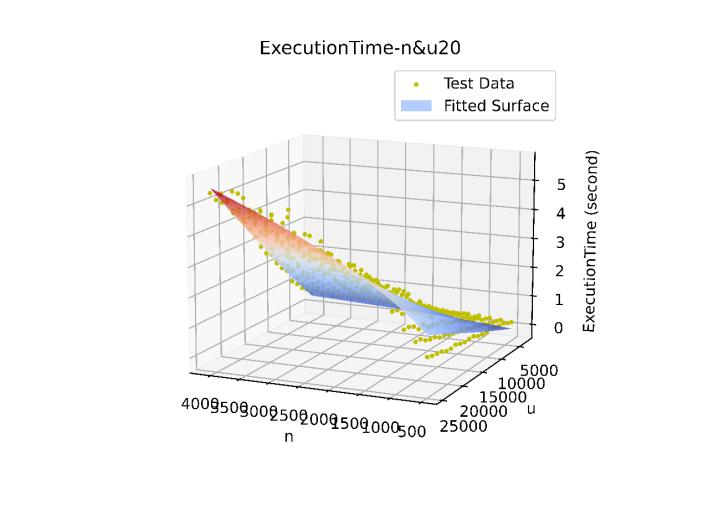
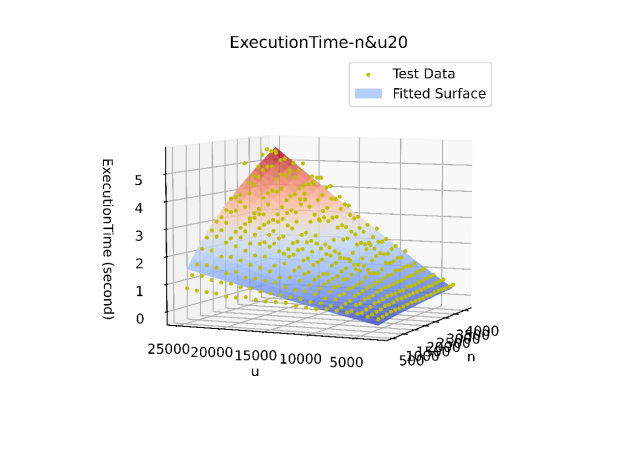
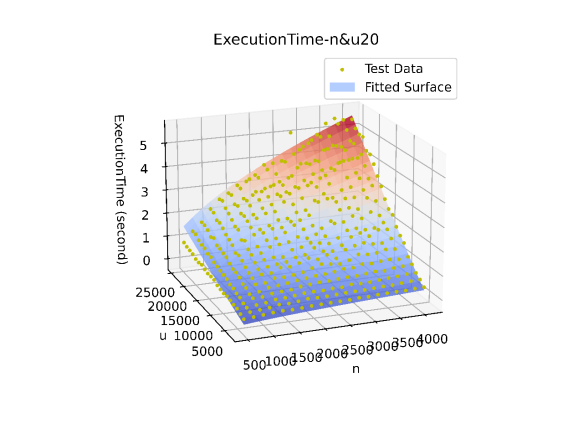


2)关于u（512个数据点）：



3)关于n和u（20\*20个数据点）：





1. 实验结果分析

由上述测试可见对于SubsetSumMadeSimple算法的代码复现很成功，性能相较dp有十分可观的提升，并且其时间复杂度的确符合。

# 4.结论与展望

（1）解决方案的整体评价，包括优点与不足

优点：本次对于arXiv:1807.08248v1 [cs.DS] 22 Jul 2018论文中的SubsetSumMadeSimple算法的实现非常成功，阅读论文时的理论理解到位，并且代码实现符合理论预期。

不足：测试分析时耗时较长，按照上文中的测试数据量，取4次运行时间平均值，在本人的主机（CPU为i513490F）上进行一轮测试需要近8小时，或可利用python多进程等方法进行测试上的改进，以提高测试分析的效率。

1. 进一步工作的展望

在SubsetSum问题方面，可以继续研究arXiv:1807.11597v3 [cs.DS] 26 Nov 2018中时间复杂度为Õ(n+u)的随机化算法（为的简写，表示多对数时间）。

# 5.参考文献

[1] Konstantinos Koiliaris and Chao Xu. Subset Sum Made Simple. arXiv:1807.08248

[cs], July 2018. arXiv: 1807.08248.