《数据结构》课程实践报告

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院、系 | 计算机学院 | | 年级专业 | 软件工程 | 姓名 | 梅子羽 | 学号 | 2327406107 |
| 实验布置日期 | | 2024.9.9 | | 提交  日期 | 2024.9.23 | | 成绩 |  |

课程实践实验1-2：算法分析初探（选做）

## 一、问题描述及要求

在分析算法的效率时，经常要关注两种复杂度：时间复杂度和空间复杂度。

本次实践拟通过针对某一个问题的不同实现算法，使同学们认识不同算法在时空复杂度上存在的差异，从而理解算法为什么需要优化，以及如何看出算法得到了优化。

题目1：连续子序列最大和问题

该问题关注一个序列s，其元素值存储在一维整数数组s.array, 数组大小为s.n，希望从s.array中找出一个连续子序列，该子序列各元素的和最大。如果序列元素都是负数，计算结果返回0。

假设S是整数序列 A0， A1，... An-1 （可能有负数），求 A0 ~ An-1 的一个子序列 Ai ~ Aj，使得 Ai到 Aj 的和最大。

例如：序列 0, -3, 6, 8, -20, 21, 8, -9, 10, -1, 3, 6, 5，则最大子序列和为 43。

【输入形式】序列长度及序列中的元素。  
【输出形式】最大子序列的和。  
【样例输入】

13

0 -3 6 8 -20 21 8 -9 10 -1 3 6 5

【样例输出】

43 5-12（第一项是和，第二项是区间从0开始）

下面依次给出了该题目的3个实现算法，请回答下列问题

（1）请先编写程序验证这3个实现算法是否正确

（2）用事前分析法分析这3个算法的时间复杂度

（3）为s.n设定一个适当的规模（比如50，100），可通过随机函数产生这批数组元素，应包含负数。假设算法开始时间为t1，结束时间是t2，通过t2-t1可以衡量算法实际需要的运行时间。请记录并对比这3个算法的实际运行时间的差异，看看它们与（2）中估量的时间数量级是否一致。

(4) 最后请给出你对这3个算法的分析结论

题目2：递归的空间复杂度

假设需要输出由小到大，从1 到n的所有的数字，提供了下面两种算法：

在你的电脑环境中，测试当n达到多少时，算法2-1会崩溃，这时算法2-2工作正常吗？

递归程序的时间复杂度的确定有时比较难以分析，有一种方法是将递归转换成非递归后来分析其的时间复杂度，

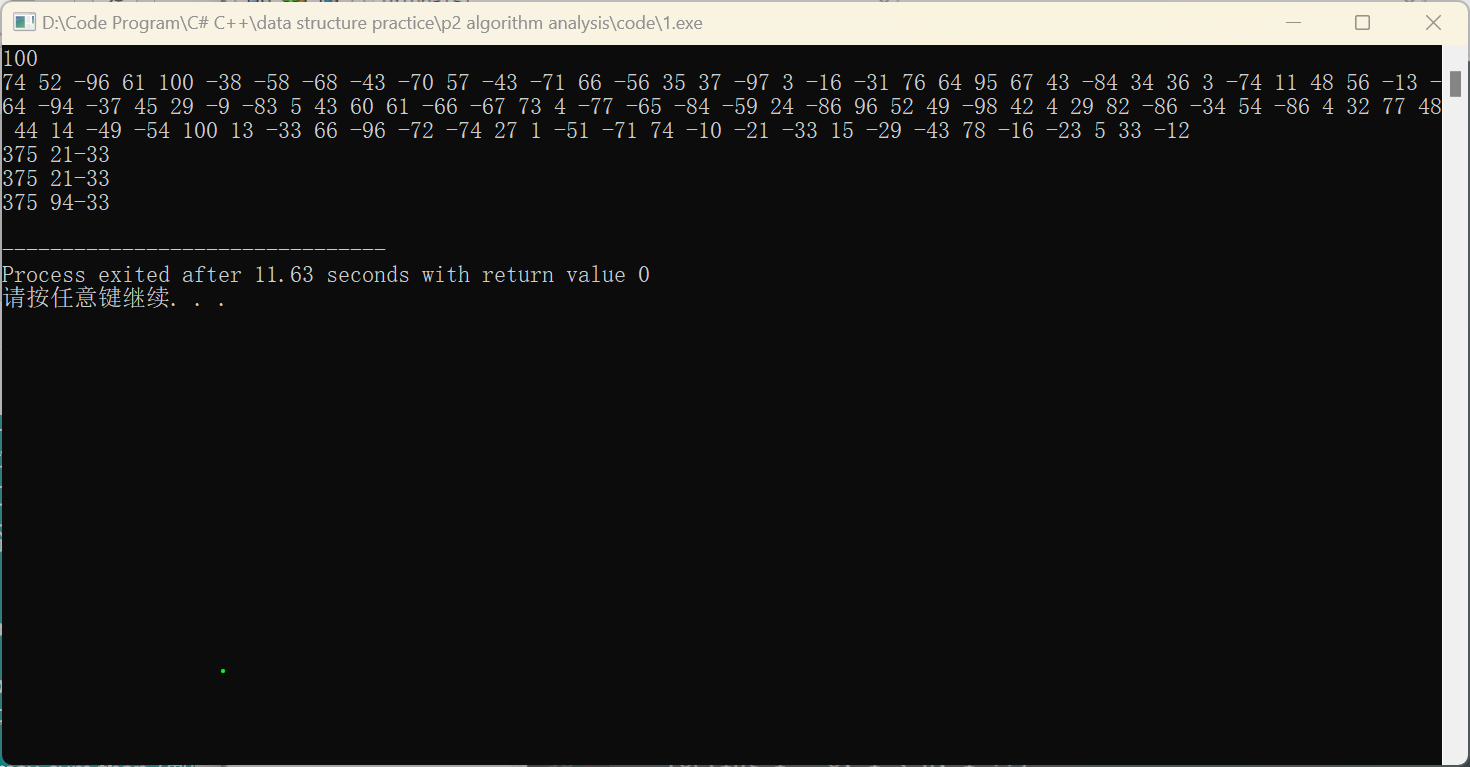
当然两者是不完全等价的，本题两个算法时间复杂度都为O（n），

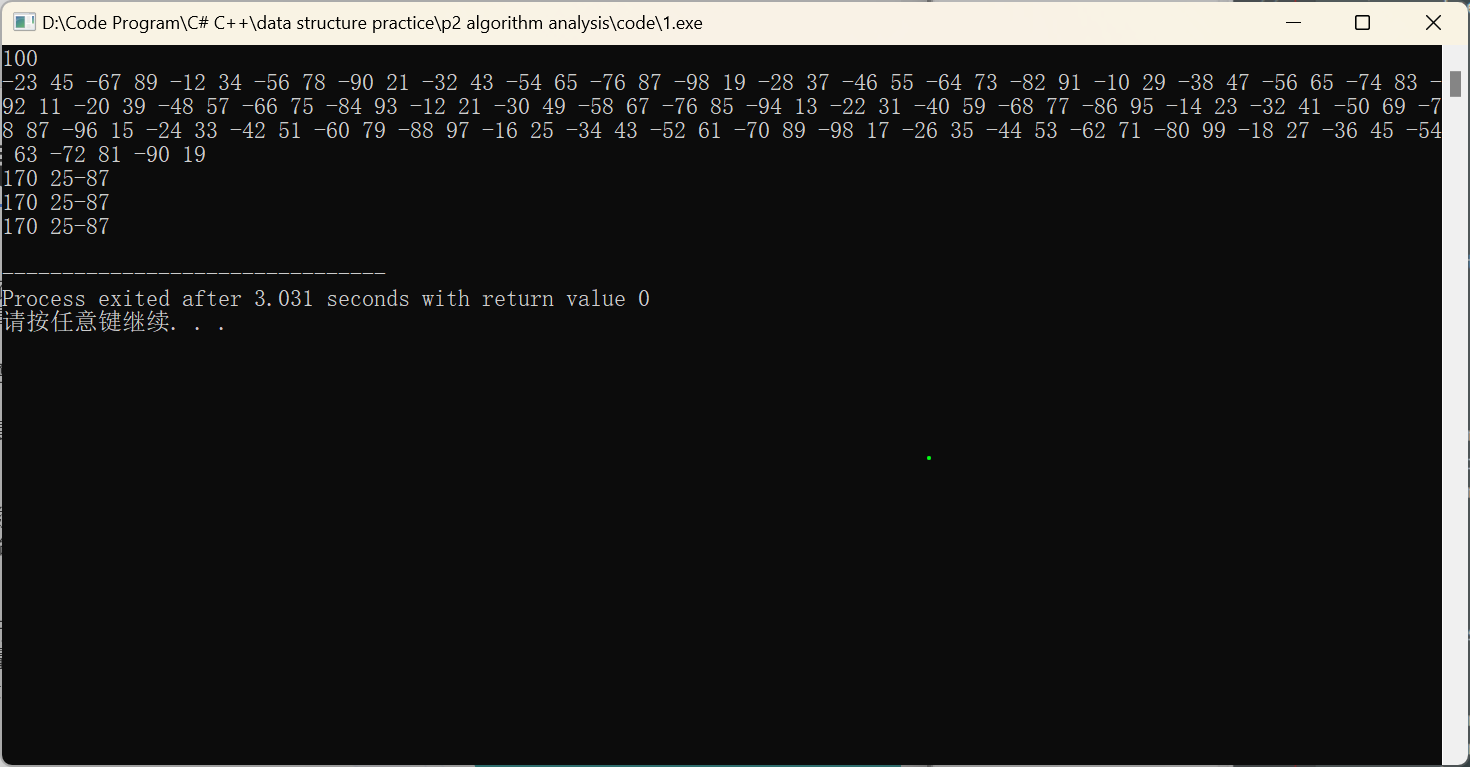
当n取一个适当值时，请计算程序实际运行时间，看看这个结论是否正确。

## 二、实验分析与探讨

问题一：

（1）用两组含有100个整数的数据对三个算法进行测试，用来检验其正确性（答案按顺序出现），以下是程序运行结果截图：





我们可以发现，三种算法在最大子序列和的处理上都是正确的，但是在子序列区间的确定上，只有第一二种算法是正确的，第三种算法的子序列起点没有找对，这是其本身的一些细节代码出现了问题。

（2）对于算法1：

主体代码为三重循环，外层n次，中层也为n次，内层j – i +1次，中层和内层用等差公式求和，其结果为1 + 2 + 3 + ... + n = n \* (n + 1) / 2次。简单计算即得时间复杂度为O(n^3)

对于算法2：

主体代码为二重循环，实际上就是算法1中层和内层的结构，直接是n \* (n + 1) / 2次。简单计算即得时间复杂度为O(n^2)

对于算法3：

主体代码为一重循环，时间复杂度为O(n)

（3）由于要大量的测试数据，所以添加一个随机数生成函数。

void randNums(vector<int> &nums, int n)

{

srand(static\_cast<unsigned int>(std::time(0)));

for (int i = 0; i < n; ++i)

{

int randomNumber = rand() % 201 - 100;

nums.push\_back(randomNumber);

cout << randomNumber;

if (i < n - 1)

cout << " ";

else

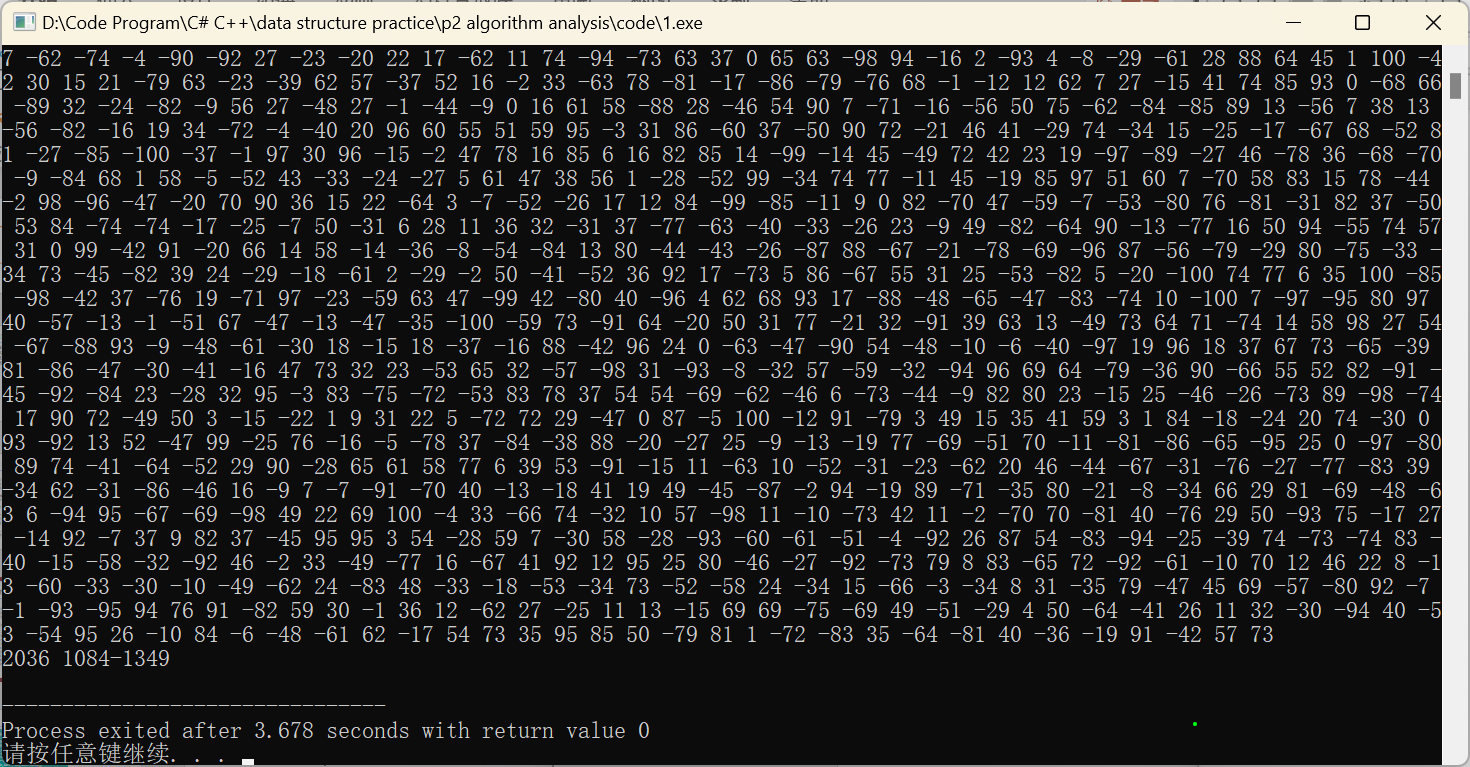
cout << endl;

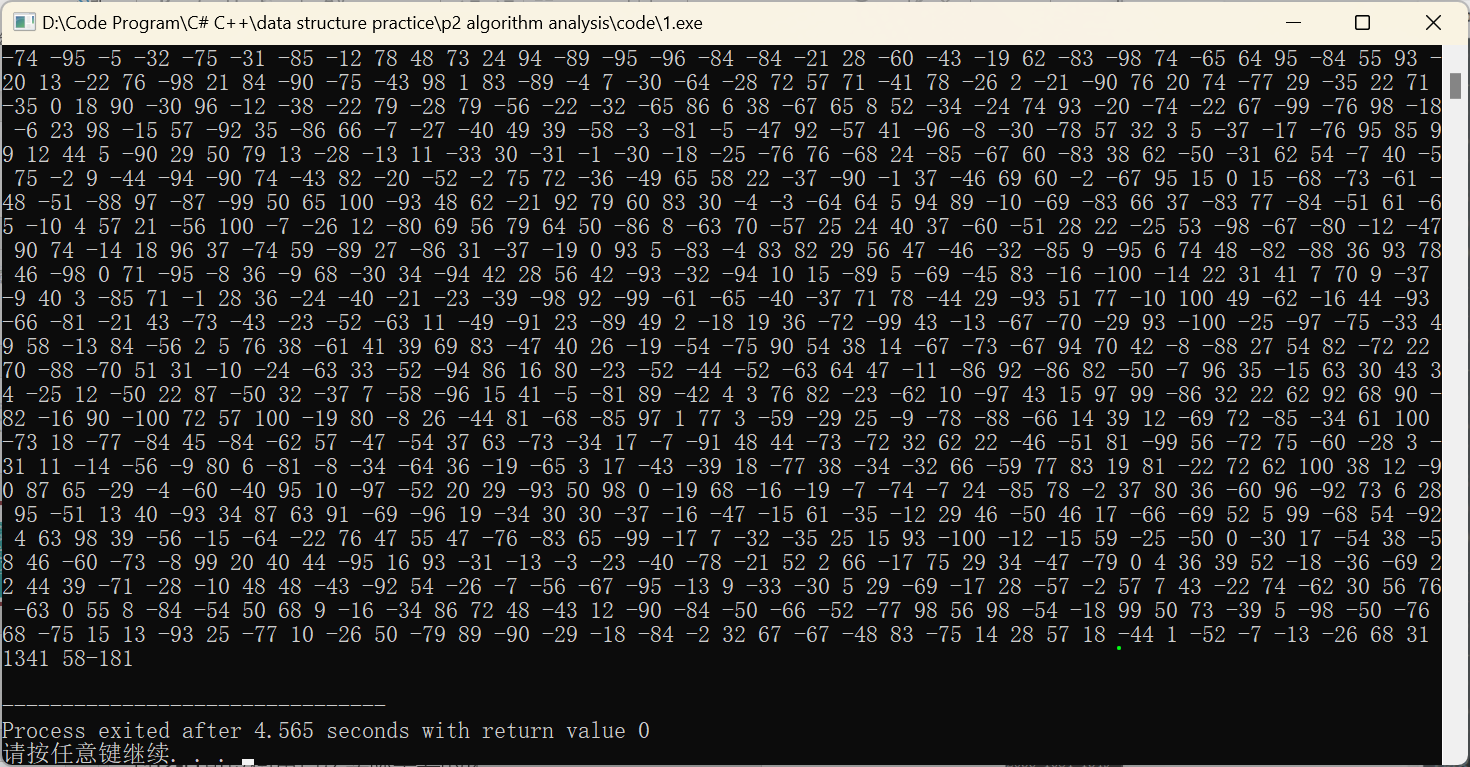
}

}

测试时，我的电脑上数据量在1000个整数之前三种算法都没有明显差距，但是到2000左右就开始有答案延迟呈现的情况了，故我选取这一规模的数据来进行算法实际运行时间的测量。

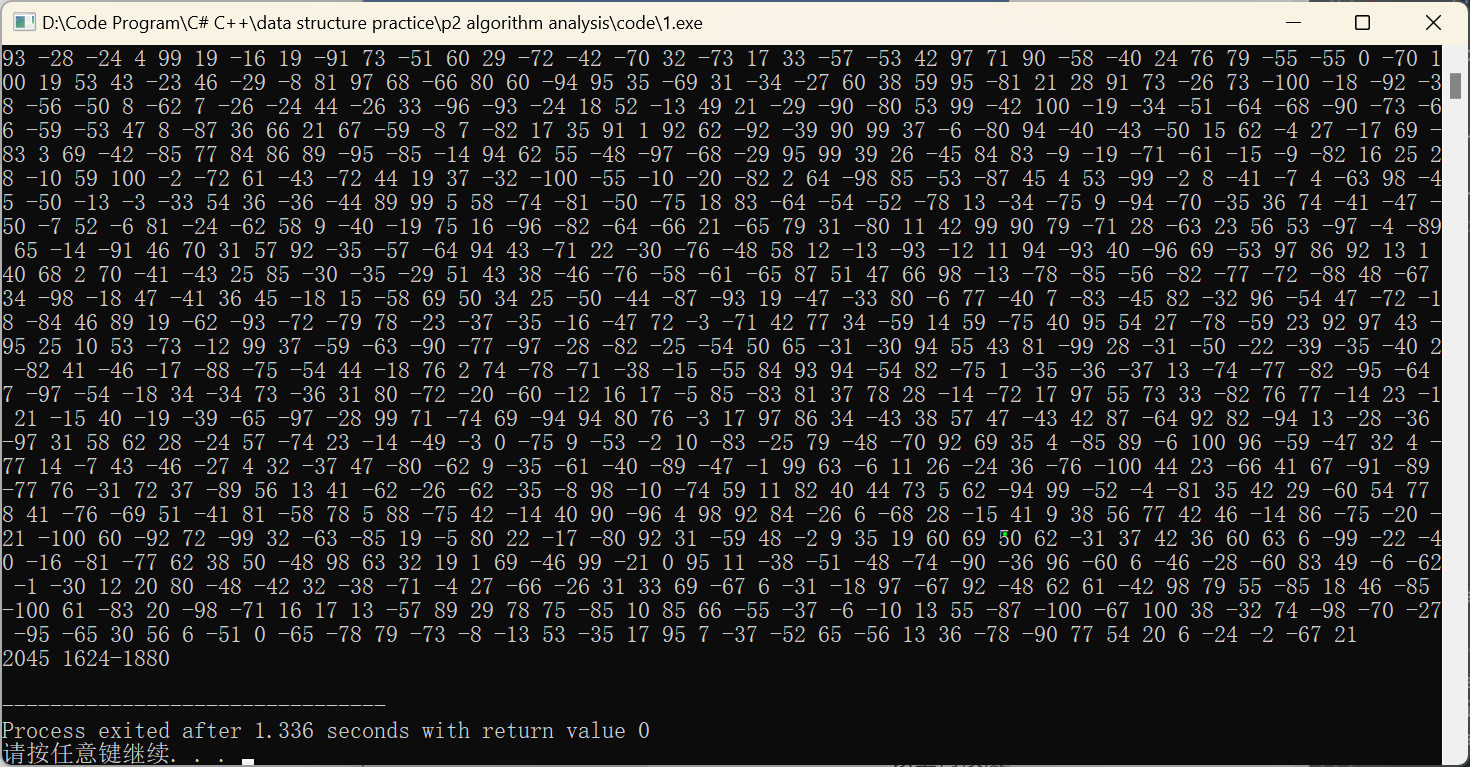
对于算法1：



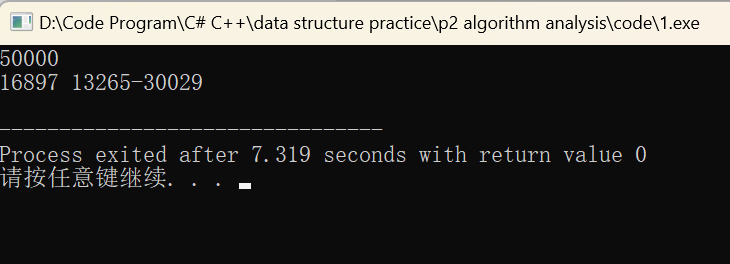


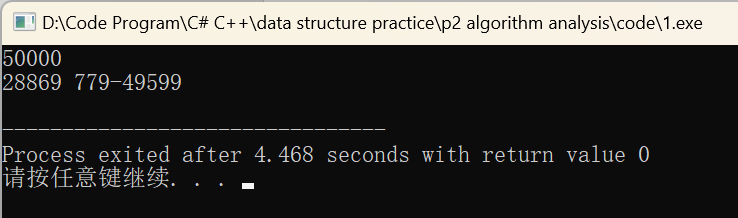
以上两段随机数文本，算法1的处理时间在3-5秒左右，我还测试了几组，基本就是这个平均时间，此处就不粘贴图片了。

对于算法2;



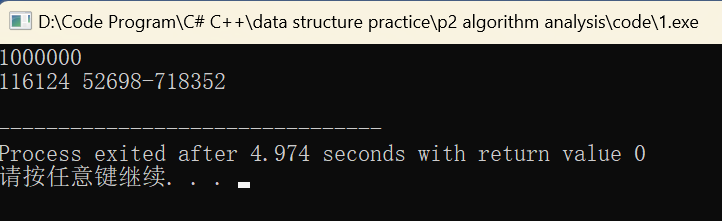
测试了几组就发现算法2处理的速度比算法1要快的多，所以我直接用100000数量规模的数据进行测试，发现太久都不出现答案。这时候我估计该算法的理想数据处理能力大概在50000数量规模左右，于是我进行了几组数据的测试。

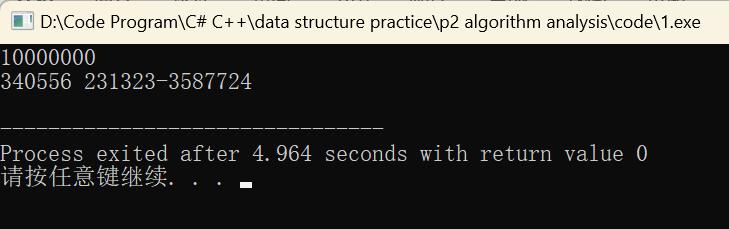


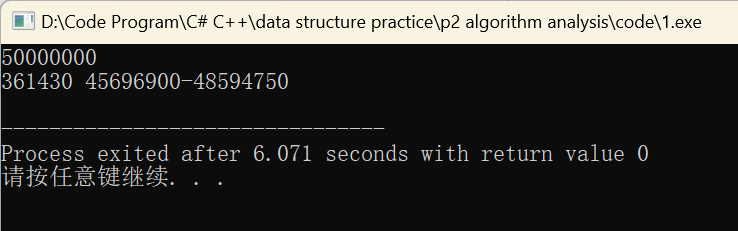


这时算法2的平均运行时间也在3-5秒左右，由于数据太多这里就把打印数据的代码部分注释掉了。

对于算法3：







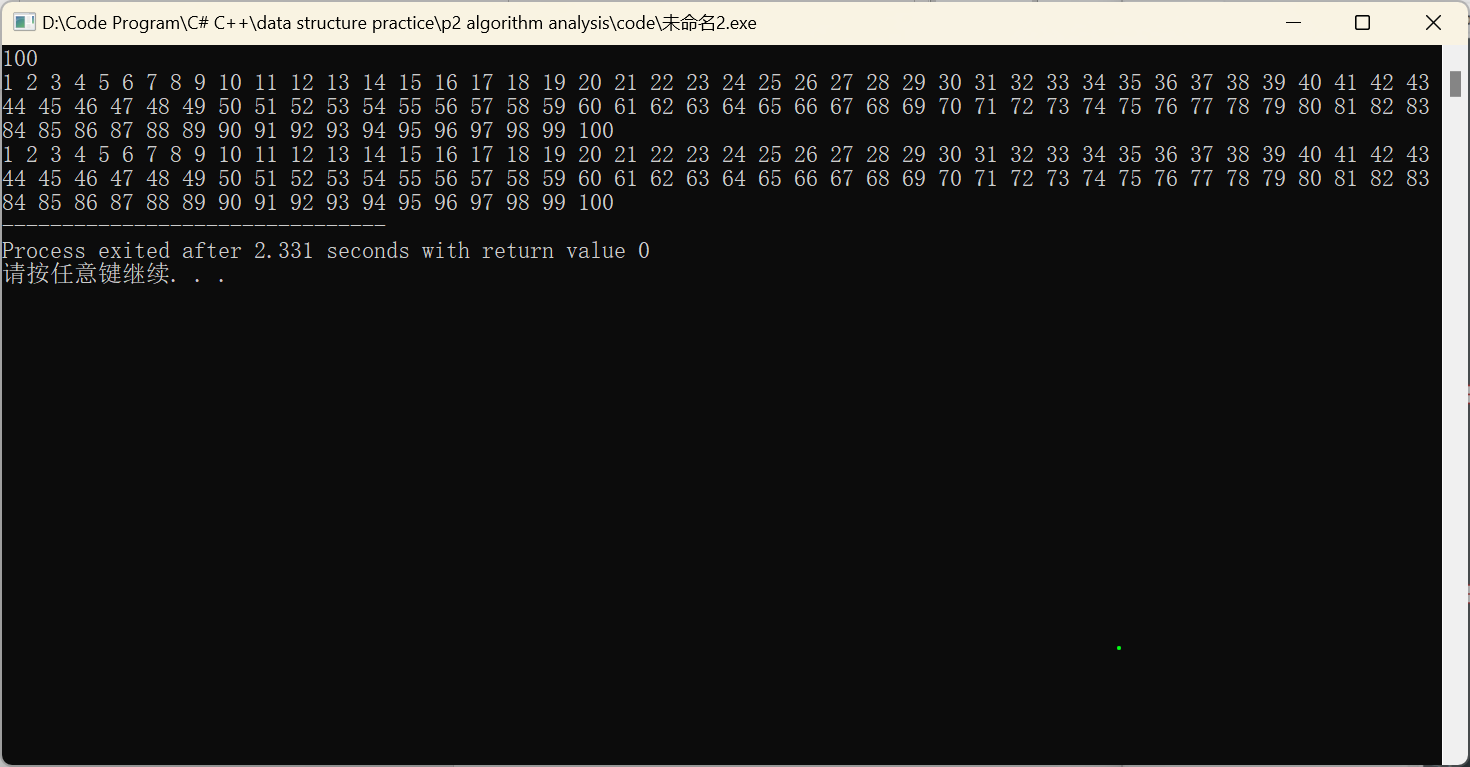
这里就不多废话了，当数据量达到50000000数据规模时，算法3才呈现出了平均3-5秒的运行时间，其算法效率显而易见。

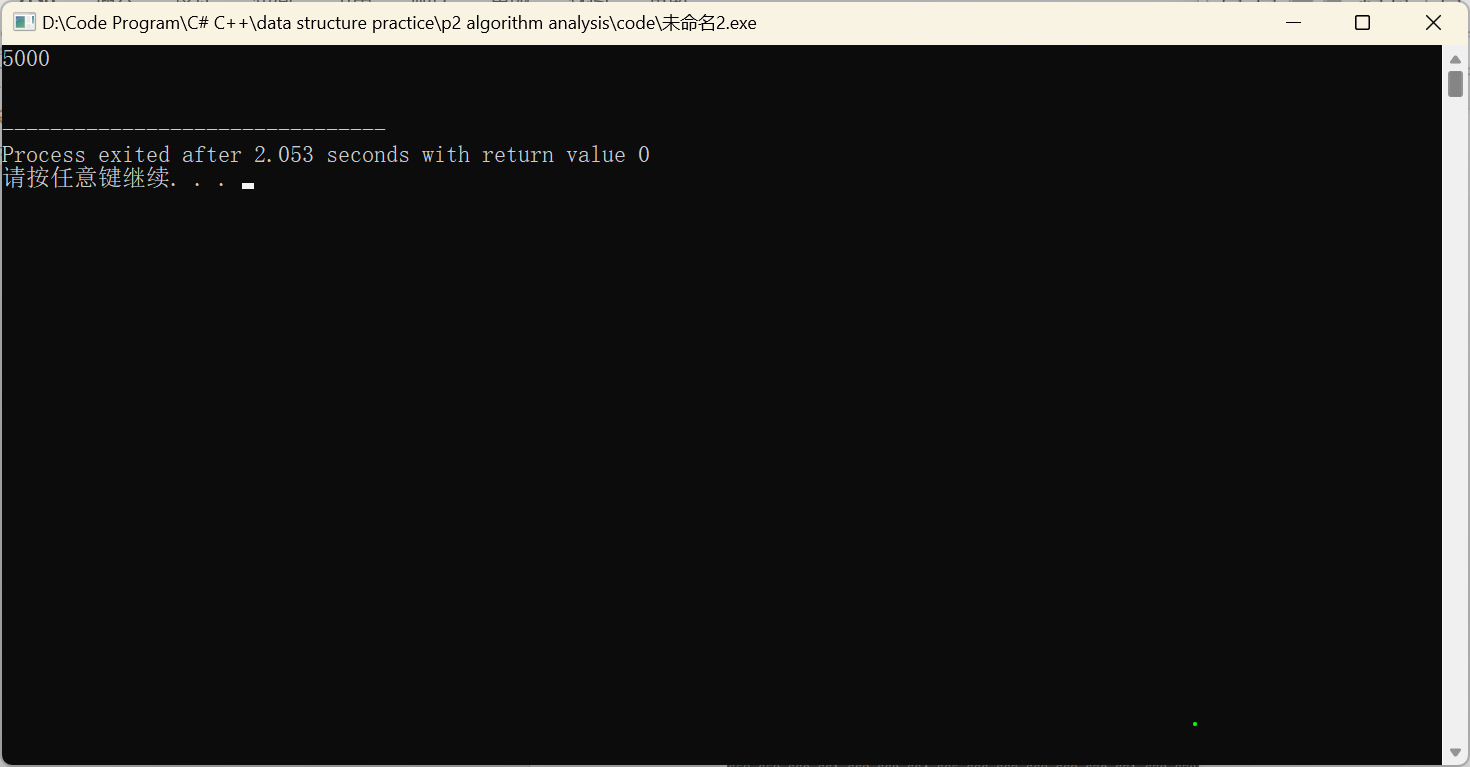
综上所述，对于平均运行时间在3-5秒的情况，算法1的数据规模截止到5000，算法2截止到50000，算法3截止到50000000，与预估的时间复杂度是一致的。

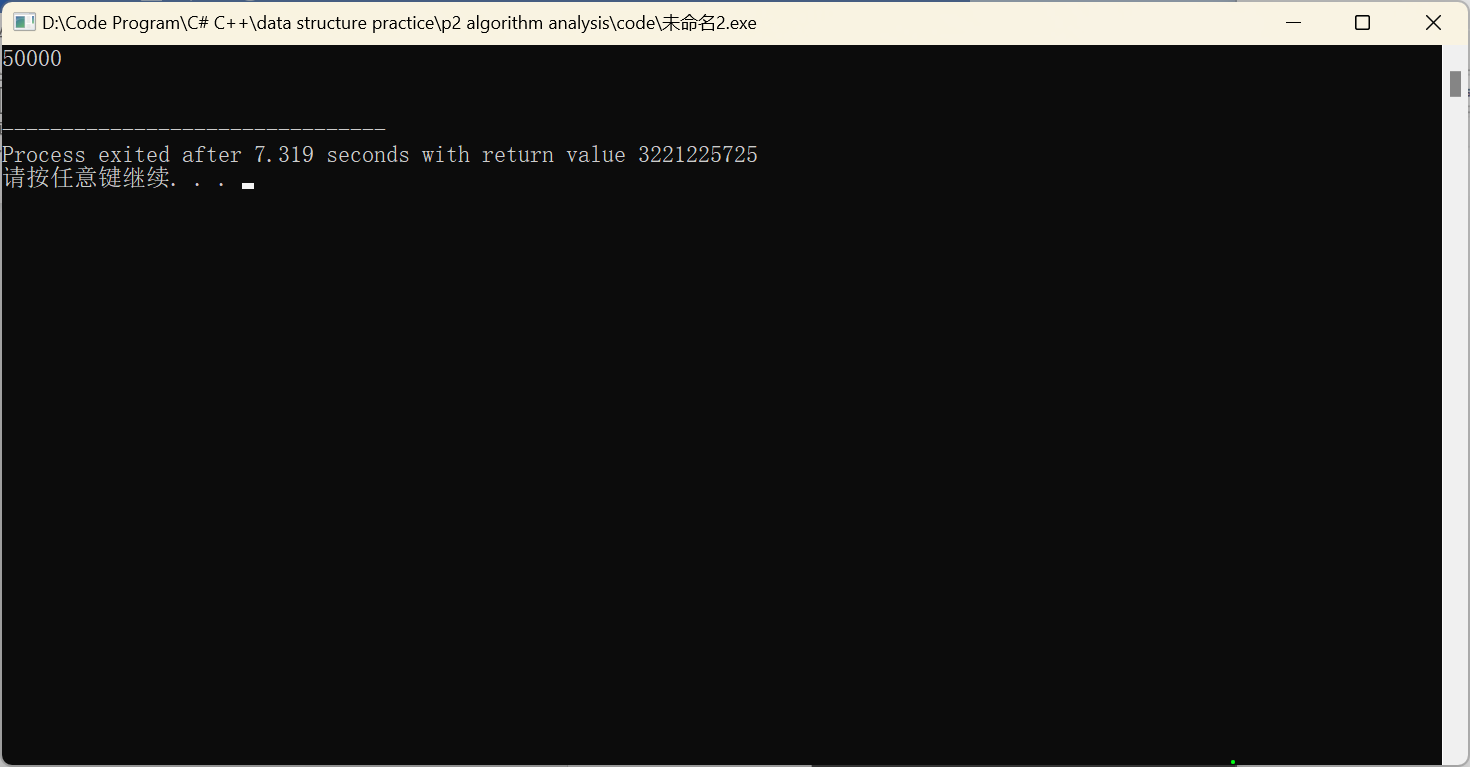
（4）最后对三个算法进行分析，显然效率排名算法3>算法2>算法1，且这种效率的差异呈现会随着数据量增长而出现愈发显著的效应。所以在处理程序问题时，一定要对所处理的数据规模有大概的估计，从而制定出适合的算法来解决问题，而不是盲目编写。

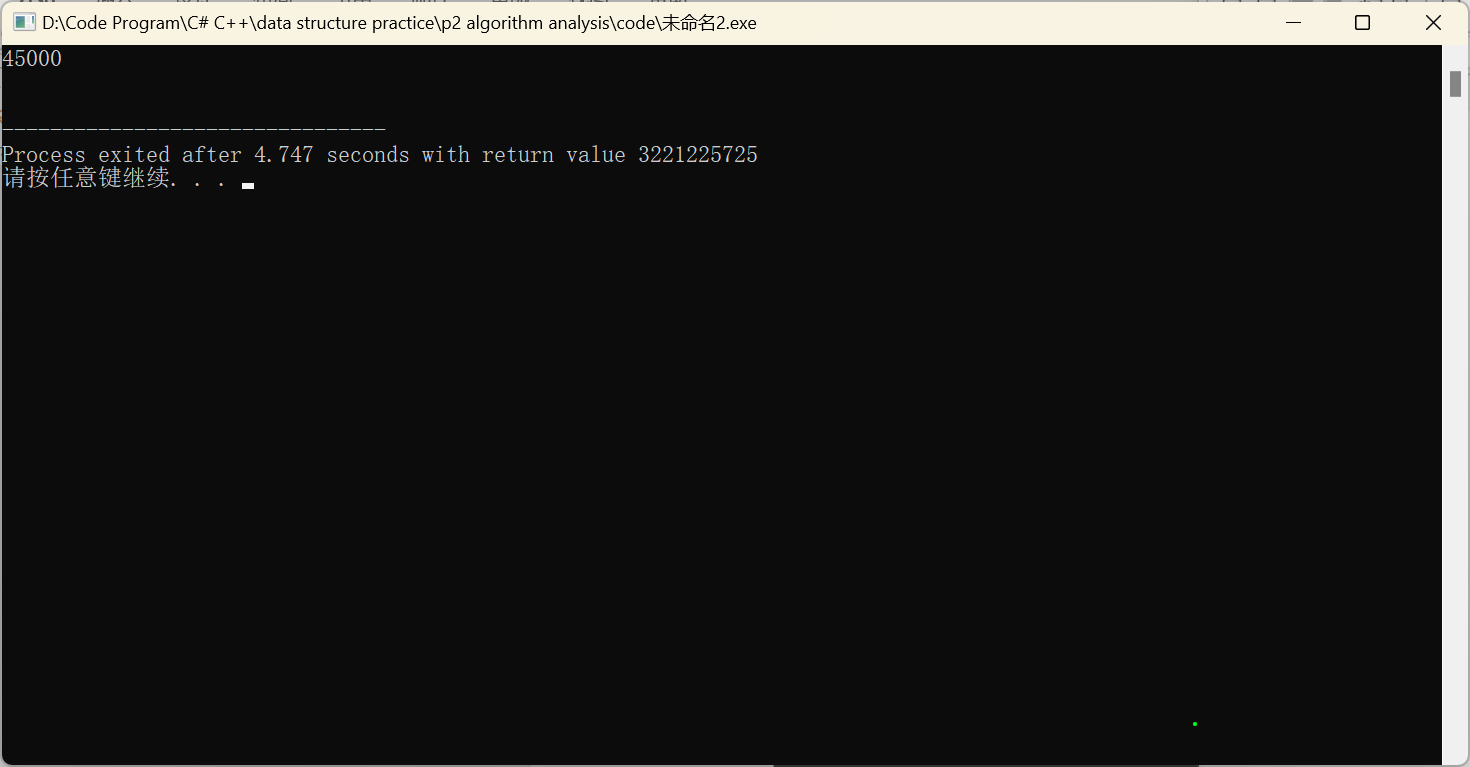
问题2：

复杂度分析









这里找出递归报错的数据量有两个小细节，一是递归的函数放在循环输出函数的后面，这样可以利用顺序执行来找到循环不报错，递归报错的数据规模；二是数据量太大的时候就直接省略打印步骤了，for循环里用一条continue语句占位即可，我的电脑运行递归函数时大概是在n=45000处报错无法再继续运行了，而循环此时正常。而对于时间复杂度的验证问题，我简单计算了程序运行的实际时间，与所估计的相差无几，可以认为是正确的。

## 附录：源代码

**1、实验环境：Dev-C++ 5.11 C++11标准**

2、

（1）问题一

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

using namespace std;

void maxSubArray1(vector<int> &nums, int n)

{

int maxsum = 0, start = -1, end = -1;

for(int i = 0; i < n; i ++)

{

for(int j = 0; j < n; j ++)

{

int tempsum = 0;

for(int k = i; k <= j; k ++)

tempsum += nums[k];

if(tempsum > maxsum)

{

maxsum = tempsum;

start = i;

end = j;

}

}

}

cout << maxsum << " ";

if(end != -1)

cout << start << "-" << end;

cout << endl;

}

void maxSubArray2(vector<int> &nums, int n)

{

int maxsum = 0, start = -1, end = -1;

for(int i = 0; i < n; i ++)

{

int tempsum = 0;

for(int j = i; j < n; j ++)

{

tempsum += nums[j];

if(tempsum > maxsum)

{

maxsum = tempsum;

start = i;

end = j;

}

}

}

cout << maxsum << " ";

if(end != -1)

cout << start << "-" << end;

cout << endl;

}

void maxSubArray3(vector<int> &nums, int n)

{

int maxsum = 0, tempsum = 0, start = 0, end = -1;

for(int j = 0; j < n; j ++)

{

tempsum += nums[j];

if(tempsum > maxsum)

{

maxsum = tempsum;

end = j;

}

else if(tempsum < 0)

{

tempsum = 0;

start = j + 1;

}

}

cout << maxsum << " ";

if(end != -1)

cout << start << "-" << end;

cout << endl;

}

void randNums(vector<int> &nums, int n)

{

srand(static\_cast<unsigned int>(std::time(0)));

for (int i = 0; i < n; ++i)

{

int randomNumber = rand() % 201 - 100;

nums.push\_back(randomNumber);

cout << randomNumber;

if (i < n - 1)

cout << " ";

else

cout << endl;

}

}

int main()

{

int n, num;

vector<int> nums;

cin >> n;

randNums(nums, n);

maxSubArray1(nums, n);

maxSubArray2(nums, n);

maxSubArray3(nums, n);

return 0;

}

（2）问题二

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

using namespace std;

void RecursivePrint(int n)

{

if(n > 1)

RecursivePrint(n - 1);

cout << n << " ";

}

void ForPrint(int n)

{

for(int i = 1; i <= n; i ++)

//continue;

cout << i << " ";

}

int main()

{

int n;

cin >> n;

ForPrint(n);

cout << endl;

RecursivePrint(n);

return 0;

}