数据结构及算法实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| **学院** | **理学院** | **专业班级** | **数据科学与大数据技术** |
| **姓名** | **孟廷轩** | **学号** | **202211107119** |
| **实验时间** | **2023.9.24** | **得分** |  |

**实验一 Fibonacci数列三种求和方式及常见的排序算法**

1. **实验目的**
2. 了解Fibonacci数列
3. 掌握Fibonacci数列求和的算法
4. 熟悉通过Matplotlib画数据图
5. 掌握常见的排序算法（冒泡排序，插入排序，选择排序必须掌握）
6. **实验内容**
7. 实现Fibonacci数列求和的算法
8. 测试不同算法对Fibonacci数列求和的影响
9. 用点阵图与折线图分别展示Fibonacci数列求和项数与时间的关系
10. 实现常见的排序算法（C语言实现）
11. **实验过程及结果（算法、流程图、含注释的源代码、测试和运行结果分析等）**

**3.1用递归实现Fibonacci数列求和**

递归是一种通过调用自身来解决问题的方法。在斐波那契数列求和的递归算法中，我们可以这样实现：1）定义递归函数：首先，我们定义一个递归函数，例如 Fib(n)，用于计算斐波那契数列的前n项和；2）定义基本情况：接下来，我们定义递归函数的基本情况。当n等于1或2时，斐波那契数列的和为1;3）定义递归关系：然后，我们定义递归函数的递归关系。斐波那契数列的第n项是前两项的和，因此斐波那契数列的第n项和可以表示为第n-1项和加上第n-2项，即 Fib(n) = Fib(n-1)+Fib(n-2)；4）递归调用：在递归关系中，我们可以看到递归函数Fib(n) 不停地调用，直到n = 1或2时停止。5）递归结束条件：为了避免无限递归，我们需要定义递归函数的结束条件。在这个例子中，结束条件是当n等于1或2时，直接返回1。通过以上几步我们可以实现递归求和，代码与运行结果如下图一:



图一

**3.2用迭代实现Fibonacci数列求和**

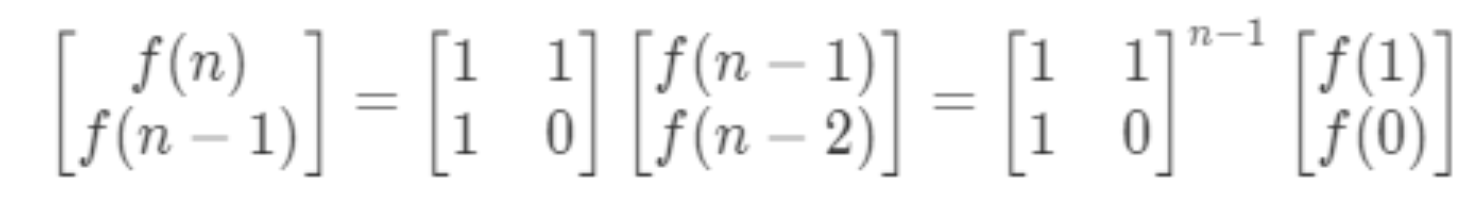
使用迭代方式实现斐波那契数列的求和，可以通过以下步骤实现：1）定义函数：首先，定义了一个名为Fib2的函数，用于计算斐波那契数列的前n项和；2）初始化变量：在函数内部，定义了变量a、b、c和sum，分别表示斐波那契数列的前两项、当前项和求和结果。将a和b初始化为1，c和sum初始化为0；3）处理特殊情况：如果n等于1，直接返回1；如果n等于2，直接返回2；4）迭代求和：使用while循环来迭代计算斐波那契数列的前n项和。从第3项开始，每一次循环都将a和b相加得到c，然后更新a为原来的b，b为原来的c，并将c累加到sum中。同时，将n减1，直到n减少到2为止；5）返回结果：循环结束后，返回sum，即斐波那契数列的前n项和。通过以上几步我们可以实现迭代求和，代码与运行结果如下图二:



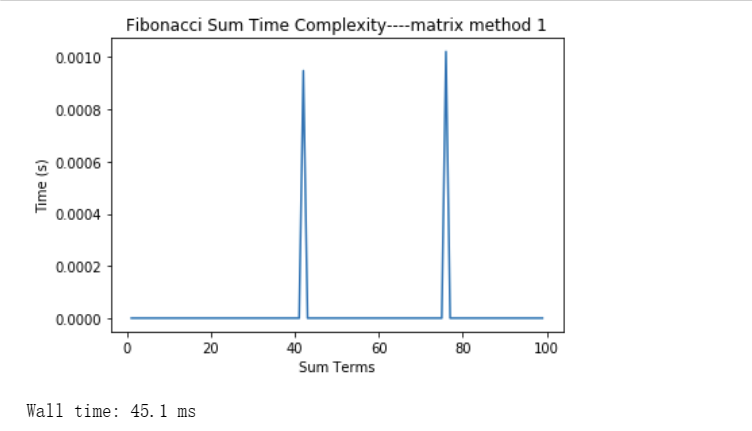
图二

**3.3用矩阵快速幂实现Fibonacci数列求和**

1. 矩阵快速幂是一种高效计算斐波那契数列的方法，通过矩阵的乘法运算，可以快速求解斐波那契数列的第n项。斐波那契数列的递推公式为：F(n) = F(n-1) + F(n-2)，其中F(1)=1，F(2)=1。我们可以将斐波那契数列的递推公式表示为矩阵形式：



根据上述公式，我们可以通过矩阵快速幂的方法来计算斐波那契数列的第n项。具体步骤如下：1）定义矩阵A：定义一个2x2的矩阵A，其中A[0][0]=1，A[0][1]=1，A[1][0]=1，A[1][1]=0。2）定义单位矩阵：定义一个2x2的单位矩阵I，其中I[0][0]=1，I[0][1]=0，I[1][0]=0，I[1][1]=1。3）定义矩阵乘法函数：实现一个矩阵乘法函数，用于计算两个2x2矩阵的乘法。4）定义矩阵快速幂函数：实现一个矩阵快速幂函数，用于计算矩阵A的乘方。5）计算斐波那契数列的第n项：通过调用矩阵快速幂函数，计算矩阵A的乘方，然后将结果与单位矩阵I相乘，得到一个新的2x2矩阵B。B[0][0]即为斐波那契数列的第n项。6）返回结果：返回斐波那契数列的第n项。通过矩阵快速幂的方法，我们可以在O(logn)的时间复杂度内计算斐波那契数列的第n项，大大提高了计算效率。代码与运行结果如下图三、四：

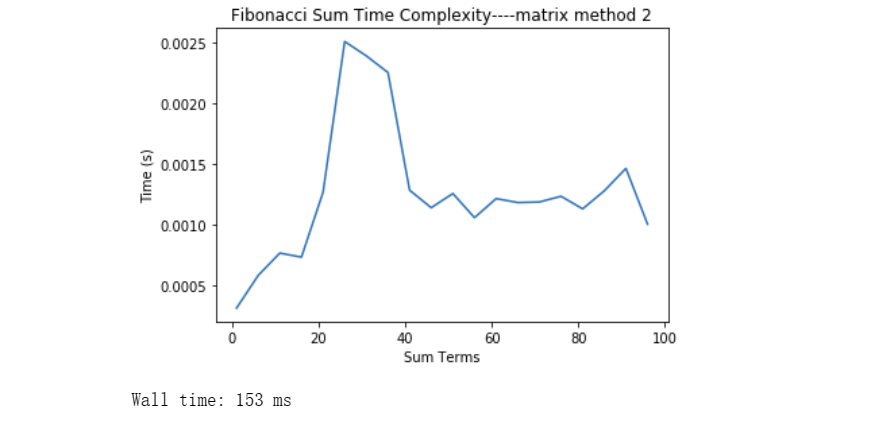


图三



**图四**

1. 利用numpy库进行简化，原理同上，只不过从numpy库中直接调用相应矩阵计算方法，简化一些代码，但运行时间变长了。代码与运行结果如下图五、六：

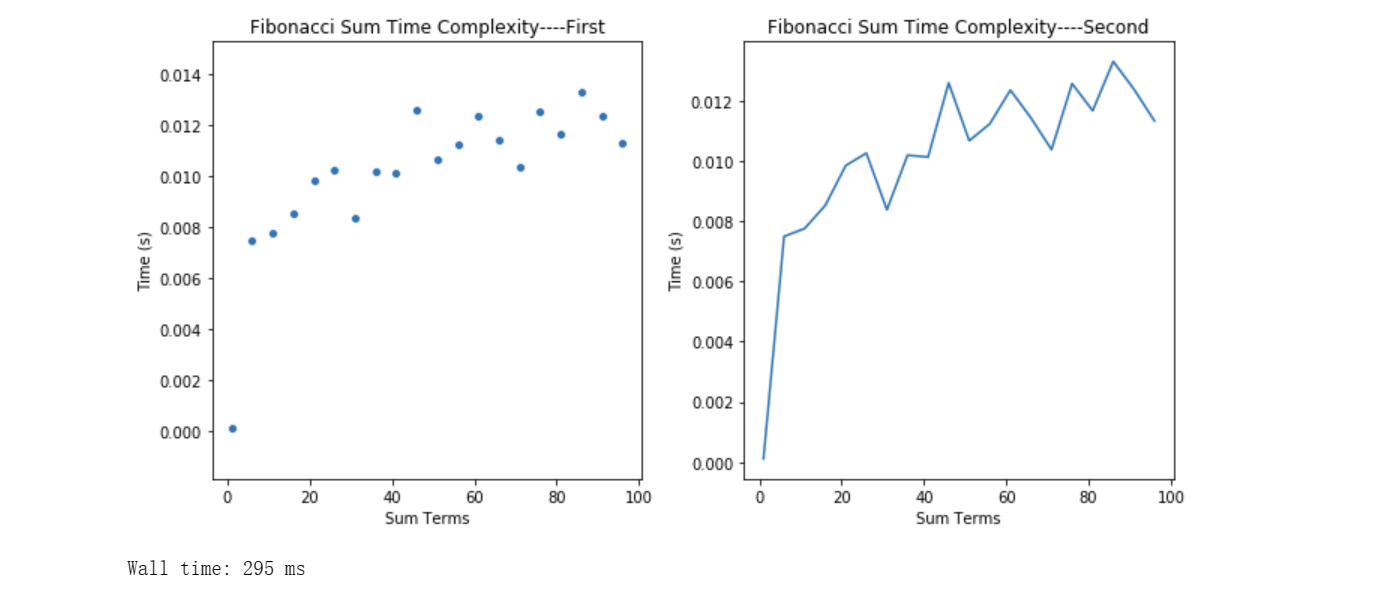


**图五**



**图六**

**3.4用点阵、折线图分别展示Fibonacci数列求和项数与时间的关系**



**图七**

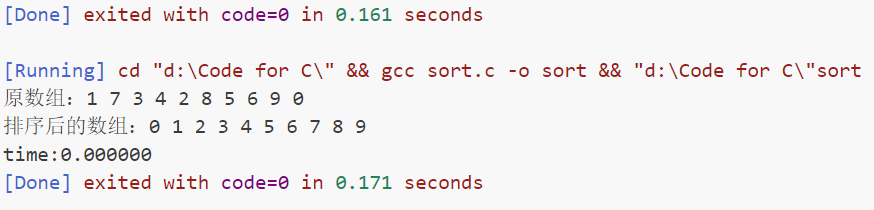


**图八**

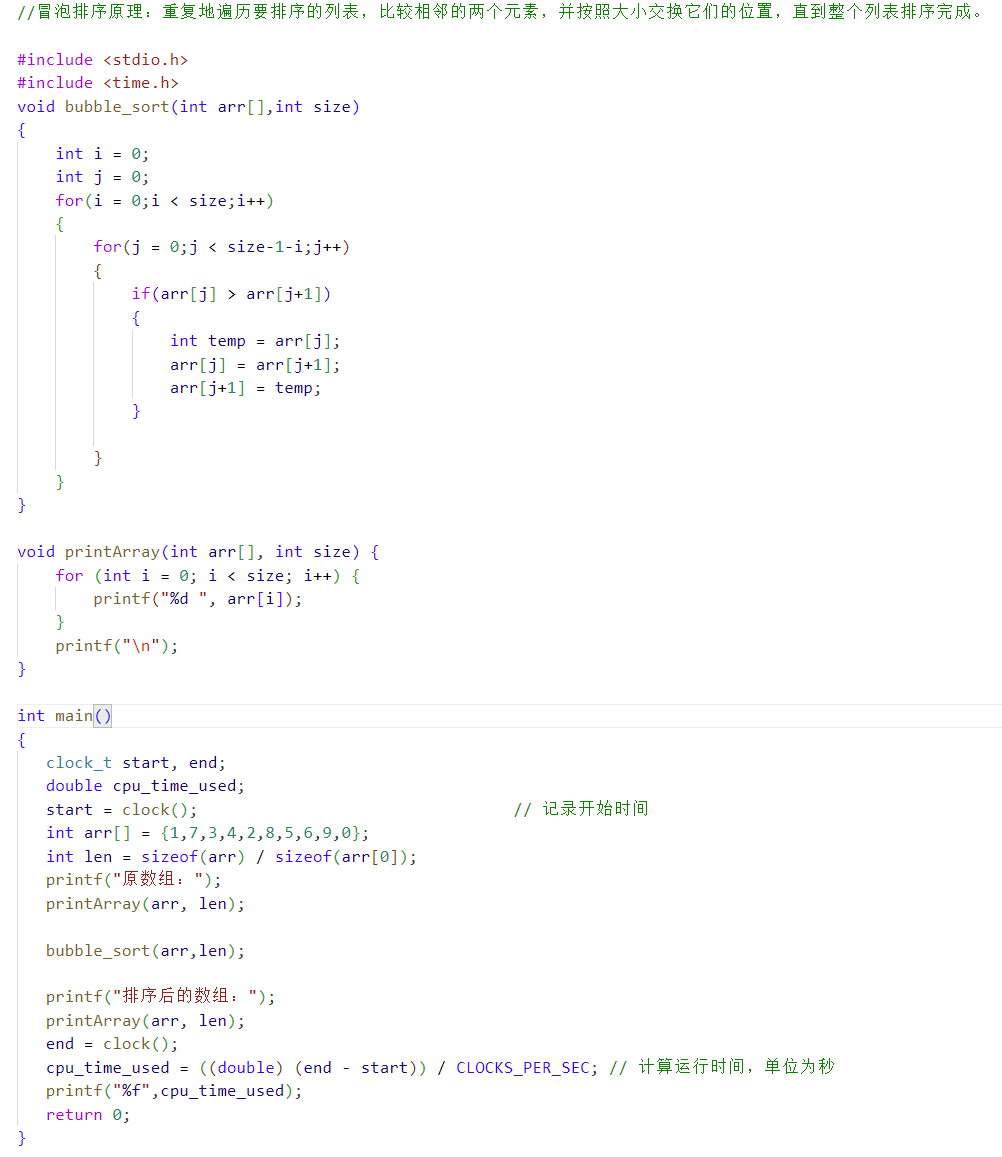
**3.5实现常见的排序算法**

常见的排序算法有以下几种：冒泡排序，插入排序，选择排序，快速排序，堆排序，希尔排序和归并排序。本实验展示其中的三种--冒泡排序，插入排序与选择排序。

1. 冒泡排序：冒泡排序是一种简单的排序算法，它重复地遍历待排序的元素，比较相邻的两个元素，并按照一定的规则交换它们的位置，直到整个序列有序为止。具体步骤如下：1）从待排序序列的第一个元素开始，依次比较相邻的两个元素。如果前一个元素大于后一个元素，则交换它们的位置，使较大的元素向后移动。2）继续遍历序列，重复上述比较和交换的过程，直到没有相邻的元素需要比较为止。一轮遍历结束后，最大的元素会被移动到序列的最后一个位置。3）重复上述步骤，每次遍历序列的元素减少一个，直到所有元素都有序排列。冒泡排序的时间复杂度为O(n^2)，n是待排序序列的长度。在最坏情况下，即待排序序列是逆序的情况下，冒泡排序需要进行n-1轮比较和交换操作。代码实现与运行结果如下图九、十：

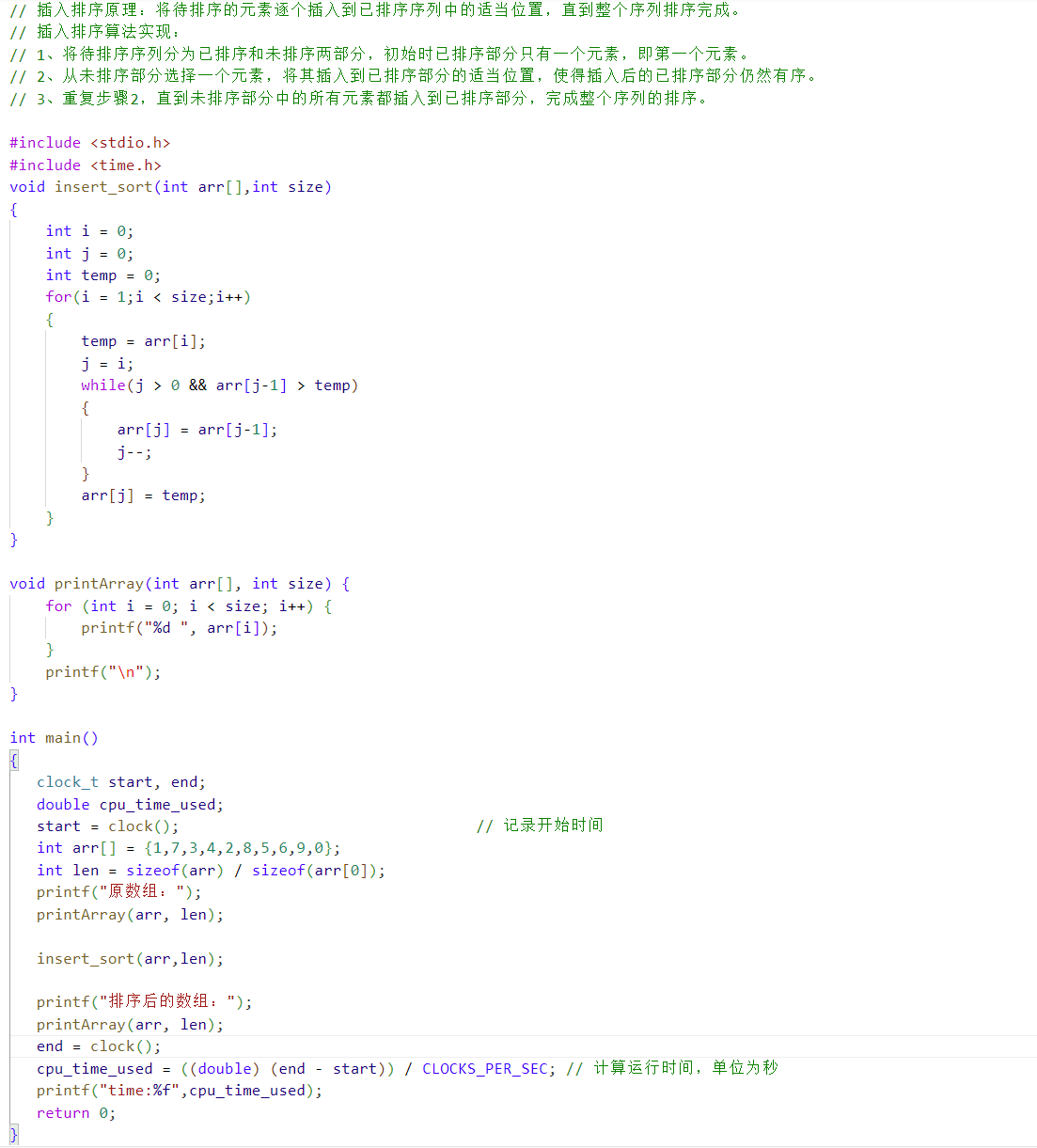


图九

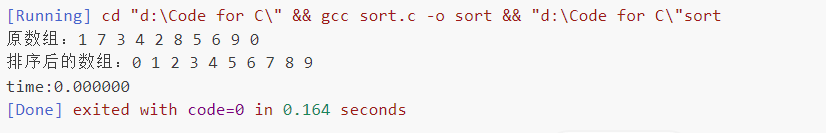


图十

1. 插入排序：插入排序是一种简单直观的排序算法，它将待排序的序列分为已排序和未排序两部分，每次从未排序部分取出一个元素，插入到已排序部分的合适位置，使得已排序部分仍然有序。具体步骤如下：1）假设待排序序列为arr，长度为n。从第二个元素开始，将当前元素插入到已排序部分的合适位置。假设已排序部分的最后一个元素位置为i，初始时i=0。2）将当前元素与已排序部分的元素从后往前依次比较，如果当前元素小于已排序部分的某个元素，则将该元素后移一位。3）重复步骤3，直到找到当前元素的合适位置，将其插入。4）重复步骤2和步骤3，直到所有元素都插入到已排序部分。插入排序的时间复杂度为O(n^2)，其中n是待排序序列的长度。在最坏情况下，即待排序序列是逆序的情况下，插入排序需要进行n\*(n-1)/2次比较和移动操作。然而，在最好情况下，即待排序序列已经有序的情况下，插入排序的时间复杂度为O(n)，因为只需比较n-1次即可确定每个元素的位置。插入排序是稳定的排序算法，适用于小规模数据或基本有序的序列。代码实现与运行结果如下图十一、十二所示：



图十一

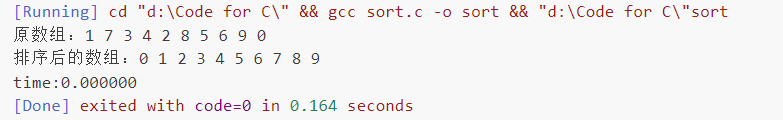


图十二

1. 选择排序：选择排序是一种简单直观的排序算法，它将待排序序列分为已排序和未排序两部分，每次从未排序部分选择最小（或最大）的元素，放到已排序部分的末尾。具体步骤如下：1）假设待排序序列为arr，长度为n。从第一个元素开始，将其视为已排序部分。在未排序部分选择最小（或最大）的元素，记其下标为minIndex。2）将最小（或最大）的元素与已排序部分的最后一个元素交换位置。3）将已排序部分的长度增加1，未排序部分的长度减少1。4）重复步骤3至步骤5，直到所有元素都被放到已排序部分。选择排序的时间复杂度为O(n^2)，其中n是待排序序列的长度。无论待排序序列的初始状态如何，选择排序总共需要进行n-1轮比较和交换操作。选择排序是不稳定的排序算法，因为交换操作可能改变相同元素的相对顺序。选择排序适用于小规模数据，但对于大规模数据效率较低。代码实现与运行结果如下图十三、十四：



图十三



图十四