班级：计算机科学与技术21留学生1 学号：2021529620004姓名：LOW REN HONG

实验一 递归与分治

**一、实验目的**

1、理解分治算法的概念和基本要素；

2、理解递归的概念；

3、掌握设计有效算法的分治策略。

**二、实验内容和要求**

**实验要求**：通过上机实验进行算法实现，保存和打印出程序的运行结果，并结合程序进行分析，上交实验报告。

**实验内容**：

**1、归并排序问题**

对*n*个元素组成的序列进行排序。

将待排序元素分成大小大致相同的两个子集合，分别对两个集合进行排序，最终将排序好的子集合合并成所要求的排好序的集合。

**2、极值问题**

已知m、n为整数，且满足下列两个条件：

① m、n∈{１，２，…，k}，即1≤m，n≤k

②（n2－m\*n－m2）2＝1

你的任务是：编程输入正整数k（1≤k≤1e9），求一组满足上述两个条件的m、n，并且使m2＋n2的值最大。例如，从键盘输入k=1995，则输出：m=987    n=1597。

1e9 = 1000000000

输入

输入k

输出

按样例输出m和n

样例输入

1995

样例输出

m=987

n=1597

**3、黑白棋子的移动**

有2n个棋子（n≥4）排成一行，开始位置为白子全部在左边，黑子全部在右边，如下图为n=5的情形：

○○○○○●●●●●

移动棋子的规则是：每次必须同时移动相邻的两个棋子，颜色不限，可以左移也可以右移到空位上去，但不能调换两个棋子的左右位置。每次移动必须跳过若干个棋子（不能平移），要求最后能移成黑白相间的一行棋子。如n=5时，成为：

○●○●○●○●○●

任务：编程打印出移动过程。

输入

输入n。

输出

移动过程

样例输入

7

样例输出

step 0:ooooooo\*\*\*\*\*\*\*--

step 1:oooooo--\*\*\*\*\*\*o\*

step 2:oooooo\*\*\*\*\*\*--o\*

step 3:ooooo--\*\*\*\*\*o\*o\*

step 4:ooooo\*\*\*\*\*--o\*o\*

step 5:oooo--\*\*\*\*o\*o\*o\*

step 6:oooo\*\*\*\*--o\*o\*o\*

step 7:ooo--\*\*\*o\*o\*o\*o\*

step 8:ooo\*o\*\*--\*o\*o\*o\*

step 9:o--\*o\*\*oo\*o\*o\*o\*

step10:o\*o\*o\*--o\*o\*o\*o\*

step11:--o\*o\*o\*o\*o\*o\*o\*

提示

注意格式

**三、程序代码**

**(1)**

# 归并排序函数

def merge\_sort(arr):

# 如果数组长度小于等于1，直接返回数组

if len(arr) <= 1:

return arr

# 否则，将数组分为左右两部分，并对每部分进行递归调用归并排序函数

mid = len(arr) // 2

left = arr[:mid]

right = arr[mid:]

# 最后将左右两部分合并排序，调用merge函数实现

left = merge\_sort(left)

right = merge\_sort(right)

return merge(left, right)

# 合并函数

def merge(left, right):

# 初始化一个空列表 merged

merged = []

# 使用两个索引 left\_index 和 right\_index 分别追踪左右两部分的位置

left\_index = 0

right\_index = 0

# 当左右两部分都有元素时，比较当前元素大小，将较小的元素添加到 merged 中，同时更新索引

while left\_index < len(left) and right\_index < len(right):

if left[left\_index] <= right[right\_index]:

merged.append(left[left\_index])

left\_index += 1

else:

merged.append(right[right\_index])

right\_index += 1

# 当其中一部分已经遍历完时，将剩余部分的元素依次添加到 merged 中

while left\_index < len(left):

merged.append(left[left\_index])

left\_index += 1

while right\_index < len(right):

merged.append(right[right\_index])

right\_index += 1

# 返回合并后的有序列表 merged

return merged

# 提示用户输入待排序的元素列表

input\_str = input("请输入待排序的元素列表，以空格分隔: ")

arr = list(map(int, input\_str.split()))

# 调用归并排序

sorted\_arr = merge\_sort(arr)

# 显示排序后的结果

print("排序后的结果:", sorted\_arr)

**(2)**

def find\_maximum\_values(k):

# 初始化m、n和max\_sum的初始值

m = 1

n = 1

max\_sum = 2

# 遍历i和j的取值范围

for i in range(1, k+1):

for j in range(1, k+1):

# 判断条件：(j^2 - i\*j - i^2)^2 = 1 并且 i^2 + j^2 > max\_sum

if pow((pow(j,2) - i\*j - pow(i,2)),2) == 1 and pow(i,2) + pow(j,2) > max\_sum:

# 更新m、n和max\_sum的值

m = i

n = j

max\_sum = pow(i,2) + pow(j,2)

# 返回满足条件的m和n

return m, n

# 提示用户输入k

k = int(input("请输入k的值："))

# 调用函数查找满足条件的m和n，并使m^2 + n^2的值最大

m, n = find\_maximum\_values(k)

# 显示结果

print("m =", m)

print("n =", n)

**(3)**

#定义了print\_chessboard函数，用于打印棋盘的当前状态。

# 通过格式化字符串打印出step和棋盘的字符串表示。

def print\_chessboard(a, step):

print('step{:2d}:{}'.format(step, ''.join(a)))

#定义了move函数，用于移动棋子的位置。

def move(a, k, sp, step):

#通过交换元素的方式将棋子从位置k移动到sp+j的位置，同时将原来的位置置为'-'。

for j in range(2):

a[sp+j] = a[k+j-1]

a[k+j-1] = '-'

#调用print\_chessboard函数打印移动后的棋盘状态。

print\_chessboard(a, step)

#更新sp的值为k-1，并返回更新后的sp。

sp = k - 1

return sp

#定义了mv函数，用于递归移动棋子。

def mv(a, n, sp, step):

#当n为4时，按照规定的移动顺序依次调用move函数移动棋子，并更新sp的值。

if n == 4:

sp = move(a, 4, sp, step)

sp = move(a, 8, sp, step)

sp = move(a, 2, sp, step)

sp = move(a, 7, sp, step)

sp = move(a, 1, sp, step)

#当n不为4时，先调用move函数移动棋子，然后递归调用mv函数，将n减1，并更新step的值。

else:

sp = move(a, n, sp, step)

sp = move(a, n\*2-1, sp, step)

sp = mv(a, n-1, sp, step+1)

#返回更新后的sp。

return sp

n = int(input("Please insert a number:"))

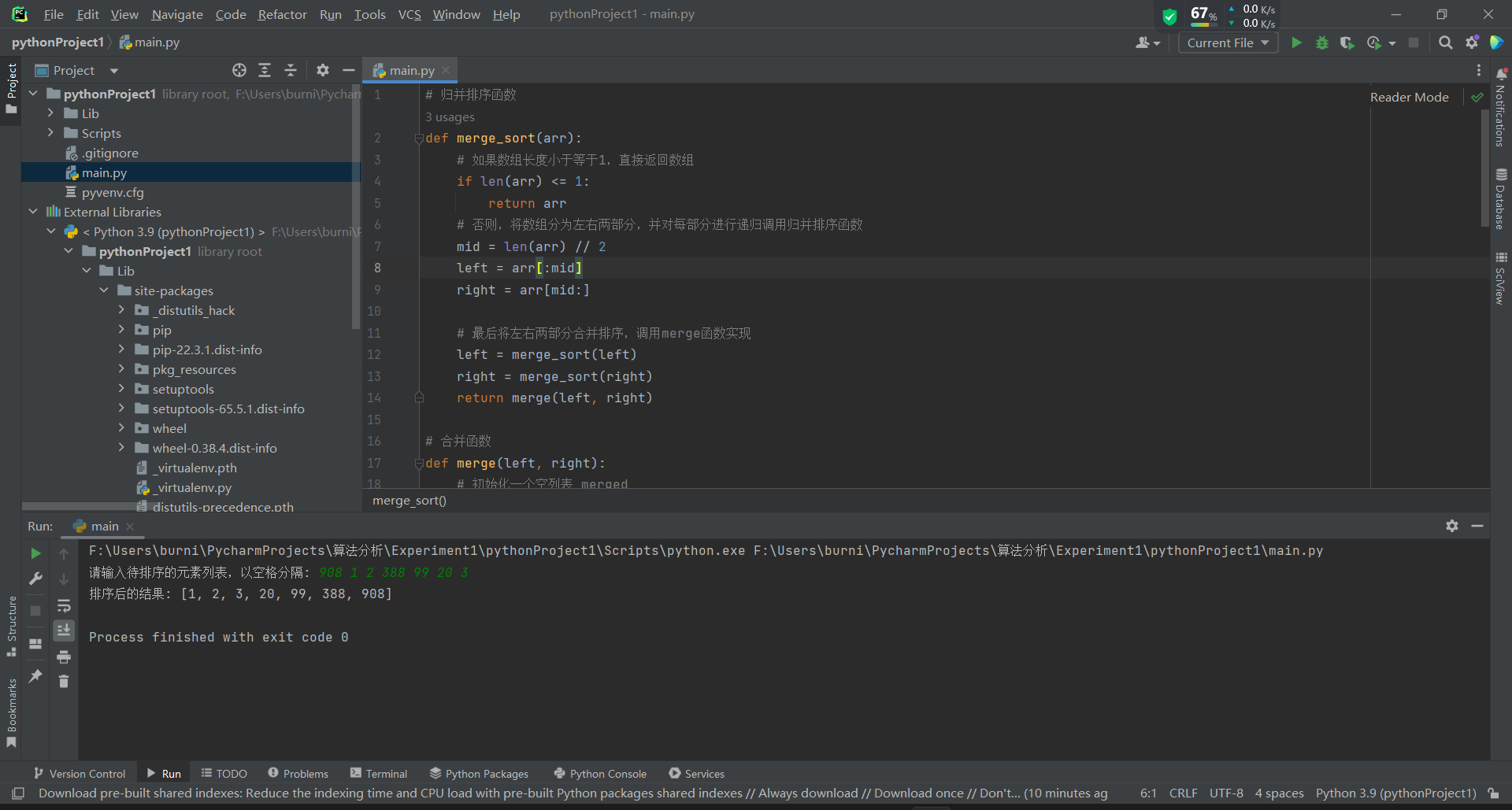
a = ['o' for \_ in range(n)] + ['\*' for \_ in range(n)] + ['-', '-']

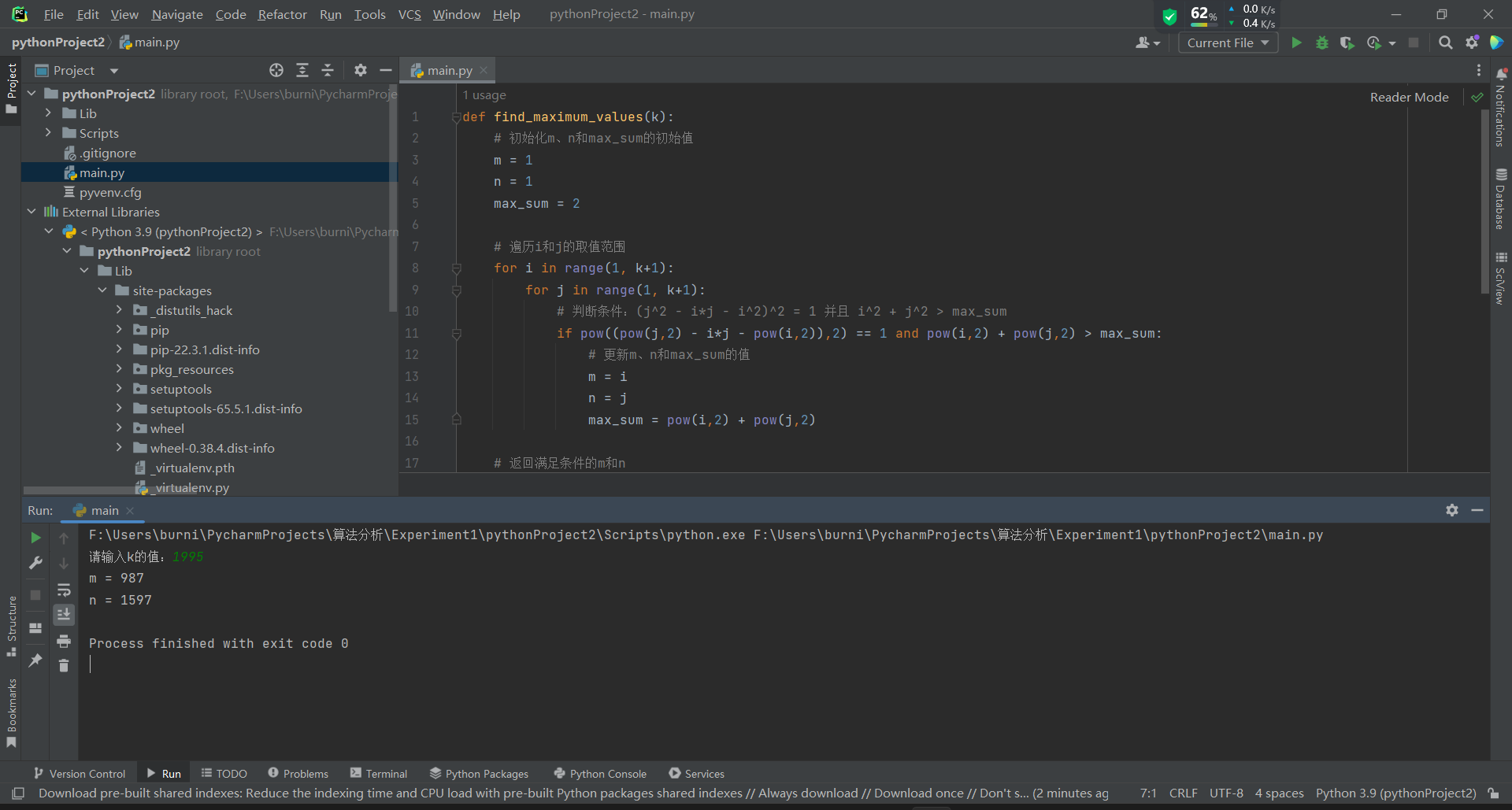
sp = 2 \* n

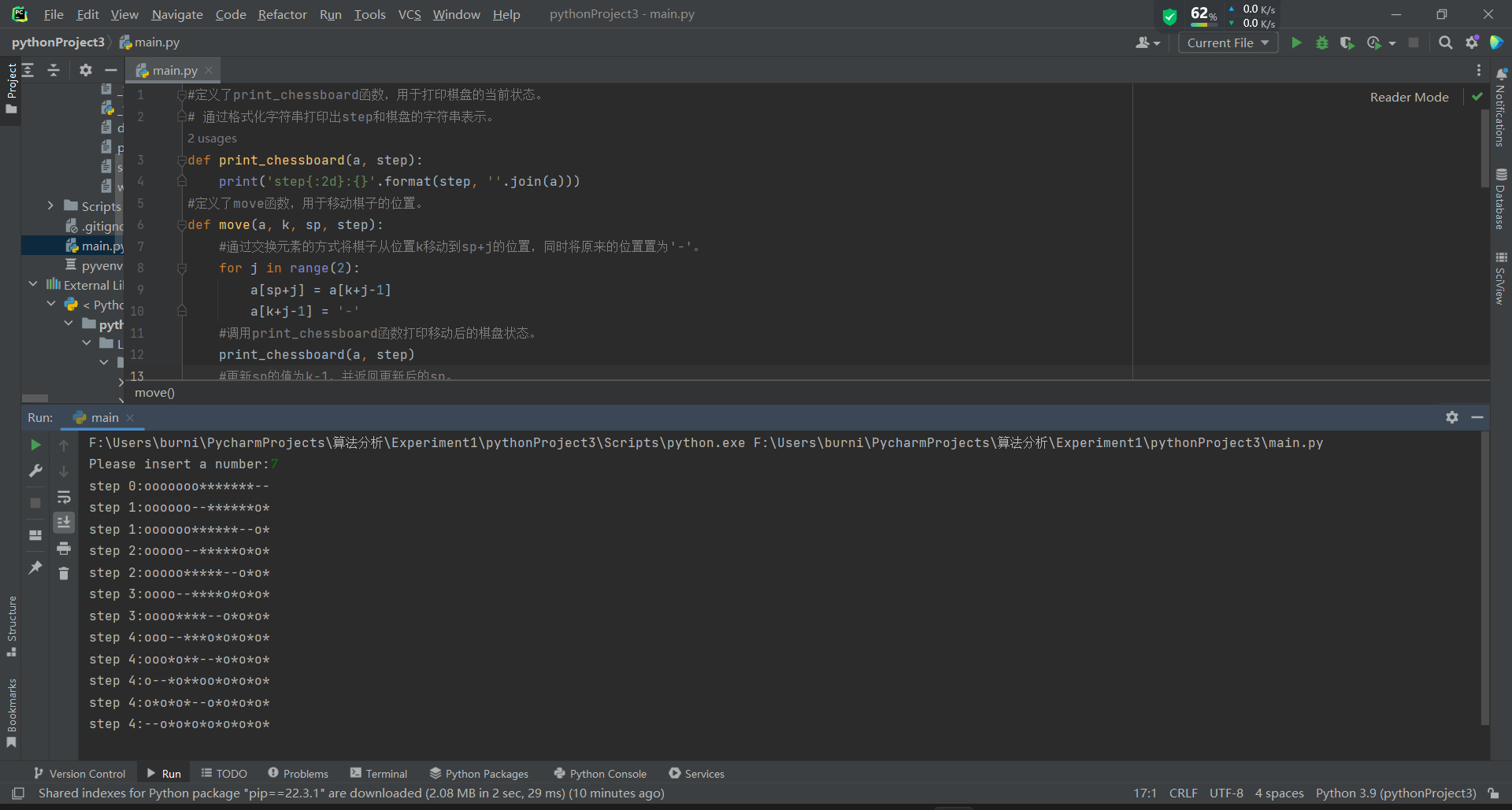
print\_chessboard(a, 0)

sp = mv(a, n, sp, 1)

**四、结果运行与分析**

定义了merge\_sort函数，用于对输入的数组进行归并排序。对于长度小于等于1的数组，直接返回。对于长度大于1的数组，将其分为左右两部分并递归调用merge\_sort函数对其进行排序。然后调用merge函数将排序后的左右两部分进行合并排序。在merge函数中，初始化一个空列表merged，使用索引left\_index和right\_index追踪左右两部分的位置。比较左右两部分的元素大小，将较小的元素添加到merged中，并更新相应的索引。将剩余部分的元素依次添加到merged中。返回合并后的有序列表merged。提示用户输入待排序的元素列表，将输入的字符串转换为整数列表。调用merge\_sort函数对输入的列表进行归并排序，并将排序后的结果赋值给sorted\_arr。输出排序后的结果。

定义了find\_maximum\_values函数，用于查找满足条件的m和n，并使m^2 + n^2的值最大。初始化m、n和max\_sum的初始值为1和2。通过两个嵌套循环遍历i和j的取值范围，并根据条件筛选满足条件的m和n，同时更新max\_sum的值。循环结束后，返回满足条件的m和n。提示用户输入k的值，并将其转换为整数类型。调用find\_maximum\_values函数，查找满足条件的m和n，并使m^2 + n^2的值最大。将结果赋值给变量m和n。输出结果m和n。



定义函数print\_chessboard(a, step)用于打印棋盘状态。函数move(a, k, sp, step)用于移动棋子位置。函数mv(a, n, sp, step)用于递归移动棋子。提示用户输入数字n。创建棋盘状态列表a，初始化空位位置sp。打印初始棋盘状态。调用mv(a, n, sp, 1)递归移动棋子。简化完成。

**五、心得与体会**

本次的实验，我学会了分治算法和递归算法的概念，并掌握了设计有效算法的分治策略。通过实验的编程代码，加深了对这些概念的理解，并通过实现代码加深了对算法的理解以及掌握。归并排序问题是分治算法的典型应用之一。实现了归并排序算法，将一个待排序的序列分为两个子集合，对每个子集合进行排序，然后将排序好的子集合合并成最终的有序序列。这个过程中，我学会了如何使用递归进行子问题的解决，并通过合并函数将子问题的解合并成原问题的解。