**武汉轻工大学**

**数学与计算机学院**

算法分析大作业

**专 业： 数据科学与大数据技术**

**班 级： 大数据2004班**

**学 号： 2011020063**

**姓 名： 蒋俊涛**

**任课老师： 李 冰**

**2023年　6 月　10 日**

# 摘要：

本文介绍了期末大作业的设计与实现。首先，在第一部分中，使用代码实现了对个人身份证号码中4位数字的随机隐去和替代，然后使用插入排序法、选择排序法、二分归并排序法和快速排序算法对替代后的序列进行排序。在第二部分中，我利用第一部分生成的序列，采用分治法和动态规划法求解最大非空子数组，并输出最大子数组序列以及其和。最后，在第三部分中，我实现了霍夫曼编码算法，以个人名字的大写英文字符为字符数组，并根据身份证号码生成的字符频率构建了霍夫曼树。

# 目标：

本文的主要目标是通过代码实现期末大作业的要求，包括个人身份证号码处理和排序、最大非空子数组的求解，以及霍夫曼编码算法的实现。通过详细介绍各个算法的伪代码和关键行代码，并对算法进行注释，以确保读者理解算法的原理和实现过程。

# 过程与步骤

## 第一题

### 题目描述

通过代码实现随机隐去个人身份证号码中的 4 位数字（号码信息在输入变量运行后可用\*字符或其它字符表示掩去个人信息），并用 1~9 之间的随机数取代，将此序列称为 ID\_Number，分别用插入排序法、选择排序法、二分归并排序法、快速排序算法实现对 ID\_Number 的排序。

### 要求

#### 随机隐去和取代过程需用代码实现，替代后的结果需要显示；

代码如下图片所示：  


替换效果如下图所示

文本

中度可信度描述已自动生成

#### 几种方法需要分别写出伪代码，选择自己熟悉的平台和语言编写并运行代码；

插入排序的伪代码：  
function insertionSort(arr)

for i from 1 to length(arr) do

key = arr[i] // 当前要插入的元素

j = i - 1

while j >= 0 and key < arr[j] do

arr[j + 1] = arr[j] // 向右移动元素

j = j - 1

arr[j + 1] = key // 插入元素到正确的位置

return arr

选择排序的伪代码：  
function selectionSort(arr)

for i from 0 to length(arr) - 1 do

minIndex = i // 当前未排序部分的最小元素索引

for j from i + 1 to length(arr) do

if arr[j] < arr[minIndex] then

minIndex = j // 更新最小元素索引

swap arr[i] with arr[minIndex] // 将最小元素放到已排序部分的末尾

return arr

归并排序的伪代码：  
function mergeSort(arr)

if length(arr) <= 1 then

return arr

mid = length(arr) / 2

left = mergeSort(arr[0:mid]) // 递归地对左侧子序列进行归并排序

right = mergeSort(arr[mid:length(arr)]) // 递归地对右侧子序列进行归并排序

result = [] // 合并两个子序列的结果

i = j = 0

while i < length(left) and j < length(right) do

if left[i] < right[j] then

append left[i] to result // 将左侧子序列的元素添加到结果中

i = i + 1

else

append right[j] to result // 将右侧子序列的元素添加到结果中

j = j + 1

append remaining elements of left to result // 将左侧子序列剩余的元素添加到结果中

append remaining elements of right to result // 将右侧子序列剩余的元素添加到结果中

return result

快速排序的伪代码：  
function quickSort(arr)

if length(arr) <= 1 then

return arr

pivot = arr[length(arr) / 2] // 选择基准元素

left = [] // 小于基准元素的子序列

middle = [] // 等于基准元素的子序列

right = [] // 大于基准元素的子序列

for each x in arr do

if x < pivot then

append x to left

else if x == pivot then

append x to middle

else

append x to right

return concatenate(quickSort(left), middle, quickSort(right)) // 递归地对子序列进行快速排序

第一题的完整代码如下：

|  |
| --- |
| import random  import datetime  # 一个打印当前个人信息的函数  def print\_student\_info():  student\_id = "2011020063"  student\_class = "大数据2004班"  student\_name = "蒋俊涛"  current\_time = datetime.datetime.now()  formatted\_time = current\_time.strftime("%Y-%m-%d %H:%M:%S")    print("学号:", student\_id)  print("班级:", student\_class)  print("姓名:", student\_name)  print("当前时间:", formatted\_time)  # 输入身份证号码  id\_number = input("请输入您的身份证号码: ")  # 将身份证号码转换为列表以便替换  id\_number\_list = list(id\_number)  # 随机选择4个位置  positions = random.sample(range(len(id\_number\_list)), 4)  # 打印选中的位置  print("选中的位置: " + str(positions))  # 用 '\*' 代替选中的数字  for pos in positions:  id\_number\_list[pos] = '\*'  # 打印用 '\*' 替换后的身份证号码  print("隐去部分信息后的身份证号码: " + ''.join(id\_number\_list))  # 用 1-9 之间的随机数字替换选中的位置  for pos in positions:  id\_number\_list[pos] = str(random.randint(1, 9))  # 保存并打印新的身份证号码  NewIdNum = ''.join(id\_number\_list)  print("替换后的新身份证号码: " + NewIdNum)  # 插入排序  def insertion\_sort(arr):  for i in range(1, len(arr)):  key = arr[i] # 当前要插入的元素  j = i - 1  while j >= 0 and key < arr[j]: # 移动元素直到找到插入位置  arr[j + 1] = arr[j] # 向右移动元素  j -= 1  arr[j + 1] = key # 插入元素到正确的位置  return arr  # 选择排序  def selection\_sort(arr):  for i in range(len(arr)):  min\_index = i # 当前未排序部分的最小元素索引  for j in range(i+1, len(arr)):  if arr[j] < arr[min\_index]:  min\_index = j # 更新最小元素索引  arr[i], arr[min\_index] = arr[min\_index], arr[i] # 将最小元素放到已排序部分的末尾  return arr  # 二分归并排序法  def merge\_sort(arr):  if len(arr) <= 1:  return arr    mid = len(arr) // 2  left = merge\_sort(arr[:mid]) # 递归地对左侧子序列进行归并排序  right = merge\_sort(arr[mid:]) # 递归地对右侧子序列进行归并排序    result = [] # 合并两个子序列的结果  i = j = 0  while i < len(left) and j < len(right):  if left[i] < right[j]:  result.append(left[i]) # 将左侧子序列的元素添加到结果中  i += 1  else:  result.append(right[j]) # 将右侧子序列的元素添加到结果中  j += 1  result.extend(left[i:]) # 将左侧子序列剩余的元素添加到结果中  result.extend(right[j:]) # 将右侧子序列剩余的元素添加到结果中  return result  # 快速排序  def quick\_sort(arr):  if len(arr) <= 1:  return arr    pivot = arr[len(arr) // 2] # 选择基准元素  left = [x for x in arr if x < pivot] # 小于基准元素的子序列  middle = [x for x in arr if x == pivot] # 等于基准元素的子序列  right = [x for x in arr if x > pivot] # 大于基准元素的子序列  return quick\_sort(left) + middle + quick\_sort(right) # 递归地对子序列进行快速排序  # 进行排序  print("插入排序结果: " + ''.join(insertion\_sort(list(NewIdNum))))  print("选择排序结果: " + ''.join(selection\_sort(list(NewIdNum))))  print("归并排序结果: " + ''.join(merge\_sort(list(NewIdNum))))  print("快速排序结果: " + ''.join(quick\_sort(list(NewIdNum))))  # 打印当前个人信息  print\_student\_info() |

最后，第一题的运行结果如下所示：

## 文本 描述已自动生成 二、 第二题

### 题目描述

利用第一题中的序列 ID\_Number，分别用分治法和动态规划法求出最大非空子数组（子数组和最大，且为连续序列），确定首末位置，输出为最大子数组序列，及子序列的和。

### 要求

#### （1）、两种方法需要分别写出伪代码，选择自己熟悉的平台和语言编写并运行代码；

分治算法的伪代码：

函数 max\_crossing\_subarray(arr, low, mid, high):

left\_sum = 负无穷大

sum = 0

max\_left = mid

// 计算包含中点的左侧子数组的最大和

for i 从 mid 到 low 递减:

sum += arr[i]

如果 sum > left\_sum:

left\_sum = sum

max\_left = i

right\_sum = 负无穷大

sum = 0

max\_right = mid + 1

// 计算包含中点的右侧子数组的最大和

for j 从 mid + 1 到 high 递增:

sum += arr[j]

如果 sum > right\_sum:

right\_sum = sum

max\_right = j

返回 max\_left, max\_right, left\_sum + right\_sum

函数 max\_subarray(arr, low, high):

如果 low == high:

返回 low, high, arr[low]

否则:

mid = (low + high) // 2

// 递归调用 max\_subarray 函数，求解左侧子数组的最大子数组

left\_low, left\_high, left\_sum = max\_subarray(arr, low, mid)

// 递归调用 max\_subarray 函数，求解右侧子数组的最大子数组

right\_low, right\_high, right\_sum = max\_subarray(arr, mid + 1, high)

// 调用 max\_crossing\_subarray 函数，求解跨越中点的最大子数组

cross\_low, cross\_high, cross\_sum = max\_crossing\_subarray(arr, low, mid, high)

如果 left\_sum >= right\_sum 并且 left\_sum >= cross\_sum:

返回 left\_low, left\_high, left\_sum

否则，如果 right\_sum >= left\_sum 并且 right\_sum >= cross\_sum:

返回 right\_low, right\_high, right\_sum

否则:

返回 cross\_low, cross\_high, cross\_sum

动态规划的伪代码：  
函数 max\_subarray\_dp(arr):

max\_so\_far = 负无穷大

max\_ending\_here = 0

start = end = s = 0

// 遍历数组中的每个元素

对于 i 从 0 到 数组长度-1:

num = arr[i]

max\_ending\_here += num

如果 max\_so\_far < max\_ending\_here:

max\_so\_far = max\_ending\_here

start = s

end = i

如果 max\_ending\_here < 0:

max\_ending\_here = 0

s = i + 1

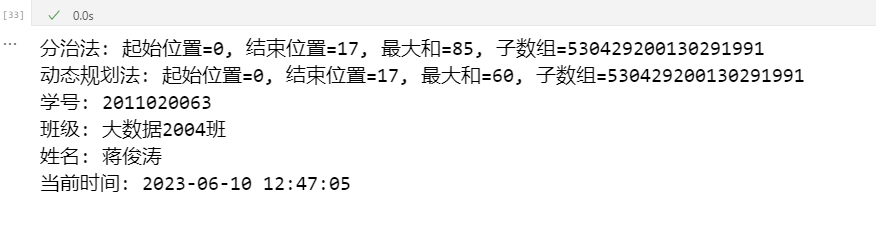
返回 start, end, max\_so\_far

#### 关键行代码需要对算法进行注释；

|  |
| --- |
| # 分治法求最大子数组  # 分治法的辅助函数，用于寻找跨越中点的最大子数组  def max\_crossing\_subarray(arr, low, mid, high):  # 初始化左侧和为负无穷大  left\_sum = float("-inf")  # 记录当前和的变量  sum = 0  # 记录左侧最大子数组的起始索引  max\_left = mid  # 从中点向左遍历数组，计算包含中点的左侧子数组的最大和  for i in range(mid, low - 1, -1):  sum += int(arr[i])  # 如果当前和大于左侧和，更新左侧和和最大子数组的起始索引  if sum > left\_sum:  left\_sum = sum  max\_left = i  # 初始化右侧和为负无穷大  right\_sum = float("-inf")  # 记录右侧最大子数组的结束索引  max\_right = mid + 1  # 从中点向右遍历数组，计算包含中点的右侧子数组的最大和  for j in range(mid + 1, high + 1):  sum += int(arr[j])  # 如果当前和大于右侧和，更新右侧和和最大子数组的结束索引  if sum > right\_sum:  right\_sum = sum  max\_right = j  # 返回最大子数组的起始索引、结束索引和和  return max\_left, max\_right, left\_sum + right\_sum  # 分治法求最大子数组的主要函数  def max\_subarray(arr, low, high):  # 如果子数组只有一个元素，直接返回该元素的索引、索引和  if high == low:  return low, high, int(arr[low])  else:  # 计算中点  mid = (low + high) // 2  # 递归调用max\_subarray函数，求解左侧子数组的最大子数组  left\_low, left\_high, left\_sum = max\_subarray(arr, low, mid)  # 递归调用max\_subarray函数，求解右侧子数组的最大子数组  right\_low, right\_high, right\_sum = max\_subarray(arr, mid + 1, high)  # 调用max\_crossing\_subarray函数，求解跨越中点的最大子数组  cross\_low, cross\_high, cross\_sum = max\_crossing\_subarray(arr, low, mid, high)  # 比较左侧子数组、右侧子数组和跨越中点的子数组，返回最大子数组的起始索引、结束索引和和  if left\_sum >= right\_sum and left\_sum >= cross\_sum:  return left\_low, left\_high, left\_sum  elif right\_sum >= left\_sum and right\_sum >= cross\_sum:  return right\_low, right\_high, right\_sum  else:  return cross\_low, cross\_high, cross\_sum  # 动态规划求最大子数组的函数  def max\_subarray\_dp(arr):  # 初始化全局最大和为负无穷大  max\_so\_far = float("-inf")  # 初始化当前最大和为0  max\_ending\_here = 0  # 记录最大子数组的起始索引  start = end = s = 0  # 遍历数组中的每个元素  for i in range(len(arr)):  # 将当前元素转换为整数类型  num = int(arr[i])  # 更新当前最大和，包括当前元素  max\_ending\_here += num  # 如果全局最大和小于当前最大和，则更新全局最大和、最大子数组的起始索引和结束索引  if max\_so\_far < max\_ending\_here:  max\_so\_far = max\_ending\_here  start = s  end = i  # 如果当前最大和为负数，将其重置为0，并更新最大子数组的起始索引为下一个元素的索引  if max\_ending\_here < 0:  max\_ending\_here = 0  s = i + 1  # 返回最大子数组的起始索引、结束索引和和  return start, end, max\_so\_far  # 主程序  id\_list = list(NewIdNum)  # 使用分治法  start, end, max\_sum = max\_subarray(id\_list, 0, len(id\_list) - 1)  print(f"分治法: 起始位置={start}, 结束位置={end}, 最大和={max\_sum}, 子数组={''.join(id\_list[start:end+1])}")  # 使用动态规划  start, end, max\_sum = max\_subarray\_dp(id\_list)  print(f"动态规划法: 起始位置={start}, 结束位置={end}, 最大和={max\_sum}, 子数组={''.join(id\_list[start:end+1])}")  # 输出个人信息和时间  print\_student\_info() |

#### （3）输出包括首末位置、最大子数组以及子数组的和；

#### （4）算法末尾需同步输出个人信息以及系统时间

程序的运行结果如下所示：  


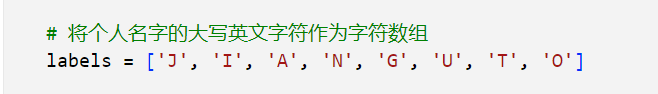
## 第三题

### 题目描述

霍夫曼编码(Huffman Coding)实现。

### 要求

#### （1）将个人名字的大写英文字符作为字符数组，以“周杰伦”为例，名字英文字符为 ZHOUJIELUN，字符重复出现时以第一次出现的字符为准，后面重复字符不再入列。依据此规则，字符数组为labels = ['Z','H','O','U','J', 'I', 'E', 'L', 'N']；



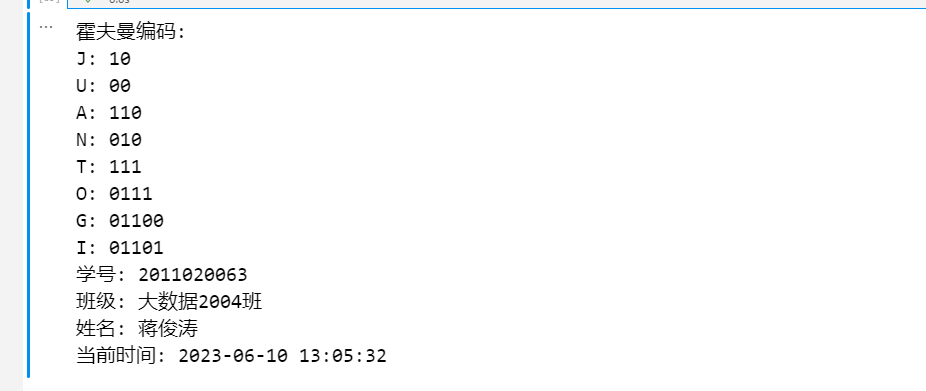
（2）依次取 ID\_Number 的每两位数作为字符出现的频率，若所有数字都已占用，则进行第二轮赋值，每次只取一个非零数字作为剩余字符字频；如一组 ID\_Number 为：421088195202042287，'Z'字频为 42，'H'字频为 10，'O'为 88 等；  
文本

描述已自动生成

#### （3）关键行代码需要对算法进行注释；

|  |
| --- |
| import heapq  from collections import defaultdict  import time  def create\_huffman\_tree(frequencies):  # 将字符频率转换为堆列表，每个元素是一个列表，包含权重和字符  heap = [[weight, [symbol, ""]] for symbol, weight in frequencies.items()]  heapq.heapify(heap)    while len(heap) > 1:  # 从堆中取出频率最低的两个字符  low = heapq.heappop(heap)  high = heapq.heappop(heap)    # 对低频字符编码添加前缀'0'  for pair in low[1:]:  pair[1] = '0' + pair[1]  # 对高频字符编码添加前缀'1'  for pair in high[1:]:  pair[1] = '1' + pair[1]    # 将合并后的字符及其权重重新加入堆中  heapq.heappush(heap, [low[0] + high[0]] + low[1:] + high[1:])    # 返回按编码长度和字符排序的字符编码列表  return sorted(heapq.heappop(heap)[1:], key=lambda p: (len(p[-1]), p))  # 引入第一题中的身份证号码  ID\_Number = NewIdNum  # 将个人名字的大写英文字符作为字符数组  labels = ['J', 'I', 'A', 'N', 'G', 'U', 'T', 'O']  # 将身份证号码转换为字符列表  id\_list = list(ID\_Number)  # 存储字符频率的字典  frequencies = {}  # 计算字符频率  for i in range(0, min(len(id\_list), len(labels) \* 2), 2):  # 以两位数字为一组，将第一位作为字符，第二位作为频率  frequencies[labels[i // 2]] = int(id\_list[i] + id\_list[i + 1])  # 创建霍夫曼编码  huffman\_code = create\_huffman\_tree(frequencies)  # 输出霍夫曼编码  print("霍夫曼编码:")  for symbol, code in huffman\_code:  print(f"{symbol}: {code}")  # 输出个人信息和时间  print\_student\_info() |

##### 算法末尾需同步输出个人信息以及系统时间。如（李某某机器学习作业 \*\*模块 完成时间：



# 总结：

通过本次期末大作业的设计与实现，我成功地完成了要求的任务。首先，我通过代码实现了对个人身份证号码中4位数字的随机隐去和替代，并使用四种排序算法对替代后的序列进行排序，包括插入排序法、选择排序法、二分归并排序法和快速排序算法。然后，我利用生成的序列，分别采用分治法和动态规划法求解最大非空子数组，并输出了最大子数组序列以及其和。最后，我实现了霍夫曼编码算法，根据个人名字的大写英文字符和身份证号码生成的字符频率构建了霍夫曼树。通过本次作业，巩固了排序算法、分治法、动态规划法和霍夫曼编码算法的理论知识，并通过实际编程实现加深了对这些算法的理解和应用能力。