1. **Python编程实现二分法查找**

*# 代码实现：*l = []  
item = int(input(**'请输入您想查找的元素：'**))  
**while True**:  
 num = int(input(**'请输入数字：'**))  
 l.append(num)  
 **if** len(l) == 5:  
 **break**print(**'输入的列表为：%s'**%l)  
  
**for** i **in** range(len(l)-1):  
 **for** j **in** range(len(l)-1-i):  
 **if** l[j] > l[j+1]:  
 l[j],l[j+1] = l[j+1],l[j]  
print(**'排序后的列表为：%s'**%l)  
  
low = 0  
high = len(l)-1  
res = 0  
  
*# print(l)***while** low < high:  
 middle = (low + high) // 2  
 guess = l[middle]  
 **if** guess > item:  
 high = middle - 1  
 **elif** guess < item:  
 low = middle + 1  
 **else**:  
 res = middle  
 **break**print(**'您查找的元素：%d,其位置为第%d位'**%(item,res))

1. **Python编程实现狄克斯特拉算法**

graph={}  
graph[**'start'**]={} *#定义图中的各个邻居节点*graph[**'start'**][**'a'**]=6  
graph[**'start'**][**'b'**]=-1  
graph[**'a'**]={}  
graph[**'a'**][**'end'**]=1  
graph[**'b'**]={}  
graph[**'b'**][**'a'**]=3  
graph[**'b'**][**'end'**]=5  
graph[**'end'**]={}  
cost={} *#首尾已知点的代价*infinity=float(**'inf'**)  
cost[**'a'**]=6  
cost[**'b'**]=-1  
cost[**'end'**]=infinity  
parent={} *#首尾已知点的负节点*parent[**'a'**]=**'start'**parent[**'b'**]=**'start'**parent[**'end'**]=**None***#print(cost.keys())*process=[]  
**def** select\_lowest\_cost(cost):  
 lowest\_cost=float(**'inf'**)  
 lowest\_cost\_node=**None  
 for** node **in** cost.keys() :  
 **if** cost[node]<lowest\_cost **and** node **not in** process: *#保证所有的初始节点（第一步）都被考虑到* lowest\_cost=cost[node]  
 lowest\_cost\_node=node  
 **return** lowest\_cost\_node  
*#print(select\_lowest\_cost(cost))*node=select\_lowest\_cost(cost)  
**while** node **is not None**:  
 neighbor=graph[node]  
 costs=cost[node]  
 *#neighbor[node]=graph[node].keys()* **for** i **in** neighbor.keys():  
 c=costs+neighbor[i]  
 **if** c<cost[i]:  
 cost[i]=c  
 parent[i]=node  
 process.append(node) *#处理完的点表示该节点所有的邻居节点都考虑完全* node = select\_lowest\_cost(cost)  
print(cost[**'end'**])

1. **解读一下：程序 = 数据结构 + 算法 这个经典范式的含义。**

算法:解决问题的流程/步骤(顺序、分支、循环...)

  数据结构:将数据按照某种特定的结构来保存  
 通俗的说 算法相当于逻辑，小部分已为人们发掘出来（这里的小部分指的是书本里讲的各种算法，属于人们对于特定模式抽象出来的核心，比如排序），可以看做一种模式。对应于业务来说，一种逻辑（可能由其他元子逻辑组合而成）一旦确定下来，便可看做常量，固定不变。

数据结构即数据表示，说白了就是数据，比如用户数据，属于互联网玩的主要部分。这里面有一个问题，就是如何合理高效表示数据。为此，人们想出了各种各样的数据结构，比如数组，比如树。还有一点就是代码通用性的考量。对于一个设计良好的数据（结构）来讲，应当可以保证在代码逻辑不变的基础上，功能的增加只需在数据层动点手脚完成：下拉菜单数据中追加一条“详情页”的数据和对应的回调方法，即可完成新菜单项的添加工作。

 问题 —> 数据结构+算法 == 程序 —> 解决问题

“算法“ -> 逻辑

“数据结构“ -> 存储

1. **有这样一种说法：‘数据结构是抽象数据类型的物理实现’。请问你怎么理解这句话**

在Clifford A.Shaffer在《数据结构与算法分析》一书中，对于数据结构的定义是这样的：数据结构是抽象数据类型的物理实现。在理解这句话之前，我们先了解一下什么是抽象数据类型。

抽象数据类型(Abstract Data Type，简称ADT)，是指一个数学模型以及定义在此数学模型上的一组操作。抽象数据类型需要通过固有数据类型（高级编程语言中已实现的数据类型）来实现。抽象数据类型是与表示无关的数据类型，是一个数据模型及定义在该模型上的一组运算。对一个抽象数据类型进行定义时，必须给出它的名字及各运算的运算符名，即函数名，并且规定这些函数的参数性质。一旦定义了一个抽象数据类型及具体实现，程序设计中就可以像使用基本数据类型那样，十分方便地使用抽象数据类型(来自百度百科).  
 通俗解释：最开始的计算机语言，关注的都是如何更加有效率地计算，可以说其目的是计算层面的抽象。然而随着这个行业的不断发展，计算机不仅仅用于计算，开发也不仅只关注计算过程了，数据层面的抽象也变得同样重要。虽然计算机语言一开始就有对数据的抽象，但是那些都只是对一些最基本的数据类型而不包括我们想要用的，多种多样的数据。  
 程序处理的数据，通常是不同的类型的。只有事先约定好的不同类型的数据的存储方式，计算机才能正确理解逻辑上不同的数据类型。所有编程语言都会有一组内置的基本数据类型。另外在实际工作过程中，或早或晚总会碰到一些没法用现有数据类型解决的问题，这时就需要自定义一些数据类型来解决。像Python这样比较高级的语言的话，在基本类型的基础上还添加了一些额外的数据结构如tuple,list,dict（这些如果从广义上来说也算是Python的数据类型）。

以上基本数据类型都是比较simple的结构，但是他们具有一个共通的问题，就是都会把数据暴露在外。这点类似于一个人如果有了对某个变量的权限的话他就可以看到这个变量代表的数据结构中的所有数据。而如果我们不希望被看到，那么这个问题可能就比较严重了。为了解决这个问题，就必须要有一种数据类型，它可以让使用者只需要考虑如何去使用，而不需要去关注类型内部的具体实现方式以及数据的表示等等。这样的类型从概念上来说就是抽象数据类型了。

也就是说，各大编程语言自带的基本数据类型满足不了日常编程开发的需求，同时又因为基本数据类型将数据直接暴露在外影响了数据的安全性及可靠性，所以，需要通过抽象数据类型，将数据封装，只对外提供可用的操作及可访问的属性。而抽象数据类型是人为定义的一种自定义类型，所以抽象数据类型可以是任何一种数据。

抽象数据类型的实现思路：基本思想均是把数据定义为抽象的数据对象集合，只为他们定义可用的合法操作，而不暴露具体的内部细节，注意，不论是操作细节还是数据存储细节均不暴露。  
 基于这种思路，一般而言，抽象数据类型应该至少具备以下三种操作：  
 1.构造操作，即如何通过抽象数据类型创建抽象数据对象  
 2.解析操作(也可理解为获取操作)：一定会有一系列getXXX()方法去获取到该抽象 类型内部的任意数据及操作  
 3.修改操作：一定会有一系列setXXX()方法去给当前抽象数据类型修改任意数据 及操作。  
 而我们说，在Python中，对象有可变与不可变之分，那么基于以上的三种基本操作，可以判断一个数据类型的可变性。  
 如果一个类型同时具备以上三种类型，那么这个类型就是可变的。  
 如果一个类型只具备一二两种操作，那么这个类型就是不可变的。

那么其实，讲到这里，抽象数据类型到底是什么我们其实应该明白了。没错，其实抽象数据类型应用在数据结构中，就是一个一个的类。这就是为什么，我们说数据结构是抽象数据类型的物理实现了，原因很简单，数据结构中有很多所谓的数据结构，比如线性表，树，森林等等。其实这些理论应用在编程上就是对应的各种各样的类以及由类实例化得到的对象。

1. **请解释：什么是有向图？详细说一说拓扑排序算法？**

一个有向图D是指一个有序[三元组](https://baike.baidu.com/item/%E4%B8%89%E5%85%83%E7%BB%84/2434943" \t "https://baike.baidu.com/item/%E6%9C%89%E5%90%91%E5%9B%BE/_blank)(V(D)，A(D)，ψD)，其中ψD)为[关联函数](https://baike.baidu.com/item/%E5%85%B3%E8%81%94%E5%87%BD%E6%95%B0/8320961" \t "https://baike.baidu.com/item/%E6%9C%89%E5%90%91%E5%9B%BE/_blank)，它使A(D)中的每一个元素(称为有向边或弧)对应于V(D)中的一个有序元素(称为顶点或点)对．

拓扑排序：

1.有向无环图：如果一个有向图无法从某个顶点出发经过若干条边回到该点，则这个图是一个有向无环图（DAG图）  
 2.拓扑序列：对一个有向无环图(Directed Acyclic Graph简称DAG)G进行拓扑排序，是将G中所有顶点排成一个线性序列，使得图中任意一对顶点u和v，若边(u,v)∈E(G)，且u在线性序列中出现在v之前。通常，这样的线性序列称为满足拓扑排序(TopologicalOrder)的序列，简称拓扑序列。  
 3.拓扑排序算法： 循环执行以下两步，直到不存在入度为0的顶点为止  
 1.选择一个入度为0的顶点并输出之；  
 2.从网中删除此顶点及所有出边。

1. **数据结构都包含哪些呢？**

线性表，树，图，森林等等。

1. **线性表的顺序存储和链式存储有何区别？**

顺序表示：指的是用一组地址连续的存储单元依次存储线性表的数据元素，称为线性表的顺序存储结构或顺序映像（sequential mapping）。它以“物理位置相邻”来表示线性表中数据元素间的逻辑关系，可随机存取表中任一元素。

链式表示：指的是用一组任意的存储单元存储线性表中的数据元素，称为线性表的链式存储结构。它的存储单元可以是连续的，也可以是不连续的。在表示数据元素之间的逻辑关系时，除了存储其本身的信息之外，还需存储一个指示其直接后继的信息（即直接后继的存储位置），这两部分信息组成数据元素的存储映像，称为结点（node）。它包括两个域；存储数据元素信息的域称为数据域；存储直接后继存储位置的域称为指针域。指针域中存储的信息称为指针或链。

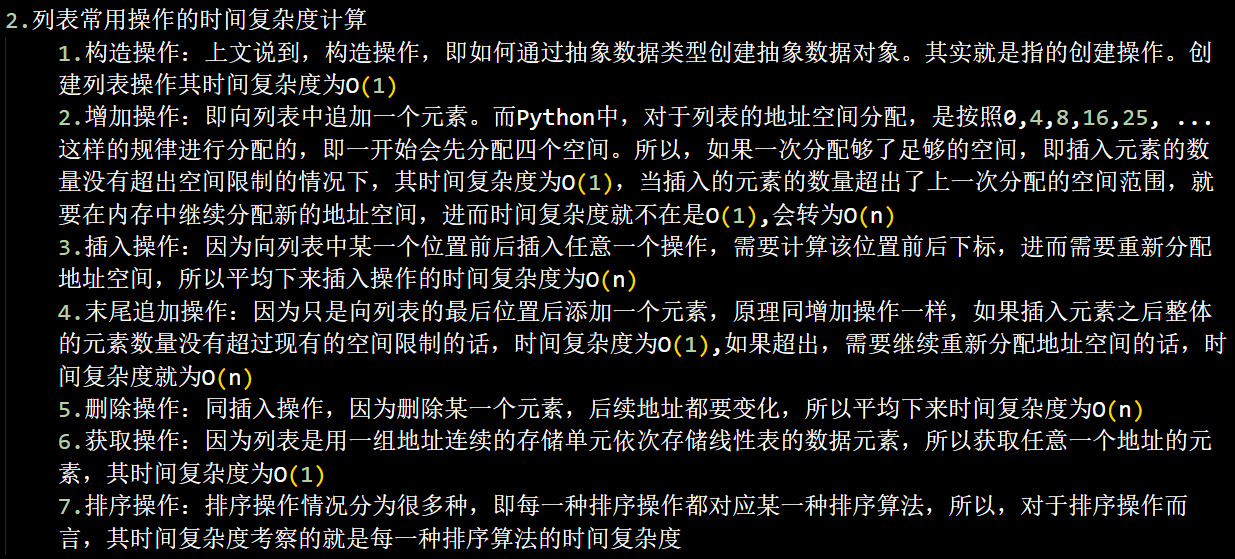
1. **什么叫堆？完全二叉树形成条件是什么？**

堆：数据结构——树中，一类特殊的数据结构的总称。定义如下：n个元素的序列[k1,k2,k3,...ki,...,kn](i = 1,2,3,4,...,n/2),当且仅当满足以下条件：(ki <= k2i,ki <= k2i+1)或>=时，就称之为堆。  
 注意：如果用一个列表存储一个堆，则该堆一定对应一个完全二叉树(堆的本质就是一棵树，一颗二叉树，一颗完全二叉树)。那么该完全二叉树的根结点就叫做堆的堆顶元素。  
当此条件：(ki <= k2i,ki <= k2i+1)均为小于时，就会产生小顶堆，反之产生大顶堆。所以由堆的定义可以看出，第一个元素，即堆顶元素必为最小项或最大项，同时，所有非叶子结点的值均不大于或不小于其子女的值  
 比如：  
 大顶堆：[87,84,85,36,11,32]  
 87  
 84 45  
 36 11 32  
 小顶堆：[12,23,26,32,24,30,28,43,35]  
 12  
 23 26  
 32 24 30 28  
 43 35

1. **八大常用排序算法，请列举**

****

1. **顺序表的插入操作，删除操作，遍历操作，末尾追加操作，末尾删除操作，排序操作的时间复杂度为多少？**

****

1. **什么叫算法的稳定性呢？怎么解释？**

算法的稳定性：假定在待排序的列表中，存在多个相同的元素，若经过排序，这些想等元素的相对次序保持不变，即在原序列中，l[i] = l[i + 1]或l[i - 1] = l[i],且i-1 在 i 之前，i+1 在 i 之后，而在排序后的序列中，仍然为i-1 在 i 之前，i+1 在 i 之后，则称这种排序算法是稳定的，反之即为不稳定

1. **非线性时间比较类排序算法都有哪些呢？**

非线性时间比较类算法：直接插入排序，希尔排序，冒泡排序，快速排序，简单选择排序，堆排序

1. **八大排序算法中，稳定性较好的算法有哪些？**

答案见第九题

1. **Python编程实现简单选择排序，直接插入排序，堆排序，冒泡排序，快速排序。**

简单选择排序

l = [3,1,5,7,2,4,9,6]  
**for** i **in** range(0, len(l)):  
 *# 默认设置最小值的下标为当前值* min = i  
 *# 在剩余n-1个元素的序列中找到最小元素的下标* **for** j **in** range(i + 1, len(l)):  
 *# if l[i] > l[i+1]  
 # 如果找到，就把最小元素的下标赋值给min* **if** l[min] > l[j]:  
 min = j  
 *# 将找到的最小值的元素和当前元素做位置交换* temp = l[min]  
 l[min] = l[i]  
 l[i] = temp  
print(l)

直接插入排序

*#初始化序列*l = [45,34,78,78,98,23,12]  
*# 拿出序列第一个元素后，从第二个开始用for循环遍历序列***for** i **in** range(1,len(l)):  
 *# 后面的和前面的比较* j = i - 1  
 *# 哨兵，临时存放元素* key = l[i]  
 *# 每一次遍历决定插入的最终位置* **while** j >= 0:  
 *# 如果后面的值小于前面的值* **if** key < l[j]:  
 *# 后面的值赋值给前面的值，相当于交换位置* l[j+1] = l[j]  
 *# 前面的值拿出来存在哨兵中* l[j] = key  
 *# 继续向前比较* j -= 1  
print(l)

冒泡排序：

**import** random  
**import** math  
l = []  
counts = int(input(**"请输入您想生成的序列的元素个数："**))  
**while True**:  
 num = math.floor(random.random()\*100+1)  
 l.append(num)  
 **if** len(l) == counts:  
 **break**print(**'排序前，生成好的随机列表为:%s'**%l)  
**for** i **in** range(len(l)-1):  
 **for** j **in** range(len(l)-i-1):  
 **if** l[j] > l[j+1]:  
 temp = l[j]  
 l[j] = l[j+1]  
 l[j+1] = temp  
print(**"排序后的列表为：%s"**%l)

堆排序

**import** random  
**import** math  
  
*#随机生成0~100之间的数值，此随机生成的序列无序***def** get\_randomNumber():  
 list=[]  
 i=0  
 num = int(input(**"请输入元素个数："**))  
 **while** i<num:  
 list.append(math.floor(random.random()\*100+1))  
 i += 1  
 **return** list  
  
*# 将一个无序序列形成完全二叉树，此完全二叉树同样无序***def** PrintList\_Dui(l):  
 *# 提供一个初始化好的列表  
 # 第一行时只有一个数  
 # 每行的数字数量* number\_count=1*#2 4  
 # 行数* row\_number=1 *#2 3  
 # 遍历这个列表  
 # 形成的堆结构* print(**"形成的完全二叉树结构："**)  
 **for** i **in** range(len(l)):  
 *# 因为要形成一个完全二叉树，每一层(行)的元素数量为2 \*\* (行数-1)个  
 # 使用一个巧妙的算法来实现打印出一个完全二叉树  
 # 此段算法分析：  
 # i = 0时，循环刚开始，因为0和number\_count不等，if跳过，直接先打印arr[0]的值，10  
 # i = 1时，1和1相等，执行if,number\_count(每行元素个数)变为3，然后换行，行数加1，再输出arr[1] 9  
 # i = 2时，2 和 3 不等，直接打印arr[2] 8  
 # i = 3时，3和3相等，if执行，number\_count变为7，换行，行数变为3，输出arr[3] = 7  
 # ...  
 # 直至程序结束  
 #-----------------------------#* **if** i==number\_count:  
 number\_count += 2 \*\* row\_number*# i = 1时，a = 3* print(**"\n"**)  
 row\_number += 1*# i = 1时 row = 2  
 #-----------------------------#* print (l[i],end=**" "**)*# i = 0,if跳过，直接运行这行，10被打印 i = 1时，if执行，换到第二行，打印9  
 #-----------------------------#* print(**"Over"**)  
 **return** l  
  
*# 至此，基本功能实现，接下来就是详细实现步骤(大顶堆，形成升序序列)  
# 第一步：创建堆。  
# 将一个无序列表写为顺序二叉树，从最后一个非叶子节点开始，按照从下至上，从左至右的顺序遍历，依次寻找每一个非叶子节点的左右子节点，然后将其与父节点大小进行比较  
# ，如果子节点比父节点大，就将父子节点交换值  
  
# 第一步：创建堆***def** build\_heap(l, size):  
 *# 从最后一个非叶子结点开始遍历，从下至上，从左至右遍历* **for** i **in** range(0, (size // 2))[::-1]:  
 adjust\_heap(l, i, size)  
 *# 此时需要进行调整堆，所以调用调整堆的函数，创建堆其实只是先需要创建一个初始化的堆结构，但因为形成堆结构就需要进行依次  
 # 调整堆，所以在这里需要调用调整堆函数  
  
# 第二步：调整堆***def** adjust\_heap(l, i, size):  
 *# 父节点i的左子节点* lchild = 2 \* i + 1  
 *# 父节点i的右子节点* rchild = 2 \* i + 2  
 *# 堆顶(最大值)的下标* max = i  
 **if** i < size // 2:  
 *# 如果没有超出数组的深度，且非叶子节点左子节点的值大于父节点的值* **if** lchild < size **and** l[lchild] > l[max]:  
 *# 将该非叶子节点的左子节点的值存在在max变量* max = lchild  
 *# 如果没有超出数组的深度，且非叶子节点右子节点的值大于父节点的值* **if** rchild < size **and** l[rchild] > l[max]:  
 *# 将该非叶子节点的右子节点的值存放在max变量* max = rchild  
 *# 挑出来的非叶子节点的值的下标此时已经存在max(堆顶，也是最大值)中，将max(带着的就是挑出来的左或者右子节点的下标)跟父节点进行交换* **if** max != i:  
 l[max], l[i] = l[i], l[max]  
 *# 到此，准备遍历第二遍，第二遍仍然是在重复调整堆，所以调整函数需要再执行一遍，形成递归。* adjust\_heap(l, max, size)  
  
*# 第三步：堆排序***def** heap\_sort(l):  
 *# 获取列表长度* size = len(l)  
 *# 堆排序之前，先创建堆* build\_heap(l, size)  
 *# 从后向前遍历，将每一次调整堆之后获得的堆顶和最后一个元素交换位置，先得到第一大最大值* **for** i **in** range(0, size)[::-1]:  
 *# 循环内，继续进行调整堆，不断得到第二大最大值，第三大最大值，...第n大最大值  
 # 这里是交换位置* l[0], l[i] = l[i], l[0]  
 *# 重复进行调整堆，直至剩下最后两个元素交换位置，排序完成* adjust\_heap(l, 0, i)  
 *# 整个完整的调整堆的过程就是一个堆排序过程  
 # 所以，其实可以发现，创建堆的过程其实只需要一遍调整堆就ok，但因为第二步是重复调整堆，  
 # 所以很自然的直接过度到调整堆函数，  
 # 而重复调整堆之后每一次调整堆都会交换堆顶和最后一个元素的位置  
 # 不断的形成升序方向的最大值，所以完整的重复调整堆的过程就是堆排序* **return** l  
  
  
l = get\_randomNumber()  
print(**"排序之前(随机生成的列表)：%s"** %l)  
b = PrintList\_Dui(l)  
print(**"排序之前，形成完全二叉树结构之后的无序列表:%s"  
 "(其实跟原列表一样一样的，只不过是对照着这个无序列表形成完全二叉树结构)"**%b)  
finall\_list = heap\_sort(l)  
print(**'排序之后的列表为：%s'**%finall\_list)

1. **输出序列逆序对儿，如输入数组[1,2,5,4,3]，输出(5,4),(5,3),(4,3)**
2. **二维数组中的查找：请完成一个函数，输入一个二维数组和一个整数，判断数组中是否有该整数**
3. **Python中比较常用的搜索算法都有哪些呢？**
4. **Python数据类型有可变和不可变之分。请问，站在数据结构的角度上分析，为什么会有可变与不可变之分呢？**
5. **请用Python来编程，模拟实现线性表的顺序存储。**
6. **我们可以用2\*1的小矩形横着或者竖着去覆盖更大的矩形。请问用n个2\*1的小矩形无重叠地覆盖一个2\*n的大矩形，总共有多少种方法？**
7. **Python编程实现：查找完全二叉树任意两个节点的最近公共祖先**
8. **线性表与树形结构之间是否可以转换呢？举例说明Python中用到的线性表与树形结构之间的转换实例。**