云南大学数据结构和算法考研复试试题

1．什么是算法的时间复杂度？

算法是对特定问题求解步骤的一种描述，是指令的有限序列。

算法的 5 个特性：

1.可行性 2 确定性 3 有穷性 4 输入 （大于等于 0 个）5 输出（必须有输出）

时间复杂度是将一个算法转换成程序并在计算机上执行时，其运行所需要的时间。

空间复杂度是指算法在计算机内执行时所需存储空间的度量

程序=数据结构+算法

软件=程序+文档

2．简述数据的逻辑结构和物理结构的概念和两者的关系

数据的逻辑结构指的是数据元素之间的关系。

数据的物理结构指的是：数据结构在计算机中的标识，具体存放在哪个位置。

3．快速排序在什么情况下效率最高？什么情况下最差？

快速排序是对冒泡排序的一种改进。它的基本思想是：通过一躺排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一不部分的所有数据都要小，然后再按次方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

在几乎有序的情况下效率最差，无序的情况下效率最高。

4．什么是 NP 问题？

NP 问题叫非确定性问题。NP 就是 Non-deterministic Polynomial 的问题，也即是多项式复杂程度的非确定性问题。这种问题的答案，是无法直接计算得到的，只能通过间接的“猜算”来得到结果。解决这个猜想，无非两种可能，一种是找到一个这样的算法，只要针对某个特定 NP 完全问题找到一个算法，所有这类问题都可以迎刃而解了，因为他们可以转化为同一个问题。另外的一种可能，就是这样的算法是不存在的。那么就要从数学理论上证明它为什么不存在。

5．什么是递归算法

递归算法：是一种直接或者间接地调用自身的算法。

递归算法的特点

递归过程一般通过函数或子过程来实现。

递归算法：在函数或子过程的内部，直接或者间接地调用自己的算法。

递归算法的实质：是把问题转化为规模缩小了的同类问题的子问题。然后递归调用函数(或过程)来表示问题的解。

6．如何用顺序结构存储完全二叉树

用一组连续的存储单元存储二叉树的数据元素。将二叉树中编号为 i 的结点的数据元素存放在分量 bt[i]中。根据完全二叉树的特性，结点在向量中的相对位置蕴含着结点之间的关系。显然，这种顺序存储结构仅适合于完全二叉树，因为在顺序存储结构中，仅以结点在向量中的相对位置表示结点之间的关系，因此，一般的二叉树也必须按完全二叉树的形式来存储，这将造成存贮的浪费。在最坏的情况下，一个深度为 k 且只有 k 个结点的单支树（树中无度为 2 的结点）却需 2k- 1 个 存储分量。

7．堆得特征是什么？如何利用堆进行排序？

n 个关键字序列 Kl，K2，…，Kn 称为堆，当且仅当该序列满足如下性质(简称为堆性质)：

（ 1 ）ki≤K2i 且 ki≤K2i+1 或(2)Ki≥K2i 且 ki≥K2i+1(1≤i≤)

树中任一非叶结点的关键字均不大于(或不小于)其左右孩子(若存在)结点的关键字。

大根堆排序算法的基本操作：

① 初始化操作：将 R[1..n]构造为初始堆；

② 每一趟排序的基本操作：将当前无序区的堆顶记录 R[1]和该区间的最后一个记录交换，然后将新的无序区调整为堆(亦称重建堆)。

注意：

①只需做 n-1 趟排序，选出较大的 n-1 个关键字即可以使得文件递增有序。

②用小根堆排序与利用大根堆类似，只不过其排序结果是递减有序的。堆排序和直接选择排序相反：在任何时刻，堆排序中无序区总是在有序区之前，且有序区是在原向量的尾部由后往前逐步扩大至整个向量为止。

8.. 贪心算法的思想是什么？能得到最佳效果吗？

所谓贪心算法是指，在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，他所做出的仅是在某种意义上的局部最优解。贪心算法不是对所有问题都能得到整体最优解，但对范围相当广泛的许多问题他能产生整体最优解或者是整体最优解的近似解。

贪心算法的基本思路如下：

1.建立数学模型来描述问题。

2.把求解的问题分成若干个子问题。

3.对每一子问题求解，得到子问题的局部最优解。

4.把子问题的解局部最优解合成原来解问题的一个解。

9.简述用非递归实现递归的方法？

原理都是用堆栈，区别就是一个由系统来管理堆栈，另一个由自己来管理堆栈，都可以实现递归的功能。

10.树的遍历方法有哪些？：--先序　后序

1.树的先序遍历与其转换的相应二叉树的先序遍历的结果相同。树的后序遍历与其转换的相应二叉树的中序遍历的结果序列相同。

2.森林的先序遍历和后序遍历与所转换的二叉树的先序遍历和中序遍历的结果序列相同。

3.二叉树的遍历方法：先根　中根　后根

11.对链表设置头结点的好处是什么/

【解答】 其好处有:

(1) 对带头结点的链表，在表的任何结点之前插入结点或删除表中任何结点，所要做的都是修改前一结点的指针域，因为任何元素结点都有前驱结点。若链表没有头结点，则首元素结点没有前驱结点，在其前插入结点或删除该结点时操作会复杂些。

(2) 对带头结点的链表，表头指针是指向头结点的非空指针，因此空表与非空表的处理是一样的

12. 分治法的基本思想是什么？

分治法的基本思想是将一个规模为 n 的问题分解为 k 个规模较小的子问题，这些子问题互相独立且与原问题相同。递归地解这些子问题，然后将各个子问题的解合并得到原问题的解。它的一般的算法设计模式如下：divide-and-conquer(P)

```
{
    if(|P|<=n0) adhoc(P);
    divide P into smaller subinstances P1,P2,...,Pk;
    for(i=1;i<=k;i++)
        yi=divide-and-conquer(Pi);
    return merge(y1,...,yk);
```

}

其中，|P|表示问题 P 的规模。n0 为一阀值，表示当问题 P 的规模不超过 n0 时，问题已容易解出，不必再继续分解。adhoc(P)是该分治法中的基本子算法，用于直接解小规模的问题 P。当 P 的规模不超过 n0 时，直接算法 adhoc(P)求解。算法 merge(y1,y2,...,yk)是该分治法中的合并子算法，用于将 P 的子问题 P1,P2,...,Pk 的解 y1,y2,...,yk 合并为 P 的解。

根据分治法的分割原则，应把原问题分为多少个子问题才比较适宜？每个子问题是否规模相同或怎样才为适当？这些问题很难给予肯定的回答。但人们从大量实践中发现，在用分治法设计算法时，最好使子问题的规模大致相同。即将一个问题分成大小相等的 k 个子问题的处理方法是行之有效的。许多问题可以取 k=2。这种使子问题规模大致相等的做法是出自一种平衡(banlancing)子问题的思想，它几乎总是比子问题规模不等的做法要好。

从分治法的一般设计模式可以看出，用它设计出的算法一般是递归算法。因此，分治法的计算效率通常可以用递归方程来进行分析。一个分治法将规模为 n 的问题分成 m 个规模为 n/m 的子问题，其中 k(k<=m)个子问题需要求解。为方便起见，设分解阀值 n0=1，且 adhoc 解规模为 1 的问题耗费 1 个单位时间。另外再设将原问题分解为 k 个问题以及用 merge 将 k 个子问题的解合并为原问题的解需用 f(n)个单位时间。