一、判断题：20分（10题，2分/每题）

二、填空题：12分（8空，1.5分/每空）

三、单选题：32分（16题， 2分/每题）

四、完善各算法代码：20分（4题，5分/每题）

五、分析题：16分（2题，8分/每题）

1. 判断题（20分，2分/每题）/填空题（12分，1.5/每空）/单选题(32分， 2分/每题）
2. 顺序存储方式能用于存什么结构的数据？

**顺序存储结构**

**所有元素存放在一片连续的存储单元中，逻辑上相邻的元素存放到计算机内存仍然相邻**

**链式存储结构**

**所有元素存放在可以不连续的存储单元中，但元素之间的关系可以通过地址确定，逻辑上相邻的元素存放到计算机内存后不一定是相邻的**

1. 数据结构是研究什么的学科？数据结构处理的最小单位是什么？

**数据结构是一门研究“非数值计算的程序设计问题中”计算机的操作对象 +关系 + 操作等的学科。**

**数据项（ Data Item ）是数据结构中处理的最小单位。**

1. 什么叫二叉树？有什么特点？ “左孩子- 右兄弟”方式在树结构中的意义是什么？

**二叉树是一个由有限结点构成的集合，它可能为空或者包含一个根节点和两棵不相交的二叉树，这两棵子树分别称为它的左子树和右子树。**

**二叉树与树的区别：树不能为空，二叉树可以；在树中不关心孩子的顺序，二叉树区分孩子的顺序。**

1. 什么叫算法的复杂度？它包括什么内容？复杂度如何表示？

**算法是为了解决某类问题而规定的一个有限长的操作序列**

**时间特性(时间复杂度T(n) )**

**空间特性(空间复杂度S(n) )**

1. 数据结构中什么是“逻辑结构”与“物理结构”，两者间的关系是什么？

**逻辑结构：是对数据元素之间的逻辑关系S的描述。它可以用一个数据元素的集合D和定义在此集合上的若干关系S来表示;**

**物理结构：是逻辑结构在计算机中的表示和实现，故又称“存储结构”。即：逻辑结构在存储器中的映象**

1. 最小代价生成树中的“克鲁斯卡尔”、“普里姆”和“索林”的贪心算法思想是什么？

**克鲁斯卡尔算法（适用于既密也可疏的网络）：通过每次向T中添加一条边来构造最小代价生成树T。算法按照边的代价以非减的顺序来选择包含在T中的边。**

**普里姆算法（适用于稠密的网络）：逐条边地构造最小代价生成树。但是在算法执行的整个过程中，已选择的边的集合始终形成一棵树。**

**索林算法：索林算法在每个阶段都选择若干条边。在一个阶段开始时，已选择的边和全部n个顶点一起构成了一个森林。**

1. 在什么情况下才能得到一个惟一的最小生成树？

**若连通图上各权值均不相同，则该图的最小生成树是惟一的**

**最小代价原则如下：**

**必须使用图中的边**

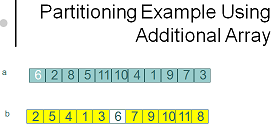
**必须刚好使用n-1条边**

**不可使用产生环的边**

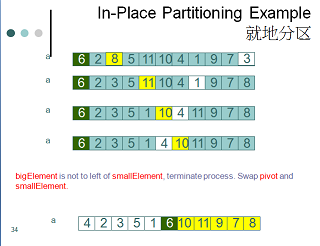
1. 快速排序法的几种策略是什么？知道他们的区别吗？最坏情况与平均情况下的时间复杂度是什么？

**Median-of-Three rule. From the leftmost, middle, and rightmost elements of the list to be sorted, select the one with median key as the pivot.**

**附加数组分区**



**In-Place Partitioning Example  
就地分区**



**快速分区**



1. “栈”与“队列”是两种运算受限的线性表，他们的区别是什么？

**栈（stack）是指只允许在(表的末端)进行插入和删除的线性表。**

**栈又叫做*后进先出（LIFO）*的线性表**。

**队列是只允许在表的一端插入，在另一端删除的线性表。**

**队列又叫*先进先出（FIFO）*的线性表**

**当front= =rear，队列为空；**

**当rear= =maxSize，队列满**

**回答前面提出的为什么重定义front变量的问题的理由1：**

**为了统一操作。**

1. 一棵满二叉树的结点及高度如何计算？当h=n时是一棵怎么样的二叉树？

**2^k-1 h=log(n+1) 偏斜树**

1. 二叉树可以用几种方式存储？他们叫什么名字？

**顺序存储和链式存储**

1. 二叉查找树的递归算法是如何查找一个关键k的？

**1.从根开始检索，如果根为0，那么树中不含任何元素，查找失败。**

**否则我们将k和根的键值进行比较。如果k小于根的键值，那么只需要查找左子树。**

**K大于根的键值，查找右子树。K等于根的键值，则查找成功**。

**2.二叉查找树是一棵二叉树。它可以为空。如果不为空，则它满足如下性质：**

**每一个元素有一个键值，不同元素的键值不同。**

**位于左子树的键值小于根结点的键值。**

**位于右子树的键值大于根结点的键值。**

**左右子树都是二叉查找树。**

1. 为什么要采用循环队列？它与线性队列相比有什么异同？

**采用循环队列，在最坏的情况下添加和删除元素时间复杂度就变成了O(1)。与线性队列相比，两者都是在队首删除元素，在队尾插入元素，不同的是，循环队列front指向元素沿逆时针方向的前一个，队列为满时，front=(rear+1)%capacity。**

1. 哈希算法的思想是什么？它能在O(1) 时间复杂度内实现关键字的查找吗？为什么？

**哈希函数的构造原则：既容易计算又能将冲突数目最小化。**

**哈希函数将一个关键字映射为哈希表中的一个桶的地址。能**

1. 哈希算法分为静态散列与动态散列，静态散列可以将多个关键字映射到一个定长区间？

**可以。**

1. 什么是装载密度与关键字密度？

**关键字密度：n/t，n是哈希表中记录的数目，t是所有可能的关键字数目。**

**装填密度：n/（sb），s->槽数，b->桶的个数。**

1. 如何解决哈希算法中的“冲突”与“溢出”？

**最常用的溢出处理方法有两种：开放定址法和链地址法。**

**开放定址法4种：线性探测法；二次探测法；再散列法；随机探测法。**

**二次探测与再哈希法是限制关键字聚集的有效方法**

1. 能说出树的离散集合中“并”的策略吗？

**加法权则，折叠法则。**

1. 线性表的不同映射方式是什么？及这些方式带来操作的异同?

**数组，栈，队列，链表。都可用来存储数据，并对数据进行插入和删除操作。不同点:数组执 行插入和删除操作时，需大量移动元素，预先分配空间，后三个不需要。**

1. 线性队列与循环队列的异同？为什么用循环队列？使用循环队列时应注意什么问题？与第13题同。
2. **各部份的知识点例题**
3. 只能从线性表一端进行操作的数据结构称为 （栈结构）。
4. 多关键字排序是基于关键字的重要程度来进行排序。（Y）
5. 二叉树的中序遍历是指：先访问父结点的右子树，然后访问该父结点，再访问其左子树。 （N）
6. 除头尾位置的元素外，每个元素有且仅有一个前驱，有且仅有一个后继元素的数据结构称为 （ 线性表 ）。
7. 完全二叉树一定存在度为1的结点 。（N）
8. 排序算法中的比较次数与初始元素序列的排列无关。（N）
9. 若连通图上各权值均不相同，则该图的最小生成树是惟一的。（Y）
10. 循环链表不是线性表。（N）
11. 一棵结点数为n、高度h=n的二叉树一定是一棵（偏斜树 ）。
12. 图的广度优搜索BFS算法采用了( 队列 )的数据结构来实现。
13. 链式存储结构克服了顺序存储结构的（ 插入、删除不需移动元素 ），（ 不需预先分配空间 ），表容量只受限于内存空间三个弱点。
14. 一个完全有向图最多有 ( n\*(n-1) )条边。
15. 散列函数是从关键字集合到 （ 桶地址集合 ）的映象。
16. 数据结构是一门主要研究数值计算问题的学科，如，预测人口增长、人员信息管理等都属于数值计算问题。 （N）
17. 文件型病毒的传播问题在数据结构学科中是属于图的问题。 （N） （树的遍历问题）
18. 线性表采用顺序存储，便于进行插入和删除操作。 （N）
19. 已知一算术表达式的中缀形式为 A+B\*C-D/E，后缀形式为ABC\*+DE/-。 （Y）
20. 只有一个结点的二叉树的度为0 。 （Y）
21. 二叉树的度为2 或者是0，1 。 （Y）
22. 二叉树的左右子树可任意交换。 （N）
23. 深度为K的完全二叉树的结点个数小于或等于深度相同的满二叉树。 （Y）
24. 设无向图的顶点个数为n，则该图最多有（ n(n-1)/2 ）条边。
25. 顺序存储设计时，存储单元的地址（一定连续 ）。
26. 散列运算是一种能够在（ O(1) ）预期的时间内完成对字典的获取、插入或删除操作技术。
27. 动态散列函数h（k,p）中的p表示（目录深度），表示空间增长按幂规律增长。
28. 动态散列函数h（k, r+q）中的r+q表示（空间按幂规律+线性因子增长度）。
29. 最大容量为n的循环队列，队尾指针是rear，队头是front，则队列空的条件是 （ rear=front=0 ）。
30. 如想要将两个有序序列汇总为一个有序序列，应采用下列算法中的（ 归并 ）排序法。
31. 一个多元多项式可以用一个广义链表来表示，其广义表的深度由多元多项式的变元个数来确定。 （Y）
32. 要连通具有n个顶点的无向图，至少需要（ n-1 ）条边。
33. 有向图的邻接矩阵一定是对称矩阵。 （N）
34. 一个有n个结点的图，最多有（ n ）个连通分量。

**二、完善各算法代码** （20分，4题，5分/每题）

完善一段链式栈/链式队列/链式稀疏矩阵/链式多项式相加算法

完善一段中序遍历算法/先序遍历算法/后序遍历/BST的查找算法

**Preorder Traversal VLR先序遍历**

**template** <**class** T>

**void** PreOrder(TreeNode<T> \*t)

{

**if** (t != **NULL**)

{

**Visit(t);**

PreOrder(t->leftChild);

PreOrder(t->rightChild);

}

}

**Recursive Searching a BST**

**递归查找实现**

**emplate <class** *K*, **class** *E***> //** Driver

***pair*<*K*, *E*>\* *BST*<*K*, *E*> :: *Get*(const *K& k*)**

**{ //** Search the binary search tree (**\*this**) for a pair with key *k*.

**//** If search a pair is found, return a pointer to this pair; otherwise, return 0.

**return** ***Get*(*root*, *k*);**

**}**

**template <class** *K*, **class** *E***> //** Workhorse

*pair***<***K*,*E***>**\* *BST***<***K*,*E***>** **:: *Get* (*TreeNode* <*pair* <*K*, *E*> >\* *p*, const *K& k*)**

**{**

**if (!*p*) return 0;**

**if (k < *p→data.first*) return *Get* (*p→leftChild, k*);**

**if (k > *p→data.first*) return *Get (p→rightChild, k);***

**return *&p→data*;**

**}**

**Iterative Searching a BST  
迭代查找实现template <class** *K*, **class** *E***> //** Iterative version

*pair***<***K*,*E***>**\* *BST***<***K*,*E***>** :: *Get*(**const** *K& k***)**

**{**

*TreeNode* **<** *pair***<***K*,*E***> >** \**currentNode* = *root***;**

**while (*Nocurrentde*)** **{**

**if** (*k* < *currentNode →data.first*)

***currentNode* = *currentNode →leftChild*;**

**else if** (*k* > *currentNode →data.first*)

***currentNode* = *currentNode →rightChild*;**

**else return** & *currentNode →data***;**

**}**

**//** no matching pair

**return** 0**;**

**}**

完善一段广度优先算法/深度优先算法/克鲁斯卡尔算法/普里姆算法/索林算法/多源多目标算法/单源多目标算法

**单源多目标算法**

**void Graph::ShortestPath(const int n, const int v)**

{ for (int i = 0; i < n; i++) {s[i] = FALSE; dist[i] = length[v][i];} // initialize

s[v] = TRUE;

dist[v] = 0;

Out(n);

for (int i = 0; i < n-2; i++) { // determine n-1 paths from vertex v

int u **= choose(n);** // choose returns a value u:

// dist[u] = minimum dist[w] where s[w] = FALSE

s[u] = TRUE;

for (int w = 0; w < n; w++)

if (! s[w])

if (dist[u] + length[u][w] < dist[w])

**dist[w] = dist[u] + length[u][w];**

Out(n);

} // end of for (i = 0; ...)

}

完善一段堆排序/插入排序/多关键字排序/链和列表排序/归并排序/快速排序

**快速排序**

**template <class** *T***>**

**void** *QuickSort*(*T* \**a*, **const int** *left*, **const int** *right*)

**{** // Sort *a*[*left*:*right*] into nondecreasing order.

// *a*[*left*] is arbitrarily chosen as the pivot. Variables *i* and *j*

// are used to partition the subarray so that at any time *a*[*m*] ≤ *pivot*, *m* < *i,*

// and *a*[*m*] ≥ *pivot*, *m* > *j*. It is assumed that *a*[*left*] ≤ *a*[*right* + 1].

**if** (*left* < *right*) **{**

**int** *i* **=** *left*,

*j* = *right* + 1,

*pivot* = *a*[*left*]**;**

**do {**

**do** *i*++**; while** (*a*[*i*] < *pivot*)**;**

**do** *j*--**; while** (*a*[*j*] > *pivot*)**;**

**if** (*i* < *j*) *swap* (*a*[*i*], *a*[*j*])**;**

**} while** (*i* < *j*)**;**

*swap* (*a*[*left*], *a*[*j*])**;**

*QuickSort*(*a*, *left*, *j* - 1)**;**

*QuickSort*(*a*, *j* + 1, *right*)**;**

**}**

**}**

**堆排序**

**template** <**class** *T*>

**void** *Adjust*(*T* \**a*, **const int** *root*, **const int** *n*)

**{**// Adjust binary tree with root *root* to satisfy heap property. The left and right

//subtrees of *root* already satisfy the heap property. No node index is > *n*.

*T e* = *a*[*root*]**;**

// find proper place for *e*

**for** (**int** *j* =2\**root***;** *j* <= *n***;** *j* \*=2) **{**

**if** (*j* < *n* && *a*[*j*] < *a*[*j*+1]) *j*++**;** // *j* is max child of its parent

**if** (*e* >= *a*[*j*]) **break;** // *e* may be inserted as parent of *j*

*a*[*j* / 2] = *a*[*j*]**;** // move *j*th record up its parent node

**}**

*a*[*j* / 2] = *e***;**

**template** <**class** *T*>

**void** *HeapSort*(*T* \**a*, **const int** *n*)

**{**// Sort *a*[1:*n*] into nondecreasing order.

**for** (**int** *i* = *n*/2**;** *i* >= 1**;** *i*--) // heapify

*Adjust*(*a*, *i*, *n*)**;**

**for** (*i* = *n*-1; *i* >= 1; *i*--) // sort

**{**

*swap*(*a*[1], *a*[*i*+1])**;**// swap first and last of current heap

*Adjust*(*a*, 1, *i*); // heapify

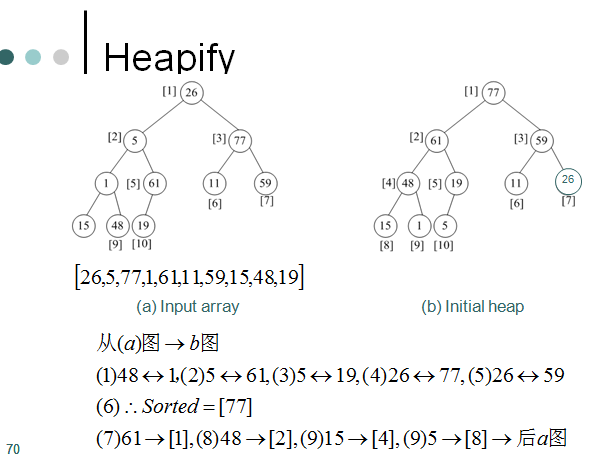
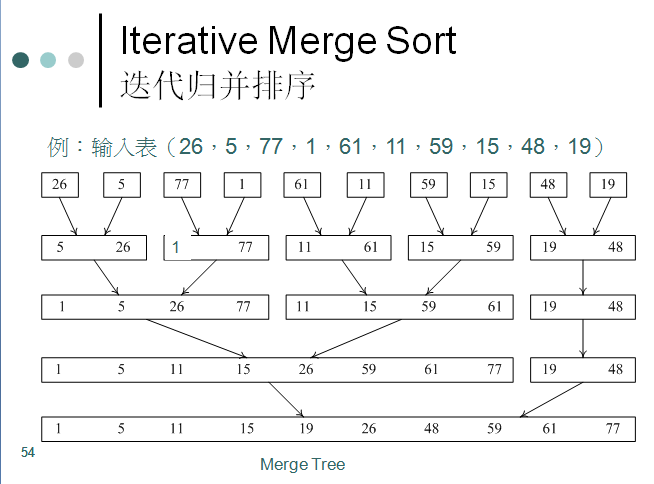
**}**

**}**

**}**

**三、分析题（**16分，2题，8分/每题） ）

用图表达排序过程：堆排序/插入排序/多关键字排序/链和列表排序/归并排序/快速排序



重构一棵二叉树方法：中序与后序/中序与先序/中序与层序