1.程序 for(i=0;i<n;i=i*2)　　的时间复杂度是（　d　）

for (j=0;j<n;j=j+2)

a=a+i;

A. O(2n)　　B. O(n^2)　　C. O(logn)　　D. O(nlogn)

2.属于逻辑结构的是（　c　）

A. 顺序表　　B. 哈希表　　C.有序表　　D. 单链表

有序表中所有元素以递增或递减方式排列，对数据之间的关系进行了描述，是一种逻辑结构。而 顺序表（是指用一组地址连续的存储单元依次存储数据元素的线性结构。），哈希表（用散列法存储的线性表叫散列表），单链表（用一组地址任意的存储单元存放线性表中的数据元素），均只是一种存取结构，不是逻辑结构。

3.在双向链表指针 p 的结点前插入一个指针 q 的结点操作是（　c　）

A.　p->Llink=q;q->Rlink=p;p->Llink->Rlink=q;q->Llink=q;

B.　p->Llink=q;p->Llink->Rlink=q;q->Rlink=p;q->Llink=p->Llink;

C.　q->Rlink=p;q->Llink=p->Llink;p->Llink->Rlink=q;p->Llink=q;

先将 q 的左右指针链接好，再链接其他的

D.　q->Llink=p->Llink;q->Rlink=q;p->Llink=q;p->Llink=q;

4.对于顺序存储的线性表，访问结点和增加、删除结点的时间复杂度为（　c　）。

A. O(n)　O(n)　　B. O(n)　O(1)　　C. O(1)　O(n)　　D. O(1)　O(1)

顺序存储是指用物理上相邻的单元存储线性表的元素，简单的说就是可以用数组实现。

5. 设 abcdef 以所给的次序进栈，若在进栈操作时，允许退栈操作,则下面得不到的序列为（ d ）。

A. fedcba　　B. bcafed　　C. dcefba　　D. cabdef

如果指定元素的进栈顺序，那么它的出栈顺序就有一个特点，那就是越往后的元素如果先出栈的话，那么其前面的元素出栈顺序就不可能与进栈顺序相同

6. 表达式 a*(b+c)-d 的后缀表达式是（ b ）。

A. abcd*+-　　B. abc+*d-　　C. abc*+d-　　D. -+*abcd

7.用链接方式存储的队列，在进行删除运算时（ d ）。

A. 仅修改头指针　　B. 仅修改尾指针

C. 头、尾指针都要修改　　D. 头、尾指针可能都要修改

本题考查对链式存储队列的删除操作。题目要求对队列进行删除运算，那么在队首进行操作，由于是链式存储，删除结点后，需要修改队首的指针，使其指向下一个结点。但如果队列中只有这一个结点，这时候头、尾指针都指向这个结点，在删除结点后，头、尾指针都需要修改.

队列：只允许在表的一端进行插入，而在另一端删除元素的线性表。在队列中，允许插入的一端叫队尾（rear），允许删除的一端称为对头(front)。

链队列中，有两个分别指示队头和队尾的指针。

8. 设 S 为一个长度为 n 的字符串，其中的字符各不相同，则 S 中的互异的非平凡子串（非空且不同于 S 本身）的个数为（ d ）。

A. 2n-1　　B. n2　　C. (n2/2)+(n/2)　　D. (n2/2)+(n/2)-1

长度为 n-1 的不同子串个数为 2,长度为 n-2 的不同子串个数为 3..,长度为 1 的不同子串个数是 n，总和即为非空且不同于 S 本身的集合个数

9. 假设以行序为主序存储二维数组 A=array[1..100，1..100]，设每个数据元素占 2 个存储单元，基地址为 10，则 LOC[5，5]=（　b　）。

A. 808　　B. 818　　C. 1010　　D. 1020

按行优先存储就是把二维数组中的数据一行一行地顺次存入存储单元。二维数组 A[1..m，1..n]若按行优先存储，那么 A 的任意一个元素 A[i][j]的存储首地址 Loc(i，j)可由下式确定：Loc(i,j)=Loc(1，1)+[n×(i-1)+j-1]×b，其中，Loc(1，1)是第一个元素 A[1][1]的首地址，b 是每个元素占用的存储单元个数。代入数据便得 Loc(5，5)=10+[100×(5-1)+5-1]×2=818。注意：这是 100 行 100 列

10. 将一个 A[1..100,1..100]的三对角矩阵,按行优先存入一维数组 B[1‥298]中，A 中元素 $A_{66,65}$（即该元素下标 i=66，j=65），在 B 数组中的位置 K 为（　b　）。

A. 198　　B. 195　　C. 197　　D. 200

以按行为主序的原则转存为一维数组 M[k]中，则 A[i,j]的对应关系为

k=2*i+j-2.(i,j,k 均从 1 开始)

11. 已知广义表 L=((x,y,z)，a，(u，t，w))，从 L 表中取出原子项 t 的运算是（　d　）。

A. head (tail (tail (L))　　B. tail (head (head (tail (L))))

C. head (tail (head (tail (L))))　　D. head (tail(head (tail (tail (L)))))

getTail(L)得到的是(a,(u,t,w))

getTail(getTail(L))t 得到的就是((u,t,w))

getHead(getTail(getTail(L)))得到的就是(u,t,w)

getTail(getHead(getTail(getTail(L))))得到的就是((t,w))

getHead(getTail(getHead(getTail(getTail(L)))))得到的就是(t,w)注意这两行

getHead(getHead(getTail(getHead(getTail(getTail(L))))))得到的就是 t.

这里要注意的是,getHead得到的是一个原子,而getTail得到的却是原子外组成的新的广义表,不管是只有一个元素,但也是一个广义表,而不是直接的元素.

15. 用 DFS 遍历一个无环有向图，并在 DFS 算法退栈返回时打印相应的顶点，则输出的顶点序列是(a)。

A. 逆拓扑有序　　B. 拓扑有序　　C. 无序的

16. 若一个有向图的邻接距阵中,主对角线以下的元素均为零,则该图的拓扑有序序列（ a ）。

A. 存在　B. 不存在

只存在<ai,aj>（i<j）这种边 故无回路

17.既希望较快的查找又便于线性表动态变化的查找方法是（ d ）

A. 顺序查找　B. 折半查找　C. 索引顺序查找　D. 哈希法查找

应该是散列法~~散列法的算法代表是哈希表，通过哈希函数将值转化成存放该值的目标地址~~这种查找的性能是 O（1），对于其动态变化要求，可以进行再次散列，时间复杂度是 O(1)~~

二分法是基于顺序表的一种查找方式，体现的是折半思想，查找的时间复杂度为 O(logn)，不过要是动态变化的情况，移动次数还是 O(n)，所以不适合要求顺序法是挨个查找，这种方法最容易实现，不过查找时间复杂度都是 O(n)，动态变化时可将保存值放入线性表尾部，则时间复杂度为 O(1)，所以不满足要求分块法应该是将整个线性表分成若干块进行保存，若动态变化则可以添加在表的尾部（非顺序结构），时间复杂度是 O(1)，查找复杂度为 O(n)；若每个表内部为顺序结构，则可用二分法将查找时间复杂度降至 O(logn)，但同时动态变化复杂度则变成 O(n)

18. 将 10 个元素散列到 100000 个单元的哈希表中，则（ c ）产生冲突。

A. 一定会　　B. 一定不会　　C. 仍可能会

19. 稳定的排序方法是（ b ）

A. 直接插入排序和快速排序　　B. 折半插入排序和起泡排序

C. 简单选择排序和四路归并排序　　D. 树形选择排序和 shell 排序

除了快速排序和简单选择排序，其他都稳定；树形选择排序又称锦标赛排序；shell 排序又称希尔排序，，Shell 排序是一种不稳定的排序算法

20. 对下列四种排序方法，在排序中关键字比较次数同记录初始排列无关的是（ b ）。

A. 直接插入　　B. 二分法插入　　C. 快速排序　　D. 归并排序

记录的初始排序次序　就是 排序前的状态　1 4 5 6 7 8 9 2 3 这就是初始排序次序。

关键字的比较次数，就是比较的次数，例如从大到小排序，你得进行比较

直接插入排序(straight insertion sort)的做法是：

当插入第 i （i　　1）个对象时，前面的 V[0]，V[1]，…，V[i-1]已经排好序。这时，用 V[i]的排序码与 V[i-1]，V[i-2]，…的排序码顺序进行比较，找到插入位置即将 V[i]插入，原来位置上的对象向后顺移。直接插入排序属于稳定的排序，最坏时间复杂性为 O(n^2)，空间复杂度为 O(1)。

二分法插入:在插入第 i 个元素时，对前面的 0～i-1 元素进行折半，先跟他们中间的那个元素比，如果小，则对前半再进行折半，否则对后半进行折半，直到 left>right，然后再把第 i 个元素前 1 位与目标位置之间的所有元素后移，再把第 i 个元素放在目标位置上。

最好的情况是当插入的位置刚好是二分位置 所用时间为 O(n)；

最坏的情况是当插入的位置不在二分位置 所需比较次数为平均时间 O(n^2)

快速排序：是任取待排序对象序列中的某个对象 (例如取第一个对象) 作为基准, 按照该对象的排序码大小,将整个对象序列划分为左右两个子序列：

左侧子序列中所有对象的排序码都小于或等于基准对象的排序码

右侧子序列中所有对象的排序码都大于基准对象的排序码

归并排序：即先使每个子序列有序，再使子序列段间有序。若将两个有序表合并成一个有序表，称为二路归并。

二、程序填空题（每小题 6 分，共 12 分）

1.下面程序是通过中序遍历建立中序线索化二叉树算法，请填写完整

```
void InThreading(BiThrTree p){
    if(p){
        __InThreading(p->lchild);_;/左子树线索化
        if(!p->lchild) {p->LTag=Thread; p->lchild=pre;} /建立前驱线索


        if(_!pre->rchild_) {pre->RTag=Thread; pre->rchild=p;}
        ______pre=p_______; //建立后继线索保持 pre 为下一结点的前驱
        InThreading(p->rchild);
    }
}
```

2.下面是稀疏矩阵的快速转置算法，请补充完整

```
Status FastTransposeSMatrix(TSMatrix M, TSMatrix &T){
    T.mu=M.nu; T.nu=M.mu; T.tu=M.tu;
    if(T.tu){
        for(col=1;col<=M.nu;++col) num[col]=0;
        for(t=1;t<=M.tu;++t) __++num[M.data[t].j];___;
        cpot[1]=1;
        for(col=2;col<=M.nu;++col) __cpot[col]=cpot[col-1]+num[col-1];___;
        for(p=1;p<=M.tu;++p){
            col=M.data[p].j; q=cpot[col];
            T.data[q].i=M.data[p].j;
```

```
            T.data[q].j=M.data[p].i;
            T.data[q].e=M.data[p].e; ++cpot[col];
        }//for
    }//if
    return OK;
```

```
Status FastTransposeSMatrix(TSMatrix M, TSMatrix &T){
    T.mu=M.nu; T.nu=M.mu; T.tu=M.tu;
    if(T.tu){
        for(col=0;col<M.nu;++col) num[col]=0;//初始化num
        for(t=0;t<M.tu;++t) ++num[M.data[t].j];//求M中每列非零元个数
        cpot[0]=0;
        for(col=1;col<M.nu;++col) cpot[col]=cpot[col-1]+num[col-1];
    //求第col列中第一个非零元在T中的序号
        for(p=0;p<M.tu;++p){
            col=M.data[p].j; q=cpot[col];
            T.data[q].i=M.data[p].j;
            T.data[q].j=M.data[p].i;
            T.data[q].e=M.data[p].e;++cpot[col];//该列下一元素位置
        }//for
    }//if
    return OK;
}//FastTransposeSMatrix
```

必看题：

性质

性质1 在二叉树的第 i 层上至多有 2^{i-1} 个结点。($i \geq 1$) [证明用归纳法]

证明：当i=1时，只有根结点，$2^{i-1}=2^0=1$。

假设对所有j，i>j≥1，命题成立，即第j层上至多有2^{j-1} 个结点。

由归纳假设第i-1 层上至多有 2^{i-2}个结点。

由于二叉树的每个结点的度至多为2，故在第i层上的最大结点数为第i-1层上的最大结点数的2倍，即$2* 2^{i-2}= 2^{i-1}$。

性质2 深度为 k 的二叉树至多有 2^k-1 个结点($k \geq 1$)。

证明：由性质1可见，深度为k的二叉树的最大结点数为

$$\sum_{i=1}^{k} (\text{第 } i \text{ 层上的最大结点数})$$

$$= \sum_{i=1}^{k} 2^{i-1} = 2^0 + 2^1 + \ldots + 2^{k-1} = 2^k - 1$$

性质3　　对任何一棵二叉树T, 如果其叶结点数为 n_0, 度为2的结点数为 n_2,则$n_0=n_2+1$.

证明：若度为1的结点有 n_1 个，总结点个数为 n，总边数为 e，则根据二叉树的定义，利用总边数=度算（树从上往下看）=总结点算（树从下往上看)

$n = n_0 + n_1 + n_2$　　$e = 2n_2 + n_1 = n - 1$

因此，有 $2n_2 + n_1 = n_0 + n_1 + n_2 - 1$

$n_2 = n_0 - 1$　　=　　$n_0 = n_2 + 1$

2. 用希尔排序对数组{98，36，-9，0，47，23，1，8，10，7}进行排序，写出排序过程

答：

第一趟：gap=5　　23 1 -9 0 7 98 36 8 10 47

第二趟：gap=2　　-9, 0, 7, 1, 10, 8, 23, 47, 36, 98

第三趟：gap=1　　-9, 0, 1, 7, 8, 10, 23, 36, 47, 98

- 设待排序对象序列有 n 个对象, 首先取一个整数 gap < n 作为间隔, 将全部对象分为 gap 个子序列, 所有距离为 gap 的对象放在同一个子序列中, 在每一个子序列中分别施行直接插入排序。然后缩小间隔 gap, 例如取 gap =⌈gap/2⌉, 重复上述的子序列划分和排序工作。直到最后取 gap == 1, 将所有对象放在同一个序列中排序为止。

33

3. 用堆排序对数组{3，87，12，61，70，97，26，45}进行排序，写出排序过程

答：建堆：3，87，12，61，70，97，26，45

3，87，97，61，70，12，26，45

97，87，26，61，70，12，3，45

排序：45，87，26，61，70，12，3

87，70，26，61，45，12，3

3，70，26，61，45，12

70，61，26，3，45，12
12，61，26，3，45
61，45，26，3，12
12，45，26，3
45，12，26，3
3，12，26
23，12，3
3，12
12，3

3

堆是一种经过排序的完全二叉树，其中任一非终端节点的数据值均不大于（或不小于）其左孩子和右孩子节点的值。

最大堆和最小堆是二叉堆的两种形式。

最大堆：根结点的键值是所有堆结点键值中最大者。

最小堆：根结点的键值是所有堆结点键值中最小者。

。由上述性质可知大顶堆的堆顶的关键字肯定是所有关键字中最大的，小顶堆的堆顶的关键字是所有关键字中最小的。

堆排序分为两个步骤

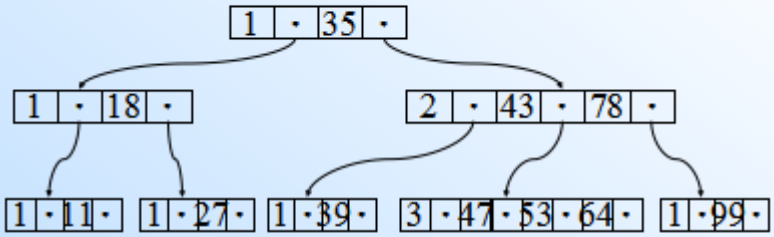
根据初始输入数据，利用堆的调整算法　HeapAdjust( ) 形成初始堆;

通过一系列的对象交换和重新调整堆进行排序。

平衡二叉树：

比左子树大，比右子树小

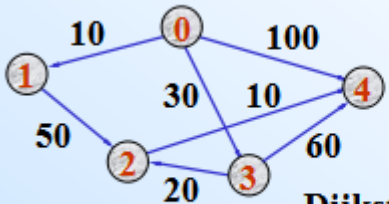
B



4阶B_树，每个节点最多4个指针3个关键字

一棵m阶B-树每个节点最多有m棵子树m-1个关键字,最少有⌈m/2⌉棵子树⌈m/2⌉-1个关键字

88

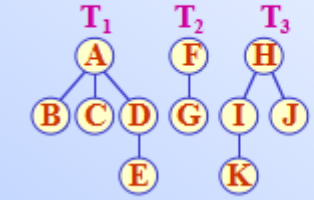


Dijkstra逐步求解的过程

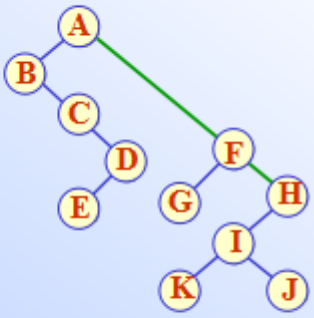
源点	终点	最短路径	路径长度
v_0	v_1	(v_0,v_1)	10
	v_2	—— (v_0,v_1,v_2) (v_0,v_3,v_2)	∞,60,50
	v_3	(v_0,v_3)	30
	v_4	(v_0,v_4) (v_0,v_3,v_4) (v_0,v_3,v_2,v_4)	100,90,60

94

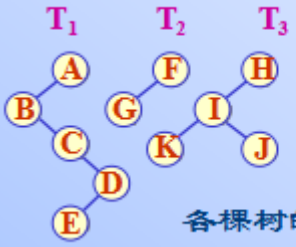
❑森林与二叉树的转换



3 棵树的森林



森林的二叉树表示



各棵树的二叉树表示

KMP算法：

```
int Index_KMP(SString S, SString T, int pos) {
   i=pos; j=1;
   while(i<=S[0] && j<=T[0]) {
        if (j==0 || S[i]==T[j]) {++i; ++j}
        else j=next[j];
   }
   if (j>T[0]) return i-T[0];
   else return 0;
}//Index_KMP
```

最小生成树（minimum cost spanning tree）

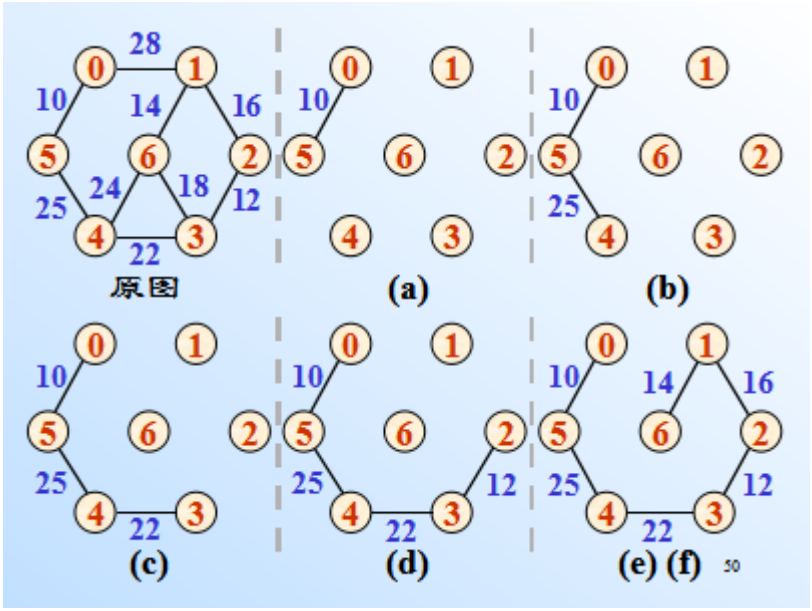
- 使用不同的遍历图的方法，可以得到不同的生成树；从不同的顶点出发，也可能得到不同的生成树。
- 按照生成树的定义，n 个顶点的连通网络的生成树有 n 个顶点、n-1 条边。
- 构造最小生成树的准则
  - 必须使用且仅使用该网络中的n−1 条边来联结网络中的 n 个顶点；
  - 不能使用产生回路的边；
  - 各边上的权值的总和达到最小。

48

普里姆(Prim)算法

- 普里姆算法的基本思想：
  从连通网络 N = { V, E }中的某一顶点 u_0 出发, 选择与它关联的具有最小权值的边 （u_0, v), 将其顶点加入到生成树顶点集合U中。
  以后每一步从一个顶点在 U 中,而另一个顶点不在 U 中的各条边中选择权值最小的边(u, v), 把它的顶点加入到集合 U 中。如此继续下去, 直到网络中的所有顶点都加入到生成树顶点集合 U 中为止。
- 采用邻接矩阵作为图的存储表示。

49


28
10
14
16
24
18
12
25
22
原图
(a)
(b)
(c)
(d)
(e) (f)