一．选择题（30 分，每小题 2 分）

1. 下列不影响算法时间复杂性的因素有（　　）。

A：问题的规模　　B：输入值

C：计算结果　　D：算法的策略

2. 链表不具有的特点是（　　）。

A: 可随机访问任意元素　　B: 插入和删除不需要移动元素

C: 不必事先估计存储空间　　D: 所需空间与线性表长度成正比

3. 设有三个元素 X，Y，Z 顺序进栈（进的过程中允许出栈），下列得不到的出栈排列是（　　）。

A：XYZ　　B：YZX　　C：ZXY　　D：ZYX

4. 判定一个无序表 Q（链表实现）为空的条件是（　　）。

A：Q.head == None　　B：　Q == None

C：Q.head == 0　　D：　Q.head != None

5. 若定义二叉树中根结点的层数为零，树的高度等于其结点的最大层数加一。则当某二叉树的前序序列和后序序列正好相反，则该二叉树一定是（　）的二叉树。

A：　空或只有一个结点　　B：高度等于其节点数

C：　任一结点无左孩子　　D：　任一结点无右孩子

6. 任意一棵二叉树中，所有叶结点在前序、中序和后序周游序列中的相对次序（　　）。

A：　发生改变　　B：　不发生改变

C：　不能确定　　D：　以上都不对

7. 假设线性表中每个元素有两个数据项 key1 和 key2，现对线性表按以下规则进行排序：先根据数据项 key1 的值进行非递减排序；在 key1 值相同的情况下，再根据数据项 key2 的值进行非递减排序。满足这种要求的排序方法是（　　）。

A: 先按 key1 值进行冒泡排序，再按 key2 值进行直接选择排序

B: 先按 key2 值进行冒泡排序，再按 key1 值进行直接选择排序

C: 先按 key1 值进行直接选择排序，再按 key2 值进行冒泡排序

D: 先按 key2 值进行直接选择排序，再按 key1 值进行冒泡排序

8. 有 n^2 个整数，找到其中最小整数需要比较次数至少为（　　）次。

A:n　　B: $\log_2 n$　　C:n^2-1　　D:n-1

9. n 个顶点的无向完全图的边数为（　　）。

A: n（n-1） B: n（n+1）

C: n（n-1）/2 D: n（n+1）/2

10. 设无向图 G=（V, E），和 G'=(V', E')，如果 G'是 G 的生成树，则下面说法中错误的是（ ）。

A：G'是 G 的子图 B： G'是 G 的连通分量

C：G'是 G 的极小连通子图且 V=V' D： G'是 G 的一个无环子图

11. 有一个散列表如下图所示，其散列函数为 h(key)=key mod 13，该散列表使用再散列函数 H2(Key)=Key MOD 3 解决碰撞，问从表中检索出关键码 38 需进行几次比较（ ）。

0	1	2	3	4	5	6	7	8	9	10	11	12
26	38			17			33		48			25

A： 1 B：2 C: 3 D： 4

12. 在一棵度为 3 的树中，度为 3 的节点个数为 2，度为 2 的节点个数为 1，则度为 0 的节点个数为（ ）。

A： 4 B：5 C： 6 D： 7

13. 由同一组关键字集合构造的各棵二叉排序树( )。

A： 其形态不一定相同，但平均查找长度相同

B：其形态不一定相同，平均查找长度也不一定相同

C： 其形态均相同，但平均查找长度不一定相同

D： 其形态均相同，平均查找长度也都相同

14. 允许表达式内多种括号混合嵌套，检查表达式中括号是否正确配对的算法，通常选用（ ）。

A： 栈 B：线性表 C： 队列 D： 二叉排序树

15. 在映射抽象数据类型（ADT Map）的不同实现方法中，适合对动态查找表进行高效率查找的组织结构是（ ）。

A： 有序表 B：堆排序 C： 二叉排序树 D： 快速排序

二．判断（10 分，每小题 1 分；对填写“Y”，错填写“N”）

1.（ ）考虑一个长度为 n 的顺序表中各个位置插入新元素的概率是相同的，则顺序表的插入算法平均时间复杂度为 O(n) 。

2.（ ）希尔排序算法的每一趟都要调用一次或多次直接插入排序算法，所以其效率比直接插入排序算法差。

3.（ ）直接插入排序、冒泡排序、希尔排序都是在数据正序的情况下比数据在逆序的情况下要快。

4.（ ）由于碰撞的发生，基于散列表的检索仍然需要进行关键码对比，并且关键码的比较次数

仅取决于选择的散列函数与处理碰撞的方法两个因素。

5.（　　）若有一个叶子结点是二叉树中某个子树的前序遍历结果序列的最后一个结点，则它一定是该子树的中序遍历结果序列的最后一个结点。

6.（　　）若某非空二叉树的先序序列和后序序列正好相同，则该二叉树只有一个根结点。

7.（　　）有 n 个节点的二叉排序树有多种，其中树高最小的二叉排序树是搜索效率最好。

8.（　　）强连通分量是有向图中的最小强连通子图。

9.（　　）用相邻接矩阵法存储一个图时，在不考虑压缩存储的情况下，所占用的存储空间大小只与图中结点个数有关，而与图的边数无关。

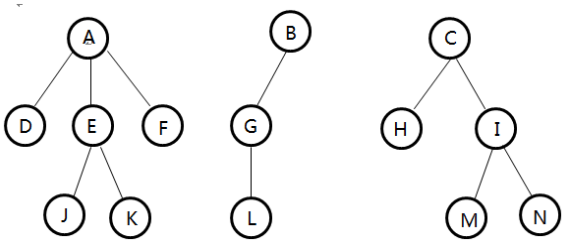
10.（　　）若定义一个有向图的根是指可以从这个结点可到达图中任意其他结点，则可知一个有向图中至少有一个根。

三．填空（20 分，每题 2 分）

1. 线性表的顺序存储与链式存储是两种常见存储形式；当表元素有序排序进行二分检索时，应采用 _____存储形式。
2. 如果只想得到 1000 个元素的序列中最小的前 5 个元素，在冒泡排序、快速排序、堆排序和归并排序中，哪种算法最快？_____
3. 设环形队列的容量为 20（单元编号从 0 到 19），现经过一系列的入队和出队运算后，队头变量（第一个元素的位置）front=18，队尾变量（待插入元素的位置）rear=11，在这种情况下，环形队列中有_____个元素。
4. 一棵含有 101 个结点的二叉树中有 36 个叶子结点，度为 2 的结点个数是 _____和度为 1 的结点个数是_____。
5. 已知二叉树的前序遍历结果（先根周游序列）为 ADC，这棵二叉树的树型有_____种可能。
6. 已知二叉树的中序序列为 DGBAECF，后序序列为 GDBEFCA，该二叉树的前序序列是_____。
7. 对于具有 57 个结点的完全二叉树，如果按层次自顶向下，同一层自左向右，顺序从 0 开始对全部结点进行编号，则有：编号为 18 的结点的父结点的编号是_____，编号为 19 的结点的右子女结点的编号是_____。
8. 有 n 个数据对象的二路归并排序中，每趟归并的时间复杂度为 _____。
9. 对一组记录进行降序排序，其关键码为（46，70，56，38，40，80，60，22），采用初始步长为 4 的希尔（shell）排序，第一趟扫描的结果是（_____）；而采用归并排序第一轮归并结果是（_____）。
10. 如果一个图节点多而边少（稀疏图），适宜采用邻接矩阵和邻接表中的_____方式进行存储。

四．简答（24 分，每小题 6 分）

1、树周游算法可以很好地应用到森林的周游上。查看下列森林结构，请给出其深度优先周游序列和广度优先周游序列。



2、哈夫曼树是进行编码的一种有效方式。设给定五个字符，其相应的权值分别为{4，8，6，9，18}，试画出相应的哈夫曼树，并计算它的带权外部路径长度 WPL 。

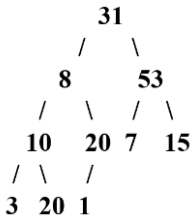
3、下图是一棵完全二叉树：

1）请根据初始建堆算法对该完全二叉树建堆，请画出构建的小根堆（2 分）；

2）基于（1）中得到的堆，删除其中的最小元素，请用图给出堆的调整过程（2 分）；

3）基于（1）中得到的堆，向其中插入元素 2，请给出堆的调整过程（2 分）。

注：每移动一个元素视为一个执行步骤，画出所有执行步骤。



4、已知图 G 的顶点集合 V={V0, V1, V2, V3, V4}，邻接矩阵如下图所示，可用 prim 算法求 G 的最小生成树。

$$\begin{vmatrix} 0 & 7 & \infty & 4 & 2 \\ 7 & 0 & 9 & 1 & 5 \\ \infty & 9 & 0 & 3 & \infty \\ 4 & 1 & 3 & 0 & 10 \\ 2 & 5 & \infty & 10 & 0 \end{vmatrix}$$

1）根据邻接矩阵，画出图 G（2 分）；

2）根据 prim 算法，求图 G 从顶点 V0 出发的最小生成树（2 分）；

3）用图表示出最小生成树每一步的生成过程（2 分）。

五、算法（16 分，每小题 8 份）

1. 已知下列 pre2post 函数的功能是根据一个满二叉树的前序遍历序列，求其后序遍历序列，请完成填空（假设序列长度不超过 32）。

```
# 返回先根序列 preorder[start:start+length]对应的后根序列
def pre2post(preorder, start, length):
    if length == 1:
        return _____________（1 分）
    else:
        length = _____________（2 分）
        left = pre2post(preorder, _____________（1 分））
        right = pre2post(preorder, _____________（2 分））
        root = _____________（2 分）
        return left + right + root


print(pre2post("ABC", 0, 3))  # 输出 BCA
print(pre2post("ABDECFG", 0, 7))  # 输出 DEBFGCA
```

2. 阅读下列程序，完成图的深度优先周游算法实现的迷宫探索。已知图采用邻接表表示，Graph 类和 Vertex 类基本定义如下：

```
class Graph:
    def __init__(self):
    def addVertex(self, key, label):  # 添加节点，id 为 key，附带数据 label
    def getVertex(self, key):  # 返回 id 为 key 的节点
    def __contains__(self, key):  # 判断 key 节点是否在图中
    def addEdge(self, f, t, cost=0):  # 添加从节点 id==f 到 id==t 的边
    def getVertices(self):  # 返回所有的节点 key
    def __iter__(self):  # 迭代每一个节点对象

class Vertex:
    def __init__(self, key, label=None):  # 缺省颜色为"white"
    def addNeighbor(self, nbr, weight=0):  # 添加到节点 nbr 的边
    def setColor(self, color):  # 设置节点颜色标记
    def getColor(self):  # 返回节点颜色标记
    def getConnections(self):  # 返回节点的所有邻接节点列表
    def getId(self):  # 返回节点的 id
    def getLabel(self):  # 返回节点的附带数据 label

mazelist = [
    "++++++++++++++++++++++",
    "+   +   ++ ++       +",
    "E     +     ++++++++++",
    "+ +    ++  ++++ +++ ++",
    "+ +   + + ++    +++  +",
    "+         ++  ++  + +",
    "+++++ + +      ++  + +",
    "+++++ +++  + +  ++   +",
    "+         + + S+ +  +",
    "+++++ +  + + +     + +",
    "++++++++++++++++++++++",
]

def mazeGraph(mlist, rows, cols):  # 从 mlist 创建图，迷宫有 rows 行 cols 列
    mGraph = Graph()
    vstart = None
    for row in range(rows):
        for col in range(cols):
            if mlist[row][col] != "+":
                mGraph.addVertex((row, col), mlist[row][col])
                if mlist[row][col] == "S":
                    vstart = _____（1 分）
```

```
    for v in mGraph:
        row, col = v.getId()
        for i in [(-1, 0), (1, 0), (0, -1), (0, +1)]:
            if (0 <= row + i[0] < rows) and (0 <= col + i[1] < cols):
                if (row + i[0], col + i[1]) in mGraph:
                    mGraph.addEdge(_____________（1 分）)

    return mGraph, vstart # 返回图对象，和开始节点

def searchMaze(path, vcurrent):  # 从 vcurrent 节点开始 DFS 搜索迷宫
    vcurrent.setColor('gray')
    path.append(vcurrent.getId())
    if vcurrent.getLabel() != "E":
        done = False
        for nbr in _____________（2 分）:
            if nbr.getColor() == "white":
                done = searchMaze(_____________（2 分）)
                if done:
                    break
        if not done:
            _____________（2 分）
            vcurrent.setColor("white")
    else:
        done = True
    return done # 返回是否成功找到通路

g, vstart = mazeGraph(mazelist, len(mazelist), len(mazelist[0]))
path = []
searchMaze(path, vstart)
print(path)
```