综合试卷六

一、填空题。(共 27 分,每空 1 分)

1. 基本的存储结构通常有两大类:_____和_____。

2. 若算法总耗时 T 与问题规模 n 的关系为 $T(n)=2n^2+3n+1$,则算法的时间复杂度可记为_____。

3. 数组 array[8][16][32]的元素按行优先方式存储,首地址为 1024H,每个元素占 4 字节,则元素 array[4][8][16]的地址为_____。

4. 设一组记录关键字序列为(95,60,45,70,25,57,49),以完全二叉树顺序方式存储,用筛选法构建大根堆,从关键码值为_____的结点开始调整。

5. 一个单链表,已知每个结点 Node 包含 data 和 next 两个成员,设指向结点的工作指针 q 和 p,q->next = =p;指针 s = new Node<int>,s->data = 10;若在 q 和 p 之间插入 s 所指结点,则执行_____和_____操作。

6. 设字符集权重表为{16,25,28,4,6,20,3,12,2},用于构建 huffman 树,若采用顺序存储,需定义_____结点的存储空间。

7. 设一棵二叉树的前序遍历序列为 *ABCDEF*,中序遍历序列为 *CBDAEF*,则该二叉树的后序遍历序列为_____。

8. 顺序表采用连续存储方式实现的,是一种_____存取结构,对表中任意结点存取操作的时间复杂度为_____;而查找链表中的结节,因为链表是一种顺序存取结构,需要从头指针起顺着链扫描才能得到,平均时间复杂度为_____。

9. 栈的进出原则是_____,仅允许在栈_____进行插入与删除操作;队列的进出原则是_____,仅允许在队_____插入元素,在队_____删除元素。

10. 设循环队列空间大小 size=1 024,front 指向队头元素的前一个位置,rear 指向队尾元素,若 front=150,rear=30,则队列中元素个数为_____。

11. 一棵高度为 h 的满 3 叉树有如下性质:第 h 层上的结点都是叶结点,其余各层上每个结点都有 3 棵非空子树;如果按层次自顶向下,同层自左向右,顺序从 1 开始对全部结点编号,试问:(1)各层结点个数是_____,(2)编号为 i 的结点的父结点(若存在)的编号为_____,(3)编号为 i 的结点的第 m 个孩子结点(若存在)的编号为_____,(4)叶子结点数 n_0 和分支结点 n_k 之间满足的关系是_____。

12. 对于 32 位计算机环境,若单链表中的数据类型为 int,则其存储密度为_____;而若为双链表,则存储密度为_____;若采用顺序表存储数据,则其存储密度为_____。

13. 含有 n 个顶点_____条边的图称为完全无向图;含有 n 个顶点、_____条边弧的

图称为完全有向图。

二、选择题。(18 分,第 9 题 2 分,其他每空 1 分)

1. 分析函数 $f()$的时间复杂度。(　　)

```
void f(int n)
{  inti = 0, s = 0;
   while(s < n)  {
       i++;  s = s + i;
```

A. $O(n)$　　B. $O(1)$　　C. $O(n^{1/2})$　　D. $O(n^2)$

2. 假设我们已经得到二叉树一对遍历序列,那么哪一对遍历序列可以唯一确定此二叉树?(　　)

(1) 前序和后序　　(2) 中序和后序

(3) 前序和中序　　(4) 层序和后序

A. 仅(1)　　B. (2)和(3)　　C. 仅(3)　　D. 仅(4)

3. 在有 n 个叶子结点的哈夫曼树中,其结点总数为(　　)。

A. n　　B. $2n$　　C. $2n+1$　　D. $2n-1$

4. 在排序元素的数目较大的情况下,选用(　　)可以获得 $O(n\log_2 n)$的时间复杂度且稳定的排序结果。

A. 归并排序　　B. 快速排序　　C. 堆排序　　D. 希尔排序

5. 将一个递归算法改为对应的非递归算法时,通常需要使用(　　)。

A. 数组　　B. 栈　　C. 队列　　D. 二叉树

6. 若一组记录的关键码为(46,79,56,38,40,84),则利用堆排序的方法建立的初始大根堆为(　　)。

A. 84,56,79,40,46,38　　B. 84,79,56,46,40,38

C. 84,79,56,38,40,46　　D. 79,46,56,38,40,84

7. 我们用循环链表表示一个队列。如果用一个独立的指针 p 用来存取队列,那么指针 p 指向哪个结点才能使得入队和出队耗费固定的时间?(　　)

A. 尾结点　　B. 头结点

C. 无法实现　　D. 头结点的下一个结点

8. 如下方法和数据结构的最佳匹配是(　　)。

X:深度优先遍历　　1:堆

Y:广度优先遍历　　2:队列

Z:排序　　3:栈

A. X−1 Y−2 Z−3　　B. X−3 Y−1 Z−2

C. X−3 Y−2 Z−1　　D. X−2 Y−3 Z−1

9. 如下代码中函数 DoSomething()的输入参数为一个指向非空树根节点的指针,那么函数的返回值是(　　)。

```
structCellNode
```

```
    int element;
    CelINode * leftchild;
    CelINode * rightChild;
}
intDosomething(struct CelINode * ptr)
{
    int value = 0;
    if (ptr ! = NULL)

        if (ptr -> leftChild ! = NULL)
            value = 1 + DoSomething(ptr - >leftChild);
        if (ptr -> rightChild! = NULL)
            value = max(value, 1 + DoSomething(ptr - >rightChild));

    return (value);
}
```

A. 该树的叶子结点数　　B. 该树的结点数

C. 该树的分支结点数　　D. 该树的高度

10. 用数组 A[1..MAXSIZE]来实现两个栈。两个栈分别从数组的两端开始增长，变量 top1 和 top2(top1<top2)分别指向每个栈的栈顶元素。如果数据空间被有效利用，那么栈满的条件是(　　)。

A. (top1 = MAXSIZE/2)并且(top2 = MAXSIZE/2 + 1)

B. top1 + top2 = MAXSIZE

C. (top1 = MAXSIZE/2) or (top2 = MAXSIZE)

D. top1 = top2 - 1

11. 如果使用数组数据结构，下述哪种排序算法在最佳情况下时间复杂度最高？(　　)

A. 堆排序　　B. 选择排序　　C. 气泡排序　　D. 插入排序

12. 在双向链表中，删除 p 所指结点的直接后继结点的操作是(　　)。

A. q = p→right;p→right→right→left = p;p→right = p→right→right

B. q = p→right;p→right→left = p; q→right = p→right;

C. p→right→right→left = p;p→right = q;p→right = p→right→right

D. p→right = p→right→right; p→right = q; p→right→right→left = p

13. 如果一个栈的进栈序列是 1,2,3,4 且规定每个元素的进栈和退栈各一次，那么不可能得到的出栈序列是(　　)。

A. 4,3,2,1　　B. 4,2,1,3　　C. 1,3,2,4　　D. 3,4,2,1

14. 如果从无向图的任一顶点出发进行一次深度优先搜索即可访问所有顶点，则该图一定是(　　)。

A. 完全图　　B. 一棵树　　C. 有回路　　D. 连通图

15. 设散列表长 $m=14$，散列函数 H(key) = key MOD 11。表中已有 4 个结点：H(15) = 4，

H(38) = 5，H(61) = 6，H(84) = 7，其余地址为空，如用线性探测法处理冲突，则关键字为 49 的地址为(　　)。

A. 8　　B. 3　　C. 5　　D. 9

16. 一棵二叉树的前序遍历序列为 30，20，10，15，25，23，39，35，42。下列哪个序列能为同一棵树的后序遍历序列？(　　)

A. 10，20，15，23，25，35，42，39，30

B. 15，10，25，23，20，42，35，39，30

C. 15，20，10，23，25，42，35，39，30

D. 15，10，23，25，20，35，42，39，30

17. 下面哪一种排序方法的时间复杂度最接近 $O(n)$？(　　)

A. 快速排序　　B. 堆排序　　C. 计数排序　　D. 归并排序

三、综合题。(38 分)

1. (7 分)设散列表的长度为 13，散列函数为 $H(k)=k\%13$，给定的关键字序列为 32,14,36,1,68,7,19,27,55,11,23,66。试画出用拉链法解决冲突时所构造的散列表(3 分)，并求出在等概率的情况下，该方法的查找成功和查找不成功的平均查找长度(4 分)。

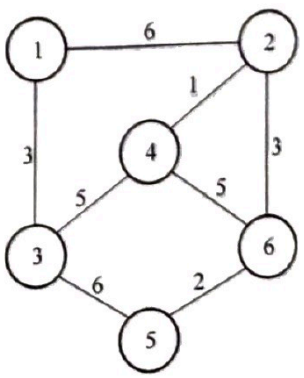
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

2. (12 分)假设用于通信的电文由 8 个字符{A、B、C、D、E、F、G、H }组成，字符在电文中出现的频次分别为{1,3,6,16,11,10,20,5}。完成以下任务：

(1) 画出哈夫曼树(规定：左子树频次＜右子树频次)。(4 分)

(2) 给出各个字符的哈夫曼编码(编码规则为左 0，右 1)。(4 分)

(3) k 叉哈夫曼问题：k 叉哈夫曼树是一颗每个节点最多有 k 棵子树的有序树。现将该电文构造一棵具有最小带权路径的 4 叉哈夫曼树，并试求该树的带权路径长度。(4 分)



3. (12 分)对于如右图所示的带权图 G，完成以下问题：

(1) 写出对应的邻接矩阵。(3 分)

(2) 分别写出从 1 号节点开始深度和广度优先遍历结果。(4 分)

(3) 画出该图的最小生成树。(2 分)

(4) 按照 Dijkstra 算法顺序,给出从 1 点出发到各个结点的最短路径。(3 分)

4. (7 分)已知序列{12,47,10,18,60,15,7,13,25,100},完成以下问题:

(1) 使用最少的调整次数将其调整为大根堆(画图),写出调整后的序列。(2 分)

(2) 写出按大根堆进行堆排序的第一趟排序序列(画图和序列)。(2 分)

(3) 写出堆排序的时间复杂度、空间复杂度、稳定性。(3 分)

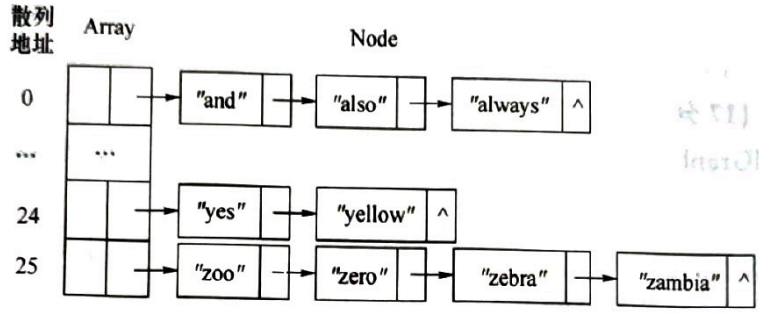
四、编程题。(17 分)

1. 下面类 MGraph<T>的成员函数 BFS 完成对图的广度优先遍历。请认真阅读程序并在空白处填写代码。(5 分)

```
#define MAXSIZE 20
template class <T> classMGraph {
    public:
        MGraph( ifstream&fin);                    //构造函数
        void BFS( int v);                         //从 v 出发广度优先遍历
private:
    T vertex[MAXSIZE];                            //顶点
    int arc[MAXSIZE][MAXSIZE];                    //边(弧)
    visited[MAXSIZE];                             //顶点访问标记数组
    int vNum, arcNum;                             //顶点数、边数
};
template <class T>
voidMGraph<T>:: BFS( int v)                       //从顶点 v 出发完成图的广度优先遍历
{
    int queue[MAXSIZE], f, r ;                    //定义队列 queue,f 为对头,r 为队尾
    _______;                                      //设置空队列
    cout << vertex[v];
    visited[v] = 1;
    queue[ ++ f] = v ;                            //v 入队(队列下标从 0 开始)
    while (_______)                               //队列不空
    {
        v = _______;                              //队头元素出队
        for( int j = 0; j < vNUM; j ++ )
        {
            if (_______)                          //判断是否存在 v 的未访问邻接点
            {
                cout << vertext[j];
                visited[j] = 1;
                _______;                          //未访问邻接点入队
```

```
            }
        }
    }
}
```

2. 下面的函数 Find 采用拉链法来实现对英文单词的查找。若找到，则返回存储地址；没找到，则返回 NULL。具体的存储结构如下图所示，请阅读程序并在空白处填写代码。(12 分)




```
struct Array{
    char data[15];
    Node * front;
};
struct Node{
    char data[15];
    Node * next;
};
Node * Find(ArrayHeadNode[], char * word)
{
    int index;
    //计算散列地址
    index = _____;
    //按散列地址在相应的"同义词"链表中查找单词 word
    Node * p;
    p = _____;                    //指向"同义词"链表中第一个词
    while(_____)
    {
        if (_____)
            _____;                //如果找到返回存储地址
        _____;                    // 下一个单词
    }
    return NULL;
}
```