班级：21级自动化6 班 姓名： 吴俊达 学号： 210320621

**第一章 绪论**

**（1）—（6）：思考下列程序的时间复杂度。**

**（1）**

for (i = 1; i <= n; i++)

    for (j = 1; j <= m; j++)

        A[i, j] = 0;

**时间复杂度为：**

**（2）**

i = 0;

s = 0;

while (s < n)

{

    i++;

    s = s + i;

}

**时间复杂度为：**

**（3）**

m = 0;

while (n >= m \* m)

    m++;

**时间复杂度为：**

**（4）**

int rec(int n)

{

    if (n == 1)

        return 1;

    else

        return (n \* rec(n - 1));

}

**时间复杂度为：**

**（5）**

i = 1;

j = 0;

while (i + j <= n)

    if (i > j)

        j++;

    else

        i++;

**时间复杂度为：**

**（6.1）**

x = 91;

y = 100;

while (y > 0)

    if (x > 10)

        y--;

**（6.2）**

x = 91;

y = 100;

while (y > 0)

    if (x > 100)

    {

        x = x - 10;

        y--;

    }

    else

        x++;

分析6.1,6.2两题中执行频度最大的语句及其频度。

**6.1 执行频度最大的语句为：**y--**，其执行频度为：100次。**

**6.2 执行频度最大的语句为：**x++**，其执行频度为：1000次。**

**（7）求下列程序的时间复杂度及语句k++的执行频度。**

k = 0;

for (i = 1; i <= n; i++)

    for (j = i; j <= n; j++)

        k++;

**时间复杂度为：；执行频度为：次。**

【第一轮循环执行*n*次，第二轮为*n*－1次，……，第*n*轮为1次，依据等差数列求和公式易得】

**（8）求下列程序的时间复杂度**

for (i = 1; i <= n; i = 2 \* i)

{

    cout << "Hello!" << endl;

}

**时间复杂度为：**

**第二章 线性表**

1. 数据结构按逻辑结构可分为两大类，它们是：  **线性**  结构， **非线性**  结构。

2. 下面程序段的时间复杂度为 **** ：

for (i=0; i<n; i++)

for (j=0; j<m; j++)  
 A[i][j]

3. 若经常需要对线性表进行插入和删除运算，最好采用 **链式** 存储结构，若经常需要对线性表进行查找运算，最好采用 **顺序** 存储结构。

4. 对于一个单链表，在表头插入结点的时间复杂度为 **** ，在表尾插入结点的时间复杂度为 **** 。

5.双向链表指针p结点之后插入指针为s的结点，执行语句：

s->next = p->next;

s->prior = p;

p->next = s;

s->next->prior = s;

6.根据二元组关系，指出它们属于何种数据结构。

a) A=（D，R），D={a，b，c，d，e，f}，

R={<a，b>，<b，c>，<c，d>，<d，e>，<e，f>}

**属于线状数据结构**

1. B=（D，R）， D={1，2，3，4，5，6}，

R={（1,2）,（2,3）,（2,4）,（3,4）,（3,5）,（3,6）,（4,5）,（4,6）}

**属于树状数据结构**

1. C=（D，R）， D={a，b，c，d，e，f，g，h}，

R={<d,b>,<d,g>,<d,a>,<b,c>,<g,e>,<g,h>,<e,f>}

**属于网状数据结构**

7.试分别以不同的存储结构实现线性表的就地逆转算法。即在原表的储存空间内将线性表（a1,a2,…,an)变为(an,an-1,…,a1)。

线性表存储方式下， C代码如下：

# define LIST\_INIT\_SIZE 100

# define LISTINCREMENT 10 //因插入元素而导致空间不足时附加内存增量

// 结构类型定义

typedef struct{

    ElemType \*elem; //存储空间基址

    int length; //当前长度

    int listsize; // 当前分配的存储容量,以sizeof(ElemType)为单位

}Sqlist;

// 就地逆转算法

void reverse(Sqlist L){

    for(i=0,j=L.length-1;i<j;i++,j--){

        temp = L.elem[i];

        L.elem[i] = L.elem[j];

        L.elem[j] = temp;

    }

}

单向链表存储方式下， C代码如下：

// 结构类型定义

typedef struct LNode{

    ElemType data; //数据域

    struct LNode \*next; //指针域

}LNode,\*LinkList;

// 就地逆转算法

void reverse(Linklist \*head){

    // head为带头结点的单链表的头指针

    LNode \*p,\*q;

    p=head->next; //p指向第一个结点

    head->next = NULL;

    // 以下依次将每个结点插入头结点之后

    while(p){

        q=p->next;

        p->next=head->next;

        head->next =p;

        p=q;

    }

}

8．线性表是具有n个\_\_**C**\_\_\_的有限序列（n>0）。

A 表元素 B 字符     C 数据元素      D 数据项      E 信息项

9. 某线性表中最常用的操作是在最后一个元素之后插入一个元素和删除第一个元素，则采用\_\_C\_\_\_\_存储方式最节省运算时间。

A. 单链表 B. 单循环链表

C. 带尾指针的单循环链表 D. 带头结点的双循环链表

(D不入选是因为D的插入和删除操作步骤比C多)

10. 静态链表中指针表示的是\_\_ **B** \_\_。

A.  内存地址 B. 数组下标

C.  下一元素地址 D. 左、右孩子地址

11. 完成在双循环链表结点p之后插入s的操作是:

s->next = p->next;

s->prior = p;

p->next = s;

s->next->prior = s;

12. 假设一个头指针为head的带头结点的单循环链表，判断该表为空表的语句是

head->next = head;

13. (判对错) 循环链表不是线性表。**（×）循环链表也是线性表。**

14. (判对错) 顺序存储方式的优点是存储密度大，且插入、删除运算效率高。**（×）顺序存储方式插入、删除运算需要移动大量元素，效率较低。**

15. (判对错) 链表中的头结点仅起到标识的作用。**（×）头结点的数据域可以存储如线性表的长度等附加信息，头结点的指针域存储指向第一个结点的指针（即第一个元素结点的存储位置）**

16. 在一个以h为头的单循环链中，p 指针指向链尾的条件是\_\_ **A** \_\_。

 A. p->next = h B. p->next = NULL

C. p->next->next = h D. p->data=-1

17. 假设静态链表A，其结点结构为（data，next)，A[0]是备用链表头结点，A[1]是线性表头结点。在线性表头插入元素e的语句是:

k = A[0].next; // 获取备用链表中第一个可用结点

A[0].next = A[k].next; // 更新备用链表中第一个可用结点

A[k].data = e; // 将上述结点设置为新结点，数据域赋值e

A[k].next = A[1].next; // 将新结点的下一个结点设置为目前线性表头节点指向的下一个结点

A[1].next = k; // 将线性表头节点指向的下一个结点设置为新结点