2.1. 一元稀疏多项式的求导算法

写出一元稀疏多项式的求导算法，用带表头结点的单链表存储该一元稀疏多项式，Lb 为头指针，用类 C 语言描述该求导算法，不另行开辟存储空间，删除无用结点，并分析算法的时间复杂度。该链表的数据结构如下：

```
typedef struct LNode{
    float    coe;  //系数
    int      exp;   //指数
    struct   LNode   *next;   //指针
} LNode , *LinkList ;
```

求导算法如下：

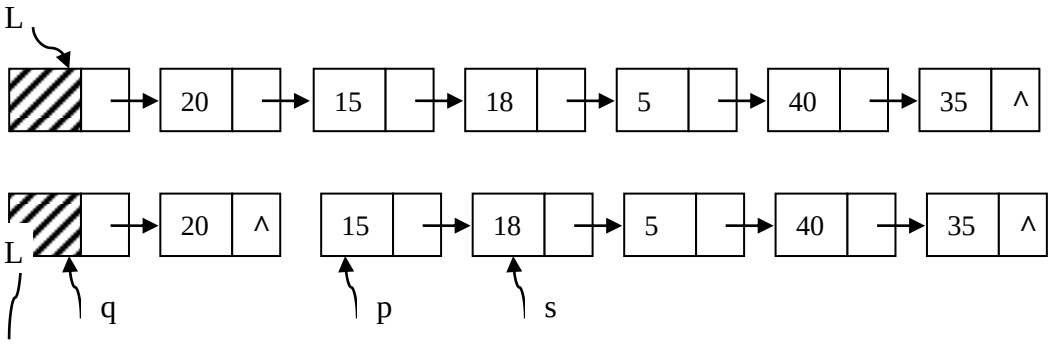
```
void   Differential(LinkList &Lb)
 { //求导算法，Lb 为链表头指针
 LinkList p, pre; //定义指针变量 p, pre
 pre=Lb; //初始化变量，pre 为 p 的前驱
 p=pre->next; // p 为当前待处理的结点
 while ( p ) //遍历链表
  {//逐个结点（数据项）求导处理
   if ( p->exp != 0 )//指数不等于零，非常数项
    {   p->coe = p->coe * p->exp ; //修改系数
        p->exp = p->exp – 1 ; //修改指数
        pre = pre->next ; //后移前驱指针 pre
    }
   else//指数等于零，常数项
    {   pre->next = p->next ; //逻辑删除常数项结点
        free ( p ); //物理删除常数项结点，即释放所占内存
    }
    p = pre->next ; //处理下一结点
   }
 }// Differential
```

时间复杂度为: O(n)

2.2. 单链表存储结构的排序算法

排序算法：将一组整数排序成非递减有序序列。用带头结点的单链表存储，L 为头指针，用类 C 语言写出该排序算法，不另行开辟存储空间，并分析算法的时间复杂度。该单链表的数据结构如下：

```
typedef struct LNode{
        int    data; //数据域
    struct    LNode    *next;    //指针域
        } LNode , *LinkList ;
void    Sort(LinkList    &L)//L 为链表头指针
{    //排序算法如下：将 L 排序成非递减单链表
    LinkList    q,p,s;    //定义变量
    if (L->next==NULL) { printf("空表\n");    return;}
//拆成 2 个链表，第 1 个链表只有头结点和首元结点，其余结点为第 2 个链表
    q=L;//q 为待处理结点的前驱
    p=q->next->next;// 第 2 个链表的头指针为 p
    q->next->next=NULL; // 第 1 个链表的头指针为 L
    //直接插入排序方法：第 2 个链表逐个结点插入到第 1 个链表
    while(p) //遍历链表
    { //比较数据域大小，定位合适的插入位置的前驱 q
        while(q->next && p->data >= q->next->data)    q=q->next;
        s=p->next; //保留下一个待处理结点
        p->next=q->next; //本条语句和下一条语句：p 结点插入到 q 结点的后面
        q->next=p;
        p=s; //处理下一结点
        q=L; //q 归位链表头结点 L
    }
}//sort
```



2.3. 设 H 为具有 n(n>0，n 很大且未知)个数据元素的单链表的头结点指针，试采用 C 语言编写一个程序，完成将单链表中第 n/2 个数据元素之后的全部数据元素倒置的功能。要求不另行开辟存储空间，算法的时间复杂度不超过 O(n)。

单链表结点的数据类型描述如下：

```
typedef struct Lnode {
  int data;    //数据元素为整数
  struct Lnode    *next;
}Lnode, *LinkList;

void ReverseN2(LinkList &H)
{//将单链表的正中间位置结点之后的全部结点倒置的功能
   LinkList p,q,s;
   p=H;   //初始化变量，p 指向头指针
   q=H;   //初始化变量，q 指向头指针
//定位链表正中间位置结点。p 结点走到表尾，q 结点在正中间位置
   while (p)
   {   if (p->next) //如果不是表尾结点
           p=p->next->next;   // p 指向下一结点的下一结点
       else//如果是表尾结点
           break;     //退出
       q=q->next;    //处理下一结点 q 指向下一结点
   }

   //拆成 2 个链表，第 1 个链表：头结点开始到正中间位置 q 结点
   //第 2 个链表：q 结点（不含 q）之后结点组成第 2 个链表
   p=q->next; // p 指向 q 结点的后继结点，p 为第 2 个链表的头指针
   q->next=NULL; //第 1 个链表的尾结点指针为 q
   //头插法（逆序），p 指向的第 2 个链表逐个逆序插入 q 结点的后面
   //实现倒置功能
   while (p)
   {   s= p->next; //保留下一个待处理结点
       p->next=q->next; //本条语句和下一条语句：p 结点插入到 q 结点的后面
       q->next=p;
       p=s; //处理下一结点
   }
}//ReverseN2
```