1. 简述下列术语：线性表，顺序表，链表。

解：（1）线性表：n 个具有相同特性的数据元素的有限序列。

（2）顺序表：在计算机内存中以数组的形式保存的线性表。通过数据元素物理存储的相邻关系来反映数据元素之间逻辑上的相邻关系，采用顺序存储结构的线性表通常称为顺序表。

（3）链表：一种物理存储单元上非连续、非顺序的存储结构。

2. 何时选用顺序表，何时选用链表作为线性表的存储结构合适?各自的主要优缺点是什么？

解：（1）基于空间的考虑：当要求存储的线性表长度变化不大，易于事先确定其大小时，为了节约存储空间，宜采用顺序表；反之，当线性表长度变化大，难以估计其存储规模时，采用动态链表作为存储结构为好。

（2）基于时间的考虑：若线性表的操作主要是进行查找，很少做插入和删除操作时，采用顺序表做存储结构为宜；反之，若需要对线性表进行频繁地插入或删除等的操作时，宜采用链表做存储结构。

3. 在顺序表中插入和删除一个结点平均需要移动多少个结点?具体的移动次数取决于哪两个因素?

解：在等概率情况下，顺序表中插入一个结点需平均移动$\frac{n}{2}$个结点。删除一个结点需平均移动$\frac{n-1}{2}$个结点。具体的移动次数取决于顺序表的长度n以及需插入或删除的位置i。i越接近n则所需移动的结点数越少。

4. 链表所表示的元素是否有序?如有序，则有序性体现于何处?链表所表示的元素是否一定要在物理上是相邻的?有序表的有序性又如何理解?

解：有序，体现在逻辑上有序，即按指针指向的顺序有序。链表所表示的元素不一定在物理上相邻。有序表的有序性体现在它在逻辑结构和物理结构上均有序。

5. 设顺序表 L 是递增有序表，试写一算法，将 x 插入到 L 中并使 L 仍是递增有序表。

解：反向遍历链表，将 x 与每个元素均进行大小比较，当 x>=某元素 y 时，若 y 的位置为 i，则将 x 插入到 y 之后的位置（即 i+1）即可。

算法如下：

```
typedef struct {
    Elemtype *temp;
    int length;
}List;
void insert(List *list,ElementType x){
    int i;
    for(i=list->length;i>0&&L->temp[i-1]>x;i--){
        list->temp[i]=list->temp[i-1];
    }
    list->temp[i]=x;
    list->length++;
}
```

6. 写一求单链表的结点数目 ListLength(L)的算法。

解:遍历链表，统计结点数：

算法如下：

```
int Length(List *list){
    int len=0;
    List* p;
    p=head;//假设该表有头节点
    while(p->next!=nullptr){
        p=p->next;
        len++;
    }
    return len;
}
```

7. 写一算法将单链表中值重复的结点删除，使所得的结果链表中所有结点的值均不相同。

解：先取开始结点中的值，然后开始遍历链表，将与其值相等的所有结点均删除。然后再取第二结点的值，重复上述操作直到最后一个结点。

算法如下：

```
void Delete(LinkList list){
    List *p,*q,*s;
    p=L->next;
    while(p->next&&p->next->next){
        q=p;//因为要做删除操作，所以q指针指向要删除元素的直接前趋
        while(q->next){
            if(p->data==q->next->data){
                s=q->next;
                q->next=s->next;
                free(s);//删除与*p的值的相同的结点
            }
            else{
                q=q->next;
            }
            p=p->next;
        }
    }
}
```

8. 写一算法从一给定的向量 A 删除值在 x 到 y(x≤y)之间的所有元素(注意： x 和 y 是给定的参数，可以和表中的元素相同，也可以不同)。

解：

```
void RangeDelete(LinkList list){
    ElementType *p,*q;
    int i,j;
    p=&A.next;
    for(i=0;i<A.length;i++){
        if(p.data>=x||p.data<=y){
            q=p;
            (p-1).next=p.next;
        }
        p++;
        free(q);
    }
}
```

9. 设 A 和 B 是两个按元素值递增有序的单链表， 写一算法将 A 和 B 归并为按按元素值递减有序的单链表 C，试分析算法的时间复杂度。

解：根据已知条件，A 和 B 是两个递增有序表，所以可以先取 A 表的表头建立空的 C 表。然后同时遍历 A 表和 B 表，将两表中最大的结点从对应表中取出，并作为开始结点插入到 C 表中。

重复此操作，直至 A 表或 B 表为空。最后将不为空的 A 表或 B 表中的结点依次取出并作为开始结点插入 C 表中。

此时，得到的 C 表就是由 A 表和 B 表归并成的一个按元素值递减有序的单链表 C。

算法如下：

```
void List::Merge(List& ptr)
{
    LinkNode* pa=head;
    LinkNode* pb=ptr.head;
    //由于链表不带头结点，所以先比较第一个数据的大小，从而确定合并后的链表的头指针的位置
    LinkNode* pd=(pa->data>=pb->data)?pa:pb;
    LinkNode* p=new LinkNode(pd->data);
    LinkNode* pHead=p;
    if(p->data==pa->data){
        pa=pa->link;
    }
    else{
        pb = pb->link;
        delete ptr.head;
        ptr.head=pb;
    }
    while (pa&&pb!=nullptr){
        if(pa->data>=pb->data){
            p->link=new LinkNode(pa->data);
            p=p->link;
            pa=pa->link;
        }
        else{
            p->link=new LinkNode(pb->data);
            p=p->link;
            pb=pb->link;
            //删除ptr的空间
            delete ptr.head;
            ptr.head=pb;
        }
    }
```

```
    if(pa!=nullptr){
        while(pa!=nullptr)
        {
            p->link=new LinkNode(pa->data);
            p=p->link;
            pa=pa->link;
        }
    }
    else{
        while(pb){
            p->link=new LinkNode(pb->data);
            p=p->link;
            pb=pb->link;
            //删除ptr的空间
            delete ptr.head;
            ptr.head=pb;
        }
    }
    p->link=NULL;
    //删除调用该函数的链表的原空间
    clear();
    head=pHead;
}
```

该算法的时间复杂度分析如下：因为算法中有三个 while 循环，其中第二个和第三个循环只执行一个。每个循环做的工作都是对链表中结点遍历处理。整个算法完成后，A 表和 B 表中的每个结点都被处理了一遍，则该算法的时间复杂度为 O(Length(A)+Length(B)).