Q1:

假设有一个带头结点的单链表 L=（a_1，b_1，a_2，b_2，…，a_n，b_n）。设计一个算法将其拆分成两个带头结点的单链表 L1 和 L2：

L1=（a_1，a_2，…，a_n）， L2=（b_n，b_{n-1}，…，b_1）

要求 L1 使用 L 的头结点。

```c
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int data;
    struct node *next;
};

// 创建新节点
struct node *create_node(int data)
{
    struct node *new_node = (struct node
*)malloc(sizeof(struct node));
    new_node->data = data;
    new_node->next = NULL;
    return new_node;
}

// 打印链表
void printList(struct node *head)
{
    struct node *current = head;
    while (current != NULL)
    {
        printf("%d,", current->data);
        current = current->next;
    }
}

// 拆分链表，奇数位置的节点放在1，偶数位置的节点放在2
void splitList(struct node *head, struct node **head1,
struct node **head2)
{
```

```
    // 初始化链表
    struct node *current = head;
    struct node *oddList = NULL;
    struct node *evenList = NULL;
    struct node *oddTail = NULL;
    struct node *evenTail = NULL;

    while (current != NULL)
    {
        if (current->data % 2 != 0)
        { // 奇数位置
            if (oddList == NULL)
            {
                oddList = current;
                oddTail = oddList;
            }
            else
            {
                oddTail->next = current;
                oddTail = oddTail->next;
            }
        }
        else
        { // 偶数位置
            if (evenList == NULL)
            {
                evenList = current;
                evenTail = evenList;
            }
            else
            {
                evenTail->next = current;
                evenTail = evenTail->next;
            }
        }
        // 移动到下一个节点
        current = current->next;
    }

    // 断开链表
    if (oddTail != NULL)
    {
        oddTail->next = NULL;
    }
    if (evenTail != NULL)
```

```
    {
        evenTail->next = NULL;
    }
    // 返回拆分后的链表
    *head1 = oddList;
    *head2 = evenList;
}

int main()
{
    struct node *head = NULL;
    struct node *head1 = NULL;
    struct node *head2 = NULL;

    // 创建链表
    head = create_node(1);
    head->next = create_node(2);
    head->next->next = create_node(3);
    head->next->next->next = create_node(4);
    head->next->next->next->next = create_node(5);
    head->next->next->next->next->next = create_node(6);
    head->next->next->next->next->next->next =
create_node(7);
    head->next->next->next->next->next->next->next =
create_node(8);
    head->next->next->next->next->next->next->next->next =
create_node(9);
    head->next->next->next->next->next->next->next->next-
>next = create_node(10);

    // 拆分链表
    splitList(head, &head1, &head2);

    // 打印链表
    printf("List 1 (Odd parts): ");
    printList(head1);

    printf("List 2 (Even parts): ");
    printList(head2);

    return 0;
}
```

Q2：某线性表最常用的操作是在尾元素之后插入一个元素和删除第一个元素，故采用（ C ）存储方式最节省运算时间。

A.单链表

B.仅有头结点指针的循环单链表

C.双链表

D.仅有尾结点指针的循环单链表

Q3：如果对含有 n（n>1）个元素的线性表的运算只有 4 种，即删除第一个元素、删除尾元素、在第一个元素前面插入新元素、在尾元素的后面插入新元素，则最好使用（ D ）。

A.只有尾结点指针没有头结点的循环单链表B.

只有尾结点指针没有头结点的非循环双链表

C.只有首结点指针没有尾结点指针的循环双链表

D.既有头指针也有尾指针的循环单链表

Q4：一个长度为L（L≥1）的升序序列S，处在第L/2个位置的数称为S的中位数。

例如：若序列S_1=(11，13，15，17，19)，则S1的中位数是15。

两个序列的中位数是含它们所有元素的升序序列的中位数。例如，若S_2=(2，4，6，8，20)，则S_1和S_2的中位数是11。

现有两个等长的升序序列A和B，试设计一个在时间和空间两方面都尽可能高效的算法，找出两个序列A和B的中位数。要求：

（1）给出算法的基本设计思想。

因为两个序列都是升序的，我们可以利用二分查找来缩小搜索范围。我们将数组划分为左右两部分，使得左半部分的最大值小于等于右半部分的最小值。如果满足条件，我们就可以确定中位数。否则，我们通过调整划分的位置继续搜索，直到找到满足条件的划分点。这样，我们可以在O(log(min(m, n)))的时间复杂度内找到两个升序数组的中位数。

（2）根据设计思想，采用C、C++或Java语言描述算法，关键之处给出注释。

```c
#include <stdio.h>
#include <stdlib.h>

// 用于找到两个升序数组的中位数
double findMedianSortedArrays(int *nums1, int nums1Size,
int *nums2, int nums2Size)
{

    if (nums1Size > nums2Size)
    {
```

```
        return findMedianSortedArrays(nums2, nums2Size,
nums1, nums1Size);
    }

    int low = 0, high = nums1Size;
    // 二分查找
    while (low <= high)
    {
        int partitionX = (low + high) / 2;
        int partitionY = (nums1Size + nums2Size + 1) / 2 -
partitionX;

        // 计算partitionX 和partitionY 的边界元素
        int maxLeftX = (partitionX == 0) ? INT_MIN :
nums1[partitionX - 1];
        int minRightX = (partitionX == nums1Size) ?
INT_MAX : nums1[partitionX];
        int maxLeftY = (partitionY == 0) ? INT_MIN :
nums2[partitionY - 1];
        int minRightY = (partitionY == nums2Size) ?
INT_MAX : nums2[partitionY];

        // 检查是否找到了中位数
        if (maxLeftX <= minRightY && maxLeftY <= minRightX)
        {
            // 奇数个元素
            if ((nums1Size + nums2Size) % 2 == 1)
            {
                return (double)max(maxLeftX, maxLeftY);
            }
            else
            { // 偶数个元素
                return (double)(max(maxLeftX, maxLeftY) +
min(minRightX, minRightY)) / 2;
            }
        }
        else if (maxLeftX > minRightY)
        {
            // 减少nums1 的右半部分
            high = partitionX - 1;
        }
        else
        {
            // 增加nums1 的左半部分
            low = partitionX + 1;
```

```
        }
    }
    // 如果没有找到中位数，返回-1
    return -1;
}

int main()
{
    //给定两个升序列
    int nums1[] = {1, 3, 5, 7, 9};
    int nums2[] = {2, 4, 6, 8, 10};
    int nums1Size = sizeof(nums1) / sizeof(nums1[0]);
    int nums2Size = sizeof(nums2) / sizeof(nums2[0]);

    double median = findMedianSortedArrays(nums1, nums1Size,
nums2, nums2Size);
    printf("The median is: %.2f\n", median);

    return 0;
}
```

3）说明你所设计算法的时间复杂度和空间复杂度。

时间复杂度和空间复杂度均为O(log n)