软件专硕模拟卷（三）答案

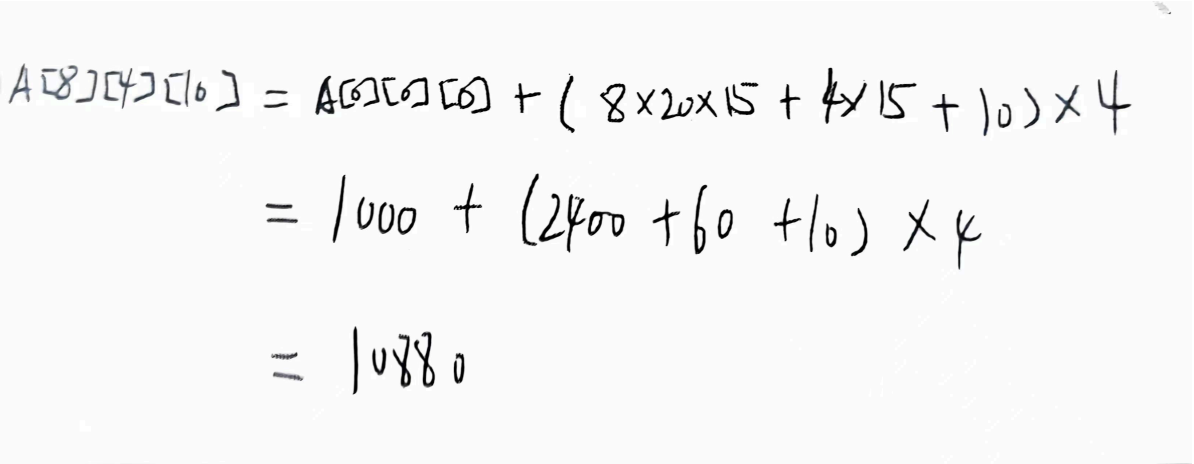
《数据结构》（50 分）

一、简答题（共20分）

1.三维数组`A[10][0][15]`采用行优先方式存储，每个元素占4个存储单元，如果`A[0][0][0]`的存储地址是1000，

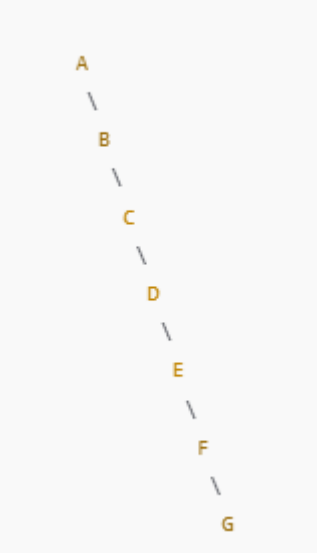
则`A[8][4][10]`的存储地址是多少，给出简要计算过程。（3分）

答案：



2. 一颗二叉树的先根序列为`ABCDEFG`，则`DACEFBG`，`CABDEFG`，`ABCDEFG`，是否是其可能得中根序列，如果是，则画出对应的二叉树形态。（4分）

答案：先序和中序一样`ABCDEFG`，最后为一条链



3. 给出一组关键字：`29，18，25，47，58，12，51，10` 分别写出按照以下各种排序方式进行排序的变化过程：

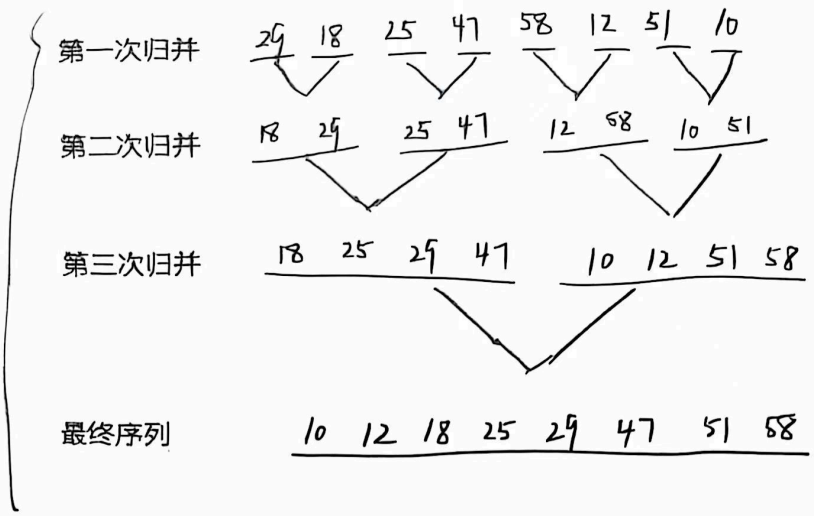
（1）合并排序，每合并一次，书写一个次序

（2）堆排序，先建一个堆，然后每从堆顶取下一个元素后，将堆调整一次。（8分）

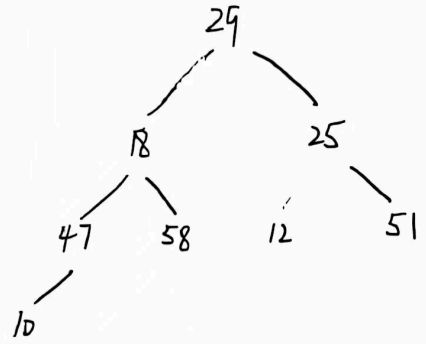
答案：

3、

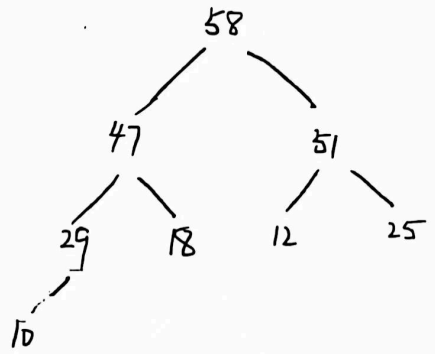
(1)



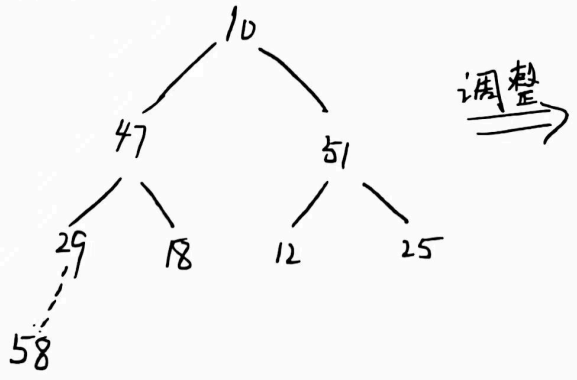
(2)



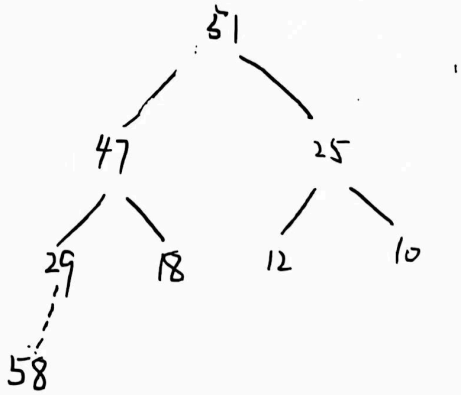
初始未建堆

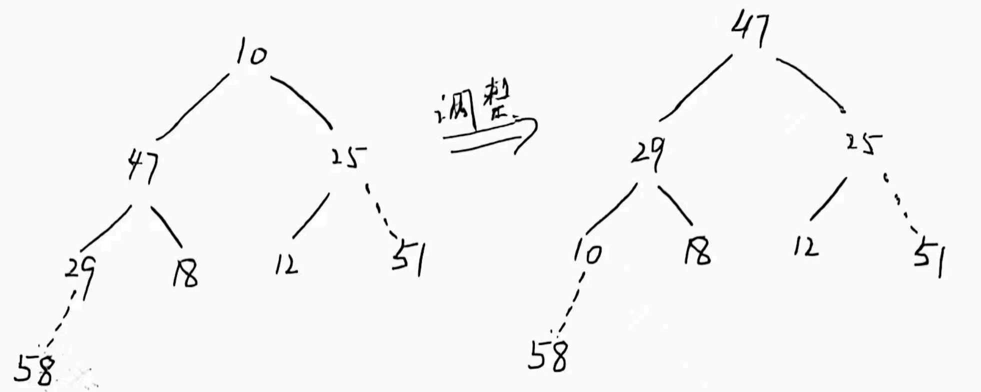


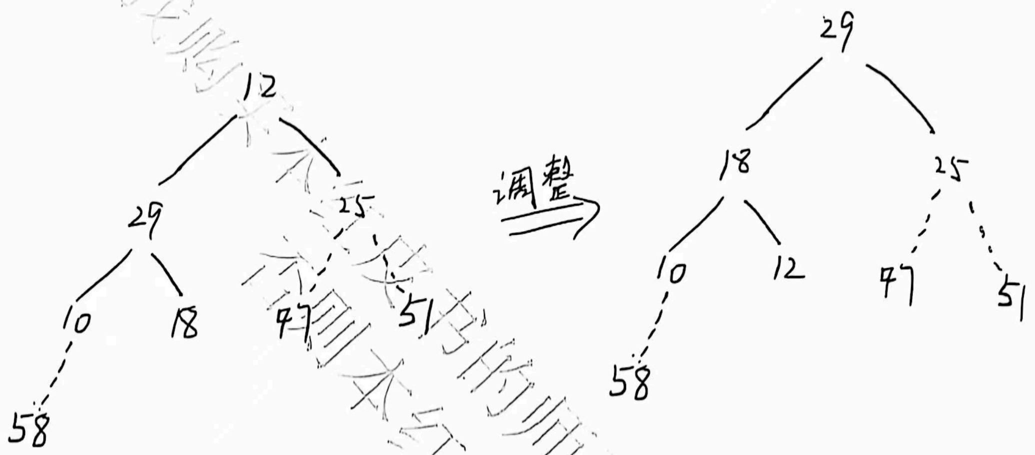
初始堆

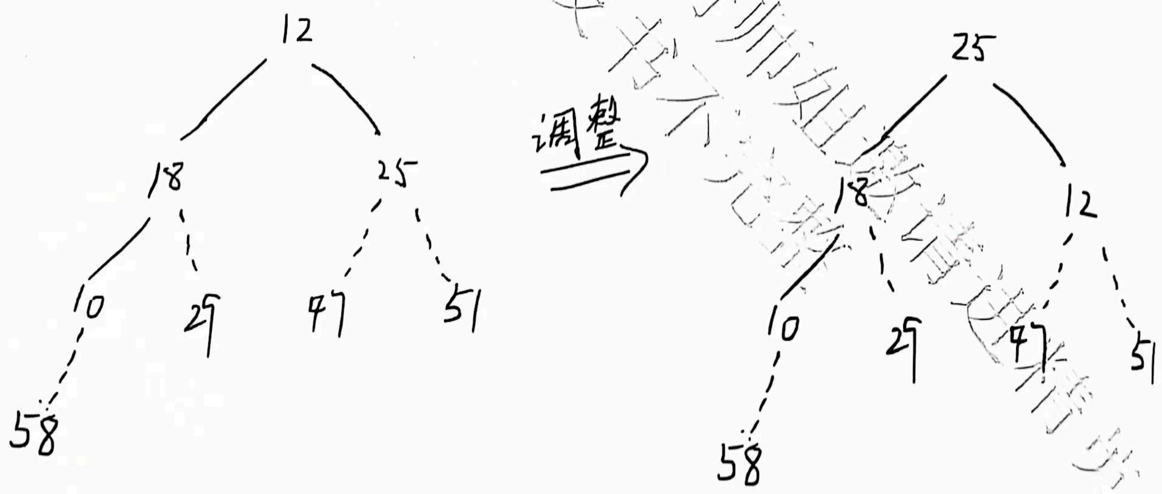


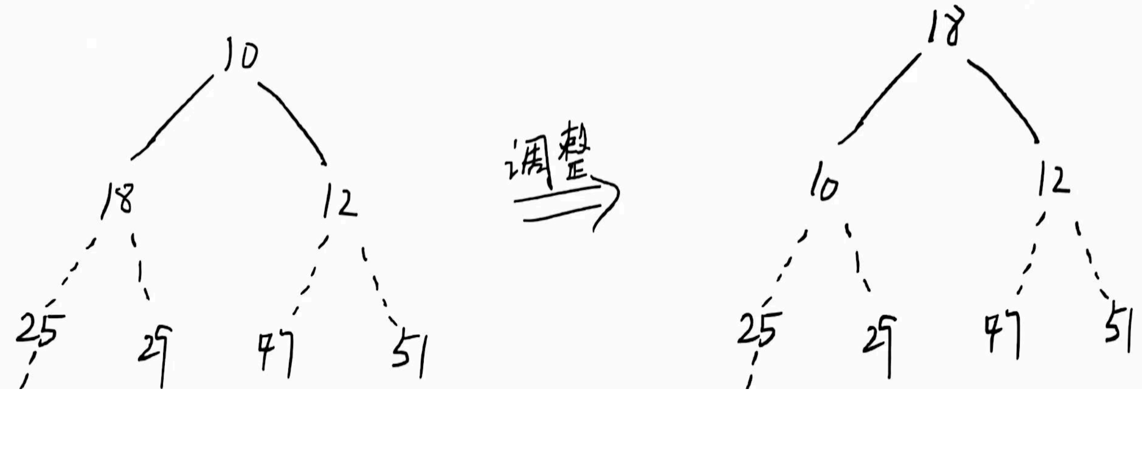
调整

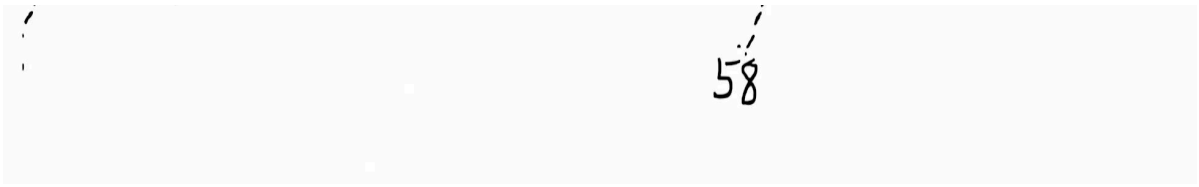


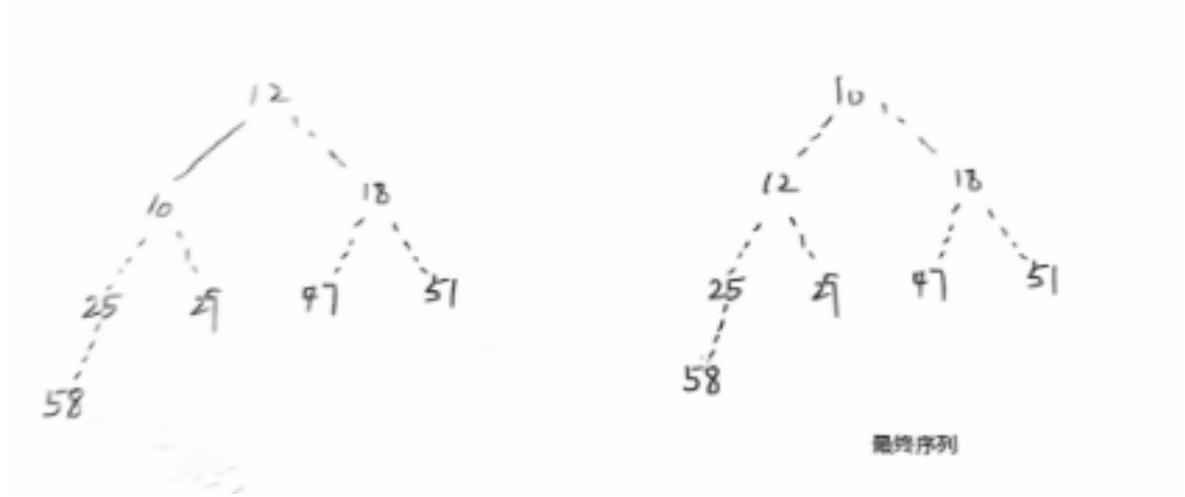

调整


调整

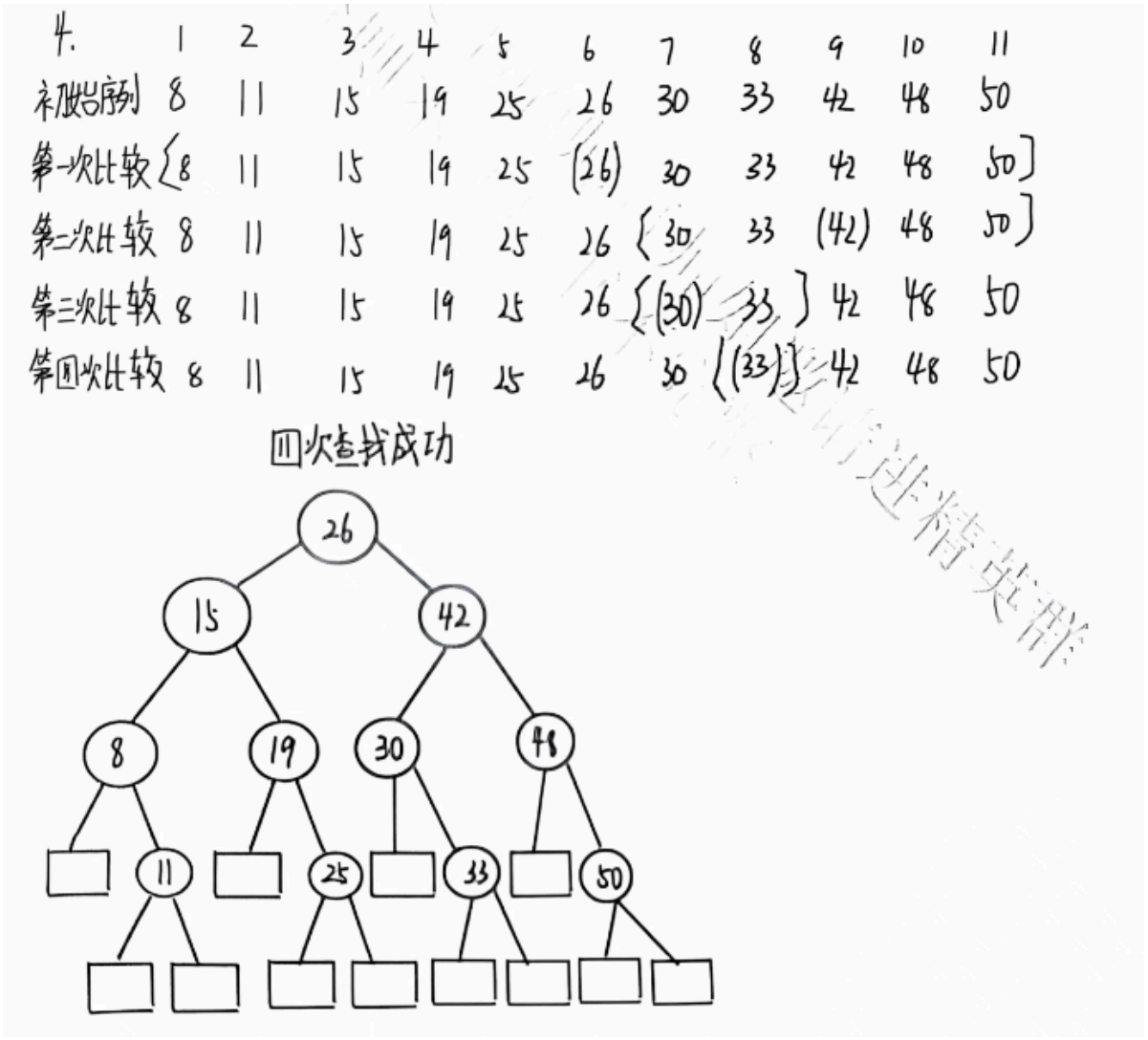

调整


调整





4. 在顺序表{8， 11， 15， 19， 25， 26， 30， 33， 42， 48， 50} 中，用二分法查找关键字33，进行多少次比较后查找成功？写出查找过程，并画出对应的二叉判定树。（5分）



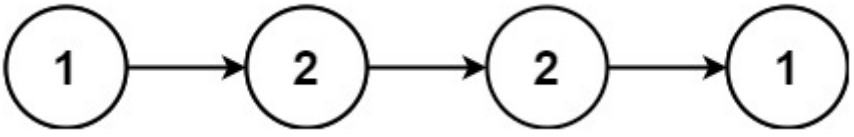
二、算法题（共30分）

答题要求：

- **（1）算法书写可采用`C`，`C++`，`ADL`等语言，使用何种语言书写要注明。**
- **（2）在算法开始出必须用自然语言书写注释，说明算法的基本思路，以及使用了那些数据结构。**
- **（3）算法的关键步骤要写注释说明其目的。**

1. 给你一个单链表的头节点 `head` ，请你判断该链表是否为回文链表。如果是，返回 `true` ；否则，返回 `false` 。（10分）

示例：



回文链表即关于中心对称的链表

思路介绍

> **链表反转**：我们可以通过将链表的后一半反转，然后与前一半进行比较来判断是否是回文链表。
>
> **快慢指针法**：
>
> - 使用 **快慢指针** 找到链表的中间节点，慢指针每次走一步，快指针每次走两步。
> - 当快指针到达链表末尾时，慢指针就正好指向链表的中间。
>
> **反转后半部分链表**：
>
> - 反转链表的后半部分。
>
> **比较前半部分和反转后的后半部分**：
>
> - 比较前半部分和反转后的后半部分的节点值，如果所有节点值都相等，则说明链表是回文的。

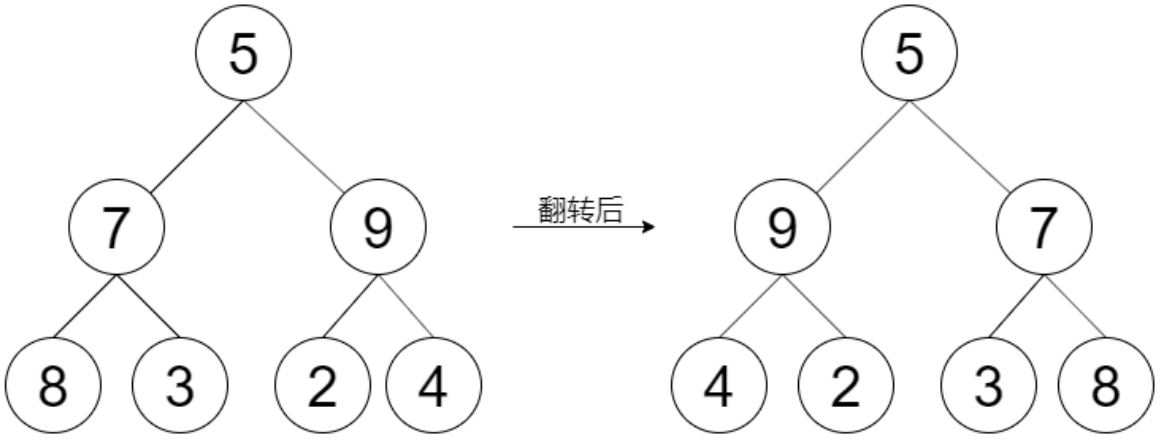
代码

```c
// 定义链表节点结构体
struct ListNode {
    int val;
    struct ListNode *next;
};

// 辅助函数：反转链表
struct ListNode* reverse(struct ListNode* head) {
    struct ListNode* prev = NULL;
    struct ListNode* curr = head;
    struct ListNode* next = NULL;

    while (curr != NULL) {
        next = curr->next; // 保存下一个节点
        curr->next = prev; // 反转当前节点的指针
        prev = curr;       // 更新 prev 为当前节点
        curr = next;       // 移动到下一个节点
    }
    return prev;
}

// 主函数：判断链表是否为回文链表
bool isPalindrome(struct ListNode* head) {
    if (head == NULL || head->next == NULL) {
        return true; // 空链表或单节点链表是回文链表
    }

    // 1. 使用快慢指针找到链表的中间节点
    struct ListNode *slow = head, *fast = head;

    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;
    }

    // 2. 反转链表的后半部分
    struct ListNode* secondHalf = reverse(slow);
```

```
    // 3. 比较前半部分和反转后的后半部分
    struct ListNode* firstHalf = head;
    while (secondHalf != NULL) {
        if (firstHalf->val != secondHalf->val) {
            return false; // 如果有不相等的节点，则不是回文链表
        }
        firstHalf = firstHalf->next;
        secondHalf = secondHalf->next;
    }

    return true; // 如果所有节点都相等，返回 true
}
```

2. 给定一棵二叉树的根节点 `root`，请左右翻转这棵二叉树，并返回其根节点。(10分)

示例 1：



思路介绍

> 从根节点开始，递归地对树进行遍历，并从叶子节点先开始翻转。如果当前遍历到的节点 root 的左右两棵子树都已经翻转，那么我们只需要交换两棵子树的位置，即可完成以 root 为根节点的整棵子树的翻转。

代码

```
struct TreeNode {
      int val;
      struct TreeNode *left;
     struct TreeNode *right;
 };

struct TreeNode* invertTree(struct TreeNode* root) {
    if (root == NULL) {
        return NULL;
    }
    struct TreeNode* left = invertTree(root->left);
    struct TreeNode* right = invertTree(root->right);
    root->left = right;
    root->right = left;
```

```
    return root;
}
```

3. 给定一个含有n个顶点（编号为1到n）和m条边的无向图，每条边都有一个非负的权重。请你找到一个最大生成树，使得树中所有边的权重和最大。（10分）

思路：

> 最小生成树代码中选最小边，转换为最大边即可。
>
> 这里使用最小生成树算法。

代码：

```
#include <stdio.h>
#include <limits.h>

// 定义常量
#define maxnum INT_MAX    // 用于初始化低权值数组的最大值
#define minnum -1         // 用于初始化低权值数组的最小值

// 邻接表结点结构体
typedef struct ArcNode {
    int adjvex;           // 边的另一端顶点
    int info;             // 边的权值
    struct ArcNode* next; // 指向下一条边的指针
} ArcNode;

// 顶点结构体
typedef struct VexNode {
    ArcNode* firstarc;  // 顶点的第一个邻接结点
} VexNode;

// 图结构体
typedef struct AGraph {
    int vexnum;          // 图的顶点数
    VexNode* adjlist;    // 邻接表
} AGraph;

// Prim 算法，最大生成树，邻接表实现
void Prim_AGraph(AGraph *G, int v, int &sum) {
    int lowcost[G->vexnum];  // lowcost数组，表示每个顶点连接到生成树的最小权重
    int visited[G->vexnum];   // visited数组，标记顶点是否已经在生成树中
    int i, j, k, max, temp;

    // 初始化lowcost和visited数组
    for (i = 0; i < G->vexnum; i++) {
        lowcost[i] = minnum;  // 初始化lowcost数组，表示初始没有连接
        visited[i] = 0;       // visited数组，表示没有顶点在生成树中
    }

    // 遍历源点v的所有邻接边，初始化lowcost数组
    ArcNode *p = G->adjlist[v].firstarc;
    while (p != NULL) {
```

```
        temp = p->adjvex;     // 取出邻接顶点
        lowcost[temp] = p->info;  // 更新该邻接顶点的权重为当前边的权重
        p = p->next;          // 遍历该顶点的所有邻接边
    }

    visited[v] = 1;  // 将源顶点v标记为已访问（即已加入生成树）

    // 开始循环，生成树的顶点数为G->vexnum-1
    for (i = 0; i < G->vexnum - 1; i++) {
        max = maxnum;  // 初始化max为最大值
        // 寻找一个没有加入生成树的顶点，且它的lowcost值是最小的
        for (j = 0; j < G->vexnum; j++) {
            if (visited[j] == 0 && lowcost[j] > max) {  // 注意：这里是选择权重最大的
边
                max = lowcost[j];  // 更新最大权重
                k = j;             // 更新该顶点的索引
            }
        }

        sum += lowcost[k];   // 将选择的最大边加入生成树，更新总权值

        visited[k] = 1;       // 将该顶点k标记为已访问，加入生成树

        // 遍历顶点k的所有邻接边，更新其相邻未加入生成树的顶点的lowcost值
        p = G->adjlist[k].firstarc;
        while (p != NULL) {
            int temp = p->adjvex;  // 获取邻接顶点
            if (visited[temp] == 0 && p->info > lowcost[temp]) {  // 如果未加入生成
树并且边的权重大于lowcost[temp]
                lowcost[temp] = p->info;  // 更新lowcost值，选择更大的权重
            }
            p = p->next;  // 遍历该顶点的其他邻接边
        }
    }

    // 输出最大生成树的权值之和
    printf("最大生成树的权值之和为%d\n", sum);
}
```

《高级语言程序设计》（100分）

1.验证角谷猜想：任意给定一个整数，若为偶数则除以2；若为奇数则乘三再加一，得到一个新的自然数之后按照上面的法则继续演算，若干次后得到的结果必为1。(25分)

思路：

> 按照题目思路模拟即可

答案：

```c
#include <stdio.h>

// 验证角谷猜想的函数
void verifyCollatz(int n) {
    if (n <= 0) {
        printf("请输入一个正整数。\n");
        return;
    }
    printf("演算过程：\n");
    while (n != 1) {
        printf("%d -> ", n);
        if (n % 2 == 0) {
            n /= 2; // 偶数除以 2
        } else {
            n = n * 3 + 1; // 奇数乘 3 加 1
        }
    }
    printf("1\n");
}

int main() {
    int num;
    printf("请输入一个正整数：");
    scanf("%d", &num);

    verifyCollatz(num);

    return 0;
}
```

2.编程序判断10阶整数方阵是否关于主对角线对称。（25分）

思路：

> 一个矩阵关于主对角线对称，意味着对于任意的 A[i][j]A[i][j]A[i][j]，都有 A[i][j]=A[j][i]A[i][j] = A[j][i]A[i][j]=A[j][i]A[i][j]=A[j][i]。判断矩阵是否对称的过程如下：
>
> 1. **矩阵输入**： 输入一个 10×1010 \times 1010×10 的整数矩阵。
> 2. **对称性检查**： 遍历矩阵的上三角区域（排除对角线），即当 i<ji < ji<j 时，检查 A[i][j]A[i][j]A[i][j]A[i][j] 是否等于 A[j][i]A[j][i]A[j][i]A[j][i]。如果存在不相等的元素，矩阵就不是关于主对角线对称。
> 3. **输出结果**： 如果所有比较都通过，矩阵是对称的；否则不是。

答案：

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

#define N 10 // 矩阵阶数
```

```c
// 函数：判断矩阵是否关于主对角线对称
bool isSymmetric(int matrix[N][N]) {
    for (int i = 0; i < N; i++) {
        for (int j = i + 1; j < N; j++) { // 只遍历上三角区域（排除对角线）
            if (matrix[i][j] != matrix[j][i]) { // 检查对称性
                return false;
            }
        }
    }
    return true;
}

int main() {
    int matrix[N][N];

    // 1. 输入矩阵
    printf("请输入一个 %dx%d 矩阵的元素：\n", N, N);
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }

    // 2. 判断是否对称
    if (isSymmetric(matrix)) {
        printf("该矩阵关于主对角线对称。\n");
    } else {
        printf("该矩阵不关于主对角线对称。\n");
    }

    return 0;
}
```

3. 编写函数，对n个字符串按照字典序排序。限定函数名:void sort(char st[][10],int n)（25分）

思路：

1. **函数定义**：
   - 输入参数是一个二维字符数组 `st[10]`，存储最多 10 个字符串，每个字符串长度不超过限制。
   - `n` 表示需要排序的字符串数量。

   排序逻辑：
   - 使用冒泡排序的方式，对每一对字符串进行比较。
   - 比较时使用 `strcmp` 函数，如果 `st[i] > st[j]`（即不符合字典序），交换两个字符串的位置。

   交换操作：
   - 为了交换两个字符串，可以使用临时数组 `temp`，避免直接修改。

> **结果输出**：
>
> - 排序完成后，输出排序后的字符串。

答案：

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

// 函数定义：对字符串数组按字典序排序
void sort(char st[][10], int n) {
    char temp[10]; // 临时存储用于交换字符串

    // 使用冒泡排序进行排序
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (strcmp(st[j], st[j + 1]) > 0) { // 若前面的字符串大于后面的，交换
                strcpy(temp, st[j]);        // 交换过程
                strcpy(st[j], st[j + 1]);
                strcpy(st[j + 1], temp);
            }
        }
    }
}

int main() {
    int n;
    char st[10][10]; // 最多存储10个字符串，每个字符串最长9个字符

    // 输入字符串数量
    printf("请输入字符串数量（最多10个）：");
    scanf("%d", &n);

    // 输入字符串
    printf("请输入每个字符串（每行一个，不超过9个字符）：\n");
    for (int i = 0; i < n; i++) {
        scanf("%s", st[i]);
    }

    // 排序
    sort(st, n);

    // 输出结果
    printf("按字典序排序后的字符串：\n");
    for (int i = 0; i < n; i++) {
        printf("%s\n", st[i]);
    }

    return 0;
}
```

4. 编写程序，由键盘输入一个字符串（仅保留数字字符，英文字符和空格），把该字符串中英文字符和空格过滤掉，提取所有整数，并将得到的整数序列输出到文件`in.txt`中。

思路：

> **核心步骤**：
>
> - 从键盘读取一行字符串。
> - 遍历字符串，检查每个字符是否为数字。
> - 如果是数字，则将其提取并存储到一个新字符串中。
> - 将提取出的数字字符写入文件 `in.txt`。
>
> **实现细节**：
>
> - 使用 `fopen` 打开文件 `in.txt`。
> - 使用 `fgetc` 或者直接通过索引遍历字符串。
> - 用 `isdigit` 函数判断字符是否为数字。

答案：

```c
#include <stdio.h>
#include <ctype.h>
#include <string.h>

int isdigit(char ch) {
    return (ch >= '0' && ch <= '9'); // 判断字符是否为数字
}

int main() {
    char input[256]; // 用于存储用户输入的字符串
    char extracted[256]; // 用于存储提取出的数字字符
    int index = 0; // 指向 `extracted` 的当前索引位置

    // 提示用户输入字符串
    printf("请输入一个字符串（仅保留数字字符）：\n");
    fgets(input, sizeof(input), stdin); // 读取一行用户输入，包含空格

    // 遍历输入字符串，提取数字字符
    for (int i = 0; input[i] != '\0'; i++) {
        if (isdigit(input[i])) { // 检查字符是否为数字
            extracted[index++] = input[i];
        }
    }

    extracted[index] = '\0'; // 为提取的数字序列添加字符串结束符

    // 打开文件 `in.txt` 并写入提取的数字
    FILE *file = fopen("in.txt", "w");
    if (file == NULL) {
        printf("无法打开文件。\n");
        return 1;
    }
```

```
    fprintf(file, "%s\n", extracted); // 将数字序列写入文件
```