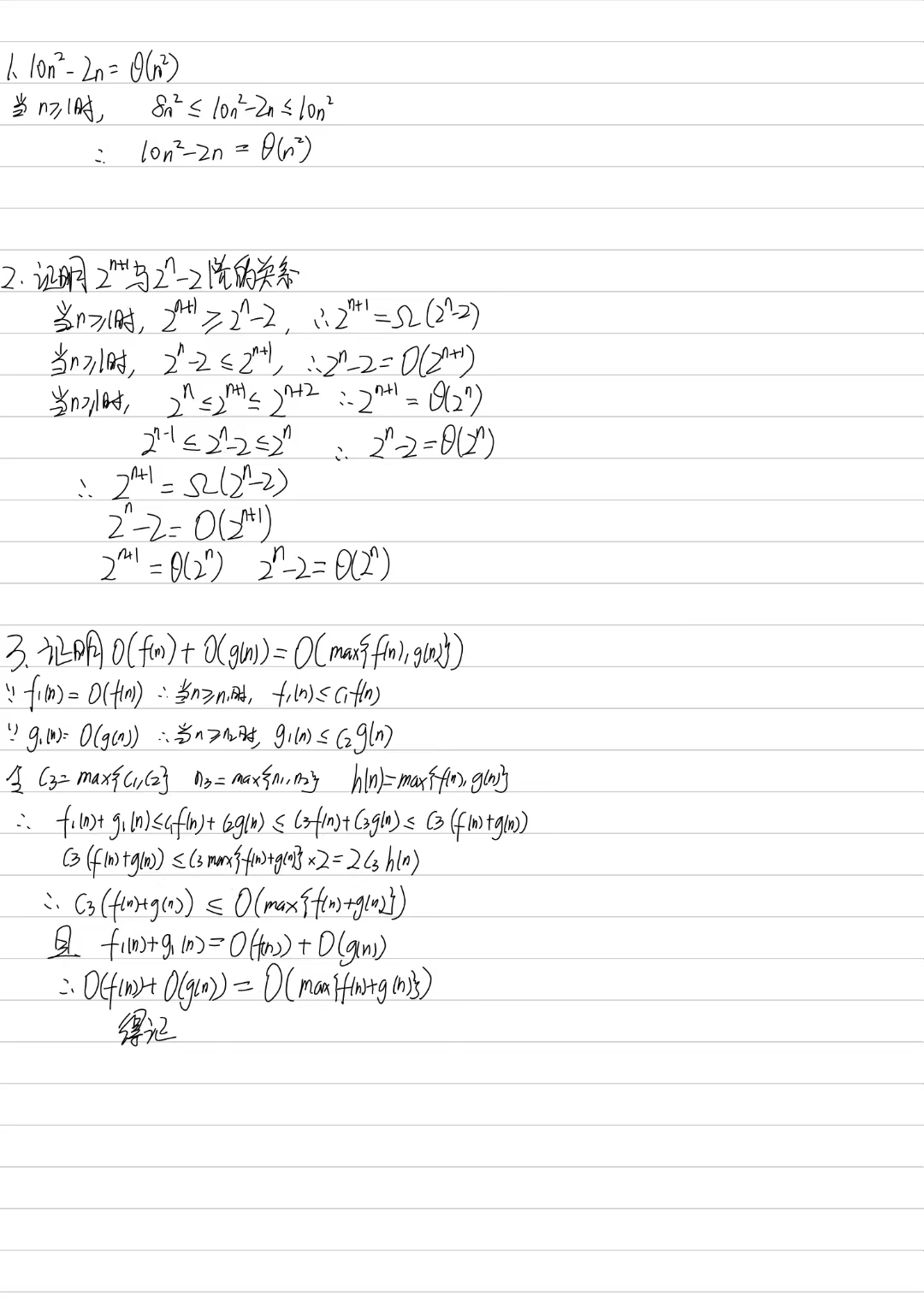
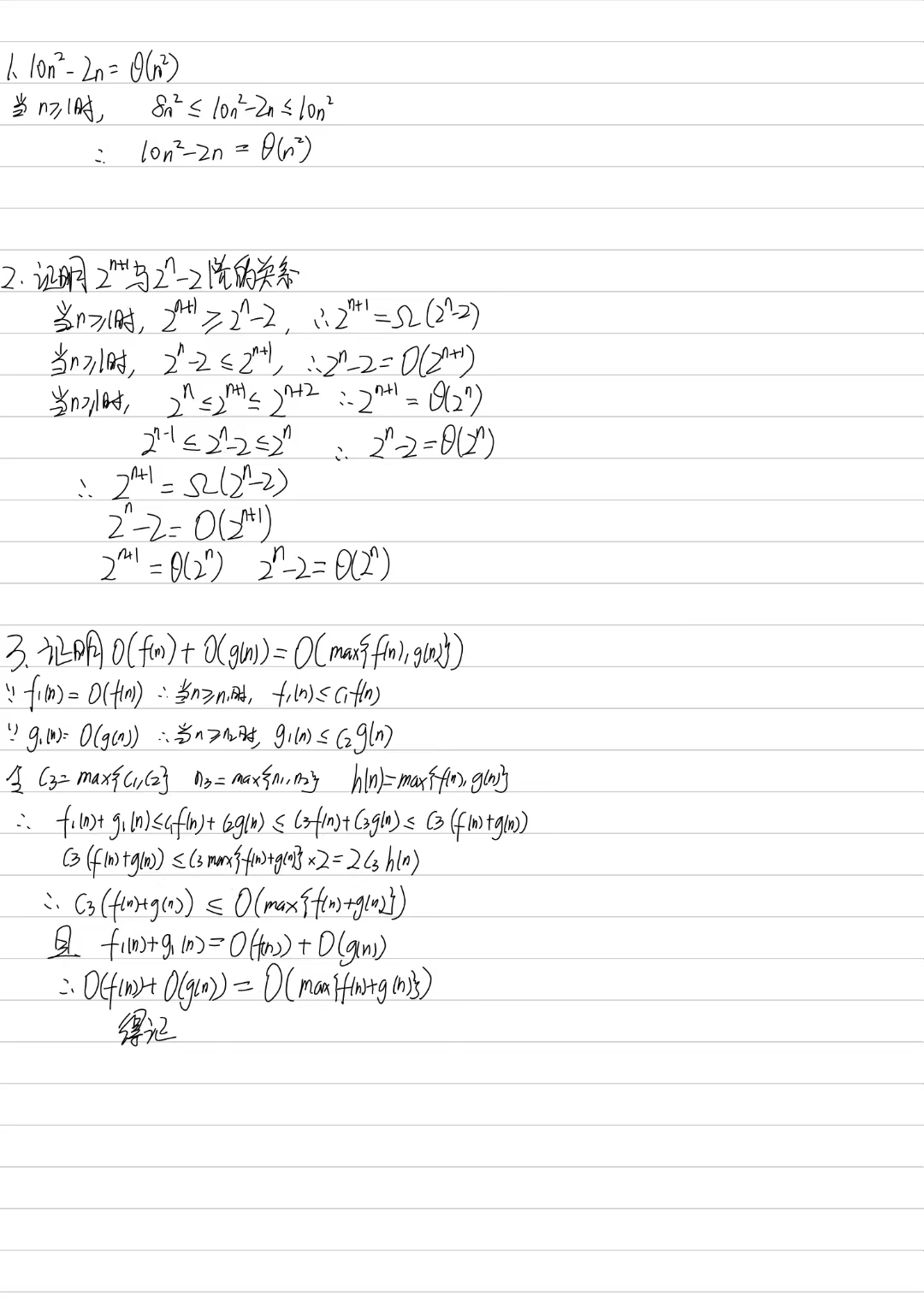
**一、证明**

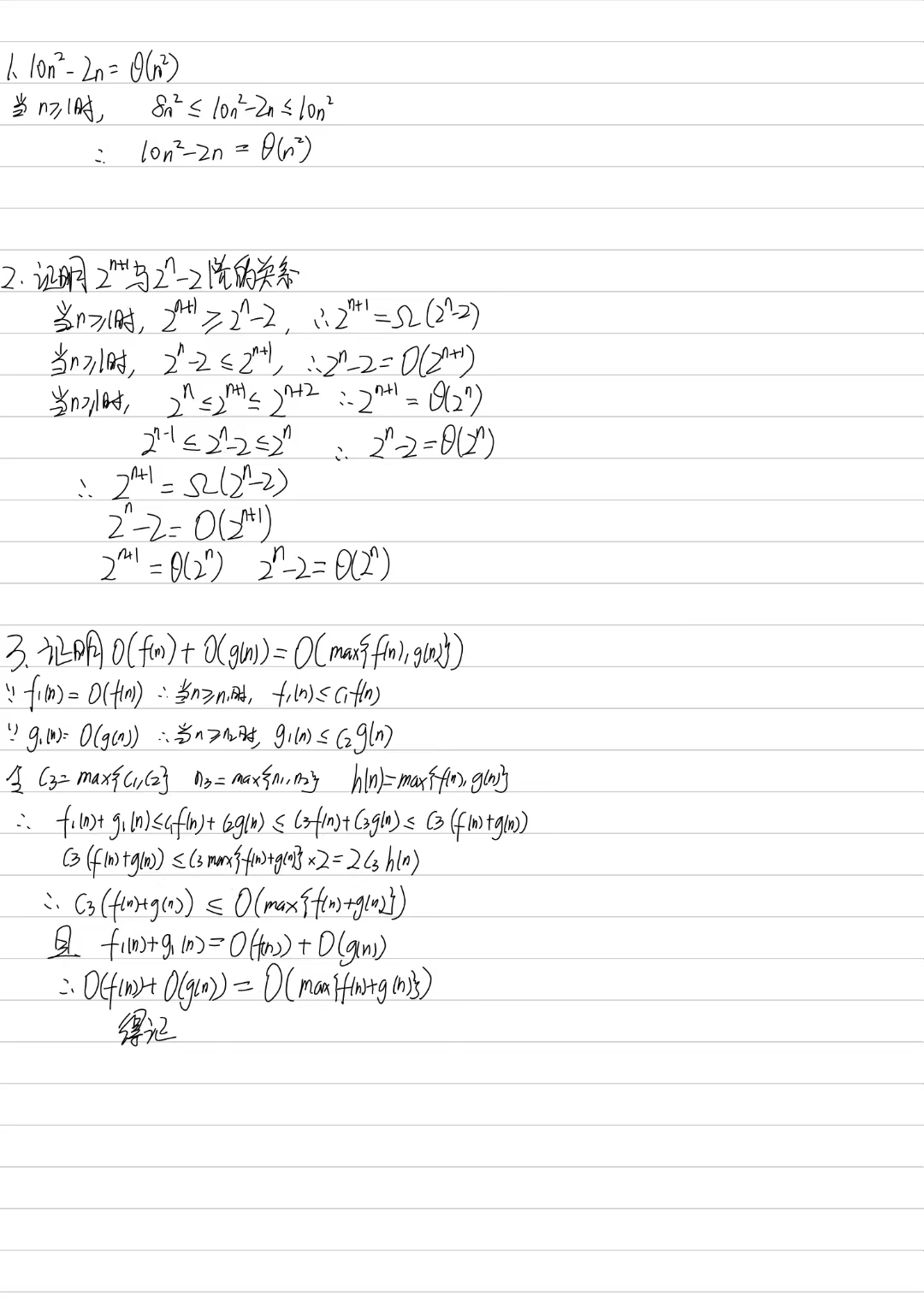
**1、**



**2、证明2阶的关系**



**3、证明**



**二、算法设计与编码实现（要求写出1、设计思想2、源码3、测试运行截图4、分析算法性能）**

**1、有一个含有n个元素的数组A，判断其中是否存在出现次数超过所有元素个数一半的元素；**

{

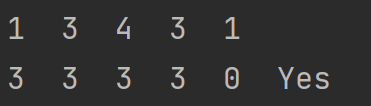
1. 设计思想

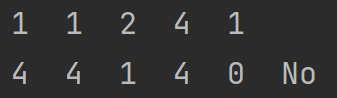
定义足够大的数组t，循环中待测数组A的值即为数组t的地址下标，每访问一次，该地址自增1，每次循环判断访问的数组t的地址的值是否大于数组A的所有元素个数的一半，若为真，跳出循环返回值1，否则待循环结束后返回值0

1. 源码

#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
  
#define n 10  
int array[n];  
  
int init(int a[]) {  
 srand((unsigned) time(NULL));  
 for (int i = 0; i < n; ++i) {  
 array[i] = rand() % 5;  
 }  
}  
  
int check(int a[]) {  
 int t[n] = {0};  
 int i = 0;  
 for (i; i < n; i++) {  
 t[a[i]]++;  
 if (t[a[i]] > (n / 2)) {  
 break;  
 }  
 }  
 if (i < n - 1) return 1;  
 else  
 return 0;  
}  
  
int main() {  
 init(array);  
 for (int i = 0; i < n; ++i) {  
 if (i % 5 == 0)  
 puts("");  
 printf("%d ", array[i]);  
 }  
 if (check(array))  
 printf("Yes");  
 else printf("No");  
 return 0;  
}

1. 测试运行截图





1. 分析算法性能

时间复杂度：O(n)

空间复杂度：O(1)

}

**2、一个字符串中含有若干'\*'字符，设计一尽可能高效的算法，将'\*'字符全部移到最前面，其他字符的先后顺序不能改变。**

**如输入：‘s\*\*tu\*de\*\*\*nt’；**

**输出‘\*\*\*\*\*\*student’**

{

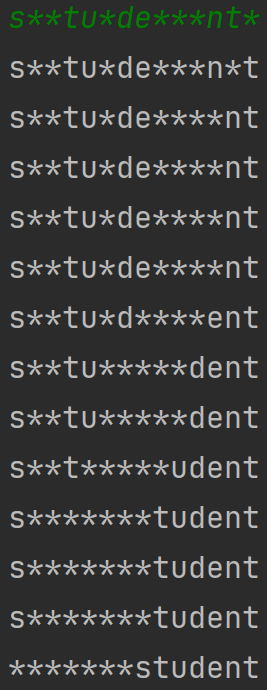
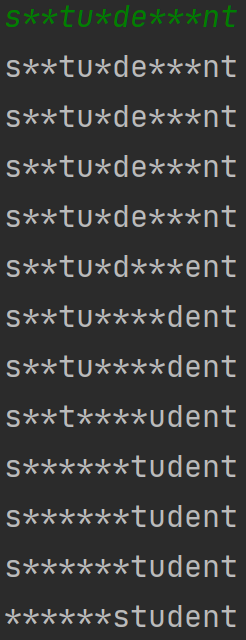
1. 设计思想

定义指针i，j，从字符串尾部开始遍历，如果i地址所在值不为\*，i自减1，若为\*，地址i和地址j的值互换，且每一次循环指针j都自减1.

1. 源码

#include <stdio.h>  
#include <string.h>  
  
#define n 20  
  
int sort(char s1[]) {  
 int i, j;  
 char temp;  
 for (i = strlen(s1) - 1, j = i; j >= 0;) {  
 if (s1[i] == '\*') {  
 temp = s1[i];  
 s1[i] = s1[j];  
 s1[j] = temp;  
 if (s1[i] != '\*')  
 i--;  
 j--;  
 }  
 if (s1[i] != '\*') {  
 i--;  
 j--;  
 }  
 }  
}  
  
int main() {  
 char str1[n];  
 scanf("%s", str1);  
 if (sort(str1))  
 printf("%s",str1);  
 else  
 printf("No");  
}

1. 测试运行截图



1. 分析算法性能

时间复杂度：O(n)

空间复杂度：O(1)

}

**3、如果字符串t是字符串s的后面若干字符串循环右移得到的，称s和t的旋转词。比如"abcdef"和"efabcd" 是旋转词；"abcdef"和"feabcd"不是旋转词。设计算法，判断输入的两个字符串是否为旋转词。**

{

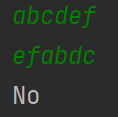
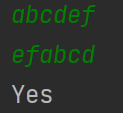
1. 设计思想

首先判断字符的t和s长度是否相等，若否，返回值0；定义两个指针i和j，遍历字符串s，找到与字符串t第一个值相等的值的地址下标，赋值为j；循环遍历字符串t，与字符串s的j地址的值对比，若不同，跳出循环返回值0；每一次循环，i自增1，j通过循环队列思想为(j+1)%L（L为字符串长度）

1. 源码

#include <stdio.h>  
#include <string.h>  
#define n 20  
int judge(char s1[], char s2[]) {  
 int i = 0, j = 0;  
 if (strlen(s1) != strlen(s2)) {  
 return 0;  
 }  
 int count = strlen(s1);  
 while (1) {  
 if (s1[i] == s2[j]) break;  
 if (j == count) break;  
 j++;  
 }  
 for (; i < count;) {  
 if (s1[i] != s2[j])break;  
 i++;  
 j = (j + 1) % count;  
 }  
 if (i == count)  
 return 1;  
 else  
 return 0;  
}  
  
int main() {  
 char str1[n], str2[n];  
 scanf("%s", str1);  
 scanf("%s", str2);  
 if (judge(str1, str2))  
 printf("Yes");  
 else  
 printf("No");  
}

1. 测试运行截图



1. 分析算法性能

时间复杂度：O(m+n)

空间复杂度：O(1)

}

**4、 数字排序：给定n（n不超过1000）个整数，统计每个整数出现的频率，并按频率大小排序输出。**

**输入：10 //整数个数n**

**3 4 1 2 6 3 3 4 1 3**

**输出： 3 4 //3出现4次**

**1 2**

**4 2**

**2 1**

**6 1**

{

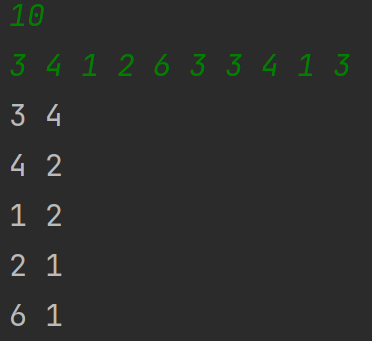
1. 设计思想

给定数组后，定义新二维数组t，第一列为值，第二列为频率，将其遍历后，定义新二维数组p，从数组t中获取有效值，以第二列的值（频率）为关键值进行冒泡排序，最后输出

1. 源码

#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <string.h>  
  
#define N 1000  
int sort(int odd[][2], int random[], int \*n, int p[][2]) {  
 for (int i = 0; i < \*n; i++) {  
 odd[random[i]][0] = random[i];  
 odd[random[i]][1]++;  
 }  
 int count=0;  
 for (int i = 0; i < \*n;i++) {  
 if (odd[random[i]][0] != -1) {  
 p[count][0] = odd[random[i]][0];  
 p[count][1] = odd[random[i]][1];  
 odd[random[i]][0] = -1;  
 count++;  
 }  
 }  
 int temp = 0;  
 (\*n)=count;  
 for (int i = 0; i < \*n - 1; i++) {  
 for (int k = 0; k < \*n - 1 - i; k++) {  
 if (p[k][1] < p[k + 1][1]) {  
 temp = p[k][1];  
 p[k][1] = p[k + 1][1];  
 p[k + 1][1] = temp;  
 temp = p[k][0];  
 p[k][0] = p[k + 1][0];  
 p[k + 1][0] = temp;  
 }  
 }  
 }  
}  
  
  
int main() {  
 int n;  
 scanf("%d", &n);  
 int odd[N][2] = {0}, random[n];  
 int put[N][2] = {0};  
 for(int i=0;i<n;i++){  
 scanf("%d",&random[i]);  
 }  
 sort(odd, random, &n, put);  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < 2; j++) {  
 printf("%d ", put[i][j]);  
 }  
 puts("");  
 }  
 return 0;  
}

1. 测试运行截图



1. 分析算法性能

时间复杂度：O(2\*m + n) <O<O(2\*m+n^2) ，n<=m

空间复杂度：O(1)

}