2-3

#include <bits/stdc++.h>

using namespace std;

/\*

    强化版二分搜索：如果该数组不包含该元素，返回最接近的两个元素（一个大于一个小于）

\*/

pair<int,int> ImprovedBinarySearch(vector<int>&a, int begin,int end,int x,int i,int j)

{

    if (begin == end)

    {

        if (a[begin]==x) return {begin,begin};

        else if (a[begin]<x) return {begin,j};

        else return {i,begin};

    }

    int mid = (begin+end)/2;

    if(a[mid]==x) return {mid,mid};

    else if (a[mid]<x) return ImprovedBinarySearch(a,mid+1,end,x,mid,j);

    else return ImprovedBinarySearch(a,begin,mid,x,i,mid);

}

int main()

{

    vector<int> a;

    pair<int,int> ans;

    int c;

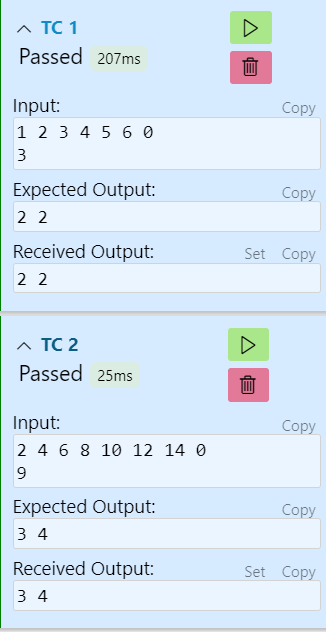
    while (cin >> c && c>0) a.push\_back(c);

    cin >> c;

    ans = ImprovedBinarySearch(a,0,a.size()-1,c,0,a.size()-1);

    cout << ans.first <<" " <<  ans.second;

}



2-4

#include <bits/stdc++.h>

using namespace std;

/\*

    仅对u进行分治，那么由分治法有：

    T(n,1) = O(n)

    T(n,m) = 3T(n,m/2)+O(n)

    易求得T(n,m) = O(nm^log(3/2))

    下面是算法实现

\*/

vector<int> BigNumberMultiple(vector<int> &u,vector<int> &v,int m,int n)

{

    vector<int> ans;

    if (m==1)

    {

        //求出反过来的乘积。

        int addflag=0;

        for (int i=n-1;i>=0;i--)

        {

            int temp = u[i]\*v[i]+addflag;

            addflag = temp/10;

            ans.push\_back(temp%10);

        }

        if(addflag !=0)

        ans.push\_back(addflag);

        //将乘积颠倒过来

        for (int i=0;i<ans.size()/2;i++)

        {

            swap(ans[i],ans[n-1-i]);

        }

    }

    else

    {

        // 求出反过来的乘积

        vector<int> u1,u2;

        for (int i=0;i<m;i++)

       {

            if(i<m/2) u1.push\_back(u[i]);

            else u2.push\_back(u[i]);

       }

       vector<int> la = BigNumberMultiple(u2,v,u2.size(),n);

       vector<int> ha = BigNumberMultiple(u1,v,u1.size(),n);

       for (int i=0;i<la.size();i++)    ans.push\_back(la[i]);

       for (int i=0;i<ha.size();i++)

       {

        if(i+la.size()>=ans.size()) ans.push\_back(ha[i]);

        else

        {

            ans[i+la.size()]+=ha[i];

            if(ans[i+la.size()]>=10)

            {

                if(i+1+la.size()>=ans.size())

                    ans.push\_back(1);

                else ans[i+1+la.size()]++;

            }

        }

       }

       //将乘积颠倒过来

        for (int i=0;i<ans.size()/2;i++)

        {

            swap(ans[i],ans[n-1-i]);

        }

    }

    return ans;

}

2-5

/\*

再用分治法求两个n位大整数u和v的乘积时，将u和v都分割成长度位n/3位的3段。证明可以用5次n/3位整数的乘法求得uv的值。

按此思想设计一个求两个大整数的分治算法，并分析算法的计算复杂性。

\*/

/\*

我们假设x = 10^(n/3)，将u,v分解成(A,B,C)和(D,E,F)，那么原来的式子就变成：

product = (Ax^2+Bx+C)(Dx^2+Ex+F) = ax^4+bx^3+cx^2+dx+e = f(x)

很显然这是一个四次方的式子，我们要求f(10^(n/3))因此我们只需要知道五个值就可以确定f(x)，自然可以求值。

因此我们选择f(0),f(1),f(2),f(-1),f(-2)，把这些算出来即可。

因此记：

S1 = f(1) = (A+B+C)(D+E+F) = a+b+c+d+e

S2 = f(2) = (4A+2B+C)(4D+2E+F) = 16a+8b+4c+2d+e

S3 = f(-1) = (A-B+C)(D-E+F) = a-b+c-d+e

S4 = f(2) = (4A+2B+C)(4D+2E+F) = 16a-8b+4c-2d+e

S5 = f(0) = CF = e

那么有：

a = (S3+S4-4S1-4S2-6S5)/24

b = (S3-S4-2S1+2S2)/12

c = (16S1+16S2-S3-S4+30S5)/24

d = (8S1-8S2-S3+S4)/12

e = S5

随后带入x = 10^(n/3)即可。

由于其合并时间复杂度是O(n)，依照主定理，该算法的时间复杂度应该满足：

T(n) = 5(Tn/3)+O(n) T(n) = n^log3(5)

\*/

2-8

/\*

交换数组问题：设a是一个有n个元素的数组，k(0<=k<=n-1)。

试设计一个算法将子数组a[0:k-1]和a[k:n-1]换位。

要求算法在最坏情况下耗时O(n)，且只用到O(1)的辅助空间。

\*/

/\*

    使用手摇算法：将两部分数组分别反转，再将整个数组反转。

\*/

#include <bits/stdc++.h>

using namespace std;

int main()

{

    vector<int> a;

    int n;

    int c;

    int k;

    cin >> n;

    for (int i=0;i<n;i++)

    {

        cin >> c;

        a.push\_back(c);

    }

    cin >> k;

    for (int i=0;i<k/2;i++)

    {

        swap(a[i],a[k-1-i]);

    }

    for(int i=0;i<(n-k)/2;i++)

    {

        swap(a[k+i],a[n-1-i]);

    }

    for (int i=0;i<n/2;i++)

    {

       swap(a[i],a[n-1-i]);

    }

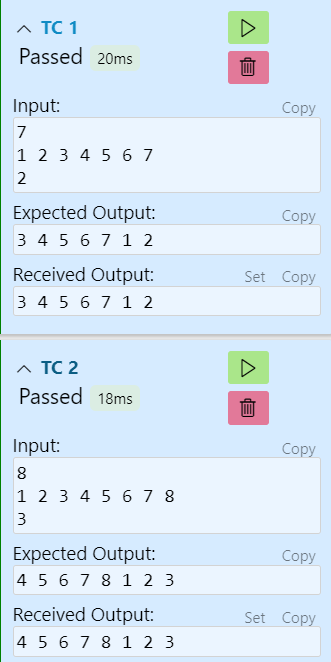
    for (int i=0;i<a.size();i++)

    {

        cout << a[i] << ' ';

    }

}



2-9

/\*

设子数组a[0~k-1]和a[k~n-1]已经排好序,设计一个合并这两个子数组为排好序的数组a[0~n-1]的算法。

要求算法在最坏情况下所用的计算时间为O(n)，且只用到O(1)的辅助空间。

\*/

/\*

使用原地归并排序, 使用两个下标i,j分别遍历两个子数组。当a[i]<=a[j]时，直接i++；当a[i]>a[j]时，

设一个变量move=0，当a[j]<a[i]时，j++,move++;最后交换a[i~j-move-1]和a[j-move~j-1],然后，i+=move，

终止条件为i==n-1

\*/

#include <bits/stdc++.h>

using namespace std;

void IterMergeSort(vector<int> a,int n,int k)

{

    int i=0,j=k;

    while(i<n-1)

    {

        if(a[i]<=a[j])

        {

            i++;

            continue;

        }

        int move = 0;

        while(a[j]<a[i] && j<n)

        {

            j++;

            move++;

        }

        //手摇算法

        for (int k=0;k<(j-move-i)/2;k++)

        {

            swap(a[i+k],a[j-move-1-k]);

        }

        for (int k=0;k<move/2;k++)

        {

            swap(a[j-move+k],a[j-1-k]);

        }

        for (int k=0;k<(j-i)/2;k++)

        {

            swap(a[i+k],a[j-1-k]);

        }

        i+=move;

    }

    for (int i=0;i<n;i++)

    {

        cout << a[i] << " ";

    }

}

int main()

{

    int n;

    cin >> n;

    vector<int> arr;

    for (int i=0;i<n;i++)

    {

        int a;

        cin >> a;

        arr.push\_back(a);

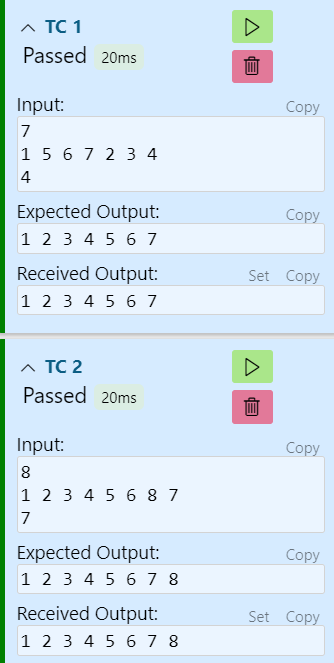
    }

    int k;

    cin >> k;

    IterMergeSort(arr,n,k);

}



附加题

/\*

设T[1:n]是一个含有n个元素的数组。如果元素x的出现次数超过n/2，称元素x为数组T的主元素。

(1)如果这n个元素存在序关系，比如n个整数

(2)如果这n个元素不存在序关系，比如n个坐标。

请分别针对上述两种情况，分别设计时间复杂性为O(n)的分治算法，判断该数组里是否有主元素。

\*/

/\*

思路：通常分治算法的话这里的复杂度为O(nlogn)，因此采用BV投票法。

下面是一个例子,假设是整数（坐标同理）

\*/

#include <bits/stdc++.h>

using namespace std;

int main()

{

    pair<int,int> ans = {0,0};

    int n,num;

    cin >> n;

    vector<int> arr(n);

    for (int i=0;i<n;i++)

    {

        cin >> num;

        arr[i] = num;

        if (ans.second == 0)

        {

            ans.first = num;

            ans.second =1;

        }

        else

        {

            if (ans.first == num)   ans.second++;

            else ans.second--;

        }

    }

    int cnt=0;

    for (int i=0;i<n;i++)

    {

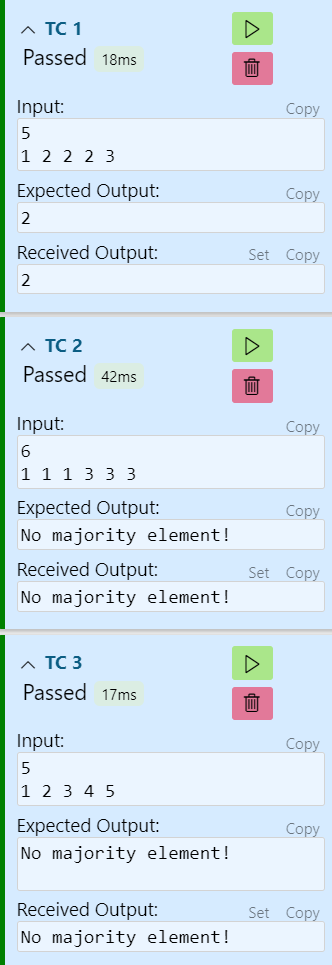
        if (arr[i]==ans.first) cnt++;

    }

    if (cnt > n/2) cout <<  ans.first;

    else cout << "No majority element!";

}



算法实现题 2-1

/\*

问题描述：给定含有n个元素的多重集合S，每个元素在S中出现的次数成为该元素的重数。

多重集S中重数最大的元素成为众数。

算法设计：对于给定的由n个自然数组成的多重集S，计算S的众数及其重数。

数据输入：输入数据由文件名为input.txt的文本文件提供，文件的第一行为多重集S中的元素个数n；在接下来的n行中，每行有一个自然数。

结果输出：将计算结果输出到文件output.txt。输出文件有两行，第一行是众数，第二行是重数。

\*/

/\*

用哈希：每输入一个多重集元素，将哈希表中对应的键的值+1；最后遍历哈希表，找到重数最大的那个。

\*/

#include <iostream>

#include <fstream>

#include <unordered\_map>

#include <vector>

void find\_mode(const std::string& input\_file, const std::string& output\_file) {

    std::ifstream infile(input\_file);

    std::ofstream outfile(output\_file);

    if (!infile.is\_open() || !outfile.is\_open()) {

        std::cerr << "Error opening file." << std::endl;

        return;

    }

    int n;

    infile >> n; // 读取元素个数

    std::unordered\_map<int, int> frequency;

    int element;

    for (int i = 0; i < n; ++i) {

        infile >> element;

        frequency[element]++;

    }

    int mode = -1;

    int max\_count = 0;

    for (const auto& pair : frequency) {

        if (pair.second > max\_count) {

            max\_count = pair.second;

            mode = pair.first;

        }

    }

    outfile << mode << std::endl;

    outfile << max\_count << std::endl;

    infile.close();

    outfile.close();

}

int main() {

    find\_mode("input.txt", "output.txt");

    return 0;

}

算法实现题 2-7

/\*

n个元素的集合可以划分为若干非空子集。

算法设计，给定正整数n，计算出n个元素的集合可以划分为多少个不同的非空子集。

\*/

/\*

    我们可以试着归纳一下：

    n=1 1

    n=2 1+1 = 2

    n=3 1\*2 +2\*1+1\*1 = 5

    n=4 1\*5 + 3\*2 +3\*1 +1\*1 = 15

    可以归纳出：

    n = k 为 C(n-1,0)a\_(n-1) + C(n-1,1)a\_(n-2) + ... + C(n-1,n-2)a\_1 + C(n-1,n-1)

    验证:

    n=5 1\*15+4\*5+6\*2+4\*1+1 = 52

    所以这是一个简单的动态规划。

 \*/

#include <bits/stdc++.h>

using namespace std;

int FindAllSubset(vector<int>ans,int n)

{

    ans.push\_back(1);

    ans.push\_back(2);

    for (int i=3;i<=n;i++)  //i表示ans对应的数

    {

        int sum=0;

        for (int j=0;j<=i-1;j++) //累计求和

        {

            int num = 1;

            for (int k=0;k<j;k++) //求组合数C(i-1,j)

            {

                num \*= (i-1-k);

            }

            for (int k=0;k<j;k++)

            {

                num /= j-k;

            }

            if(j==i-1) sum += num;

            else sum += num \* ans[i-j-2];

        }

        ans.push\_back(sum);

    }

    return ans[n-1];

}

int main()

{

    vector<int> ans;

    std::ifstream infile("input.txt");

    int n;

    infile >> n; // 读取第一行的数n

    int a=FindAllSubset(ans,n);

    ofstream o;

    o.open("output.txt",ios::out);

    o << a << endl;

}