2-3 设 a[0:n−1]是已排好序的数组。请改写二分搜索算法，使得当搜索元素 x 不在数组中时，返回小于 x 的最大元素位置 i 和大于 x 的最小元素位置 j。当搜索元素在数组中时，i 和 j 相同，均为 x 在数组中的位置。

答： 改写后的二分搜索算法如下：

```cpp
#include <iostream>
#include <vector>
using namespace std;

pair<int, int> modifiedBinarySearch(const vector<int>& a, int x) {
    int low = 0, high = a.size() - 1;
    int i = -1, j = a.size(); //i初始为-1（无更小元素），j初始为数组长度（无更大元素）

    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (a[mid] == x) {
            i = j = mid;
            break;
        }
        else if (a[mid] < x) {
            i = mid; //更新小于x的最大元素位置
            low = mid + 1;
        }
        else {
            j = mid; //更新大于x的最小元素位置
            high = mid - 1;
        }
    }
    return { i, j };
}
```

2-4 给定两个大整数 u 和 v，它们分别有 m 和 n 位数字，且 $m \leq n$。用通常的乘法求 uv 的值需要 $O(mn)$时间。可以将 u 和 v 均看作有 n 位数字的大整数，用本章介绍的分治法，在 $O(n^{\log 3})$时间内计算 uv 的值。当 m 比 n 小得多时，用这种方法就显得效率不够高。试设计一个算法，在上述情况下用 $O(nm^{\log(3/2)})$时间求出 uv 的值。

答： 当 m 比 n 小得多时，将 v 分成 n/m 段，每段 m 位。计算 uv 需要 n/m 次 m 位乘法运算。每次 m 位乘法可以用主教材中的分治法计算，耗时 O(m^log3)。因此，算法所需的计算时间为 O((n/m)m^log3)=O(nm^(log(3/2)))

2-5 在用分治法求两个 n 位大整数 u 和 v 的乘积时，将 u 和 v 都分割为长度为 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乘积的分治算法，并分析算法的计算复杂性（提示：n 位的大整数除以一个常数 k 可以在 $\theta(n)$时间内完成。符号 θ 所隐含的常数可能依赖于 k）。

答： 将 u 从左到右分为 abc 三段，v 从左到右分为 def 三段，由此可以分别得到 u，v 用 abc 和 def 表示的式子：u=a*10^(2n/3)+b*10^(n/3)+c，v=d*10^(2n/3)+e*10^(n/3)+f，其中 a,b,c,d,e,f 均为 n/3 位整数。

构造以下 5 次 n/3 位整数乘法：

P1=a*d

P2=c*f

P3=(a+b)*(d+e)

P4=(b+ c)*(e+ f)

P5=(a+c)*(d+f)

由此可以得到 uv=P1*10^(4n/3)+(P3-P1-P2)*10^(3n/3)+(P5-R3-P4+P1+P2)*10^(2n/3)

+(P4-P2-P1)*10(n/3)+P2

所以通过五次 n/3 位整数乘法即可求得 uv。

复杂度：T(n)=5T(n/3)+θ(n),由主定律计算得时间复杂度为 $O(n^{\log_3 5})$

2-8 设 a[0:n−1]是有 n 个元素的数组，k（0≤k≤n−1）是一个非负整数。试设计一个算法将子数组 a[0:k−1]与 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。

答：先将 0-(k-1)部分逆置，再将 k-n 部分逆置，最后将整个数组逆置即可将子数组逆置且耗时为 O(n),且只用 O(1)的空间

算法程序如下：

```cpp
#include <iostream>
#include <vector>
using namespace std;

void reverse(vector<int>& arr, int left, int right) {//逆置数组指定区间[left, right]
    while (left < right) {
        swap(arr[left], arr[right]);
        left++;
        right--;
    }
}
//交换子数组
void exchangeSubarrays(vector<int>& arr, int k) {
    int n = arr.size();
    reverse(arr, 0, k - 1);
    reverse(arr, k, n - 1);
    reverse(arr, 0, n - 1);
}

int main() {
    vector<int> arr = { 1, 2, 3, 4, 5 };
    int k = 2;//例如arr.size=5,k=2
    exchangeSubarrays(arr, k);
    for (int num : arr) {
        cout << num << " ";
    }
    return 0;
}
```

```
Microsoft Visual Studio 调试控
3 4 5 1 2
C:\Users\amun\source\repos\jifengsai\x64\Debug\jife
按任意键关闭此窗口. . .
```

2-9 设子数组 $a[0:k-1]$和 $a[k:n-1]$已排好序（$0 \leqslant k \leqslant n-1$）。试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。

答：算法设计：

1.定义三个指针，i 指向第一个子数组(a[0:k-1])的起始位置，初始值为 0;j 指向第二个子数组(a[k:n-1])的起始位置，初始值为 k; t 指向临时存储合并结果的数组的起始位置即 a[0]。

2.比较 a[i]和 a[j]的大小，将较小的元素放入原数组的 t 位置，然后将对应指针后移一位 t 也后移一位。

3.重复步骤 2，直到其中一个子数组的元素全部被处理完。将另一个子数组中剩余的元素依次复制到原数组的剩余位置

程序如下：

```cpp
vector<int> mergeArrays(vector<int>& a, int k) {
    int n = a.size();
    int i = 0, j = k, t = 0;
    vector<int> temp(n);

    while (i < k && j < n) {
        if (a[i] <= a[j]) {
            temp[t] = a[i];
            i++;
        }
        else {
            temp[t] = a[j];
            j++;
        }
        t++;
    }
    while (i < k) {
        temp[t] = a[i];
        i++;
        t++;
    }
    while (j < n) {
        temp[t] = a[j];
        j++;
        t++;
    }
    for (int i = 0; i < n; i++) {
        a[i] = temp[i];
    }
    return a;
}
```

复杂度分析：时间上，最坏情况是遍历整个数组也就是 O(n),空间是只借用了一个 temp 变量来存储临时数据空间复杂度为 O(1)。

2-1 众数问题。

问题描述： 给定含有 n 个元素的多重集合 S，每个元素在 S 中出现的次数称为该元素的重数。多重集 S 中重数最大的元素称为众数。例如，$S=\{1, 2, 2, 2, 3, 5\}$。多重集 S 的众数是 2，其重数为 3。

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 1 行为多重集 S 中元素个数 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 2 行，第 1 行是众数，第 2 行是重数。

输入文件示例	输出文件示例
input.txt	output.txt
6	2
1	3
2	
2	
2	
3	
5	

答： 算法设计：

通过 unordered_map 来统计每个自然数在多重集中出现的次数，键为自然数，值为出现的次数。然后再遍历 unordered_map 找到出现最多的元素和它出现的次数即为众数。

代码如下：

```cpp
#include <iostream>
#include <fstream>
#include <unordered_map>
using namespace std;
pair<int, int> findModeAndFrequency(unordered_map<int, int>& countMap) {
    int mode = 0, maxFreq = 0;
    for (const auto& pair : countMap) {
        if (pair.second > maxFreq) {
            mode = pair.first;
            maxFreq = pair.second;
        }
    }
    return { mode, maxFreq };
}

int main() {
    ifstream inputFile("input.txt");
    ofstream outputFile("output.txt");
    int n;
    inputFile >> n;
    unordered_map<int, int> countMap;
    int num;
    for (int i = 0; i < n; ++i) {
        inputFile >> num;
        countMap[num]++;
    }
    pair<int, int> result = findModeAndFrequency(countMap);
    outputFile << result.first << endl;
    outputFile << result.second << endl;
    inputFile.close();
    outputFile.close();

    return 0;
}
```

2-7 集合划分问题。

问题描述：n 个元素的集合$\{1, 2, \cdots, n\}$可以划分为若干非空子集。例如，当 n=4 时，集合$\{1, 2, 3, 4\}$可以划分为 15 个不同的非空子集如下：

$\{\{1\}, \{2\}, \{3\}, \{4\}\}$	$\{\{1, 3\}, \{2, 4\}\}$
$\{\{1, 2\}, \{3\}, \{4\}\}$	$\{\{1, 4\}, \{2, 3\}\}$
$\{\{1, 3\}, \{2\}, \{4\}\}$	$\{\{1, 2, 3\}, \{4\}\}$
$\{\{1, 4\}, \{2\}, \{3\}\}$	$\{\{1, 2, 4\}, \{3\}\}$
$\{\{2, 3\}, \{1\}, \{4\}\}$	$\{\{1, 3, 4\}, \{2\}\}$
$\{\{2, 4\}, \{1\}, \{3\}\}$	$\{\{2, 3, 4\}, \{1\}\}$
$\{\{3, 4\}, \{1\}, \{2\}\}$	$\{\{1, 2, 3, 4\}\}$
$\{\{1, 2\}, \{3, 4\}\}$	

算法设计：给定正整数 n，计算出 n 个元素的集合$\{1, 2, \cdots, n\}$可以划分为多少个不同的非空子集。

数据输入：由文件 input.txt 提供输入数据。文件的第 1 行是元素个数 n。

结果输出：将计算出的不同的非空子集数输出到文件 output.txt。

输入文件示例	输出文件示例
input.txt	output.txt
5	52

答： 算法设计：

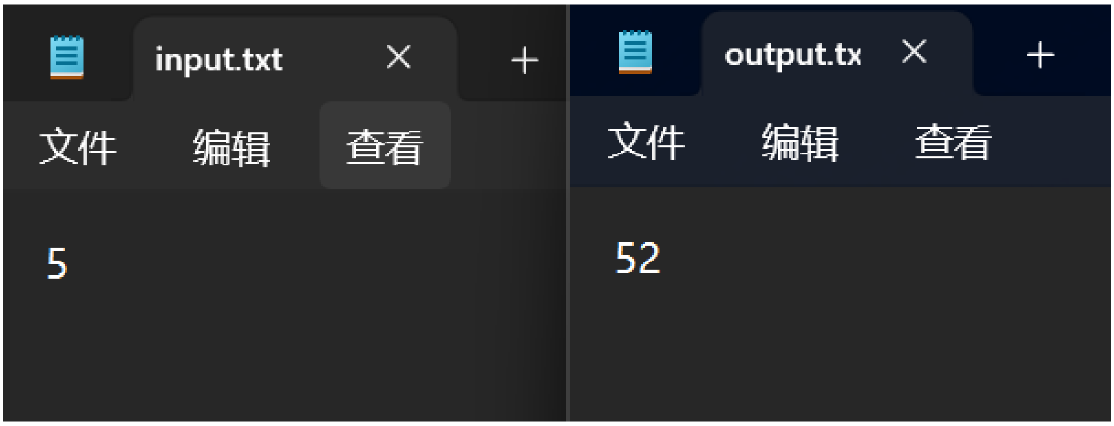
由题可知求的是 bell 数，其递归式为： $B(n)=\sum_{i=0}^{n}\begin{vmatrix} n-1 \\ i \end{vmatrix} B(i);\ B(0)=1$

得到代码实现如下：

```cpp
#include <iostream>
#include <fstream>
#include <unordered_map>
#include <vector>
using namespace std;

int combination(int n, int k) {//计算组合数
    if (k == 0 || k == n) return 1;
    int numerator = 1, denominator = 1;
    for (int i = 1; i <= k; ++i) {
        numerator *= (n - k + i);
        denominator *= i;
    }
    return numerator / denominator;
}
int bellNumber(int n) {//计算贝尔数
    vector<int> bell(n + 1, 0);
    bell[0] = 1;
    for (int i = 1; i <= n; ++i) {
        for (int j = 0; j < i; ++j) {
            bell[i] += combination(i - 1, j) * bell[j];
        }
    }
    return bell[n];
}

int main() {
    ifstream inputFile("input.txt");
    ofstream outputFile("output.txt");
    int n;
    inputFile >> n;
    int result = bellNumber(n);
    outputFile << result << endl;
    inputFile.close();
    outputFile.close();
    return 0;
}
```



补充题：

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。

答：算法分析：

(1) 将数组对半分解，分别递归的寻找两个子数组的主元素，在每个子数组中，因为元素有序，可以通过一次遍历统计每个候选主元素的出现次数，判断是否为该子数组的主元素。如果两个子数组都有主元素，比较两个主元素在整个数组中的出现次数，出现次数超过 n/2 的就是整个数组的主元素；如果只有一个子数组有主元素，判断该主元素在整个数组中的出现次数是否超过 n/2；如果两个子数组都没有主元素，那么整个数组也没有主元素。

代码实现：

```cpp
int order_cnt(const vector<int>& arr, int element, int left, int right) {//统计元素在数组指定范围内的出现次数
    int count = 0;
    for (int i = left; i <= right; ++i) {
        if (arr[i] == element) {
            count++;
        }
    }
    return count;
}

int order_find(const vector<int>& arr, int left, int right) {//分治查找主元素
    if (left == right) {
        return arr[left];
    }
    int mid = left + (right - left) / 2;
    int leftMajor = order_find(arr, left, mid);
    int rightMajor = order_find(arr, mid + 1, right);

    int leftCount = order_cnt(arr, leftMajor, left, right);
    if (leftCount > (right - left + 1) / 2) {
        return leftMajor;
    }

    int rightCount = order_cnt(arr, rightMajor, left, right);
    if (rightCount > (right - left + 1) / 2) {
        return rightMajor;
    }

    return -1;  // 表示没有找到主元素
}
```

(2) 将数组分成两个子数组 T[1:n/2]和 T[n/2+1:n]，分别递归地在两个子数组中寻找可能的主元素，在每个子数组中，利用类似摩尔投票法的方式找出一个候选主元素(在遍历子数组时，记录当前候选元素和其票数，遇到相同元素票数加一，不同元素票数减一，票数为 0 时更换候选元素)。得到两个子数组的候选主元素后，分别统计它们在整个数组中的出现次数，出现次数超过 n/2 的就是整个数组的主元素，如果都不超过，则没有主元素。

代码实现如下：

```cpp
int unorder_find(const vector<int>& arr, int left, int right) {//分治查找主元素（无序列表）
    if (left == right) {
        return arr[left];
    }
    int mid = left + (right - left) / 2;
    int leftCandidate = unorder_find(arr, left, mid);
    int rightCandidate = unorder_find(arr, mid + 1, right);

    int leftCount = 0;
    for (int i = left; i <= right; ++i) {
        if (arr[i] == leftCandidate) {
            leftCount++;
        }
    }
    if (leftCount > (right - left + 1) / 2) {
        return leftCandidate;
    }

    int rightCount = 0;
    for (int i = left; i <= right; ++i) {
        if (arr[i] == rightCandidate) {
            rightCount++;
        }
    }
    if (rightCount > (right - left + 1) / 2) {
        return rightCandidate;
    }

    return -1;  // 表示没有找到主元素
}

bool isFind_unordered(const vector<int>& arr) {
    int result = unorder_find(arr, 0, arr.size() - 1);
    if (result == -1) {
        return false;
    }
    int count = 0;
    for (int element : arr) {
        if (element == result) {
            count++;
        }
    }
    return count > arr.size() / 2;
}
```