算法分析第2次作业

小组编号：23

本次作业负责人:黄勖

1　算法分析题2-3 答案：

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。

输入:

- 已排序数组 `a[0:n-1]`，其中 n 是数组的大小。
- 要搜索的元素 `x`。

输出:

- 如果 `x` 在数组中，返回 `(i, j)` 其中 `i` 和 `j` 都等于 `x` 在数组中的位置。
- 如果 `x` 不在数组中，返回 `(i, j)` 其中 `i` 是小于 `x` 的最大元素的位置，`j` 是大于 `x` 的最小元素的位置。

算法设计:

1. 初始化两个变量 `low` 和 `high`，它们分别代表搜索范围的开始和结束位置。初始时，`low = 0`，`high = n-1`。
2. 初始化两个变量 `last_small` 和 `first_large`，它们分别用于跟踪小于 `x` 的最大元素位置和大于 `x` 的最小元素位置。初始时，`last_small = -1` 和 `first_large = n`。
3. 使用while循环执行以下步骤，直到 `low` 大于 `high`： a. 计算中间位置 `mid`，即 `(low + high) // 2`。 b. 检查 `a[mid]` 与 `x` 的关系：
   - 如果 `a[mid]` 等于 `x`，则返回 `(mid, mid)`，表示找到了 `x`。
   - 如果 `a[mid]` 小于 `x`，则将 `low` 更新为 `mid + 1`，并更新 `last_small` 为 `mid`。
   - 如果 `a[mid]` 大于 `x`，则将 `high` 更新为 `mid - 1`，并更新 `first_large` 为 `mid`。
4. 循环结束后，返回 `(last_small, first_large)`，即符合要求的最终答案。

算法分析:

这个算法和二分搜索性质类似，每执行一次算法的 while 循环，待搜索数组的大小减少一半。对于输入大小 `n` ，在最坏情况下，while 循环执行了 log n 次，循环体内的运算为 O(1) 复杂度。故**时间复杂度为 O(log n)**。

本题分工：小组共同讨论，黄勖编写

2 算法分析题2-4 答案：

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。

算法描述：

1. 由于m ≤ n，将n划分成n/m段，每段vi为m位（i=1，2，……，n/m）
2. 对n中每段vi，计算vi * u（i=1，2，……，n/m）

   Xi = vi ,Y = u对应的数字

   将Xi划分为高低位为m/2位的A和B：Xi = A*2^(m/2) + B

   将Y划分为高低位为m/2位的C和D：Y = C*2^(m/2) + D

   XiY = AC*2^m + ((A-B)(D-C)+AC+BD) * 2^(m/2) +BD
3. 累加各段和

$$uv = \sum_{i=1}^{n/m} XiY * 2^{m(i-1)}$$

算法分析：

每段Xi与Y相乘的时间复杂度为

$$O(m^{log3})$$

要进行n/m次Xi与Y相乘，故时间复杂度为

$$O(\frac{n}{m}m^{log3}) = O(nm^{log(\frac{3}{2})})$$

本题分工：小组共同讨论，李嘉琪编写

3 算法分析题2-5 答案：

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）。

算法证明：

将 u 和 v 分别拆解:

$$u = a \cdot 10^{2(n/3)} + b \cdot 10^{n/3} + c$$

$$v = d \cdot 10^{2(n/3)} + e \cdot 10^{n/3} + f$$

再计算下列表达式:

$$P1 = a \cdot d$$

$$P2 = c \cdot f$$

$$P3 = (a+b) \cdot (d+e)$$

$$P4 = (b+c) \cdot (e+f)$$

$$P5 = (a+b+c) \cdot (d+e+f)$$

$$P6 = P1 + P2 + P3 + P4 - P5$$

以上式子用 5 次 $n/3$ 位整数的乘法求得。

计算 uv 展开式，并将上述表达式代入展开式，得：
$uv = P2 + (P4 - P6 - P2) \cdot 10^{n/3} + (P5 - P3 - P4 + P6 + P6) \cdot 10^{2n/3} + (P3 - P6 - P1) \cdot 10^{3n/3} + P1 \cdot 10^{4n/3}$

证明完毕。

算法分析:

按此分解设计的分治算法需要 5 次 $n/3$ 位整数乘法。分割以及合并所需要的加减法和移位运算时间为 $O(n)$ 。设 $T(n)$ 是算法所需的计算时间，则:

$$T(n) = \begin{cases} O(1), & n = 1 \\ 5T(n/3) + O(n), & n > 1 \end{cases}$$

由此可得

$$T(n) = O(n^{\log_3 5})$$

本题分工：小组共同讨论，黄勖编写

4 算法分析题2-8 答案：

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。

输入:

数组a[0:n-1] ，非负整数k (0 ≤ k ≤ n-1)

输出:

a[k:n-1—0:k-1]，即原数组a[0:k-1—k:n]换位成a[k:n-1—0:k-1]

算法描述:

1. 根据a[0:k-1]有k个元素，a[k:n]有n-k个元素

   若 k > n - k：采用后向循环换位进入步骤2

   若 k ≤ n - k：采用前向循环换位进入步骤3

2. 后向循环换位：

定义temp保存数组末元素a[n-1]

循环n-k次依次将前面数组元素向后移动，即a[j] = a[j - 1] (j=1,2,……,n)

算法结束

3. 前向循环换位：

   定义temp保存数组首元素a[0]

   循环k次依次将后面数组元素向前移动，即a[j - 1] = a[j] (j=1,2,……,n)

   将temp保存在a[n]

   算法结束

算法分析:

使用前向和后向循环需要循环次数为min{k,n-k}，每次循环需要换位n+1次。

故算法需要移动的元素次数为**min{k,n-k}x(n+1)**，时间复杂度为O(n)

特殊情况下k=n/2，计算时间非线性为$O(n^2)$

仅使用一个辅助单元temp，空间复杂度为O(1)

本题分工：小组共同讨论，李嘉琪编写

5 算法分析题2-9 答案：

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。

输入:

- 一个数组 `a`，包含两个已排序的子数组 `a[0:k-1]` 和 `a[k:n-1]`。
- 两个索引值 `k` 和 `n`，表示子数组的分界点。

输出:

- 合并后的排好序的数组 `a[0:n-1]`。

算法设计:

1. 先找到整个数组的最大元素，把它+1，赋值到 `maxn` 变量
2. 用两个指针分别指向两个子数组的第一个元素，用`k`指向当前要插入的位置（k从0开始）
   比较 a[i] % maxn 和 a[j] % maxn 的大小
3. a[k] = a[k] + min(a[i] % maxn, a[j] % maxn) * maxn，且哪个指针指的元素小哪个指针就后移1且每次操作k往后移1。直到i > k - 1或者j > n - 1。
4. 操作完后得到的数组，每个元素除以 `maxn` 后就排序好了

算法分析:

这个算法使用了三个指针来遍历两个子数组并合并它们。它的时间复杂度为 O(n)（在最坏情况下），在辅助空间上，使用了 O(1) 的辅助变量来存储，完美的解决了这个问题。

代码辅助解释:

本质就是在数组中用一个数保存两个数，解决办法为取余和求模

```cpp
using i64 = long long;

void solve() {
    int n, m;
    std::cin >> n >> m;
    std::vector<int> a(n, 0);
    int maxn = 0;
    for (int i = 0; i < n; ++i) std::cin >> a[i], maxn = std::max(maxn, a[i]);
    ++maxn;
    int i = 0, j = m, k = 0;
    while (i < m && j < n) {
        if (a[i] % maxn <= a[j] % maxn) {
            a[k] = a[k] + (a[i] % maxn) * maxn;
            ++k; ++i;
        } else {
            a[k] = a[k] + (a[j] % maxn) * maxn;
            ++k; ++j;
        }
    }
    while (i < m) {
        a[k] = a[k] + (a[i] % maxn) * maxn;
        ++k; ++i;
    }
    while (j < n) {
        a[k] = a[k] + (a[j] % maxn) * maxn;
        ++k; ++j;
    }
    for (int i = 0; i < n; ++i) {
```

```cpp
        a[i] /= maxn;
    }
    for (int i = 0; i < n; ++i) {
        std::cout << a[i] << ' ';
    }
    std::cout << '\n';

}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);


    //freopen("in.txt", "r", stdin);
    //freopen("out.txt", "w", stdout);
    int t;
    std::cin >> t;
    while (t--) {
        solve();
    }

    return 0;
}
```

本题分工：小组共同讨论，黄勖编写

6 算法实现题2-1 答案：

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	

算法描述:

1. 打开文件读取多重集S元素个数n
2. 循环n次读入每个元素si：

   使用map记录每个si出现的次数map[si]

3. 遍历map得到max{map[si]}即重数，众数为si为所求

算法分析:

算法遍历一次文件内容并同时记录每个元素si出现的次数，故线性时间复杂度为O(n)

使用map记录故空间复杂度：O(n)

本题分工：小组共同讨论，李嘉琪编写

7　算法实现题2-7 答案：

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

算法设计:

按照题意可以采用动态规划完成。按照题目给出的划分规则，对于将 n 个元素划分为 m 个非空子集，可以由新增的1个元素分配到单独一个集合中（即 n-1 个元素划分为 m-1 个集合）和分配到已有的一个集合中（把这个元素分配到 n-1 个元素分配到 m 个集合的任意一个集合中），由这两种方式可以得到以下递推公式:

$$Sum[n][m] = Sum[n-1][m-1] + m \cdot Sum[n-1][m]$$

再考虑边界情况，得: $Sum[0][j] = 0$, $Sum[i][1] = 1$, $Sum[i][i] = 1$.

因此只需要创建一个 $n \times n$ 的二维数组 Sum，按照规则初始化数组，接着按照递推公式 $Sum[n][m] = Sum[n-1][m-1] + m \cdot Sum[n-1][m]$ 循环遍历填充数据，最后求和 $Sum[n][1]$ 到 $Sum[n][n]$ 的值即可。

算法分析:

实现该计算递推到 $Sum[n][n]$ 的值需要双重 for 循环可得时间复杂度为 $O(n^2)$.

本题分工：小组共同讨论，黄勖编写

8 补充题 答案:

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。

（1）存在序关系

输入:

有序数组T[1:n]

输出:

是否含有主元素

算法描述:

1. 二分法找到数组的中间元素mid，将T划分为T1[1:mid]和T2[mid+1:n]
2. 统计有序数组T1中<=mid的元素个数leftCount，T2中大于mid的元素个数rightCount
3. 若leftCount>n/2&&rightCount>n/2，则主元素在数组的左右两侧都可能存在，在左右子数组中查找

   若只有leftCount大于n/2，说明主元素在左子数组中，在左子数组中查找

   若只有rightCount大于n/2，说明主元素在右子数组中，在右子数组中查找

如果以上条件都不满足，那么数组中不存在主元素

算法分析:

算法通过每次线性划分查找主元素，故时间复杂度为O(n)

（2）不存在序关系

输入:

无序数组T[1:n]

输出:

是否含有主元素

算法描述:

1. 若数组只包含一个元素，该元素即是主元素。
2. 将数组T[1:n]均分成两个子数组T1[1:mid]和T2[mid+1:n]，递归左子数组T1找候选主元素c1和右子数组T2找候选主元素c2

   若c1 = c2，则该元素为主元素，返回该主元素

   若c1 ≠ c2，遍历数组T中求c1和c2候选的出现次数

      若c1出现次数 ≥ n/2，返回主元素c1

      若c2出现次数 ≥ n/2，返回主元素c2

      若c1出现次数和c2出现次数 < n/2，有序数组T[1:n]不存在主元素，算法结束

算法分析:

一般情况下

$$T(n) = \begin{cases} O(1), & \text{if } n = 1 \\ \\ 2T(\frac{n}{2}) + O(n), & \text{if } n > 1 \end{cases}$$

由Master主定理，可得时间复杂度为O(nlogn)

在某些较好的情况下，c1=c2时，仍然可以达到时间复杂度为O(n)：

$$T(n) = \begin{cases} O(1), & \text{if } n = 1 \\ \\ 2T(\frac{n}{2}) + O(1), & \text{if } n > 1 \end{cases}$$

由Master主定理，可得时间复杂度为O(n)

本题分工：小组共同讨论，李嘉琪编写