《计算机算法设计与分析》第二章作业

姓名：任宇　　　学号：33920212204567

算法分析题 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。当搜索元素在数组时，i 和 j 相同，均为 x 在数组中的位置。

答：算法设计：

1. 声明两个整数指针 left 和 right，分别初始化为 0 和 n-1。
2. 声明两个整数 i 和 j，初始化为 -1，用来存储最后的结果。
3. 进入一个循环，条件为 left <= right。
4. 在循环内，首先计算中间索引 mid，等于 (left + right) / 2。
5. 接下来，比较 mid 与 x ：
   - 如果相等，则返回 {mid, mid}。
   - 如果小于 x，将 mid 的值赋给 i，然后将 left 设置为 mid + 1。
   - 如果大于 x，将 mid 的值赋给 j，然后将 right 设置为 mid - 1。
6. 循环结束时，返回 (i, j)。

算法分析：每执行一次算法的 while 循环，待搜索数组的大小减少一半。因此在最坏情况下，while 循环被执行了 O(logn)次。循环体内运算需要 O(1)时间，因此整个算法在最坏情况下的计算时间复杂性为 O(logn)。

算法分析题 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≤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 $O(nm^{\log(3/2)})$时间内求出 uv 的值。

答：算法设计：当 m 比 n 小得多时，可以将 v 分为 n/m 段，即每段 m 位，而计算每次 m 位乘法可以用书中的分治法计算，最后合并结果。

算法分析：算法一共进行了 n/m 次乘法，每次乘法在 $O(m^{\log 3})$的时间内完成，所以 $T(n,m)=n/m*\ O(m^{\log 3})$，将 $O(m^{\log 3})$展开，可以得到 $T(n,m)=nm^{\log(3/2)}$,所以时间复杂度为 $O(nm^{\log(3/2)})$。

算法分析题 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 θ(n)时间内完成。符号 θ 所隐含的常数可能依赖于 k）。

答：算法设计：可以将 u 和 v 分别写为：

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

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

若想用 5 次 n/3 位整数的乘法求得 uv 的值，则：

P1 = a × d

P2 = c × f

P3 = (a+b) × (d+e)

P4 = (b+c) × (e+f)

P5 = (a+b+c) × (d+e+f)

P6 = P1 + P2 + P3 + P4 - P5

则 $uv = P2 + (P4-P6-P2)\times 10^{(n/3)} + (P5-P3-P4+P6+P6)\times 10^{2(n/3)} + (P3-P6-P1)\times 10^{3(n/3)} + P1\times 10^{4(n/3)}$

通过以上的计算即可得到最后的答案。

算法分析：按此思想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：

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

容易求得，其解为 $T(n)=O(\boldsymbol{n}^{\log_3 5})$。

算法分析题 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空间 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。

答：算法设计：

1. 初始化: 给定一个数组 a[0:n-1]和整数 k 满足 0 ≤ k ≤ n-1。

2. 将数组的子空间 a[0:k-1]进行逆置。

3. 接着将数组的子空间 a[k:n-1]进行逆置。

4. 最后，逆置整个数组 a[0:n-1]。

算法分析：逆置子空间 a[0:k-1]的时间复杂度为 O(k),逆置子空间 a[k:n-1]的时间复杂度为 O(n-k),而逆置整个数组的时间复杂度为 O(n)，所以总的时间复杂度是 O(k) + O(n-k) + O(n) = O(n)，逆置时用到一个临时变量，所以只用到 O(1)的辅助空间。

算法分析题 2-9 设子数组 a[0:k-1]和 a[k:n-1]已排好序（0≤k≤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：先用二分搜索算法在子数组 a[k:n-1]中搜索 a[0]的插入位置 pos，接着 a[0:pos]向右边循环换位 pos-k+1 个位置，这是，a[pos]及其左边的元素均已排好序，对于剩余的元素重复这个过程，只剩下一个数组段时，就已经将两个子数组合并为排好序的数组。

算法分析：数组段 a[0:k-1]中元素的移动次数不超过 k 次，数组段 a[0:k-1]中元素的移动次数最多一次，总的来看，该算法的平均时间复杂度为 O(n)，没有用到辅助空间，只有循环换位时用到一个辅助空间，因此只用到 O(1)的辅助空间。

算法实现题 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，其重数是 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。

答：算法设计：

1. 排序数组，对于一个有序的长度为 n 的数组 S，可以通过中位数将其分为三部分：中位数，中位数以左部分，中位数以右部分。

2. 用两个指针 left 和 right 分别指向中位数第一次出现的位置和最后一次出现的位置，right-left+1 即为中位数的重数 count，如果 count 大于最大重数 maxcount，就将众数更新为当前中位数，并将 maxcount 更新为 count。

3. 如果中位数左边部分长度大于 maxcount，说明该部分可能存在众数，递归。

4. 如果中位数右边部分长度大于 maxcount，说明该部分可能存在众数，递归。

5. 递归结束，返回 S 的众数及其重数。

算法分析：排序数组的时间复杂度为 O(nlogn),寻找中位数的时间复杂度为 O(n)，而算法中分治递归过程的时间复杂度可以表示为：

$$T(n)=\begin{cases} O(1),\ n=1 \\ 2T(n/2)+O(n),\ n>1 \end{cases}$$

因此该算法总的时间复杂度为 O(nlogn)。

代码实现：

```
#include "stdio.h"
#include "stdlib.h"
# define N 100
void Split(int a[], int n, int& l, int& r)
{
    int mid = n / 2;
    for (l = 0; l <= mid; ++l)
        if (a[l] == a[mid])
            break;
    for (r = mid + 1; r < n; ++r)
        if (a[r] != a[mid])
            break;
}
```

```
void getMaxNum(int& num, int& maxnum, int a[], int n)
{
    int l, r, s;
    int mid = n / 2;
    Split(a, n, l, r);
    s = r - l;
    if (s > maxnum)
    {
        num = a[mid];
        maxnum = s;
    }
    if (s == maxnum)
        if (num > a[mid])
        {
            num = a[mid];
            maxnum = s;
        }
    if (l + 1 > maxnum)
        getMaxNum(num, maxnum, a, l + 1);
    if (n - r > maxnum)
        getMaxNum(num, maxnum, a + r, n - r);
}
```

算法实现题 2-7 集合划分问题

问题描述：n 个元素的集合{1，2，……，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。

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

答：算法设计：

按照题目给出的划分规则，可以将问题分解成把 n 个元素划分为 m 个非空子集的集合，并求其个数总和，其中 m=1，2……，n。

对于如何将 n 个元素划分为 m 个非空子集和，可以得到以下递推公式:

Sum[n][m] = Sum[n-1][m-1] + m * Sum[n-1][m]

边界情况即：Sum[0][j]=0、Sum[i][1]=1、Sum[i][i]=1。

因此只需要创建一个 n*n 的二维数组 Sum，按照 Sum[i][i]=1 的规则初始化数组，接着按照 Sum[n][m] = Sum[n-1][m-1] + m * 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：

由代码的双重 for 循环可知其时间复杂度为 O（n^2）。

代码实现：

```
int Sum(int n)
{
    int a[N][N];
    for (int i = 0; i <= n; i++)
        for (int j = 0; j <= n; j++) {
            a[i][j] = 0;
            if (i == j)a[i][j] = 1;
        }
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++) {
            if (i >= j)
                a[i][j] = j * a[i - 1][j] + a[i - 1][j - 1];
        }
    int sum = 0;
    for(int i = 1; i <= n; i++)
        sum+=a[n][i];
    return sum;
}
```

补充题

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） **算法设计：**如果数组存在主元素，那么其中位数必定为主元素。所以可以先使用线性时间选择算法找出数组的中位数，再遍历一遍数组，计算与中位数相等的数字的数量是否过半，若超过则存在主元素，若不超过则不存在主元素。

算法分析：线性时间选择算法的时间复杂度为 O(n)，遍历数组的时间复杂度为 O(n)，故算法的时间复杂度为 O(n)。

（2） **算法设计：**经过查询资料，发现对于此类问题有一个很好用的算法----摩尔投票算法，但其并不是分治算法，因此，基于摩尔投票算法的思想，设计分治算法，即将数组两两划分，若数组存在主元素，那么至少有一组内的元素是相同的：

1. **递归地将数组分为若干组：**
   - 如果数组的长度为 1，返回这个元素。
   - 对数组进行两两分组。
   - 如果组内的两个元素相同，则只保留一个。如果组内的两个元素不同，则将这两个元素删除。
2. **对上述流程处理后的数组再次递归：**

   将得到的数组再进行上述的两两分组处理，直到数组只剩下一个元素。
3. **验证这个元素是否是主元素：**

   遍历原数组，统计这个元素出现的次数。如果该元素在原数组中的出现次数超过数组长度的一半，则它是主元素；否则不是。

算法分析：

$$T(n)=\begin{cases} O(1),\ n=1 \\ 2T(n/2)+O(1),\ n>1 \end{cases}$$

每次将数组划分为 n/2 的数组，直至最后划分为只有 2 个（或 1 个）元素的数组，组内比较的时间复杂度为 O(1)，因此可知分治的时间复杂度是 O(n)，遍历数组的时间复杂度也是 O(n)，因此总的时间复杂度是 O(n)。