算法分析第二次作业

算法分析题

2-3 题答案：

答：

2-3题答案：

答：

```
template<class Type>
int BinarySearch(Type a[ ], const Type& x, int n){ //在[0,n-1]中搜索x
    int i =0, int j = n-1; // i左,j右
    while( i <= j ){
        int middle = ( i + j )/2;
        if (x= = a[middle])
            return i,j;
        if(x > a[middle])
            i = middle+1;
        else
            j = middle-1;
    }
    return j,i;
}
```

分析：将原二分算法的返回值改为对应下标，在找不到x的情况下，原较小值下标变为较大值下标，较大变较小，故返回顺序颠倒。

2-4 题答案：

答：

2-4题答案：

答：原分治法有2种乘积手段 ①、将m，n级的XY都看为n级，进行二分，X对应A.B，Y对应C.D，进行二分

②、改变XY乘积方式：$XY = AC\times 2^n + ((A-B)(D-C)+AC+BD)\times 2^{\frac{n}{2}} + BD$，$T(n) = O(n^{\log 3}) = 3T(\frac{n}{2}) + O(n)$

故每次m位乘法的 $T(n,m) = \frac{n}{m} \cdot O(m^{\log 3})$ ∴ $O(nm^{\log\frac{3}{2}})$

2-5 题答案：

答：

该题分治算法需要 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：

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

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

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

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

$P1 = a \times d$

$P2 = c \times f$

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

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

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

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

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

2-8 题答案：

答：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:n1]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。

2-9 题答案：

答：

使用：采用双指针法，分别在两个已排序的子数组上移动指针，比较指针所指元素的大小，按顺序将较小的元素放入原数组的合适位置，直至其中一个子数组的元素全部处理完，再将另一个子数组剩余元素依次放入原数组.

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 O(1)的辅助空间。

算法实现题

2-1 题答案:

答:

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。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，并且根据要求更新。 中位数左边部分长度大于 maxcount，递归。中位数右边部分长度大于 maxcount，递归。

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：

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

$$\therefore O(n\log n).$$

```
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;
    }
    else if (s == *maxnum) {
        if (*num > a[mid]) {
            *num = a[mid];
        }
    }
    if (l + 1 > *maxnum) {
        getMaxNum(num, maxnum, a, l + 1);
    }
    if (n - r > *maxnum) {
        getMaxNum(num, maxnum, a + r, n - r);
    }
}
```

2-7 题答案：

答：

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。

递推公式：

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

边界情况：

Sum[0][j]=0、Sum[i][1]=1、Sum[i][i]=1。

双重 for 循环：时间复杂度为 O (n^2)。

```
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;
}
```