算法设计与分析第一次作业

姓名：张翼翔　　学号：21371055

1　请给出 $T(n)$ 尽可能紧凑的渐进上界并予以说明

1.

$T(1) = T(2) = 1$

$T(n) = T(n-2) + 1 \quad if \quad n > 2$

答： 渐进上界 $T(n) = O(n)$。$T(n) = T(n-2)+1 = T(n-4)+2 = \cdots = T(2) + \frac{n}{2} - 1$，$n > 2, n$ 为偶数。$T(n) = T(n-2)+1 = T(n-4)+2 = \cdots = T(1) + \lfloor\frac{n}{2}\rfloor - 2$，$n > 2, n$ 为奇数。所以 $T(n) \leq \lfloor\frac{n}{2}\rfloor \leq n$，$T(n) = O(n)$。

2.

$T(1) = 1$

$T(n) = T(n/2) + n \quad if \quad n > 1$

答： 渐进上界 $T(n) = O(n)$。不妨设 n 为 2 的幂，则 $T(n) = T(n/2) + n = \cdots = T(1) + (n + n - 2 + \cdots + 2 = \frac{2^{\log_2 n - 1}}{2-1} \times 2 + 1 = 2(n-1) + 1 < 2n$。所以 $T(n) = O(n)$。

3.

$T(1) = 1, T(2) = 1$

$T(n) = T(n/3) + n^2 \quad if \quad n > 2$

答： 渐进上界 $T(n) = O(n^2)$。因为 $2 > log_3 1 = 0$ 由主定理可知，$T(n) = O(n^2)$。

4.

$T(1) = 1$

$T(n) = T(n-1) + n^2 \quad if \quad n > 1$

答： 渐进上界 $T(n) = O(n^3)$。$T(n) = T(n-1)+n^2 = \cdots = T(1)+(n^2+(n-1)^2+\cdots+2^2) = \frac{n(n+1)(2n+1)}{6} \leq n^3$。所以 $T(n) = O(n^3)$。

5.

$T(1) = 1$

$T(n) = T(n-1) + 2^n \quad if \quad n > 1$

答： 渐进上界 $T(n) = O(n^2)$。$T(n) = T(n-1)+2^n = \cdots = T(1)+(2^n+2n-1+\cdots+2^2) = 2^n - 1$。所以 $T(n) = O(2^n)$。

6.

$$T(1) = 1$$
$$T(n) = T(n/2) + \log n \quad if \quad n > 1$$

答： 渐进上界 $T(n) = O(\log^2 n)$。$\log n = \Theta(n^{\log_2 1} \log^k n)$，$k$ 取 1。由主定理扩展形式知，$T(n) = O(\log^2 n)$。

7.

$$T(1) = 1, T(2) = 1$$
$$T(n) = 4T(n/3) + n \quad if \quad n > 2$$

答： 渐进上界 $T(n) = O(n^{\log_3 4})$。$n = O(n^{\log_3 4 - \epsilon})$，$\epsilon$ 取 $(0, \log_3 4 - 1)$。由主定理知，$T(n) = O(n^{\log_3 4})$。

2 k 路归并问题

1. 答： 这样需要合并 $k-1$ 次，第 i 次合并所需时间为 $kn \times i$。所以 $T(n) = kn(2 + 3 + \cdots + k) = n(k-1) \times (k+2)/2 = O(k^2 n)$。

2. 主体思路

将 k 个数组平均分成两份（如果不是 k 不是偶数则向下取整），分别对左边和右边的数组集合进行归并，得到两个排序好的数组，然后对这两个数组进行归并，即可得到有序的数组。

Algorithm 1: $MergeK(Array, l, r)$

Input: k 个包含 n 个元素的有序数组 $Array[1 \dots k][1 \dots n]$
区间左端点 l
区间左端点 r
Output: 包含 $(r{-}l+1)n$ 个元素的有序数组
if $l{=}r$ **then**
  return $Array[l][1 \dots n]$
end if
$mid \leftarrow \lfloor \frac{l+r}{2} \rfloor$
return $Merge(MergeK(Array, l, mid), MergeK(Array, mid+1, r))$

时间复杂度分析

答： $T(k) = 2T(k/2) + nk$，由主定理解得时间复杂度 $T(k) = O(nk \log k)$，满足题意。

3. 三余因子和问题

主体思路

从 A 到 B，依次计算这些数的三余因子，方法是用 3 除这些数，除到商不能继续整除 3 除为止，最后的商就是这个数的三余因子。然后将计算的 $B - A + 1$ 个三余因子

累加起来即可得到答案。

Algorithm 2: $SumMd3(A, B)$

Input: 区间左端点 A 和右端点 B

Output: $A \dots B$ 共 $B - A + 1$ 个连续的数的三余因子和 $\sum_{i=A}^{B} md3(i)$

```
Ans ← 0
for i ← A to B do
    num ← x
    while num%3 = 0 do
        num ← num/3
    end while
    Ans ← Ans + num
end for
return Ans
```

时间复杂度分析

时间复杂度 $O((B - A)\log B)(0 < A < B)$。一共需要计算 $B - A + 1$ 个三余因子，并计算它们的和。计算和的时间 $O(B - A)$，计算每个数的三余因子最多需要 $\log_3 B$ 的时间，时间复杂度 $O(\log B)$。故时间复杂度为 $O((B - A)\log B)$。

4. 填数字问题

主体思路

预先对数组进行分治，得到 n 个子数组，用区间 $[l_i, r_i]$ 表示。然后对这 n 个子数组以区间长度为第一关键字，区间左端点为第二关键字进行排序，得到有序排列的区间 $[l'_i, r'_j](i = 1, 2 \cdots n)$。之后将 $1 \cdots n$ 依次赋给 $A[\lfloor \frac{l'_i + r'_i}{2} \rfloor]$。

分治算法

Algorithm 3: $Partition(l, r)$

Input: 区间左端点 l 和右端点 r

Output: 分治后的区间集合 $Intervals$

```
if l > r then
    return ∅
end if
mid ← ⌊(l+r)/2⌋
return {[l,r]} ⋃ Partition(l, mid − 1) ⋃ Partition(mid + 1, r)
```

填充数字算法

Algorithm 4: $AssignArray(A, n)$

Input: 数组 A 及其长度 n

Output: 填充新数字后的 A

$Intervals \leftarrow Partition(1, n)$

将 $Intervals$ 以区间长度为第一关键字，区间左端点为第二关键字排序

for $interval_i \in Intervals$, $i \leftarrow 1$ *to* n **do**

  $l_i \leftarrow left\ end\ of\ interval_i$

  $r_i \leftarrow right\ end\ of\ interval_i$

  $A[\lfloor \frac{l_i + r_i}{2} \rfloor] \leftarrow i$

end for

return A

时间复杂度分析

时间复杂度 $O(n \log n)$。其中分治时间复杂度 $T(n) = 2T(n/2) + O(1) = O(n)$，排序时间复杂度 $O(n \log n)$。总时间复杂度 $O(n \log n)$。

5. 数字消失问题

1. 主体思路 考虑区间 $[l, r]$，取 $mid = \lfloor \frac{l+r}{2} \rfloor$，若 $[l, mid]$ 之间的数少于 $mid - L + 1$，则说明消失的数在 $[l, mid]$，反之亦然。由此递归查找即可。

见算法 5。

时间复杂度分析

时间复杂度 $T(n) = O(n)$。每个区间 $[l, r]$ 中至多只有一半的数会继续向下递归。$T(n) = T(n/2) + n/2 = O(n)$。

2. 主体思路

仍记当前区间为 $[l, r]$ 由于只能使用 bit−lookup(i, j)，考虑到 $n = 2k{-}1$，则从高到低依次枚举第 $n \cdots 1$ 个二进制位，设现在枚举到第 i 位。如果区间中这一位为 0 的数的数目小于 2^{i-1}，则说明消失的数在 $[l, l + 2^i - 1]$ 间，反之亦然。由此递归查找即可。

见算法 6。

时间复杂度分析

时间复杂度仍为 $T(n) = O(n)$，原理和前面相同。每个区间 $[l, r]$ 中至多只有一半的数会继续向下递归。$T(n) = T(n/2) + n/2 = O(n)$。

Algorithm 5: $FindMiss1(A, l, r)$

```
Input:  数组 A 和区间范围 [l, r]
Output: 消失的数 Miss
if l = r then
    return A[0]
end if
A_0 ← [ ]
A_1 ← [ ]
Miss ← 0
for j ← n downto 1 do
    cnt_0 ← 0
    cnt_1 ← 0
    for num_i ∈ A do
        mid ← ⌊(l+r)/2⌋
        if num_i ≤ mid then
            cnt_0 ← cnt_0 + 1
            将 num_i 加入 A_0
        else
            cnt_0 ← cnt_1 + 1
            将 num_i 加入 A_0
        end if
    end for
    if cnt0 < cnt1 then
        return FindMiss1(A_0, l, mid)
    else
        return FindMiss1(A_1, mid + 1, r)
    end if
end for
return FindMiss2(A, l, r)
```

Algorithm 6: $FindMiss2(A, k)$

Input: 数组 A 和区间范围 $[0, 2^k - 1]$

Output: 消失的数 $Miss$

$S_0 \leftarrow \emptyset$
$S_1 \leftarrow \emptyset$
$S \leftarrow \{1, \cdots, n\}$
$Miss \leftarrow 0$
for $j \leftarrow n$ *downto* 1 **do**
    $cnt_0 \leftarrow 0$
    $cnt_1 \leftarrow 0$
    for $i \in S$ **do**
        if $bit - lookup(i, j) = 0$ **then**
            $cnt_0 \leftarrow cnt_0 + 1$
            $S_0 = S_0 \bigcup \{i\}$
        else
            $cnt_1 \leftarrow cnt_1 + 1$
            $S_1 = S_1 \bigcup \{i\}$
        end if
    end for
    if $cnt0 < cnt1$ **then**
        $S \leftarrow S_0$
    else
        $Miss \leftarrow Miss + 2^{k-1}$
        $S \leftarrow S_1$
    end if
end for
return $Miss$