计算机学院《算法设计与分析》
(2023 年秋季学期)

第一次作业参考答案

1 请给出 $T(n)$ 尽可能紧凑的渐进上界并予以说明 (每小题 3 分，共 21 分)

1.

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

2.

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

3.

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

4.

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

5.

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

6.

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

7.

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

解：

1. $T(n) = O(n)$

2. $T(n) = O(n)$

3. $T(n) = O(n^2)$

4. $T(n) = O(n^3)$

5. $T(n) = O(2^n)$

6. $T(n) = O(\log^2 n)$

   原式展开为：

   $$\log n + \log(n/2) + \log(n/4) + \cdots + \log 2 + 1 = 1 + (1 + 2 + \cdots + \log n) \leq \sum_{i=1}^{\log n} i = O(\log^2 n)$$

7. $T(n) = O(n^{\log_3 4})$

2 k 路归并问题 (19 分)

现有 k 个有序数组（从小到大排序），每个数组中包含 n 个元素。你的任务是将他们合并成 1 个包含 kn 个元素的有序数组。首先来回忆一下课上讲的归并排序算法，它提供了一种合并有序数组的算法 $Merge$。如果我们有两个有序数组的大小分别为 x 和 y，$Merge$ 算法可以用 $O(x+y)$ 的时间来合并这两个数组。

1. 如果我们应用 $Merge$ 算法先合并第一个和第二个数组，然后由合并后的数组与第三个合并，再与第四个合并，直到合并完 k 个数组。请分析这种合并策略的时间复杂度（请用关于 k 和 n 的函数表示）。（9 分）

2. 针对本题的任务，请给出一个更高效的算法，并分析它的时间复杂度。（提示：此题若取得满分，所设计算法的时间复杂度应为 $O(nk\log k)$）。（10 分）

解：

1. 题目中给出的 $Merge$ 算法时间复杂度是线性的，根据题目中的策略对数组进行合并，每次合并的复杂度分别为 $n+n, 2n+n, \ldots, (k-1)n+n$。

   总的复杂度为：

   $$\left(n\sum_{i=1}^{k-1} i\right) + (k-1)n = n\frac{k(k-1)}{2} + (k-1)n = n\frac{k^2-k}{2} + k - 1 = O(nk^2)$$

2. 一种更高效的做法是把 k 个有序数组平均分为两份递归进行合并得到两个数组，然后再合并这两个数组。算法实现请参考 **Algorithm 1**。

Algorithm 1 $k\_Merge(A, l, r)$

Input:

k 个包含 n 个元素的有序数组, $A[1..k][1..n]$

递归区间左端点, l

递归区间右端点, r

Output:

归并后的包含 $(r-l+1)n$ 个元素的有序数组

```
if l = r then
    return A[l][1..n]
end if
m ← ⌊(l+r)/2⌋
return Merge(k_Merge(A, l, m), k_Merge(A, m+1, r))
```

这种方法的复杂度递归式为 $T(k) = 2T(k/2) + O(nk), T(1) = O(n)$，解出时间复杂度为 $O(nk\log k)$。

3 三余因子和问题 (20 分)

定义整数 i 的“3 余因子”为 i 最大的无法被 3 整除的因子，记作 $md3(i)$，例如 $md3(3) = 1, md3(18) = 2, md3(4) = 4$。请你设计一个高效算法，计算一个正整数区间 $[A, B], (0 < A < B)$ 内所有数的“3 余因子”之和，即 $\sum_{i=A}^{B} md3(i)$，并分析该算法的时间复杂度。例如，区间 $[3, 6]$ 的计算结果为 $1 + 4 + 5 + 2 = 12$。

答案:

区间 $[A, B]$ 中每一个数的“3 余因子”显然是相互独立的，因此 $\sum_{i=A}^{B} md3(i) = \sum_{i=1}^{B} md3(i) - \sum_{i=1}^{A-1} md3(i)$，原问题被简化为考虑如何对于非负整数 N，求解 $\sum_{i=1}^{N} md3(i)$。

当 N 取 0 时，这个式子是没有意义的所以答案直接取 0。首先考虑区间中无法被 3 整除的数字，例如对 3 取模为 1 或者 2 的数，这些数满足 $md3(i) = i$，所以可以直接利用等差数列的公式求解。

然后考虑区间 $[1, N]$ 内 3 的倍数，因为区间是从 1 为起点连续不断的，所以 $[1, N]$ 中 3 的倍数一定可以写为 $[3 * 1, 3 * 2, \ldots, 3 * \lfloor \frac{N}{3} \rfloor]$ 因子 3 都不能被算上，所以相当于求区间 $[1, \lfloor \frac{N}{3} \rfloor]$ 中的“3 余因子和”，那么原问题被转化为了一个规模只有原来 1/3 的问题，可以使用分治法。

详细算法如 **Algorithm** 2, **Algorithm** 3 所示。

Algorithm 2 $sum\_md3(A, B)$

Input:
　区间左端点, A
　区间右端点, B
Output:
　区间计算结果 $\sum_{i \in [A,B]} md3(i)$

return $sum\_md3(B) - sum\_md3(A - 1)$

Algorithm 3 $sum\_md3(N)$

Input:
　区间端点, N
Output:
　区间计算结果 $\sum_{i \in [1,N]} md3(i)$

if $N = 0$ **then**
　return 0
end if
$md1 \leftarrow 0$
$md2 \leftarrow 0$
if $N\%3 = 1$ **then**
　$md1 \leftarrow md1 + (1 + N) * (\lfloor \frac{N}{3} \rfloor + 1)/2$
　if $N - 2 > 0$ **then**
　　$md2 \leftarrow md2 + N * (\lfloor \frac{N-2}{3} \rfloor + 1)/2$
　end if
end if
if $N\%3 = 2$ **then**
　$md2 \leftarrow md2 + (2 + N) * (\lfloor \frac{N}{3} \rfloor + 1)/2$
　$md1 \leftarrow md1 + N * (\lfloor \frac{N-1}{3} \rfloor + 1)/2$
end if
return $md1 + md2 + sum\_md3(\lfloor \frac{N}{3} \rfloor)$

该算法将原问题转化为一个可在 $O(1)$ 时间内计算的式子与一个规模为 $n/3$ 的子问题，假设边界的数据范围为 n，故可列出递归式 $T(n) = T(n/3) + O(1)$，解得 $T(n) = O(\log n)$。如果直接对两个边界 A,B 进行分类讨论，不采用前缀和的形式，或者不采用递归的写法也能得满分。若不采用前缀和的求法，复杂度可以写为 $O(\log |B - A|)$，由于 B,A 并没有特殊的限制，所以这一题中 $|B - A|$ 的数据规模可以看做和 $|A|$ 与 $|B|$ 同阶的。

4 填数字问题 (20 分)

给定一个长度为 n 的数组 $A[1..n]$，初始时数组中所有元素的值均为 0，现对其进行 n 次操作。第 i 次操作可分为两个步骤:

1. 先选出 A 数组长度最长且连续为 0 的区间，如果有多个这样的区间，则选择最左端的区间，记本次选定的闭区间为 $[l, r]$；
2. 对于闭区间 $[l, r]$，将 $A\left[\lfloor\frac{l+r}{2}\rfloor\right]$ 赋值为 i，其中 $\lfloor x\rfloor$ 表示对数 x 做向下取整。

例如 $n = 6$ 的情形，初始时数组为 $A = [0, 0, 0, 0, 0, 0]$。
第一次操作为选择区间 $[1, 6]$，赋值后为 $A = [0, 0, 1, 0, 0, 0]$；
第二次操作为选择区间 $[4, 6]$，赋值后为 $A = [0, 0, 1, 0, 2, 0]$；
第三次操作为选择区间 $[1, 2]$，赋值后为 $A = [3, 0, 1, 0, 2, 0]$；
第四次操作为选择区间 $[2, 2]$，赋值后为 $A = [3, 4, 1, 0, 2, 0]$；
第五次操作为选择区间 $[4, 4]$，赋值后为 $A = [3, 4, 1, 5, 2, 0]$；
第六次操作为选择区间 $[6, 6]$，赋值后为 $A = [3, 4, 1, 5, 2, 6]$，为所求。
请设计一个高效的算法求出 n 次操作后的数组，并分析其时间复杂度。

解:
本题可以利用分治预先得到所有操作选择的区间，再根据规则排序依次操作赋值。
考虑若某一次操作的区间是 $[l, r]$，那么 $[l, mid)$ 和 $(mid, r]$ 是两个待操作的区间 (其中 $mid = \lfloor\frac{l+r}{2}\rfloor$)。则可以考虑如下分治算法生成所有需要操作的区间:

Algorithm 4 $SpiltAll(L, R)$

Input:
当前操作区间的左右断点 L, R;
Output:
当前区间最终分裂成的操作区间集合

```
if L > R then
    return ∅
end if
mid = ⌊(l+r)/2⌋
return [L, R] ∪ SpiltAll(L, mid − 1) ∪ SpiltAll(mid + 1, r)
```

可以证明，长度更长的区间比长度更短的区间上的数字小。关于这个结论可以使用反证法进行证明，若存在一个长度为 l 的区间 s_i，一个区间长度为 r 的区间 s_j，满足 $r > l$，而且 s_i 上数字更小。在给 s_i 赋值时，数组中已经不存在比 l 长度更短的区间。然而，s_j 区间上数字更大，那么它一定在区间 s_i 之后赋值，这与不存在比 l 小的区间矛盾，证毕。
根据上述结论，只要按照区间长度为第一关键字，区间左端点为第二关键字依次操作每个区间，就可以得到 n 次操作后的数组。故总算法框架如 **Algorithm**5所示。

Algorithm 5 $GenArray(A, n)$

Input:
数组 A 及其长度 n
Output:
数组 A

```
LIST ← SpiltAll(1, n)
将 LIST 按照区间长度为第一关键字、区间左端点为第二关键字排序
for [l_i, r_i] : LIST do
    A[⌊(l_i+r_i)/2⌋] ← i {有序枚举 LIST 里面的所有区间，记第 i 个被枚举区间为 [l_i, r_i]}
end for
return A
```

用分治得到所有可能被选择的区间需要 $T(n) = 2T(n/2) + O(1) = O(n)$ 的时间，后续排序选择区间需要 $O(n \log n)$ 的时间，故总的时间复杂度为 $O(n \log n)$。
如果使用桶排序则算法复杂度为 $O(n)$ 也可以得到满分。所有大于 $O(n \log n)$ 的算法如果正确得 10 分。

5　数字消失问题 (20 分)

给定一长度为 n 的数组 $A[1..n]$，其包含 $[0, n]$ 闭区间内除某一特定数 (记做消失的数) 以外的所有数字（例如 $n = 3$ 时，$A = [1, 3, 0]$，则消失的数是 2）。这里假定 $n = 2^k - 1$。

1. 请设计一个尽可能高效的算法找到消失的数，并分析其时间复杂度。(8 分)
2. 若假定数组 A 用 k 位二进制方式存储（例如 $k = 2$，$A = [01, 11, 00]$ 则消失的数是 10），且不可以直接访存（即不可以直接通过数组的下标访问数组的内容）。目前**唯一**可以使用的操作是 `bit-lookup`(i, j)，其作用是用一个单位时间去查询 $A[i]$ 的第 j 个二进制位。请利用此操作设计一个尽可能高效的算法找到消失的数，并分析其时间复杂度。(12 分)

答案:

1. 考虑目前数组对应的值域为 $[L, R]$，则可以利用中位数 $mid = \lfloor \frac{L+R}{2} \rfloor$ 将数组划分成两部分: $\leq mid$ 的部分和 $> mid$ 的部分。若 $\leq mid$ 的部分的元素个数少于 $mid - L + 1$，则说明消失的数在 $[L, mid]$ 之中，递归考虑，否则我们递归考虑 $> mid$ 的部分。算法伪代码请参考 **Algorithm** 6。

   每次仅有至多一半的元素需要进入下一层递归，故递归式可写为 $T(n) = T(\frac{n}{2}) + n$。由主定理可知，时间复杂度为 $T(n) = O(n)$。

2. 本题的核心思想在于，对于所有区间 $[0, 2^k - 1]$ 中的数，将他们按照顺序排列，它们二进制表示中，各个位置的数一定是交错出现的, 且 **0 和 1 数量一定相等**。那么我们就可以考虑将数组按照二进制位逐位的进行划分，然后迭代的考虑按位划分后每一位个数较少的一部分。这里举一个例子，考虑集合 $\{0, 1, 3\}$，在区间 $[0, 3]$ 中消失的数就是 2. 如果将这个集合中所有数写成 2 进制的形式，可以写为 $\{00, 01, 11\}$，那么首先考虑最低一位，利用 `bit-lookup` 操作分别统计所有数中最后一位为 1 的数的数量，以及最后一位为 0 的数的数量，发现为 0 的数量比为 1 的数量少，那么一定可以断定消失的数字最低一位为 0，继续在最低一位为 0 的数字集合中查找第一位的情况，此时数组中只有一个数字 $\{00\}$，那么第一位为 0 的数量比第一位为 1 的数量多，那么消失的数字最高位为 1，所以根据如上推理，可以求得消失的数字为 10，即数字 2。算法伪代码请参考 **Algorithm** 7。

   注意到每次循环迭代 S 的过程时，都对 S 集合进行了一次折半操作。故 `bit-lookup` 的操作次数为 $T(n) = \sum_{i=0}^{k-1} \frac{n}{2^k}$。求解该式可得时间复杂度为 $O(n)$。

问题 1 暴力的扫描做法也可以得到 8 分。第二问由于没有指定数字的位数是从 0 还是 1 开始计算，所以无论哪一种计数方式只要前后自洽都可以得到满分。

Algorithm 6 $MissingInteger(A, i, j)$

Input:
　数组 A，待寻找的值域 $[i, j]$
Output:
　消失的数

$mid \leftarrow \lfloor \frac{i+j}{2} \rfloor$
if mid 不在 A 数组中 **then**
　return mid
end if
把 A 数组划分成 $B(\leq mid)$ 和 $C(> mid)$ 两个部分
if $Size(B) < mid - i + 1$ **then**
　return MissingInteger(B, i, mid)
else
　return MissingInteger$(C, mid + 1, j)$
end if

Algorithm 7 $MissingInteger2(A, k)$

Input:

数组 A 及其对应的值域 $[0, 2^k - 1]$

Output:

消失的数

$S \leftarrow \{1, 2, \cdots, n\}$
$S_0 \leftarrow S_1 \leftarrow \varnothing$
$count0 \leftarrow count1 \leftarrow 0$
for $posn = k$ **downto** 1 **do**
    for $i \in S$ **do**
        $bit \leftarrow$ `bit-lookup`$(i, posn)$
        if $bit \leftarrow 0$ **then**
            $count0 \leftarrow count0 + 1$
            $S_0 \leftarrow S_0 \cup \{i\}$
        else
            $count1 \leftarrow count1 + 1$
            $S_1 \leftarrow S_1 \cup \{i\}$
        end if
    end for
    if $count0 > count1$ **then**
        $missing[posn] \leftarrow 1$
        $S \leftarrow S_1$
    else
        $missing[posn] \leftarrow 0$
        $S \leftarrow S_0$
    end if
    $S_0 \leftarrow S_1 \leftarrow \varnothing$
    $count0 \leftarrow count1 \leftarrow 0$
end for
return $missing$