计算机学院《算法设计与分析》
(2019 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，可以假定 n 是 2 的整数幂次。**(每小题 3 分，共 21 分)**

1.

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

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

2.

$$T(1) = 1$$

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

3.

$$T(1) = 1$$

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

4.

$$T(1) = 1$$

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

5.

$$T(1) = 1$$

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

6.

$$T(1) = 1$$

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

7.

$$T(1) = 1$$

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

解：

1. $T(n) = O(n^2)$
2. $T(n) = O(n^2)$
3. $T(n) = O(n\log n)$
4. $T(n) = O(n\log^2 n)$
5. $T(n) = O(n^2)$
6. $T(n) = O(n^{\log_2 3})$
7. $T(n) = O(n\log n)$

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 $n+n, 2n+n, \ldots, (k-1)n+n$。 ……（5 分）

   总的复杂度为：

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

   ……（4 分）

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。算法实现可以参考 **Algorithm** 1。 ……（7 分, 包括伪代码）

   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 分）

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

1: **if** $l = r$ **then**
2: **return** $A[l][1..n]$
3: **end if**
4: $m \leftarrow \lfloor \frac{l+r}{2} \rfloor$
5: **return** $Merge(k\_Merge(A, l, m), k\_Merge(A, m+1, r))$

3 战线补给问题 (20 分)

现有 2^l 个堡垒组成一条战线，编号为 $1, 2, 3, \cdots, 2^l$。其中每个堡垒都可能有一个或多个士兵驻扎，也可能没有任何士兵驻扎。已知士兵共有 n 个，且第 i 个士兵驻扎在编号为 a_i 的堡垒中。现请你用最小的费用给整条战线提供补给（即是说，为区间 $[1, 2^l]$ 中的所有堡垒提供补给）。为战线 $[1, 2^l]$ 提供补给可以按照下述两种方式进行：

1. 若当前区间组成的战线中还剩余至少两个堡垒，可以将该区间**均分**为左右两段，并分别为其提供补给。所需的总费用为补给这两段战线的费用之和。

2. 直接为当前区间组成的战线提供补给。若这段战线中的所有堡垒均没有任何士兵驻扎，则所需的费用为 A；否则，费用为 $num \times len \times B$。其中 A, B 为给定的常数，num, len 分别为这段战线中士兵的总数以及堡垒的总数。

例如，现有四个堡垒，编号为 $1, 2, 3, 4$。只有一个士兵，其驻扎在 2 号堡垒。可选的补给方式有多种：

一种方式为直接为整条战线 $[1, 4]$ 提供补给，所需费用为 $1 \times 4 \times B$，其中 1 表示当前中共有 1 名士兵，4 表示当前区间中共有 4 个堡垒；

另外一种可行的方式是将该区间平均分为两段 $[1, 2]$ 和 $[3, 4]$。并分别为这两段提供补给，和第一种方式类似，区间 $[1, 2]$ 所需的代价为 $1 \times 2 \times B$，而区间 $[3, 4]$ 由于没有任何士兵驻扎，所需的费用为 A。因此，这种补给方式的总费用为 $A + 2B$。

当然也可以将区间 $[1, 2]$ 或 $[3, 4]$ 继续均分并分别进行补给。

请设计一个高效的算法计算为整条战线体提供补给所需的最小费用，并尽可能准确地分析该算法的时间复杂度。

解法 1：

我们可以使用分治的方法解决该问题，设计递归函数 $rec(l, r, a)$ 来求解为区间 [l,r] 提供补给需要的费用。根据题意，每个区间有两种计算费用的方式，$rec(l, r, a)$ 应当为这两种方式中最小的值。伪代码如 **Algorithm** 2所示。

其中，函数 $cost(l, r, a)$ 用于计算为区间 $[l, r]$ 直接提供补给所需的花费。为了快速求出区间 $[l, r]$ 中所驻扎的士兵总数，我们可以使用二分查找来降低复杂度。首先将数组 a 排序使其变为有序数组（这一步仅需预处理一次，时间复杂度为 $O(n \log n)$）。之后分别使用二分查找算法找出该数组中第一个大于等于 l 的位置 $index_l$ 以及第一个大于 r 的位置 $index_r$，则两位置之差 $index_r - index_l$ 就是区间 $[l, r]$ 内的士兵个数。运用这种方法，每次函数 $cost(l, r, a)$ 的时间复杂度为 $O(\log n)$。

例如四个堡垒，编号为 $1, 2, 3, 4$，两个个士兵分别驻扎在 $2, 3$ 号堡垒。则排好序的数组 $a = [2, 3]$。当我们想确定区间 $[1, 4]$ 中的士兵个数时，使用二分查找算法分别找到 $1, 4$ 在数组 a 中的位置 $a[0] = 2 > 1, index_l = 0, a[1] = 3 < 4, index_r = 2$。据此可知，区间 $[1, 4]$ 中士兵人数 $num = index_r - index_l = 2$。 ……（8 分）

接下来分析该算法的复杂度。每个子问题中查找和计算的时间复杂度为 $O(\log n)$，据此可以写出递归式：$T(M) = 2T(M/2) + O(\log n), T(1) = O(\log n)$。由于区间总长度为 2^l，该算法时间复杂度为 $O(2^l \log n + n \log n)$。 ……（4 分）

Algorithm 2 $rec(l, r, a)$

Input:
提供补给的区间左右端点 (l, r)
每个士兵所驻扎的堡垒编号 $a[1..n]$
Output:
当前区间所需的最小花费 $minCost$

```
if l = r then
    return cost(l, r, a)
end if
mid ← ⌊(l+r)/2⌋
case1 ← rec(l, mid, a) + rec(mid + 1, r, a)
case2 ← cost(l, r, a)
minCost ← min(case1, case2)
return minCost
```

解法 2：
递归的子问题跟解法 1 类似，但是注意到若某个区间当前已经没有士兵驻扎，那么此区间就没有必要继续递归计算了，直接消耗 A 的花费给此区间提供补给即可。因为继续递归下去所需的花费一定比当前的花费大。考虑到这一点，可以得到如 **Algorithm** 3所示的改进后的算法。

Algorithm 3 $rec\_opt(l, r, a)$

Input:
提供补给的区间左右端点 (l, r)
每个士兵所驻扎的堡垒编号 $a[1..n]$
Output:
当前区间所需的最小花费 $minCost$

```
if l = r or 区间 [l, r] 中没有任何士兵 then
    return cost(l, r, a)
end if
mid ← ⌊(l+r)/2⌋
case1 ← rec_opt(l, mid, a) + rec_opt(mid + 1, r, a)
case2 ← cost(l, r, a)
minCost ← min(case1, case2)
return minCost
```

…… （13 分）

此时，虽然递归树仍有 l 层，但是每层最多只有 $O(n)$ 个子问题 (因为最多只有 n 个士兵，故每层最多只有 n 个子问题会继续递归求解)。每个子问题中查找和计算的时间复杂度仍为 $O(\log n)$。因此，总的时间复杂度为 $O(nl \log n)$。

…… （7 分）

4 区间计数问题 (20 分)

给定一个包含 n 个元素的数组 $A = [a_1, a_2, \cdots, a_n]$。对数组 A 中的任意区间 $[l, r](1 \le l \le r \le n)$，该区间的和可表示为 $S_{[l,r]} = \sum_{i=l}^{r} a_i$。

请设计一个高效的分治算法统计有多少个区间 $[l, r]$ 满足：$X \le S_{[l,r]} \le Y$（X, Y 为给定的常数）。并分析该算法的时间复杂度。

解：
本题可以借鉴求解数组中逆序数对个数的方法。在求解逆序数对个数的问题中，我们是要统计数组 num 中：$i < j, num[i] > num[j]$ 的数对 (i, j) 的个数，也即 $i < j, num[i] - num[j] > 0$ 的数对 (i, j) 的个数。

在本题中，我们是要求解满足：$X \le sum_{k=l}^{r} num[k] \le Y$ 的 (l, r) 的个数。对于 $sum_{k=l}^{r}$ 我们可以使用前缀和数组进行变换，前缀和数组 $preSum[k] = sum_{i=1}^{k}$。对于区间 $[l, r]$ 的和有 $sum[l, r] = preSum[r] - preSum[l-1]$ 。所以本题的求解目标可以转变为 $l \le r, X \le$

$preSum[r] - PreSum[l-1] \leq Y$ 的 (l, r) 个数，进一步转变为 $preSum[r] - PreSum[l-1] \leq Y$ 减去 $preSum[r] - PreSum[l-1] < X$ 的区间个数。过程与求解逆序数对个数相似。具体实现参考 **Algorithm** 4。

Algorithm 4 $SortAndCount(L, X, Y)$

Input:
前缀和数组 L; 区间和的上下界 X, Y;
Output:
满足 $l \leq r, X \leq L[r] - L[l-1] \leq Y$ 的 (l, r) 个数;

```
将 L 划分为两个子数组 A, B
(r_a, A) = SortAndCount(A, X, Y)
(r_b, B) = SortAndCount(B, X, Y)
(r, L) = MergeAndCount(A, B, X, Y)
return r + r_a + r_b, L
```

Algorithm 5 $MergeAndCount(A, B, X, Y)$

Input:
前缀和数组 A, B; 区间和的上下界 X, Y;
Output:
满足 $X \leq B[r] - A[l] \leq Y$ 的 (l, r) 个数;

```
ans, r_1, r_2 ← 0, 1, 1
L ← ∅
for l ← 1 to A.length do
    while r_1 ≤ B.length 并且 B[r_1] − A[l] ≤ Y do
        r_1 ← r_1 + 1
    end while
    while r_2 ≤ B.length 并且 B[r_2] − A[l] < X do
        r_2 ← r_2 + 1
    end while
    ans ← ans + r_1 − r_2
end for
L ← Merge(A, B)
return ans, L
```

…… （16 分）

每个问题划分为了两个子问题来解，两个子问题的合并过程所需时间复杂度为 $O(n)$，据此可以写出递归式 $T(n) = 2T(n/2) + O(n)$。解得总的时间复杂度为 $O(n \log n)$。 …… （4 分）

时间复杂度为 $O(n \log^2 n)$ 的算法可得 14 分，时间复杂度为 $O(n^2)$ 的算法可得 6 分。不同时间复杂度的算法只要正确地分析了时间复杂度即可得 4 分。

5 向量的最小和问题 (20 分)

给定 n 个二维向量 $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$。每一个向量 $\mathbf{v}_i = (x_i, y_i)$ 都可以变换为如下四种形式：

1. $\mathbf{v}_i^1 = (x_i, y_i)$
2. $\mathbf{v}_i^2 = (-x_i, y_i)$
3. $\mathbf{v}_i^3 = (x_i, -y_i)$
4. $\mathbf{v}_i^4 = (-x_i, -y_i)$

请你设计一个高效的算法从 n 个向量中找出两个向量，使得他们以某种形式相加后的模长最小。换言之，请找出两个向量 $\mathbf{v}_i, \mathbf{v}_j (1 \leq i, j \leq n$ 且 $i \neq j)$，以及两个整数 $k_1, k_2 (1 \leq k_1, k_2 \leq 4)$，使得 $||\mathbf{v}_i^{k_1} + \mathbf{v}_j^{k_2}||_2$ 最小。此外，请分析该算法的时间复杂度。

解:

注意到每个向量可以任意取 $\mathbf{v}^1, \mathbf{v}^2, \mathbf{v}^3, \mathbf{v}^4$ 四种状态，此时寻找两个向量的最小和问题和寻找两个向量的最小差问题是等价的。因此我们考虑如何寻找两个向量，使得他们以某种形式相减后的模长最小。 ……（答出此种模型转化方法可得 4 分）

对任意两个向量来说，在他们的第一维和第二维的符号都相同时，他们相减后的模长可以取到最小值。因此，我们将所有的向量的第一维和第二维全部变为正数，即将他们全部变换到第一象限中，并将每个向量 (x_i, y_i) 视为二维平面上的一个点，那么仅需求出该二维平面上任意两点间的最近距离就找到了我们要求的向量最小和。

该问题是一个经典的算法问题：最近点对问题 (*Closest pair of points problem*)。关于该问题有一个分治解法：

首先将所有点以 x 为第一关键字排序，然后每次按照 x 进行分治，将原集合等分为左右两个子集，左右两边分别递归求出一个最短距离 d_1, d_2，那么所有点中最近的两点间距离至少为 $d = min(d_1, d_2)$。

接下来考虑两个子集合并的过程，不妨枚举左侧的点 p_i，那么显然如果右侧的某点 p_j 到 p_i 的距离超过了 d，可以直接舍去。因此可以对每个 p_i，画一个以 p_i 为中心，边长为 $2d$ 的正方形，仅需考虑在该正方形内部的 p_j。易证该正方形内最多存在 6 个点 (详细证明过程可以参考算法导论 $Sec\,33.4$)，因此可以将这些点按照 y 排序后选择距离 p_i 最近的 6 个点来比较。

如果每次重新对左右两个子集使用快速排序等方法进行排序，则合并的时间复杂度为 $O(n \log n)$，因此可写出递归式：$T(n) \leq 2T(n/2) + O(n \log n)$，解得 $T(n) = O(n \log^2 n)$，然而，我们可以在分治的同时使用归并排序来对左右两个子集进行排序，这样合并左右两子集的时间复杂度则为 $O(n)$，算法的总复杂度为 $O(n \log n)$。

该问题给出时间复杂度为 $O(n \log n)$ 的算法可得满分，给出时间复杂度为 $O(n \log^2 n)$ 的算法可得 17 分，给出时间复杂度为 $O(n^2)$ 的算法可得 10 分。(包括模型转化)