高等理工学院《算法设计与分析》
(2020 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 $n+n, 2n+n, \ldots, (k-1)n+n$。

   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。

   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 $O(nk\log k)$。

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

$m \leftarrow \lfloor \frac{l+r}{2} \rfloor$

return $Merge(k\_Merge(A, l, m), k\_Merge(A, m+1, r))$

3 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 2 $SpiltAll(L, R)$

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 $\varnothing$

end if

$mid = \lfloor \frac{l+r}{2} \rfloor$

return $[L, R] \cup SpiltAll(L, mid-1) \cup SpiltAll(mid+1, r)$

之后，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，得到 n 次操作后的数组。故总算法框架如 **Algorithm**3所示。

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 $O(n \log n)$。

Algorithm 3 $GenArray(A, n)$

Input:

数组 A 及其长度 n

Output:

数组 A

$LIST \leftarrow SpiltAll(1, n)$

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**

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

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 $[l, r](1 \leq l \leq r \leq n)$，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：$X \leq S_{[l,r]} \leq Y$（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：$X \leq sum_{k=l}^{r} num[k] \leq Y$ 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 $l \leq r, X \leq preSum[r] - PreSum[l-1] \leq Y$ 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 $r + r_a + r_b, L$

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 $I(n) = 2T(n/2) + O(n)$。解得总的时间复杂度为 $O(n \log n)$。

5 三角形周长最小问题 (20 分)

给定平面上 n 个点 $v_1, \cdots, v_n$(保证任意三点不共线)，每个点可以表示为 $v_i = (x_i, y_i)$，记两点的距离为 $dis(v_i, v_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$。尝试在这 n 个点中选择 3 个点，使得组成的三角形的周长最小。

换言之，即求 $v_i, v_j, v_k(i, j, k \in \{1, \cdots, n\}, i < j < k)$ 使得 $dis(v_i, v_j) + dis(v_j, v_k) + dis(v_i, v_k)$ 最小。

请设计一个高效的算法求出 v_i, v_j, v_k(如果有多组解，输出任意一组即可)。此外，请分析该算法的时间复杂度。

解:

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;

$ans, r_1, r_2 \leftarrow 0, 1, 1$
$L \leftarrow \emptyset$
for $l \leftarrow 1$ **to** $A.length$ **do**
    while $r_1 \leq B.length$ 并且 $B[r_1] - A[l] \leq Y$ **do**
        $r_1 \leftarrow r_1 + 1$
    end while
    while $r_2 \leq B.length$ 并且 $B[r_2] - A[l] < X$ **do**
        $r_2 \leftarrow r_2 + 1$
    end while
    $ans \leftarrow ans + r_1 - r_2$
end for
$L \leftarrow Merge(A, B)$
return ans, L

该问题是一个经典的算法问题：最近点对问题 (*Closest pair of points problem*) 的变形问题。尝试设计如下分治算法：

1. (预操作) 将所有点按照 x 坐标排序
2. 将点均匀的分成两半，用一条垂直分割线 $x = x_{mid}$ 分开
3. 转入第二步，递归求解左半部分和右半部分，记左右回溯的答案分别为 d_l, d_r
4. 求解跨过左边和右边的答案，记作 d_{mid}
5. 返回 d_l, d_r, d_{mid} 回溯递归

预操作的复杂度是 $O(n \log n)$，后续的分治算法的复杂度，分析如下:

均分成两半的复杂度是 $O(1)$ 的,即直接找到有序数组的中间值,把两边作为参数传下去。假设合并求解跨过左边和右边的答案的时间为 $f(n)$，那么复杂度的递归表示为 $T(n) = 2T\left(\frac{n}{2}\right) + f(n)$。

现在考虑合并操作如何实现:

记 $D = \min(d_l, d_r)$, 左边点集为 L, 右边点集为 R, 我们只需要考虑所有距离中轴线小于 D 的点，即求集合 $X = \{|x_{mid} - x_p| < D\}$，故对于这些点同样，只需要考虑在 R 集合中 y 方向距离小于 D 的所有点。即需要求得 $Y(p) = \{q \in X \big| \, y_p - D < y_p \leq y_p\}$

下面证明求解 X 和 $Y(p)$ 的过程都可以是 $O(n)$ 的。

求解 X**：**注意到 X 是按照 x 坐标排好序的一个区间，故可以直接线性时间复杂度扫描数组的求出该区间 X

求解 $Y(p)$**：**如果我们可以按照 y 坐标先排序一遍所有点，那么 $Y(p)$ 也是一个区间，也可以直接求出。实际上，可以每次回溯的时候返回两个按照 y 轴排好序的点坐标数组，故这一步也是线性时间复杂度可以求出。

最后我们只需要对于 $p \in X, q \in Y(p)$ 求解 $dist(p, q)$ 和 D 取 min 就可以得到最后的答案。

故我们需要证明 $Y(p)$ 是一个常数:

考虑到实际上我们是在 $[x_{mid} - D, x_{mid} + D] \times [y_p - D, y_p]$ 的矩形上考虑，且有一点在 (x_p, y_p) 的位置，在红蓝两个矩形内部的点的距离不能小于 D 由抽屉原理，至多还有 6 个点在这两个矩形内。故 $Y(p)$ 是一个常数。

Algorithm 6 $MinTri(l, r)$

Input:

排序后 $A[l..r]$ 这一段点。

Output:

$A[l..r]$ 这一段点所能得到的三角形的最小周长的一半 $A[l..r]$ 这一段按照 y 轴排序的结果

if $l > r$ **then**
  return ∞
end if
$mid \leftarrow (l + r)/2$
$Al, ansl \leftarrow MinTri(l, mid)$
$Ar, ansr \leftarrow MinTri(mid + 1, r)$
$Aret \leftarrow Merge(Al, Ar)$ {将 Al,Ar 按照 y 坐标为关键字有序合并}
$d \leftarrow \min\{ansl, ansr\}$
$B \leftarrow \varnothing$
for $point \in Aret$ **do**
  if $abs(point.x - A[mid].x) < d$ **then**
    add $point$ to the end of B
  end if
end for
for $i : 1 \rightarrow B.size()$ **do**
  for $j : i + 1 \rightarrow B.size()$ **do**
    if $b[j].y - b[i].y \geq d$ **then**
      for $k : j + 1 \rightarrow B.size()$ **do**
        if $b[k].y - b[i].y < d$ **then**
          $d \leftarrow \min(d,$ 以$b[i], b[j], b[k]$为顶点的三角形周长$)$
        end if
      end for
    end if
  end for
end for
return $d, Aret$

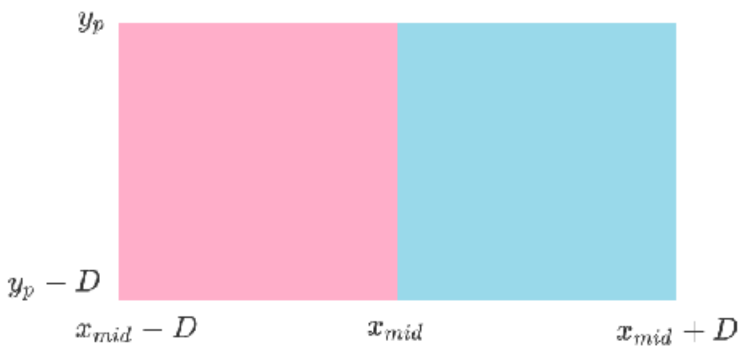

图 1: Y(p) 至多出于这些顶点

故综上所述 $O(f(n)) = O(n)$，所以由主定理，分治问题可以在 $O(n \log n)$ 的时间内解决。再考虑到对 x 的预先排序复杂度为 $O(n \log n)$，故原问题同样可以在 $O(n \log n)$ 时间求解。