

计算机学院《算法设计与分析》第一次作业

September 24, 2024

1 请给出 $T(n)$ 尽可能紧凑的渐进上界并予以说明

1.1

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

$$T(n) = T(n-1) + n = T(n-2) + n + n - 1 = \sum_1^n i = \frac{n \times (n+1)}{2}$$

由此可知，原式渐进上界为 $O(n^2)$ 。

1.2

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

采用代入法， $T(n) = T(n-2) + 1 = T(n-4) + 2 = \lceil \frac{n}{2} \rceil$ 。由此可知，原式渐进上界为 $O(n)$ 。

1.3

$$T(n) = \begin{cases} 1, & n = 1 \\ 4T(n/2) + n, & n > 1 \end{cases}$$

采用主定理法。 $\log_a b = \log_2 4 = 2 > 1$ 。由此可知，原式渐进上界为 $O(n^2)$ 。

1.4

$$T(n) = \begin{cases} 1, & n = 1 \\ 8T(n/2) + n^3, & n > 1 \end{cases}$$

采用主定理法， $\log_a b = \log_2 8 = 3$ 。由此可知，原式渐进上界为 $O(n^3 \log n)$ 。

1.5

$$T(n) = \begin{cases} 1, & n = 1, 2 \\ 2T(n/3) + n^2, & n > 2 \end{cases}$$

采用主定理法。 $\log_a b = \log_3 2 < 2$ 。由此可知，原式渐进上界为 $O(n^2)$ 。

1.6

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

采用扩展的主定理法。 $\log_2 1 = 0$ $f(n) = n^{\log_2 1} \log n$

原式展开为： $\log n + \log(n/2) + \dots + \log 2 + 1 = 1 + (1 + 2 + \dots + \log n) \leq \sum_{i=1}^{\log n} i = \log^2 n$

由此可知，原式渐进上界为 $O(\log^2 n)$ 。

1.7

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

$$T(n) = T(n-1) + 2^n = T(n-2) + 2^n + 2^{n-1} = \sum_{i=0}^n 2^i = 2^{n+1} - 1$$

由此可知，原式渐进上界为 $O(2^n)$ 。

2 游戏获奖问题

在一场射击比赛中，有 n 名选手参加比赛，每名选手的得分都不同，记为 $s[1..n]$ ，其中第 i 名选手的得分为 $s[i]$ ，且对于任意的 $i \neq j$ $s[i] \neq s[j]$ 。按照比赛规则，分数排名为前 $\lfloor \frac{n}{3} \rfloor$ 的选手可以获得与其得分相等的奖金（包括第 $\lfloor \frac{n}{3} \rfloor$ 名）。例如，当有 6 名选手参加比赛，得分为 $s = [100, 50, 90, 60, 65, 80]$ 时，排名位于前 $\lfloor \frac{6}{3} \rfloor = 2$ 的选手得分为 100, 90，则主办方应该发放奖金 190 元。请你设计一个算法，计算这次比赛主办方一共要发放多少奖金。请描述算法的核心思想，给出算法伪代码并分析其对应的时间复杂度。

2.1 作业题目 2 分析和算法

本题相当于要我们求出 n 个数中最大的 $\lfloor \frac{n}{3} \rfloor$ 个数之和，很容易想到利用排序算法完成求解。我们采用“分而治之”算法的典范快速排序算法来完成本题，具体为选定参考点之后进行交换，该算法的复杂度为 $O(n \log n)$ 。最后我们将分数相加时时间复杂度为 $O(n)$ ，因此算法时间复杂度为 $O(n \log n)$ 。

2.2 作业题目 2 伪代码

Algorithm 1: 快速排序后求和算法

Input: 选手得分数组 $s[n]$

Output: 一个正整数 sum , 为主办方发放奖金之和

```
1 global s[n]
2 function PARTITION(s, p, r):
3     x ← s[r];
4     i ← p - 1;
5     for j ← p to r - 1 do
6         if s[j] ≥ x then
7             i ← i + 1;
8             exchange(s[i], s[j]);
9         end
10    end
11    exchange(s[i + 1], s[r]);
12    return (i + 1);
13 end
14 function QUICKSORT(s, p, r):
15    if p ≤ r then
16        q ← PARTITION(s, p, r);
17        QUICKSORT(s, p, q - 1);
18        QUICKSORT(s, q + 1, r);
19    end
20 end
21 function SOLVE(n):
22    QUICKSORT(s, 1, n);
23    sum ← 0; for i ← 1 to ⌊n/3⌋ do
24        sum ← sum + s[i];
25    end
26    return sum;
27 end
```

3 奇数因子和问题

定义 $od(i)$ 为整数 i 的最大奇数因子, 例如 $od(3) = 3, od(14) = 7$ 。请你设计一个高效算法, 计算一个整数区间 $[A, B]$ 内所有数的最大奇数因子和, 即 $\sum_{i=A}^B od(i)$ 。请描述算法的核心思想, 给出算法伪代码并分析其对应的时间复杂度。例如, 区间 $[3, 9]$ 的计算结果为 $3 + 1 + 5 + 3 + 7 + 1 + 9 = 29$ 。

3.1 作业题目 3 分析和算法

对于这道题, 我们首先想到将所有整数的因子 2 除尽, 这样得出的就是最大奇数因子, 这样对于 n , 除以 2 次数为 $1 \rightarrow n$ 中 2 的幂的个数和, 即 $\sum_{i=1}^{\log_2 n} \frac{n}{2^i}$, 时间复杂度为 $O(n)$, 显然不合格。

我们采用递归思想, 利用前缀和再做算法设计。我们设 $\sum_{i=1}^n od(i) = sum(i)$, 则显然本题所求答案为 $sum(B) - sum(A - 1)$, 经过分析可知:

$$sum(n) = \text{所有奇数项之和} + \text{所有的偶数项} / 2 \text{所有的偶数项} / 2 = \text{当前所有奇数项之和} + \text{当前所有的偶数项} / 2$$

由此可递推得

$$sum(n) = \begin{cases} 1, & n = 1, 2 \\ \frac{(n+1)\times(n+1)}{4} + sum(n/2), & n \bmod 2 = 1 \\ \frac{n^2}{4} + sum(n/2), & n \bmod 2 = 0 \end{cases}$$

由于每次计算都对 $n/2$, 而最后相减求值步骤时间复杂度为 $O(1)$, 因此算法总体时间复杂度为 $O(\log n)$ 。

3.2 作业题目 3 伪代码

Algorithm 2: 前缀和加递归算法

Input: 正整数 A, B , 与题目中意义相同

Output: 一个正整数 res , 结果

```
1 global A, B
2 function Sum(n):
3     if n = 1 then
4         | return 1;
5     end
6     else if n mod 2 = 1 then
7         | return  $\frac{(n+1)\times(n+1)}{4}$  + Sum(n/2);
8     end
9     else
10        | return  $\frac{n^2}{4}$  + Sum(n/2);
11    end
12 end
13 function SOLVE(n):
14     sum  $\leftarrow$  Sum(B) - Sum(A - 1);
15     return sum;
16 end
```

4 施工点对计数问题

在一个城市的修建工程中, 有 n 个施工点, 其中 $costs[i]$ 表示第 i 个施工点的成本。为完成任务, 工程师们需要选择两个不同的施工点进行联合施工, 联合施工的成本为每个施工点的成本加和, 并且需要保证联合施工的总成本不能超过预算上限。形式化地说: 需要计算所有满足预算限制 l 的施工点对 (i, j) 的数量, 其中 $1 \leq i < j \leq n$ 且 $costs[i] + costs[j] \leq l$ 。请你设计并实现一个算法来解决该问题, 描述算法的核心思想, 给出算法伪代码并分析其对应的时间复杂度。

4.1 作业题目 4 分析和算法

如果我们不对 $costs$ 数组率先进行排序, 那么只能采取二重循环的扫描解法, 时间复杂度为 $O(n^2)$, 但如果我们提前对数组进行排序 (采用复杂度为 $O(n \log n)$ 的排序算法), 利用双端指针维护数对和, 那么在复杂度上还能做到更好。

我们如下分析, 对于排序过后的数组 $cost$, 有 $i < j_1 < j$, 则有 $costs[i] + costs[j_1] < costs[i] + costs[j]$, 那么, 若 $costs[i] + costs[j] \leq l$, 则有 $\forall m, i < m < j, costs[i] + costs[m] \leq l$ 。因此对于此 (i, j) , 我们可以确定有 $j - i$ 个数对满足条件。因此我们维护双端指针 l, r , 从两段遍历数组, 最终将每一个上述位置 (i, j) , 所确定的数对求和则可求解, 此扫描过程时间复杂度为 $O(n)$, 排序算法时间复杂度为 $O(n \log n)$, 因此算法总时间复杂度为 $O(n \log n)$ 。

4.2 作业题目 4 伪代码

Algorithm 3: 双端指针扫描算法

Input: 施工点成本数组 $costs[1 \dots n]$

Output: 一个正整数 res , 所有满足预算限制 l 的施工点对 (i, j) 的数量

```
1 global costs[n], l
2 function Sum(costs):
3     left ← 1;
4     right ← n;
5     sum ← 0;
6     while left < right do
7         while left < right and costs[left] + costs[right] > l do
8             right ← right - 1;
9         end
10        ans ← ans + right - left;
11        left ← left + 1;
12    end
13    return sum;
14 end
15 function SOLVE(n):
16     QUICKSORT(costs, 1, n);
17     sum ← Sum(costs);
18     return sum;
19 end
```

5 最近村庄距离问题

在一个广阔的平原上, 有 n 个村庄。每个村庄 i 都有一个独特的位置, 记为 (x_i, y_i) , 表示在平面上横纵坐标的位置。现在, 政府打算优先在两个距离最接近的村庄之间修建一条公路, 以便加强它们之间的交通联系, 其中村庄之间的距离定义为欧几里得距离, 即对于两个村庄 $A(x_1, y_1)$ 和 $B(x_2, y_2)$, 它们之间的距离为: $\sqrt{(x_1-x_2)^2 + (y_1-y_2)^2}$ 请你设计一个高效算法, 计算在平原上距离最近的两个村庄之间的距离。请描述算法的核心思想, 给出算法伪代码并分析其对应的时间复杂度。

5.1 作业题目 5 分析和算法

本题颇为经典, 将题中修饰语句删除, 题意即为给定平面上 n 个点, 求出两点间欧几里得距离的最小值。我们很容易想到朴素遍历方法, 通过二重循环得出最短距离, 这样的方法时间复杂度为 $O(n^2)$ 。我们需要在时间复杂度上有所精进, 此时, 最容易想到的便是求助于“分治算法”。

我们将 n 个点按 x 坐标从大到小排序, 取 x 坐标的中位数, 将点集分成两半, 分别求解两区域下距离最小值, 最后再处理跨区域距离最小值的情况, 由此我们知道, 最短距离只可能在三种情况下产生:

1. 2 个点均在左侧, 距离为 ld
2. 2 个点均在右侧, 距离为 rd
3. 2 个点跨左右两侧, 距离为 md

显然，最复杂的情况为求解 2 点跨两侧的情况。下面，我们进行算法具体步骤的描述：这里我们先假设合并操作的时间复杂度为 $O(n)$ ，可知算法总复杂度为 $T(n) = 2T(\frac{n}{2}) + O(n) = O(n \log n)$ 。我们先将所有点按照 x_i 为第一关键字、 y_i 为第二关键字排序，并以点 $p_m (m = \lfloor \frac{n}{2} \rfloor)$ 为分界点，拆分点集为 A_1, A_2 ：

$$A_1 = \{p_i \mid i = 0 \dots m\}$$

$$A_2 = \{p_i \mid i = m + 1 \dots n - 1\}$$

并递归下去，求出两点集各自内部的最近点对，设距离为 h_1, h_2 ，取较小值设为 h 。我们试图找到这样的一组点对，其中一个属于 A_1 ，另一个属于 A_2 ，且二者距离小于 h 。因此我们将所有横坐标与 x_m 的差小于 h 的点放入集合 B ：

$$B = \{p_i \mid |x_i - x_m| < h\}$$

对于 B 中的每个点 p_i ，我们当前目标是找到一个同样在 B 中、且到其距离小于 h 的点。为了避免两个点之间互相考虑，我们只考虑那些纵坐标小于 y_i 的点。显然对于一个合法的点 p_j ， $y_i - y_j$ 必须小于 h 。于是我们获得了一个集合 $C(p_i)$ ：

$$C(p_i) = \{p_j \mid p_j \in B, y_i - h < y_j \leq y_i\}$$

如果我们将 B 中的点按照 y_i 排序， $C(p_i)$ 将很容易得到，即紧邻 p_i 的连续几个点。

由此我们得到了合并的步骤：

1. 构建集合 B 。
2. 将 B 中的点按照 y_i 排序。
3. 对于每个 $p_i \in B$ 考虑 $p_j \in C(p_i)$ ，对于每对 (p_i, p_j) 计算距离并更新答案（当前所处集合的最近点对）。

这个算法似乎仍然不优，因为 $|C(p_i)|$ 将处于 $O(n)$ 数量级，导致合并总复杂度超出 $O(n)$ 。事实上，我们可以证明 $|C(p_i)|$ 最大值为 7，该证明由 Preparata 和 Shamos 在 1975 年提出：

现在已经知道， $C(p_i)$ 中的所有点的纵坐标都在 $(y_i - h, y_i]$ 范围内；且 $C(p_i)$ 中的所有点，和 p_i 本身，横坐标都在 $(x_m - h, x_m + h)$ 范围内。这构成了一个 $2h \times h$ 的矩形。

再将这个矩形拆分为两个 $h \times h$ 的正方形，不考虑 p_i ，其中一个正方形中的点为 $C(p_i) \cap A_1$ ，另一个为 $C(p_i) \cap A_2$ ，且两个正方形内的任意两点间距离大于 h 。

将一个 $h \times h$ 的正方形拆分为四个 $\frac{h}{2} \times \frac{h}{2}$ 的小正方形。可以发现，每个小正方形中最多有 1 个点：因为该小正方形中任意两点最大距离是对角线的长度，即 $\frac{h}{\sqrt{2}}$ ，该数小于 h 。

示意图如下：

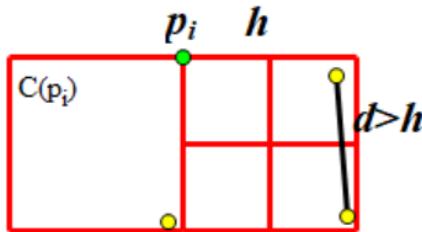


Figure 1: C 点集示意图

由此，每个正方形中最多有 4 个点，矩形中最多有 8 个点，去掉 p_i 本身， $\max(C(p_i)) = 7$ 。

在合并过程中，我们使用了两次排序，因为点坐标全程不变，第一次排序可以只在分治开始前进行一次。我们令每次递归返回当前点集按 y_i 排序的结果，对于第二次排序，上层直接使用下层的两个分别排序过的点集归并即可。

因此合并操作复杂度为 $O(n)$ ，算法总复杂度为 $T(n) = 2T(\frac{n}{2}) + O(n) = O(n \log n)$ 。

5.2 作业题目 5 伪代码

Algorithm 4: 平面最近点对算法

Input: n 个点的点集 $point[1 \dots n]$
Output: 一个浮点数 res ，表示最近的村庄距离

```
1 global point[n], res
2 function SOLVE(n):
3     |  QUICKSORT(point, 1, n);      对 point 按 x 坐标进行快速排序
4     |  CAL(1, n);
5     |  return res;
6 end
```

Algorithm 5: 分治算法

```
global point[n], res
function UPDATE(a, b):
    dist =  $\sqrt{((a.x - b.x)^2 + (a.y - b.y)^2)}$ ;
    if dist < res then
        |  res  $\leftarrow$  dist;
    end
end

function CAL(l, r):
    m  $\leftarrow$   $\frac{(l+r)}{2}$ ;
    midx  $\leftarrow$  point[mid].x;
    CAL(l, m);
    CAL(m + 1, r);
    MERGE(l, m + 1, r + 1);
    flag  $\leftarrow$  0;
    i  $\leftarrow$  l;
    while i  $\leq$  r; do
        |  if  $|point[i].x - midx| < res$  then
            |      j  $\leftarrow$  flag - 1;
            |      while j  $\geq$  0 and point[i].y - t[j].y < res do
            |          |  UPDATE(point[i], t[j]);
            |          |  j  $\leftarrow$  j - 1;
            |      end
            |      t[flag]  $\leftarrow$  a[i];
            |      flag  $\leftarrow$  flag + 1;
        end
        |  i  $\leftarrow$  i + 1;
    end
end
```

Algorithm 6: 合并算法

global $point[n], res$

function $MERGE(a, b, c)$:

$i \leftarrow 1;$

$j \leftarrow b + 1;$

while $i \leq b + 1 \text{ or } j \leq c$ **do**

if $j > r \text{ or } (point[i].y > point[j].y \text{ and } i > b)$ **then**

$t[k] \leftarrow point[i];$

$i \leftarrow i + 1;$

end

if $i > b \text{ or } (point[i].y > point[j].y \text{ and } j > c)$ **then**

$t[k] \leftarrow point[j];$

$j \leftarrow j + 1;$

end

end

$point[a \dots b] \leftarrow t;$

end
