计算机学院《算法设计与分析》
(2022 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.

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

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

2.

$$T(1) = 1$$

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

3.

$$T(1) = 1$$

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

4.

$$T(1) = 1$$

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

5.

$$T(1) = 1$$

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

6.

$$T(1) = 1$$

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

7.

$$T(1) = 1$$

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

解：

1. $T(n) = O(n)$
2. $T(n) = O(\log n)$
3. $T(n) = O(n)$
4. $T(n) = O(n)$
5. $T(n) = O(n^2)$
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. $T(n) = O(n^2)$

2 寻找中位数问题 (19 分)

一组数据 $x = (x_1, \cdots, x_n)$ 从小到大排序后的序列为 $x'_1, \cdots, x'_n$，则这组数据的**中位数** M 为

$$M = \begin{cases} x'_{\frac{n+1}{2}} & 若n为奇数； \\ \frac{1}{2}(x'_{\frac{n}{2}} + x'_{\frac{n}{2}+1}) & 若n为偶数。 \end{cases}$$

给定两个长度分别为 n、m 的有序数组 $A[1..n], B[1..m]$（A 与 B 均已按从小到大排序）。请设计尽可能高效的算法，求出这两个数组中所有数据的中位数，并分析其时间复杂度。

解：
根据中位数的定义，问题即为寻找两个有序数组中第 k 小的数，其中

$$k = \begin{cases} \frac{n+m+1}{2} & 若n+m为奇数； \\ \frac{n+m}{2}和\frac{n+m}{2}+1 & 若n+m为偶数。 \end{cases}$$

此时，尝试比较 $A[\lfloor \frac{k}{2} \rfloor]$ 和 $B[\lfloor \frac{k}{2} \rfloor]$。由于 $A[\lfloor \frac{k}{2} \rfloor]$ 和 $B[\lfloor \frac{k}{2} \rfloor]$ 的前面有 $A[1..\lfloor \frac{k}{2} \rfloor - 1]$ 和 $B[1..\lfloor \frac{k}{2} \rfloor - 1]$，故对于 $A[\lfloor \frac{k}{2} \rfloor]$ 和 $B[\lfloor \frac{k}{2} \rfloor]$ 中的较小值，最多只有 $\lfloor \frac{k}{2} \rfloor - 1 + \lfloor \frac{k}{2} \rfloor - 1 \leq k - 2$ 个元素，故其不可能是第 k 小的元素了。

根据如上观察，可以归纳出两种情况:

1 如果 $A[\lfloor \frac{k}{2} \rfloor] \leq B[\lfloor \frac{k}{2} \rfloor]$，则可以排除 $A[1..\lfloor \frac{k}{2} \rfloor]$。

2 如果 $A[\lfloor \frac{k}{2} \rfloor] > B[\lfloor \frac{k}{2} \rfloor]$，则可以排除 $B[1..\lfloor \frac{k}{2} \rfloor]$。

再排除之后，对于新的两个数组和新的 k，可以继续迭代去寻找第 k 小的数。该算法的伪代码如 **Algorithm** 2所示。

注意到，每轮循环后需要查找的 k 至少减小了一半，而初始时 $k = (m+n)/2$ 或 $k = (m+n)/2 + 1$。故时间复杂度为 $O(\log(m+n))$。

时间复杂度为 $O(n+m)$ 的算法可得 10 分。

Algorithm 1 $getKthEle(A[1..n], B[1..m], k)$

Input:

两个数组 $A[1..n], B[1..m]$;

Output:

两个数组的第 k 小元素

```
1: i, j ← 1, 1
2: while True do
3:    if i = m then
4:       return B[i + k − 1]
5:    end if
6:    if j = n then
7:       return A[j + k − 1]
8:    end if
9:    if k = 1 then
10:      return min{A[i], B[j]}
11:   end if
12:   ni ← min{i + k/2 − 1, m}
13:   nj ← min{j + k/2 − 1, n}
14:   pi, pj ← A[ni], B[nj]
15:   if pi ≤ pj then
16:      k ← k − (ni − i + 1)
17:      i ← ni + 1
18:   else
19:      k ← k − (nj − j + 1)
20:      j ← nj + 1
21:   end if
22: end while
```

Algorithm 2 $FindMedian2(A[1..n], B[1..m])$

Input:

两个数组 $A[1..n], B[1..m]$;

Output:

两个数组的中位数

```
1: tot ← n + m
2: if tot is odd then
3:    return   getKthEle(A, B, (tot + 1)/2)
4: else
5:    return   (getKthEle(A,B,tot/2)+getKthEle(A,B,tot/2+1))/2
6: end if
```

3 双调序列最大值问题 (20 分)

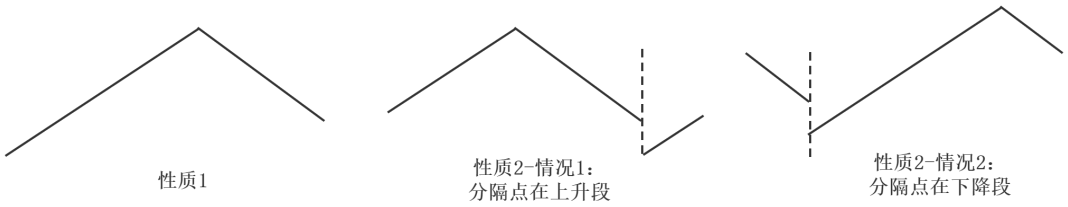
若一个序列 $a[1...n]$ 满足以下两个性质之一，则该序列称为双调序列：

- 性质 1：存在 $k \in [1,n]$，使得 $a_1 \leq a_2 \leq ... \leq a_{k-1} \leq a_k \geq a_{k+1} \geq ... \geq a_{n-1} \geq a_n$
- 性质 2：序列经循环移位后满足性质 1

例如，序列 $[1,2,4,8,7,5,3]$ 为双调序列（满足性质 1），序列 $[5,3,1,2,4,8,7]$ 也为双调序列（循环移位后可得序列 $[1,2,4,8,7,5,3]$，满足性质 2）。

给定一个所有元素互不相同的双调序列 $a[1...n]$，请设计一个算法来求出这个序列中的最大值，并对该算法的复杂度进行分析。

解：



由于性质 2 可以看作性质 1 的一种特殊情况，因此只需要对满足性质 2 的序列进行分析，按照循环移位分割点的位置将满足性质 2 的序列分为两类。

- 情况 1：分割点在上升段时，双调序列是递增-最大值-递减-递增的，且满足 $a_1 > a_n$
- 情况 2：分割点在下降段时，双调序列是递减-递增-最大值-递减的，且满足 $a_1 < a_n$

因此，算法应先通过 a_1 和 a_n 的大小关系判断出是上述两种情况的哪一种。

使用二分的思想对情况 1 进行分析，令 $m = \lfloor n/2 \rfloor$，检查元素 a_m 与 a_{m+1} 的大小关系，若 $a_m > a_{m+1}$（a_m 位于递减段）或 $a_m < a_1$（a_m 位于第二个递增段），则在区间 $a[1...m]$ 中递归查找最小值；否则，在区间 $a[m+1...n]$ 中递归查找。用相同的方法可以对情况 2 进行分析，算法的伪代码如 **Algorithm 3**所示。

注意到每一次循环会使查找的范围减半，因此算法的时间复杂度为 $O(\log n)$。时间复杂度为 $O(n)$ 的算法可得 10 分。

Algorithm 3 $FindMax(a[1..n])$

Input:

双调序列 $a[1..n]$;

Output:

$a[1..n]$ 的最大值

```
1: left ← 1, right ← n
2: if a_1 > a_n then
3:    while left < right do
4:       m ← ⌊(left + right)/2⌋
5:       if a_m > a_{m+1} or a_m < a_1 then
6:          right ← m
7:       else
8:          left ← m + 1
9:       end if
10:   end while
11: else
12:   while left < right do
13:      m ← ⌊(left + right)/2⌋
14:      if a_m < a_{m+1} or a_m < a_n then
15:         left ← m + 1
16:      else
17:         right ← m
18:      end if
19:   end while
20: end if
21: return a[left]
```

4 字符串等价关系判定问题 (20 分)

给定两个长度为 n 的字符串 A 和 B，若称 A 与 B 是**等价**的，当且仅当它们满足如下关系之一：

1. A 和 B 完全相同;
2. 若将把 A 分成长度相等的两段 A_1 和 A_2，也将 B 分成长度相等的两段 B_1 和 B_2。且他们之间满足如下两种关系之一：
   a. A_1 和 B_1 等价且 A_2 和 B_2 等价;
   b. A_1 和 B_2 等价且 A_2 和 B_1 等价;

请你设计一个高效的算法来判断两个字符串是否等价并分析你的算法的时间复杂度。

解法 1：

通过观察可以发现，题目中所描述的**等价**事实上是一种等价关系。它满足自反性、对称性和传递性，这意味着我们可以把这些字符串划分为不同的等价类。如果对每个等价类找出它们中字典序最小的一个字符串作为其代表元，这样仅需判断它们的代表元是否相等就可以判断两个字符串是否等价了。

给出一个字符串，找和它属于同一等价类的字典序最小元的方法也很简单，就是递归处理字符串的左右两个子串，把字典序较小的子串放在前面。算法实现请参考 **Algorithm** 4。

Algorithm 4 $Smallest(S[1..n])$

Input:
一个长度为 n 的字符串, $S[1..n]$;
Output:
字符串 S 所在等价类的最小元;

```
1: if n%2 = 1 then
2:     return S;
3: end if
4: m ← n/2;
5: S1 ← Smallest(S[1..m]);
6: S2 ← Smallest(S[m + 1..n]);
7: if S1 < S2 then
8:     return S1 + S2;
9: else
10:    return S2 + S1;
11: end if
```

注：伪代码中的 $<$ 运算为字典序的比较，$+$ 运算为两个字符串直接拼接。例如 $ab < ac, bd < cd, ab + cd = abcd, cd + ab = cdab$。

每次递归都将问题分解为两个规模缩减一半的问题，之后合并由于涉及到字符串的比较以及拼接，时间复杂度为 $O(n)$，可以写出递归式 $T(n) = 2T(n/2) + O(n)$，解出算法的时间复杂度为 $O(n \log n)$。

解法 2（答出本解法可得 16 分）：

直接从等价关系入手，根据定义直接进行四次递归判断，可以写出一个递归算法 **Algorithm** 5。

上述算法在最坏的情况下需要进行四次递归计算，递归式为 $T(n) \leq 4T(n/2) + O(n)$，从而得到算法的时间复杂度为 $O(n^2)$。

然而，通过如下分析，可以发现该算法仍有可以改进的地方。

1. 如果 A1 和 B1 等价且 A2 和 B2 等价，那么可以直接返回 A 和 B 等价。
2. 在 A1 和 B1 等价且 A2 和 B2 不等价的时候，我们没有必要再判断 A2 和 B1 以及 A1 和 B2 是否等价了，因为如果 A1 和 B2 等价，又因为之前已经判断过了 A1 和 B1 是等价的，因此 B1 和 B2 一定是等价的，又因为之前已经知道 A2 和 B2 不等价，因此 A2 和 B1 一定是不等价的。

Algorithm 5 $Trivial\_equivalence(A, B)$

Input:

两个长度为 n 的字符串, $A[1..n], B[1..n]$;

Output:

两个字符串是否等价;

1: **if** $A = B$ **then**
2: **return true**;
3: **end if**
4: **if** n%2=1 **then**
5: **return false**;
6: **end if**
7: $m \leftarrow \frac{n}{2}$;
8: $A1 \leftarrow A[1..m]$;
9: $A2 \leftarrow A[m+1..n]$;
10: $B1 \leftarrow B[1..m]$;
11: $B2 \leftarrow B[m+1..n]$;
12: **return** $Trivial\_equivalence(A1, B1)$ **and** $Trivial\_equivalence(A1, B1)$ **or** $Trivial\_equivalence(A1, B2)$ **and** $Trivial\_equivalence(A2, B1)$;

3. 在 A1 和 B1 不等价的时候，我们显然也不需要判断 A2 和 B2 是否等价了，仅需要判断 A2 和 B1 以及 A1 和 B2 是否等价。

该算法的伪代码如 **Algorithm** 6所示。

因此，该算法最多仅需进行 3 次递归计算。递归式为 $T(n) \leq 3T(n/2) + O(n)$，解出其时间复杂度为 $O(n^{\log_2 3})$。

Algorithm 6 $Equivalence(A, B)$

Input:

两个长度为 n 的字符串, $A[1..n], B[1..n]$;

Output:

两个字符串是否等价;

```
1: if A = B then
2:    return true;
3: end if
4: if n%2=1 then
5:    return false;
6: end if
7: m ← n/2;
8: A1 ← A[1..m];
9: A2 ← A[m + 1..n];
10: B1 ← B[1..m];
11: B2 ← B[m + 1..n];
12: if Equivalence(A1, B1) then
13:    if Equivalence(A2, B2) then
14:       return true;
15:    else
16:       return false;
17:    end if
18: else
19:    return Equivalence(A1, B2) and Equivalence(A2, B1);
20: end if
```

5 向量的最小和问题 (20 分)

给定 n 个二维向量 $v_1, v_2, \ldots, v_n$。每一个向量 $v_i = (x_i, y_i)$ 都可以变换为如下四种形式：

1. $v_i^1 = (x_i, y_i)$
2. $v_i^2 = (-x_i, y_i)$
3. $v_i^3 = (x_i, -y_i)$
4. $v_i^4 = (-x_i, -y_i)$

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 $v_i, v_j (1 \le i, j \le n$ 且 $i \ne j)$，以及两个整数 $k_1, k_2 (1 \le k_1, k_2 \le 4)$，使得 $||v_i^{k_1} + v_j^{k_2}||_2$ 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 v^1, v^2, v^3, 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。

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。可以证明该正方形内最多存在 6 个点（参考下图，使用鸽巢原理证明），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。

Algorithm 7 $MinVector(V)$

Input:
n 个二维向量 $V = [v_1, v_2 ... v_n]$
Output:
向量的最小和

```
for i ← 1 to n do
    v_i.x = abs(v_i.x)
    v_i.y = abs(v_i.y)
end for
将 v_1, v_2...v_n 按照 x 坐标排序
return MinPointPair(V[1...n])
```

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：$T(n) \le 2T(n/2) + O(n \log n)$，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。

Algorithm 8 $MinPointPair(V)$

Input:

点坐标数组 $V[l...r]$

Output:

最近点对的距离按 y 轴排序后的 $V[l...r]$

if $l > r$ **then**
  return ∞
end if
$mid \leftarrow (l + r)/2$
$ansl, Al \leftarrow MinPointPair(V[l, mid])$
$ansr, Ar \leftarrow MinPointPair(V[mid + 1, r])$
$Aret \leftarrow Merge(Al, Ar)$ {将 Al, Ar 按照 y 坐标为关键字有序合并}
$d \leftarrow \min\{ansl, ansr\}$
for $p_i \in Aret$ **do**
  if $p_i.x \in [V[mid].x - d, V[mid].x]$ **then**
    找到纵坐标与 p_i 最接近的 6 个 p_j，且 p_j 满足 $p_j.x \in [V[mid].x, V[mid].x + d]$
    更新 $d = min(dist(p_i, p_j), d)$
  end if
end for
return d