《算法设计与分析》
第一次作业

姓名：曹建钦

学号：20375177

合： 100

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 的整数次幂

21

1、

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

解：使用代入法：

- 当 n 为偶数时：

$$T(n) = T(n-2) + 1 = T(n-4) + 1 + 1 = T(2) + 1 + \ldots + 1 = n/2$$

- 当 n 为奇数时：

$$T(n) = T(n-2) + 1 = T(n-4) + 1 + 1 = T(1) + 1 + \ldots + 1 = (n+1)/2$$

综上：

$$T(n) = O(n)$$

2、

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

解：使用 Master 定理：由 $d = 0$ 和 $\log_b a = 0$ 得 $d = \log_b a$ 可得：

$$T(n) = O(\log n)$$

3、

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

解：使用 Master 定理：由 $d = 1$ 和 $\log_b a = 0$ 得 $d > \log_b a$ 可得：

$$T(n) = O(n)$$

4、

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

解：使用 Master 定理：由 $d = 0$ 和 $\log_b a = 1$ 得 $d < \log_b a$　可得：

$$T(n) = O(n)$$

5、

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

解：使用 Master 定理：由 $d = 0$ 和 $\log_b a = 2$ 得 $d < \log_b a$　可得：

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

6、

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

解：使用代入法：

$$\begin{aligned} T(n) &= O(n/2) + \log n \\ &= O(n/4) + \log(n/2) + \log n \\ &= O(1) + \log 2 + \log 4 + \ldots + \log n/2 + \log n \\ &= \log(n \cdot \frac{n}{2} \cdot \frac{n}{4} \ldots 4 \cdot 2) \\ &= \log^2 n - \frac{\log 2}{2}(\log^2 n - \log n) \\ &= O(\log^2 n) \end{aligned}$$

7、

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

解：使用 Master 定理：由 $d = 2$ 和 $\log_b a = \log_2 3$ 得 $d > \log_b a$　可得：

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

2 寻找中位数问题

2.1 Main Idea

根据中位数的定义,将已排好序的数组 $A[1..n]$ 和已排好序的数组 $B[1..m]$ 整合成一组从大到小排列好的序列 $x = (x_1, \ldots, x_{m+n})$，其中位数为:

$$M = \begin{cases} x_{\frac{m+n+1}{2}} & if\ (m+n)\%2 == 1 \\ \frac{1}{2}(x_{\frac{m+n}{2}} + x_{\frac{m+n}{2}+1}) & if\ (m+n)\%2 == 0 \end{cases}$$

问题可转化为: 找到序列 x 中第 k 小的元素。在 (m+n) 为奇数的情况下 $k = \frac{m+n+1}{2}$, 中位数 M 就是 $x[k]$; 而在 (m+n) 为偶数的情况下 $k = \frac{m+n}{2}$, 此时中位数 M 为 $\frac{x[k]+x[k+1]}{2}$。

要想找到序列中第 k 小的元素，受到 Selection Problem 解法的启发，这里采用分治算法的思想，方法如下:

1. 取 $A[k/2]$ 和 $B[k/2]$(无特殊说明除法均向下取整)。
2. 对于 $A[k/2]$ 和 $B[k/2]$ 中的较小者, 最多只会有 $(k/2-1)+(k/2-1) = (k-2)$ 个元素比它小，那么它不可能是第 k 小的元素，因此可以直接把较小者所在数组的第 1 到第 k/2 个元素全部移除；如果 $A[k/2]$ 和 $B[k/2]$ 相等，则移除 $A[1..\frac{k}{2}]$ 和 $B[1..\frac{k}{2}]$ 中的一个，伪代码中默认为 A 数组。
3. 第 2 步可以排除 k/2 个不可能是第 k 小的元素，在新的 A，B 数组中再找第 $k - \lfloor k/2 \rfloor$ 小的元素即可，此时可用递归进行查找，当 k 等于 1 时返回两个数组首元素的最小值。第 3 步中有以下情况需要特别考虑:
   - 如果 A[k/2] 或者 B[k/2] 越界（最多只可能有一个数组越界），则选取对应越界数组中的最后一个元素进行比较，在这种情况下需要根据所排除数的个数减小 k 的值，而不能直接减去 k/2
   - 如果过程中一个数组为空，则说明该数组所有的元素均被排除，此时直接返回另一个数组对应第 k(更新之后的 k) 小的元素

2.2 Pseudo Code

Algorithm 1: $getKthElement(A, la, ra, B, lb, rb, k)$

Input: An array **A**,the range of index **la**, **ra**， An array **B**,the range of index **lb**, **rb**,and the k_{th} smallest element that we want to select

Output: 当 $(ra + rb = n + m)$ 为奇数时输出第 k 个数，而当 $(ra + rb = n + m)$ 为偶数时输出第 k 和 k+1 个数的平均值。

```
1  Function getKthElement(A, la, ra, B, lb, rb, k)
2      if k == 1 then
3          if n + m 为奇数 return min(A[la], B[lb]);
4          else return average value of A[la] and B[lb]
5      end
6      lengthA ← ra − la + 1 ;                  // the length of A
7      lengthB ← rb − lb + 1 ;                  // the length of B
8      // 如果过程中一个数组为空，则说明该数组所有的元素均被
           排除，此时直接返回另一个数组对应第 k 小的元素
9      if lengthA == 0 then
10         if n + m 为奇数 return B[lb + k − 1] ;
11         else return average value of B[lb + k − 1] and B[lb + k]
12     end
13     if lengthB == 0 then
14         if n + m 为奇数 return A[la + k − 1] ;
15         else return average value of A[lb + k − 1] and A[lb + k]
16     end
17     // deleteNum 表示需要排除的数的个数，其在特殊情况下
           不取 k/2
18     if k/2 > ra or k/2 > rb then
19         deleteNum ← 对应越界数组的数组长度
20     else
21         deleteNum ← k/2
22     end
23     if A[la + deleteNum] ≤ B[lb + deleteNum] then
24         getKthElement(A, la + deleteNum, ra, B, lb, rb, k −
             deleteNum)
25     else
26         getKthElement(A, la, ra, B, lb + deleteNum, rb, k −
             deleteNum)
27     end
28 end
```

Algorithm 2: $findM(A, n, B, m)$

Input: 有序数组 $A[1..n]$ 和有序数组 $B[1..m]$

Output: 所有数据的中位数 M

```
1 Function findM(A, n, B, m)
2     if (n+m) is odd number then
3         midIndex ← (m+n+1)/2
4     else
5         midIndex ← (m+n)/2
6     end
7     return getKthElement(A, 1, n, B, 1, m, midIndex)
8 end
```

2.3 Complexity Analysis

以 $T(n, m)$ 代表算法 2 中进行判断的次数，其中 n 和 m 分别代表两有序数组的长度。

- 函数 $findM(A, n, B, m)$ 首先进行一次奇偶数判断
- 调用函数 $getKthElement(A, 1, n, B, 1, m, midIndex)$ 以 $T^{'}(k)$ 代表该函数进行判断的次数，其中 k 在 (m+n) 为奇数的情况下等于 $\frac{m+n+1}{2}$，而在 (m+n) 为偶数的情况下 $k = \frac{m+n}{2}$
  - 函数先进行了四次判断
  - 递归调用进行 $T^{'}(k/2)$ 次判断

  即 $T^{'}(k) = T^{'}(k/2) + 4 = T^{'}(k/2) + O(1) = O(\log k) = O(\log(m+n))$

综上有：

$$T(n, m) = T^{'}(k) + 1 = O(\log(m+n))$$

3 双调序列的最大值问题 20

3.1 Main Idea

认定给定的双调序列中的所有元素互不相同，将双调序列平均分为两半，比较分割线左右两边元素的大小，因为序列为双调序列，其的最大值一定在大元素一侧。剔除小元素所在一侧后对大元素一侧（依旧为双调序列）递归地进行二分、分割，最后剩下的最后一个元素就是双调序列的最大值。

算法的伪代码如下，主程序调用 $findMax(a, 1, n)$ 即可。

3.2 Pseudo Code

Algorithm 3: $findMax(a, p, r)$

Input: 双调序列 $a[p \dots r]$，$1 \le p \le r \le n$，p, r 分别为双调序列 a 的首元素和尾元素的下标

Output: 双调序列 $a[p, r]$ 的最大值

```
1  if p=r then
2  |  return a[p]
3  end
4  lpivot ← ⌊(p+r)/2⌋
5  rpivot ← lpivot + 1
6  if a[lpivot] < a[rpivot] then
7  |  return findMax(a, rpivot, r)
8  else
9  |  return findMax(a, p, lpivot)
10 end
```

3.3 Complexity Analysis

以 $T(n)$ 对算法进行时间复杂度的描述，其中 n 代表双调序列中元素的个数，以比较作为基本运算，行 1 及行 6 进行了两次比较，递归进行了 $T(n/2)$ 次比较。

故有：

$$\begin{cases} T(n) = T(n/2) + O(1) \\ T(1) = 1 \end{cases}$$

根据 Master 定理可得：

$$T(n) = O(\log n)$$

4 字符串等价关系判定问题

20

4.1 Main Idea

如果长度为 n 的字符串 A 和字符串 B 完全相同则二者必然等价。

否则将 A 字符串平均分为两半：A_1 和 A_2，B 字符串平均分为两半：B_1 和 B_2。（能够分为长度相等的两半意味着这种情况下字符串长度 n 必须为 2 的整数次幂）

先判断 A_1 和 B_1 是否等价和 A_2 和 B_2 是否等价，如果均等价则可得到 A 和 B 等价。如果不均等价，再判断 A_1 和 B_2 是否等价且 A_2 和 B_1 是否等价，如果均等价，则可得到 A 和 B 等价。

对于其两个子字符串是否等价的子问题，可以递归地进行判断。

算法如下，主程序调用 $isEquivalent(A, B, n)$ 即可。

4.2 Pseudo Code

Algorithm 4: $isEquivalent(a, b, len)$

Input: 字符串 a 和字符串 b,, 且长度均为 len

Output: 字符串 a 和字符串 b 是否等价, 等价则输出 $True$，不等价则输出 $False$

```
1  if a == b then
2      return True
3  else
4      if log_2(len) 不为整数 then
5          return False
6      else
7          midIndex ← len/2
8          a_1 ← a[0 : midIndex]
9          // 字符 a 进行切片操作得到的左半字符串 a_1
10         a_2 ← a[midIndex : len]
11         // 字符 a 进行切片操作得到的右半字符串 a_2
12         b_1 ← a[0 : midIndex]
13         // 字符 b 进行切片操作得到的左半字符串 b_1
14         b_2 ← a[midIndex : len]
15         // 字符 b 进行切片操作得到的右半字符串 b_2
16         return
             [isEquivalent(a_1, b_1, len/2) and isEquivalent(a_2, b_2, len/2)]
             or
             [isEquivalent(a_1, b_2, len/2) and isEquivalent(a_2, b_1, len/2)]
17     end
18 end
```

4.3 Complexity Analysis

用 $T(n)$ 表示算法的时间复杂度, 其中 n 代表字符串 A、B 的长度。以判断两字符串是否完全相同作为基本运算。

- 行 1 进行了一次字符串的判断
- 行 4 可以看出若 n 不为 2 的整数次幂则 $T(n) = 1$，考虑最坏情况下

的时间复杂度——即 n 为 2 的整数次幂

- 行 16 进行递归时，or 语句只有一种情况能进入下一层而不会产生分支，而一种情况中需要递归调用两次该算法，但算法规模为原来的一般，则 $T(n) = 2(n/2) + O(1)$

综上，最坏情况下的算法复杂度为：

$$\begin{cases} T(n) = 2T(n/2) + O(1) \\ T(1) = 1 \end{cases}$$

根据 Master 定理可得：

$$T(n) = O(n)$$

5 向量的最小和问题

20

5.1 Main Idea

对于两个二维向量 v_i 和 $v_j(1 \le i, j \le n$ 且 $i \ne j)$ 而言，$\|v_i^{k_1} + v_j^{k_2}\|_2$ 最小值一定为：

$$\min_{v_i^{k_1}, v_j^{k_2}} \sqrt{(|x_i| - |x_j|)^2 + (|y_i| - |y_j|)^2}$$

为了得到绝对值的效果，每个二维向量需要变化为相应四种形式的一种以保证变换后横、纵坐标均为正数，假设 n 个二维向量 $v_1, v_2, \ldots, v_n$ 变换为 $v_1^{k_1}, v_2^{k_2}, \ldots, v_n^{k_n}$。二维向量可对应平面直角坐标系第一象限的点：$V_1, V_2, \ldots, V_n$。

那么问题可转化为：平面直角坐标系第一象限上有 n 个点 $V_1, V_2, \ldots, V_n$，$n > 1$，V_i 的直角坐标是 (x_i, y_i)，求距离最近的两个点 V_i、V_j。而向量 $v_i^{k_i}, v_j^{5-k_j}$ 或 $v_i^{5-k_i}, v_j^{k_j}$ 便是原问题的答案。

考虑分治算法，用一条垂直于 x 轴的线 l 将 n 个点构成的点集 V 划分为左半平面 V_L 和右半平面 V_R 两部分，使得 V_L 和 V_R 的点数近似相等，即 $x_l = median\ of\ \{x_i | (x_i, y_i) \in V\}$。

V 中的最邻近点对可能有三种情况：两个点都在 V_L 中；个点都在 V_R 中；或者一个点在 V_L 中，另一个点在 V_R 中。算法分别考虑这三种情况。对于前两种情况，可以分别计算 V_L 和 V_R 中的最邻近点对，这是两个 n/2 规

模的子问题。对于第三种情况，算法需要找到由一个 V_L 中的点和一个 V_R 中的点构成的最邻近点对。假设 V_L 和 V_R 中的最邻近点对的距离分别是 δ_L 和 δ_R，令 $\delta = min\{\delta_L, \delta_R\}$，如果出现了第三种情况，那么这一对点的距离应该不大于 δ，于是，为了找到这样的两个点，只需要检查在直线 l 两侧距 l 不超过 δ 的窄缝内的点即可。

伪代码描述如下，调用 $mainMinDistance(v, \{x_i | x_i \in (x_i, y_i)\}, \{y_i | y_i \in (x_i, y_i)\})$ 即可。

5.2 Pseudo Code

Algorithm 5: $mainMinDistance(v, X, Y)$

Input: n 个二维向量的集合 v。X 和 Y 分别为 v 中所有点的横、纵坐标的集合

Output: 两个向量 v_i, v_j，以及两个整数 k_1, k_2

将 n 个二维向量 $v_1, v_2, \ldots, v_n$ 变换为 $v_1^{k_1}, v_2^{k_2}, \ldots, v_n^{k_n}$ 使得其全落在第一象限，记 $v_1^{k_1}, v_2^{k_2}, \ldots, v_n^{k_n}$ 形成的点集为 V。点 V_i 坐标是 $(x_i^{k_i}, y_i^{k_i})$。

$X_k \leftarrow \{x_i^{k_i} | x_i^{k_i} \in (x_i^{k_i}, y_i^{k_i})\}$

$Y_k \leftarrow \{y_i^{k_i} | y_i^{k_i} \in (x_i^{k_i}, y_i^{k_i})\}$

遍历一遍 X_k 和 Y_k，建立 $y_i^{k_i}$ 对 $x_i^{k_i}$ 的索引，旨在通过 $y_i^{k_i}$ 一次查找到对应 $x_i^{k_i}$

对集合 X_k 和 Y_k 各自从小到大进行快速排序

$minDistance(V, X_k, Y_k)$

// $minDistance$ 函数得到了两个向量 $v_i^{k_i}$ 和 $v_j^{k_j}$

return v_i,v_j 和 k_i,$5 - k_j$ 以及 v_i,v_j 和 $5 - k_i$,k_j

Algorithm 6: $minDistance(V, X, Y)$

Input: n 个点的集合 V，X 和 Y 分别为集合 V 中点的从小到大排好序的横坐标的集合和从小到大排好序的纵坐标的集合

Output: 距离最近的两个点

1 如果 V 中点的数量小于或等于 3，则直接计算其中的最小距离

2 // 作垂直线 l 将 V 近似划分为大小相等的点集 V_L 和 V_R，V_L 的点在 l 左边，V_R 的点在 l 右边

3 $X_L \leftarrow$ X 中前一半元素

4 $X_R \leftarrow$ X 中后一半元素

5 // 需要注意的是集合 Y 不能简单进行拆分而得到排好序的 Y_L 和 Y_R，这时候 $mainMinDistance(v, X, Y)$ 行 4 中建立的索引就可以派上用场

6 顺序扫描 Y 集合，每个元素查找对应 x 坐标，如果 x 坐标小于 x_l 则将元素放在集合 Y_L 中最后一个元素的后面，否则放在集合 Y_R 中最后一个元素的后面，这样就能保证集合 Y_L 是点集 V_L 对应的纵坐标的集合；集合 Y_R 是点集 V_R 对应的纵坐标的集合。且 Y_L、Y_R 均从大到小排序完成。

7 $minDisatance(V_L, X_L, Y_L)$；$\delta_L = V_L$ 中的最小距离

8 $minDisatance(V_R, X_R, Y_R)$；$\delta_R = V_R$ 中的最小距离

9 $\delta \leftarrow min(\delta_L, \delta_R)$

10 // 对于在线 l 左边距离 δ 范围内的每个点，检查 l 右边是否有点与它的距离小于 δ，如果存在则将 δ 修改为新值

11 假设所有距线 l 不超过 δ 的窄缝的点构成集合 S，通过顺序扫描 Y（Y 在执行此算法前已经完成了排序），检查每个点的横坐标，计算它是否距 l 小于 δ，如果是，则把它放到集合 S 中。之后按照 S 中点的纵坐标以从小到大的顺序考查，如果一个点的纵坐标为 y，那么只需要检查那些纵坐标不大于 $y + \delta$ 的点，查看其中是否存在分布在另一侧，且与该点的距离小于 δ 的点。

12 **return** 最小的 δ 对应的 $v_i^{k_i}$ 和 $v_j^{k_j}$

5.3 Complexity Analysis

用 $T(n)$ 表示算法的时间复杂度，其中 n 为二维向量的个数。

- $mainMinDistance(v, X, Y)$ 中行 1 需要遍历一遍集合 v，时间复杂度为 $O(n)$
- $mainMinDistance(v, X, Y)$ 中行 4 建立索引需要遍历一遍集合 V，时间复杂度为 $O(n)$
- $mainMinDistance(v, X, Y)$ 中行 5 对两个大小为 n 的集合进行快速排序，时间复杂度为 $O(n \log n)$
- $mainMinDistance(v, X, Y)$ 中行 6 调用 $minDistance(V, X_k, Y_k)$，记其时间复杂度为 $T_1(n)$，n 同样为二维向量的个数：
  - 行 1 计算距离时间复杂度为 $O(1)$
  - 行 3、行 4 只需要进行一次二分即可，时间复杂度为 $O(1)$
  - 行 6 通过索引拆分集合 Y，需要扫描一遍集合，则时间复杂度为 $O(n)$
  - 行 7、行 8 递归调用，总时间复杂度为 $2T_1(n/2)$
  - 分析行 11 的过程：需要从小到大顺序扫描集合 Y，每一个点都需要检查相邻纵坐标不大于 $y + \delta$ 的点，而检查相邻的点需要几次呢？设 V_i 是在线 l 左边的任一点，坐标为 (x_i, y_i)。在右边窄缝内距离 V_i 小于 δ 的点其纵坐标一定在 $y_i + \delta$ 与 $y_i - \delta$ 之间，即这些点只能位于右边高度不超过 2δ，宽度不超过 δ 的矩形区域内，将矩形分成 6 个相等的小矩形，每个小矩形的宽是 $\delta/2$，高是 $2\delta/3$, 小矩形对角线长度为 $5\delta/6$，而前面 δ 为 $min(\delta_L, \delta_R)$，小矩形在线 l 的同一侧，其中不可能有两个点（否则两点之间距离就比 δ 小了，与 δ 为同侧两点间距离的最小值相矛盾）。即任何小矩形内至多只有一个点，则右边与 V_i 的距离小于 δ 的点数最多可能有六个。因此每遍历一个元素时检查相邻的点的次数是一个小于十二的常数，则时间复杂度为 $O(n)$

综上：$T_1(n) = 2T_1(n/2) + O(n)$，根据 Master 定理可得 $T_1(n) = O(n \log n)$。因此：$T(n) = O(n) + O(n \log n) + T_1(n) = O(n \log n)$。

该算法的时间复杂度为 $O(n \log n)$