# Problem Set 2 Answer Sheet

**151220131 谢旻晖**

## Problem 2.1

1.

冒泡排序中的循环不变量为：每次循环后，数组 $A[i......n]$ 为原数组中 $n-i+1$ 个最大的数且已排好。

下面利用循环不变量证明算法的正确性：

**初始化**：当 $i=n$ 时, 内层循环迭代了 $n-1$ 次使得整个数组中最大的数交换至 $A[n]$ 处。此时，循环不变量成立。

**保持**: 设 $i=k$ 时循环不变量成立，当 $i=k-1$ 时，内层循环从 $j=1$ 迭代到 $j=k-2$, 将 $A[1......k-1]$ 中的最大数交换至 $A[k-2]$, 且由假设的循环不变量得 $A[k......n]$ 已排好易得 $A[k-1,k,......n]$ 也已排好，循环不变量仍然成立。

**结束**: 当 $i=2$ 时算法结束，此时 $A[2......n]$ 已排好，且是 $n$ 个元素中较大的那 $n-1$ 个，所以 $A[1......n]$ 自然有序，算法正确性得证。

2.

我们从逆序对的角度考虑，由于是相邻元素比较，一次比较最多消除**一个**逆序对。

**在最坏情况下:** 最多可以产生 $\frac{n(n-1)}{2}$ 个逆序对，所以最坏情况下算法的时间复杂度为 $\Theta(n^2)$。

**在平均情况下:** 我们考虑任意的一种序列 $x_1,x_2......x_i......x_j......x_n$ 与他的转置 $x_n,x_{n-1}......x_j......x_i......x_1$, 逆序对 $(x_i,x_j)$ 只有一个。而总的逆序对 $(x_i,x_j)$ 的组合有 $\binom{n}{2}=\frac{n(n-1)}{2}$ 种，所以逆序对出现的平均次数为 $\frac{n(n-1)}{4}$, 即在平均情况下算法的时间复杂度为 $\Theta(n^2)$。

3.

并不会改变，从逆序对的角度来看，题述的方法并没有改变算法”一次比较至多减少一个逆序对”的本质。

## Problem 2.2

同高度下，满二叉树叶子结点最多，高度为 h 的满二叉树叶子结点数目为 $2^h$，所以 $L <= 2^h$。

## Problem 2.5

1. 见 **Algorithm 1**

**Algorithm 1** PARTITION$(A,1,n)$

```
i = 0
for j = 0 to n do
  if A[j] <= 0 then
    i = i + 1
    swap(A[i], A[j])
  end if
end for
return i
```

2. 见 **Algorithm 2**

## Problem 2.6

1. $D-ARY-PARENT(i)=\left\lfloor \frac{i-2}{d}+1 \right\rfloor$
2. $D-ARY-CHILD(i,j)=d(i-1)+j+1$

定义 $(a,b)$ 为深度为 $a$ 的第 $b$ 个节点（从 0 开始计数，且 $1 \leq b \leq d^{a-1}$）。

**Algorithm 2** PARTITION$(A, 1, n)$

```
i = 0
j = n + 1
for x = p to r do
  if the color of A[x] is yellow then
    i = i + 1
    swap(A[x], A[i])
  else if the color of A[x] is blue then
    j = j − 1
    swap(A[x], A[j])
  end if
end for
return
//1 到 i 为黄球,i+1 到 j-1 为红球，j 到 n 为蓝球
```

那么由等比求和公式，我们易得 $(a, b)$ 在数组中的下标 $i$ 为:

$$i = \frac{1-d^a}{1-d} + b$$

**首先证明 1 式成立：** 显然地，$(a, b)$ 的父结点可用上述坐标表示为 $\left(a-1, \left\lfloor \frac{b}{d} \right\rfloor + 1\right)$。那么由上面的公式，我们有

$$\left(a-1, \left\lfloor \frac{b}{d} \right\rfloor + 1\right) \quad = \tfrac{1-d^{a-1}}{1-d} + \left\lfloor \tfrac{b}{d} \right\rfloor + 1$$

那么我们只要证明等式

$$\frac{1-d^{a-1}}{1-d} + \left\lfloor \frac{b}{d} \right\rfloor + 1 = \left\lfloor \frac{i-2}{d} + 1 \right\rfloor$$

要证明上式成立，只要证明

$$\frac{1-d^{a-1}}{1-d} + \left\lfloor \frac{b}{d} \right\rfloor + 1 = \left\lfloor \frac{i-2}{d} + 1 \right\rfloor$$

只要证明

$$\frac{1-d^{a-1}}{1-d} + \left\lfloor \frac{b}{d} \right\rfloor + 1 = \left\lfloor \frac{\frac{1-d^a}{1-d} + b - 2}{d} + 1 \right\rfloor$$

等式两边同时-1, 再乘以 (1-d)，只要证明

$$1-d^{a-1}+(1-d)\left\lfloor \frac{b}{d} \right\rfloor = \left\lfloor \frac{1-d^a}{d} - \frac{2(1-d)}{d} + \frac{b(1-d)}{d} \right\rfloor$$

只要证明

$$1 = \left\lfloor 2 - \frac{1}{d} \right\rfloor$$

由于 $d \geq 1$, 所以上式子成立，命题得证。

**再证明 2 式成立**:

设节点下标为 $i$ 的结点用坐标表示为 $(a, b)$，那么他的第 $j$ 个子女结点可用坐标表示为 $(a+1, (b-1)*d+j)$

$$i = (a, b) = \frac{1-d^a}{1-d} + b$$

$$(a+1, (b-1)*d+j) = \frac{1-d^{a+1}}{1-d} + (b-1)*d + j$$

要证明

$$D - ARY - CHILD(i, j) = d(i-1) + j + 1$$

只要证

$$\frac{1-d^{a+1}}{1-d} + (b-1)*d + j = d\left(\frac{1-d^a}{1-d} + b - 1\right) + j + 1$$

只要证

$$1-a^{a+1}+d(1-d)(b-1) = d(1-d^a)+d(1-d)(b-1)+(1-d)(j+1)$$

上式展开，左右相等，证明完毕。

# Problem 2.7

合并时, 每次总是要么取 A 中的最小的，要么取 B 中的最小的。如果该次取了 A 中的，则记 $'a'$，如果该次取了 B 中的，则记 $'b'$。那问题就转化为有 $k$ 个 $'a'$,$m$ 个 $'b'$ 的字母排列有多少种,很显然是 $\binom{n}{k} = \binom{n}{m}$。

# Problem 2.10

采用一种类似于快速排序的算法, 从未匹配的螺母中拿一个 $x$，将所有的螺钉逐个进行匹配，将螺钉比螺母 $x$ 小的归为一组 $A$，螺钉比螺母 $x$ 大的归为一组 $B$，并且还找到了一个匹配的螺钉，记为 $X$。将 $x$ 匹配的螺钉 $X$ 旋入所有除了 $x$ 的螺母，将比螺钉 $X$ 大的螺母归为一组 $a$, 将比螺钉 $X$ 小的螺母归为一组 $b$。分别再对组 $(A, a)$, 组 $(B, b)$ 再进行上述操作。

算 法 伪 代 码 表 示 见 **Algorithm 3**

**Algorithm 3** QUICK_MATCH (S,s)

**Input:** 螺钉集合 S，螺母集合 s

**Output:** 形如 $\{(X_1, x_1), (X_2, x_2)……(X_n, x_n)\}$ 的匹配集合

**if** $|S| == 0$ **then**
  **return** $\varnothing$
**end if**
$RESULT = \varnothing$
$A = \varnothing, a = \varnothing$
$B = \varnothing, b = \varnothing$
取 $x \in s$
**for** each $I \in S$ **do**
  **if** $I > x$ **then**
    $A = A \cup I$
  **else if** $I < x$ **then**
    $B = B \cup I$
  **else**
    $X = I$
    $RESULT = RESULT \cup (X, x)$
  **end if**
**end for**
**for** each $i \in s$ **do**
  **if** $i > X$ **then**
    $a = a \cup i$
  **else if** $i < X$ **then**
    $b = b \cup i$
  **else**
    *do nothing*
  **end if**
**end for**
**return** $RESULT \cup QUICK\_MATCH(A, a) \cup QUICK\_MATCH(B, b)$

**QUCIK_MATCH**

我们易得

$$T(n) = 2T(\frac{n}{2}) + \Theta(n)$$

$$T(n) \in \Theta(n \log n)$$

# Problem 2.11

1. 算法见 **Algorithm 4** $COUNT\_INVERSIONS$, 调用时候令 $p = 1, r = n$

2. 算法见 **Algorithm 5** $COUNT\_INVERSIONS\_MODIFIED$, 调用时候令 $p = 1, r = n$, 只要对上题算法稍作改动即可 (改动主要于 $line$ 25)

# Problem 2.12

1.i)

如果存在 14 个不同的常见元素，则总元素个数大于 n，矛盾。

1.ii)

x 是 $R[1...n]$ 的常见元素，所以至少有 $\frac{n}{13}$ 个 $x$ 在数组中，将数组 $R[1...n]$ 划分为 $R[1...\frac{n}{2}]$ 和 $R[\frac{n}{2} + 1...n]$ 两个子数组, 则至少有一个子数组中的 $x$ 的个数大于等于 $\frac{n}{26}$, 根据定义，$x$ 是这个子数组的常见元素。

1.iii)

算法思想：由 ii) 问，$x$ 若是 $R[1..n]$ 的常见元素，那也至少是两个子数组中的一个数组的常见元素。所以可以先递归调用两次，把两个子数组的常见元素找出（最多有 26 个）, 再逐个统计他们在 $R[1...n]$ 中出现的个数，$T(n) = 2T(\frac{n}{2}) + 26n$,$T(n) \in \Theta(n \log n)$

算法伪代码见 **Algorithm** 6, 初次调用时，令 p=1,r=n

2.

仍然能正常工作，算法与 k 的值无关。

**Algorithm 4** COUNT_INVERSIONS($A[\,], p, r$)

**if** $p < r$ **then**
  $q = \lfloor \frac{p+r}{2} \rfloor]$
  $subcount_1 = COUNT\_INVERSIONS(A, p, q)$
  $subcount_2 = COUNT\_INVERSIONS(A, q+1, r)$
  //MERGE(A,p,q,r)
  $n_1 = q - p + 1$
  $n_2 = r - q$
  let $L[1..n_1 + 1]$ $and$ $R[1..n_2 + 1]$ be new arrays
  **for** $i = 1$ $to$ $n_1$ **do**
    $L[i] = A[p + i - 1]$
  **end for**
  **for** $j = 1$ $to$ $n_2$ **do**
    $R[j] = A[q + j]$
  **end for**
  $L[n_1 + 1] = \infty$
  $R[n_2 + 1] = \infty$
  $i = 1$
  $j = 1$
  $count = 0$
  **for** $k = p$ $to$ $r$ **do**
    **if** $L[i] \leq R[j]$ **then**
      $A[k] = L[i]$
      $i = i + 1$
    **else**
      $count = count + n_1 - i + 1$//which is attached to mergesort
      $A[k] = L[j]$
      $j = j + 1$
    **end if**
  **end for**
  **return** $count + subcount_1 + subcount_2$
**else**
  **return** 0
**end if**

**Algorithm 5** COUNT_INVERSIONS_MODIFIED($A[\,], p, r, C$)

**if** $p < r$ **then**
  $q = \lfloor \frac{p+r}{2} \rfloor]$
  $subcount_1 = COUNT\_INVERSIONS\_MODIFIED(A$
  $subcount_2 = COUNT\_INVERSIONS\_MODIFIED(A$
  $1, r, C)$
  //MERGE(A,p,q,r)
  $n_1 = q - p + 1$
  $n_2 = r - q$
  let $L[1..n_1 + 1]$ $and$ $R[1..n_2 + 1]$ be new arrays
  **for** $i = 1$ $to$ $n_1$ **do**
    $L[i] = A[p + i - 1]$
  **end for**
  **for** $j = 1$ $to$ $n_2$ **do**
    $R[j] = A[q + j]$
  **end for**
  $L[n_1 + 1] = \infty$
  $R[n_2 + 1] = \infty$
  $i = 1$
  $j = 1$
  $count = 0$
  **for** $k = p$ $to$ $r$ **do**
    **if** $L[i] \leq R[j]$ **then**
      $A[k] = L[i]$
      $i = i + 1$
    **else**
      **if** $L[i] \geq C * R[j]$ **then**
        $count = count + n_1 - i + 1$//which is attached to mergesort
      **end if**
      $A[k] = L[j]$
      $j = j + 1$
    **end if**
  **end for**
  **return** $count + subcount_1 + subcount_2$
**else**
  **return** 0
**end if**

**Algorithm 6** FOUND_COMMON_ELEMENTS(R[ ],p,r)

**Input:** R 为数组，初次调用时 $p = 1, r = n$

**Output:** R 中的常见元素集合

**const** $frequence = 13$
**if** $r - p + 1 \leq frequence$ **then**
  //当数组数目比较小的时候，每个元素都是常见元素
  **return** $set(R)$//集合化 R
**end if**
$q = \lfloor \frac{p+r}{2} \rfloor]$
$A = FOUND\_COMMON\_ELEMENTS(R, 1, q)$
$B = FOUND\_COMMON\_ELEMENTS(R, q+1, n)$
$RE = \varnothing$//结果集合
**for each** $i \in A \cup B$ **do**
  $count = 0$//统计 $i$ 在 $S$ 中出现的次数
  **for each** $j \in R$ **do**
    **if** (check(i,j))=true) **then**
      $count = count + 1$
    **end if**
  **end for**
  **if** $count \leq \frac{|S|}{frequence}$ **then**
    $RE = RE \cup \{i\}$
  **end if**
**end for**
**return** RE

3.

可以有 $\Theta(n)$ 的算法解决。算法见 **Algorithm** 7。采用 PK 法，存储当前遇到“最强”的元素以及他的“强度”，当新的元素到来时，如果与他相等，则“强度”加二，如果不等，则“强度”不加，当遍历的 i 和“强度”碰头的时候，将“最强”元素替换为新的元素。这样遍历一遍数组之后，这个“最强”元素就是 $k = 2$ 时的疑似常见项，再遍历一遍数组数一数他的个数，确定他是否为常见项。

**Algorithm 7** FOUND_COMMON_ELEMENTS(k=2)($R[0...n-1]$)

$strong\_ele = R[0]$
$strength = 2$
**for** $i$ in $[1, 2...n-1]$ **do**
  **if** $check(R[i], strong\_ele) == true$ **then**
    $strength += 2$
  **else if** $strength == i$ **then**
    $strength += 2$
    $strong\_ele = b[i]$
  **end if**
**end for**
$count = 0$
**for each** in $R[0...n-1]$ **do**
  **if** $check(each, strong\_ele) == true$ **then**
    $count++$
  **end if**
**end for**
**if** $count \geq n/2$ **then**
  **return** $strong\_ele$
**else**
  **return** $\varnothing$
**end if**

4.

常见项问题的下界是 $O(n \log k)$，是与基于比较的排序同等难度。采用决策树模型，令 $r = n/k$，利用老师给出论文[1]中的 **r-lists** 证明。论文中证明

[1] Misra, Jayadev, and David Gries. "Finding repeated elements."

了对于 $b[0...n-1]$ 至少有 $(\frac{k}{e})^n$ 种不同的 **r-lists**，且对于每种 **r-lists** 执行基于比较的算法，决策树终止于不同的叶节点。因此，我们就有 $(\frac{k}{e})^n$ 个叶节点。

$$O(\log{(\frac{k}{e})^n}) = O(n*\log k - n*\log e) = O(n*\log k)$$

## Problem 2.14

**1.1** 参见 **Algorithm 8**

**Algorithm 8** FOUND_MAXIMA_WITH_SORT($S[\,], n$)

**Input:** $S[1...n]$ ={所有点坐标的数组}//约定可以通过 $S[1].x$，这种形式访问 x 坐标分量，y 类似
**Output:** *maxima*

```
sort(S)//将 S 以 x 轴坐标降序排序，当 x 轴坐标相同时，以 y 轴坐标降序排。
maxy = S[1].y
index = 1
while S[index + 1].x == S[1].x do
  index++
end while
//横坐标最大的点一定是 maxima
add S[1...index] to maxima
//对于剩下的点
for each in S[index + 1 ...n] do
  if each.y ≥ maxy then
    add each to maxima
    maxy = each.y//更新 maxy
  end if
end for
return maxima
```

**1.2** 参见 **Algorithm 9**

**Algorithm 9** FOUND_MAXIMA_DIVIDE&CONQUER($S$)

**Input:** $S = \{(x_1, y_1), (x_2, y_2)...(x_n, y_n)\}\}$//S 为输入点坐标无序集
**Output:** *maxima* 集合

```
//basecase:
if S == ∅ then
  return ∅
else if S 中每个点的 x 坐标均相同, 即点在一条竖线上 then
  return S
end if
通过 Θ(n) 的算法，将 S 集合分为左一半和右一半–黄鱼说的可以这样假设.
S = {A, B}
RA=FOUND_MAXIMA_DIVIDE&CONQUER(A)
RB=FOUND_MAXIMA_DIVIDE&CONQUER(B)
RESULT = RB
find the point which has the biggest y value in RB, we assume its y value is maxy
for each_point in RA do
  if each_point.y ≥ maxy then
    RESULT = RESULT ∪ {each_point}
  end if
end for
return RESULT
```

**2.**

两种思路的错误本质之处都是一样的

- 对于第一种思路，思路想均衡得分开那些点，但我们总能找到情况使得这四个象限里

Science of computer programming 2.2 (1982): 143-152.

的点数差别很大。比如右上角全是点，而其他象限点则只有一个。此时，递归式应该是 $T(n) = T(n-2) + T(1) + T(1) + O(n)$, 原递归式 $T(n) = 3T(\frac{n}{4}) + O(n)$ 也就不成立了。

- 对于第二种思路有两个错误。一是如果左下象限都没有点，我们就没有办法减少问题规模进行 divide&conquer 了，递归式也就不成立了; 二是我们无法在 $O(n)$ 的复杂度之内完成 Conquer 步骤。

**3.**

只需要证明该算法的下界是 $\Theta(n\log n)$ 即可说明问题。

决策树的叶节点是具有如下性质的 maxima 点**序列**

- 定义一种集合叫 **binding** 集合, 他是具有这样性质的集合: 集合中只有一个 $maxima$ 点或者两个 $maxima$ 点，分别叫 $1-binding$ 和 $2-binding$,$2-binding$ 内部是无序的。
- 由于一次比较最多产生一对 $maxima$ 点，也即最多产生一个 **binding** 集合。
- 决策树叶节点序列是这样的若干个 **binding** 集合的有序排列。

下面我证明这颗决策树至少有 $\Theta(n!)$ 个叶节点　　设某次算法执行后的结果序列中有 $k$ 个 maxima 点，其中有 $i$ 个 **2-binding**,$k-2i$ 个 **1-binding**, 我们计算可能的结果序列有多少种。
首先从 $i+(k-2i)$ 个 **bindings** 中选 i 个位置给 **1-binding**, 是 $C_{k-i}^{i}$。接着我们把 k 个 maxima 点，随机地排入这些 **bindings** 中，注意到其中的 **2-binding** 的内部两个点的在结果序列中的顺序是我们不关心的，所以是 $\frac{A_k^k}{2^i}$。

接着，我们对所有的 k 和 i 求和，注意到 k 的取值是从 1 到 n,i 的取值是 0 到 $\frac{k}{2}$

$$\begin{aligned}\sum_{k=1}^{n}\sum_{i=0}^{\frac{k}{2}} C_{k-i}^{i}\frac{A_k^k}{2^i} &> \sum_{k=1}^{n} A_k^k \sum_{i=0}^{\frac{k}{2}} \frac{1}{2^i} \\ &= \sum_{k=1}^{n} A_k^k (2-(\frac{1}{2})^{\frac{k}{2}}) \\ &> A_n^n(2-(\frac{1}{2})^{\frac{k}{2}}) \\ &= \Theta(n!)\end{aligned}$$

证得之后我们即可有决策树叶节点个数 $l \geq n!$, 即有

$$\begin{aligned} n! \leq l &\leq 2^h \\ h &\geq \log n! \\ h &= \Omega(n\log n)\end{aligned}$$

因此，我们有算法下界为 $\Theta(n\log n)$

## Problem 2.17

根据水平方向的坐标把平面上的 N 个点均分成两部分 Left 和 Right，且这两个部分的点数差不多。假设递归求出了 Left 和 Right 两个部分最近的点对之最短距离为 MinDist(Left) 和 MinDist(Right)，现在我们只要考虑点对中一个点在 Left 部分，另一个点在 Right 部分。

我们取 MDist=$min(MinDist(Left), MinDist(Right))$, 则在中轴线两边各 MDist 距离的点，不可能是最近点对。

我们在这个带状区域考虑，如果一个点对的距离小于 MDist，那么它们一定在一个 MDist*(2*MDist) 的区域内。

而在这左右两个 MDist*MDist 正方形区域内，最多都只能含有 4 个点。如果超过 4 个点，则这个正方形区域内至少有一个点对的距离小于 Mdist，矛盾了。因此，一个 MDist*(2MDist) 区域中最多有 8 个点。因此我们可以遍历 $O(n)$ 个如图所示的矩形，每遍历一个矩形最多尝试 8 个点一定能找到最短距离。

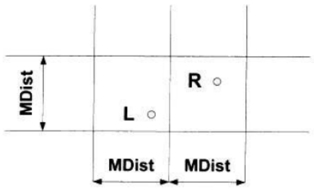


图 1

因此，我们可以写出递归式：$T(n) = 2T(n/2) + O(n)$，可以用主项定理 (master method) 解得时间复杂度 $T(n) = O(n \log n)$。加上排序一次的时间 $O(n \log n)$，因此整个算法的运行时间 $T(n) = O(n \log n)$。