Notes for Tutorial Two

一、 分析：递归方程 vs 直接展开 【*Ref: 算法导论*】

Def：指标随机变量 X_{ij}

$$X_{ij}=\begin{cases}0, & \textbf{若}\boldsymbol{x_i}\textbf{与}\boldsymbol{x_j}\textbf{发生了比较}\\ 1, & \textbf{若}\boldsymbol{x_i}\textbf{与}\boldsymbol{x_j}\textbf{没发生比较}\end{cases}$$

快速排序的分析：

引入了指标随机变量后，快速排序的算法可以表述为：

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n}\mathrm{X}_{ij} \qquad ①$$

由于期望的线性特征，我们进一步有：

$$\mathrm{E}\left[\sum_{i=1}^{n}\sum_{j=i+1}^{n}\mathrm{X}_{ij}\right]=\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\mathrm{E}[\mathrm{X}_{\mathrm{ij}}] \qquad ②$$

注意到输入的随机性，我们不妨考虑以下这种情况：

$$\underbrace{Z_1<Z_2<\cdots<Z_{i-1}}_{\text{无影响}}<Z_i<\underbrace{Z_{i+1}<\cdots<Z_{j-1}}_{\text{不比较}}<Z_j<\underbrace{Z_{j+1}<\cdots<Z_n}_{\text{无影响}}$$

比较　　比较

根据快速排序的性质，我们可知：处于 pivot 左边与右边的元素不再会发生比较。故可得：

$$\Pr\{Z_i\textit{和}Z_j\textit{发生了比较}\}=\frac{2}{j-i+1}$$

代入②式可直接求出快速排序的时间复杂度为：$\mathrm{O}(n\log n)$。

二、 蛮力排序

几个时间复杂度为$\mathrm{O}(n^2)$的排序算法：$\begin{cases}\text{选择排序}\\ \text{冒泡排序}\\ \text{插入排序}\end{cases}$

其中，选择和冒泡的最坏情况和平均情况的代价相同，
　　插入排序的最坏情况比平均情况略好。

例：改进（无效）的冒泡排序分析

Record

2 1 **3** 4 5

IDEA：假定序列中从某一个数开始，其后的序列都有序，则记录该位置，每次比较只需要截止到 Record 的位置，如果被比较的数小于 Record 数，则不往后比较。例如，在序列 2,1,3,4,5 中，令 Record=3，则当冒泡比较到 2<3 时，停止比较，2 无需再与 3 之后的 4 和 5 发生比较。

分析：记序列中有序的尾部序列长度为 k，则该改进算法只对$\frac{1}{k!}$ 的情况有效；

结论：在大 O 意义下，该改进算法并不能提升算法效率；

理解：用消除 inversion 的视角去审视，该算法并不能减少消除逆序对的次数；

三、 堆结构的各种性质

例 1：证明 $\sum h \leq n-1$ 在堆中始终成立。
性质：由堆的性质可知，两个子树之中至少有一个为完美二叉树。
提示：用数学归纳法证明时，针对完美二叉树的一边子树可以用公式计算，而非完美二叉树的一边子树用归纳假设计算。

例 2：给定一个有 n 个元素的堆，它最大的元素在哪？根。它第二大的元素可能出现在哪？根的左右子节点。那么它第 k 大的元素在哪？请给出找到堆中第 k 大元素的算法。（记根节点为第 1 层）
Brute Force 解：采用堆排序，在排好序的序列中直接找到位于第 k 个位置的元素。
此种解法的时间复杂度即位堆排序的时间复杂度：$\mathrm{O}(n\log n)$
Challenge：如何找到与 n 无关，而仅与 k 有关的算法？
性质：由此可知：第 k 大的元素至多出现在第 k 层中，因为由堆的性质可知，第 k 层的节点至少有 k-1 个祖先（比它大的元素），第 k+1 层往后的元素不可能为第 k 大；
IDEA：首先找出 n 个元素组成的堆中前 k 层的所有2^k个节点，然后在其中找第 k 大的元素，每次在这个子堆中删除最大的元素后 fix，如此往复 k 次后即可得到答案。此种解法的时间复杂度为：$\mathrm{k}\cdot\log 2^k = k^2$。

四、 Anagram（换位词、变位词）问题【*Ref: Online Judge 作业二 Problem A*】

解法：PHASE ONE：根据单词的字长排序；
PHASE TWO：对每个单词排字母序，记为该单词的 ID；
PHASE THREE：对所有等长的单词根据 ID 排序；

五、 常见项问题 【*Ref: finding repeated elements*】

以 k=2 为例，常见项问题即为找到数组中出现次数大于$\frac{n}{2}$的项。

IDEA：PK 法求解，将数组中的元素取出并两两比较，如果两个数相同，则保留这两个数，否则将它们都删掉，如此往复直到数组中只剩下相同值的元素，即为常见项。

用分治法求解：假定时间复杂度为$\mathrm{O}(n\log n)$，则可得递推方程为：

$$\mathrm{T}(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

如何保证不会漏数：原问题的解一定是某个子问题的解；
如何保证不会多数：将每个 candidate 依次扫描确认；

六、 （递推算法的例子）找最近的点对 【*Ref: Algorithm Design p225-231*】

题目描述：给定二维平面上的 n 个点(x_1, y_1),(x_2, y_2),…,(x_n, y_n)。求出这 n 个点中欧几里得距离最近的两个点。
IDEA：类似于最大和子串问题，采用分治法求解；

七、 求坐标系中 Maxima 点的问题【*Ref: Problem Set 第二季 Problem 2.14*】

题目描述：

给定二维平面上的 n 个点(x_1, y_1),(x_2, y_2),⋯ ,(x_n, y_n)。我们定义 "(x_1, y_1)支配(x_2, y_2)" ，如果x_1>x_2，y_1>y_2。一个点被称作为 Maxima，如果没有任何其它点支配它。

IDEA：

1、 按照横坐标对所有点进行排序, 最右边的点（横坐标的值最大）一定是 Maxima；$(\mathrm{O}(n \log n))$

2、 排好序的点从后往前依次扫描，并且记录当前最大的纵坐标的值y_{max}，如果扫面到一个点的纵坐标大于y_{max}，则该点为一个 Maxima，并且更新它的纵坐标的值为y_{max}，继续向前扫描，直至遍历完所有的点； $(\mathrm{O}(n))$