Homework 2

Name: 方嘉聪 ID: 2200017849

Problem 1 (Textbook 2.7). 设 A 是含有 n 个元素的数组, 如果元素 x 在 A 中出现的次数大于 $n/2$, 则称 x 是 A 的主元素.

(1) 如果 A 中的元素是可以排序的, 设计一个 $O(n\log n)$ 时间的算法判断 A 是否有主元素.

(2) 对于 (1) 中可排序的数组, 能否设计一个 $O(n)$ 时间的算法.

(3) 如果 A 中元素只能进行"是否相等"的测试, 但无法排序, 设计一个算法判断 A 是否有主元素.

◀

Answer. (1) 算法一: **基于排序的算法**. 先对数组进行排序, 再遍历数组记录出现次数最多的元素, 最后检查是否为主元素. 时间复杂度 $T(n) = O(n\log n) + O(n) + O(n) = O(n\log n)$, 具体思路见下:

Algorithm 1: Sort and Count(By Vote)

Input: An array A of size n

Output: Whether A has a majority element

1 Sort the array A in $O(n\log n)$ time.
2 Initialize $count = 1$, $major = A[0]$.
3 Find the maximum number of the same element in A **by vote** in $O(n)$ time.
4 Check the number of the $major$ in A in $O(n)$ time.

算法二: **基于分治的算法**. 首先证明下面一个引理:

Lemma 1. 如果数组 A 有主元素, 则数组 A 的子数组 $A[0:n/2]$ 和 $A[n/2:n]$ 中至少有一个有主元素.

证明. 设数组 A 的主元素为 x, $L_1 = A[0:n/2], L_2 = A[n/2:n]$, 设 L_1 中含有 x 的个数为 i, L_2 中含有 x 的个数为 j, 如果 $i \le \lfloor n/2 \rfloor, j \le \lceil n/2 \rceil$, 那么有 $i + j \le n$, 矛盾. 因此至少有一个子数组中含有主元素. □

Algorithm 2: Divide and Conquer

Input: An array A of size n

Output: the majority element x and whether A has a majority element

1 **if** $n = 1$ **then**
2 **return** $A[0]$, *True*
3 $m_1 \leftarrow DivideAndConquer(A[0:n/2])$, $m_2 \leftarrow DivideAndConquer(A[n/2:n])$
4 **if** $m_1 = m_2$ **then**
5 **return** m_1, *True*
6 **else if** m_1 *and* m_2 *don't exist* **then**
7 **return** *None, False*
8 **else**
9 check the number of m_1 and m_2(if it exists) in A

第 9 行的检查操作可以在 $O(n)$ 时间内完成, 因此整个算法的时间复杂度为 $T(n) = 2T(n/2) + O(n) = O(n \log n)$.

(2) 先证明一个引理:

Lemma 2. *A 中的主元素一定是中位数.*

证明. 假设主元素不是中位数, 那么在有序的数组中主元素只能分布在中位数的一侧, 那么主元素的个数一定不超过 $n/2$, 矛盾. □

Algorithm 3: Find Median and Check

Input: An array A of size n

Output: Whether A has a majority element

1 Find the median of A in $O(n)$ time.

2 Check the number of the median in A in $O(n)$ time.

第 1 行可以使用课上证明的**确定性快速选择算法**. 总的时间复杂度 $T(n) = O(n) + O(n) = O(n)$.

(3) 采用两两比较并将不同的元素删除的方法 (**类似课上提及的芯片测试算法**), 最后剩下的元素即为主元素. 时间复杂度为 $O(n)$. 伪代码如下:

Algorithm 4: Delete Different Elements

Input: An array A of size n

Output: Whether A has a majority element

```
1  while |A| > 1 do
2  |  Group the elements in A in pairs
3  |  for each group do
4  |  |  if the two elements are different then
5  |  |  |  Delete the two elements.
6  |  |  else
7  |  |  |  Keep one element.
8  |  end
9  end
10 Check the remaining element in A in O(n) time.
```

时间复杂度 $T(n) = T(\frac{n}{2}) + O(n) \implies T(n) = O(n)$.

◁

Problem 2 (Textbook 2.27). 如下图所示, 城市街道都是水平或垂直分布, 有 $m+1$ 条, 不妨设任何两个相邻位置之间的距离都为 1. 在街道的十字路口有 n 的商店, 图中 $n = 3, m = 8$, 三个商店的位置坐标分别为 $(2,4), (5,3), (6,6)$. 现在需要在某个路口位置见一个合用的仓库. 若仓库选择 $(3,5)$ 的位置, 那么这 3 个商店到仓库的路程 (只能沿街道行进) 总长至少为 10. 请设计一个算法找到仓库的最佳位置, 使得所有商店到仓库路程的总长最小. ◀

Answer. 不妨设 n 个点的坐标为 $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$, 那么目标为

$$\min_{(x,y)} \sum_{i=1}^{n} |x - x_i| + |y - y_i|$$

注意到横纵坐标是独立的, 因此可以分别求解. 不妨先考虑横坐标, 问题转化为

$$\min_{x} \sum_{i=1}^{n} |x - x_i|$$

由于 $|x - x_i|$ 是关于 x 的绝对值函数, 因此可以得到 x 的最优解为 $x = x_{(n+1)/2}$, 即中位数. 同理, y 的最优解为 $y = y_{(n+1)/2}$. 因此仓库的最佳位置为 $(x_{(n+1)/2}, y_{(n+1)/2})$. 那么只要使用快速选择算法 即可在 $O(n)$ 时间内找到中位数, 因此整个算法的时间复杂度为 $O(n)$. ◁

Problem 3 (Forces Between Particles). You guys have learned about how to compute the electrostatic forces between two charged particles in high school physics. Consider a case where we have particles at points $\{1, 2, \cdots, n\}$ on the real axis, and the particle at point j has charge q_j (the charge can be either positive or negative). By Coulomb's law, the total force on each particle j is:

$$F_j = \sum_{i<j} \frac{Cq_iq_j}{(j-i)^2} - \sum_{i>j} \frac{Cq_iq_j}{(j-i)^2}$$

where C is a given constant (the Coulomb constant). Design an algorithm that computes all the forces $F_j, j = 1, 2, \cdots, n$ in $O(n \log n)$ time. ◀

Answer. 基本思路: 找到两个向量 α 和 β, 使得 α 和 β 的卷积的第 $n + j + 1$ 个分量为

$$(\alpha * \beta)[n + j + 1] := \sum_{i<j} \frac{q_i}{(j-i)^2} + \sum_{i>j} \frac{-q_i}{(j-i)^2}, (j \in \{1, 2, \cdots, n\}) \tag{1}$$

而后我们调用 FFT 算法计算卷积, 最后再对对应分量乘上 Cq_j 即可得到 F_j.
事实上我们考虑

$$\alpha = (q_1, q_2, \cdots, q_n), \beta = \left(-\frac{1}{n^2}, -\frac{1}{(n-1)^2}, \cdots, -1, 0, 1, \cdots, \frac{1}{(n-1)^2}, \frac{1}{n^2}\right)$$

容易证明 α, β 如上取值满足1式. 由于 FFT 算法的时间复杂度为 $O(n \log n)$, 因此整个算法的时间复杂度为 $O(n \log n)$.

◁

Problem 4 (Median of Medians). The $Quickselect(A, k) algorithm$ for finding the $k-th$ smallest element in an unsorted array A picks an arbitrary pivot, then partitions the array into three pieces: the elements less than the pivot, the elements equal to the pivot, and the elements that are greater than the pivot. It is then recursively called on the piece of the array that still contains the $k - th$ smallest element.

(a) Consider the array A = $[1, 2, \cdots, n]$ shuffled into some arbitrary order. What is the worst-case runtime of $Quickselect(A, n/2)$ in terms of n? Construct a sequence of bad pivot choices that achieves this worst-case runtime.

(b) The above worst case has a chance of occurring even with randomly-chosen pivots, so the worst-case time for a random-pivot Quickselect is $O(n^2)$.
Lets define a new algorithm Better-Quickselect that deterministically picks a better pivot. This pivot-selection strategy is called Median of Medians, so that the worst-case runtime of Better-Quickselect(A, k) is $O(n)$.

Algorithm 5: Median of Medians

Input: An array A of size n and an integer k

Output: The $k-th$ smallest element in A

(a) Group the array into $\lfloor \frac{n}{5} \rfloor$ groups of 5 elements each (ignore any leftover elements)

(b) Find the median of each group of 5 elements (as each group has a constant 5 elements, finding each individual median is $O(1)$)

(c) Create a new array with only the $\lfloor \frac{n}{5} \rfloor$ medians, and find the true median of this array using Better-Quickselect.

(d) Return this median as the chosen pivot.

Let p be the chosen pivot. Show that for least $3n/10$ elements x we have that $p > x$, and that for at least $3n/10$ elements we have that $p \leq x$.

(c) Show that the worst-case runtime of Better-Quickselect(A, k) using the Median of Medians strategy is $O(n)$. *Hint: Using the Master theorem will likely not work here. Find a recurrence relation for $T(n)$, and try to use induction to show that $T(n) \leq cn$ for some $c > 0$.*

◀

Answer. (a) 最坏情况下的时间复杂度为 $O(n^2)$, 例如: 以序列

$$\left[1, 2, 3, \cdots, \frac{n}{2} - 1, n, n-1, \cdots, \frac{n}{2} + 1, n/2\right]$$

取 pivot, 这样每次调用只删去了一个元素, 因此时间复杂度为 $O(n^2)$.

(b) 总共分成了 $n/5$ 份, 由于 p 是中位数的中位数, 因此 p 至少大于一半的中位数 (即至少大于 $3 * (n/10) = 3n/10$ 个元素), 至少小于一半的中位数 (即至少小于 $3n/10$ 个元素).

(c) 设小于 p 的元素个数为 S_1, 大于 p 的元素个数为 S_2, 那么 $S_1 \geq 3n/10, S_2 \geq 3n/10 \implies 3n/10 \leq S_1, S_2 \leq 7n/10$. 设最坏情况下时间复杂度为 $T(n)$, 那么有

$$T(n) = T(\frac{n}{5}) + T(\frac{7n}{10}) + O(n)$$

其中, $T(n/5)$ 为求中位数的中位数的时间 (算法中 (c)), $T(7n/10)$ 为递归调用的时间, $O(n)$ 为找到各组中位数的时间 (算法中 (b)). 我们可以通过递归树的方法证明 $T(n) = O(n)$.

◁

Problem 5 (Draw the Skyline!). You are given n non-vertical lines in the plane, labeled $L_1, \cdots, L_n$, with the i^{th} line specified by the equation $y = a_i x + b_i$. We assume that **no three of the lines all**

meet at a single point. We say line L_i is **uppermost** at a given x-coordinate x_0 if its y-coordinate at x_0 is greater than the y-coordinates of all the other lines at x_0, i.e., $a_i x_0 + b_i > a_j x_0 + b_j, \forall j \neq i$. We say line L_i is **visible** if there exists some x-coordinate at which it is uppermost. Give an algorithm that takes n lines as input, and in $O(n \log n)$ time returns all the visible ones.

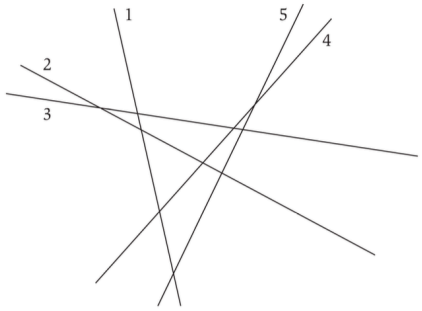

Fig. 1. An example of finding all the visible lines. All the lines except for line 2 are visible.

◀

Answer. 大致思路: 先将所有的线段按照**斜率** a_i **升序排列**为 $[L_{k_1}, L_{k_2}, \cdots, L_{k_n}]$, 注意到斜率最大和最小的线段一定是 *visible* 的, 类似的将所有线段两两分组为 $[L_{k_1}, L_{k_n}], \left[L_{k_2}, L_{k_{n-1}}\right], \cdots$. 对每一组, 我们记录下交点的横坐标 x_i 以及在区间 $(-\infty, x_i), (x_i, \infty)$ 哪条线段是 *uppermost* 的. 而后我们调用 ***Merge*** 算法 (思路为: 在合并两个线段集中的点的同时, 根据该点对应值的大小确定合并后的 *uppermost* 线段) 递归地将线段合并起来, 得到最终的的 *visible* 线段.
Merge 算法基本思路与伪代码见下:

Algorithm 6: Merge

Input: Tow line sets L_1, L_2 with **sorted** intersection-points set $\{x_i\}, \{y_j\}$
Output: One merged line sets L with **sorted** intersection-points set $\{r_t\}$

```
Like merge step in merge-sort, merge {x_i} , {y_i} and new intersection points {n_s} in O(n).
for z_k in {x_i} ∪ {y_j} ∪ n_s do
    if L_1(z_k) > L_2(z_k) then
        Add the uppermostline of L_1 at z_k to L
    else if L_1(z_k) < L_2(z_k) then
        Add the uppermostline of L_2 at z_k to L
    else
        Add the uppermostline of L_1 at z_k to L
    update the intersection-points set {r_t}
end
```

这里 $L_1(z_k)$ 表示 L_1 在 $x = z_k$ 上对应 *uppermost* 线段的 y, 时间复杂度和归并排序的 **Merge** 操作一样, 为 $O(n)$. 下面我们来证明整个算法的时间复杂度为 $O(n \log n)$.

设 $T(n)$ 为 n 个线段的 *visible* 算法的时间复杂度, 那么有 $T(n) = 2T(n/2) + O(n)$, 由 **Master Theorem** 可知 $T(n) = O(n \log n)$.

◁

Problem 6 (Local Minima of A Chessboard). Given a $n \times n$ matrix M, where each element $M[i][j]$,$1 \leq i, j \leq n$ is labeled with a real number $m_{i,j}$; you may assume that all these values are distinct. For each element $M[i][j]$, you can determine the value $m_{i,j}$ by probing it. We define that $M[i][j]$ is a *local minimum iff.* Its value is smaller than the values of all the adjacent (left, right, up, down) elements (except for those falling out of the range of the matrix). Show how to find **a local minimum** of M using only $O(n)$ probes to the elements of M.

◀

Answer. 基本思路, 先从中间行列找到最小值 m, 如果是 *local minimum*, 那么返回 m, 否则, 选择 m 所在的行或列中最小的值, 递归地在这一值所在的象限调用上述算法, 直到找到 *local minimum*. 伪代码如下:

Algorithm 7: Find Local Minimum

Input: A $n \times n$ matrix M
Output: A local minimum of M

```
Find the minimum value m in the middle row and middle column of M in O(n) time.
if m is a local minimum then
    return m
else
    // Find the minimum value m' in the row or column of m in O(n) time.
    Find the minimum value m' in nerighbourhood of m.
    if m' is a local minimum then
        return m'
    else
        Recursively call Find Local Minimum on the submatrix of M that contains m'.
    end
end
```

时间复杂度分析: 注意到每次矩阵大小变为原来的 1/2, 因此时间复杂度为 $T(n) = T(n/2) + cn$, 那么由主定理有 $T(n) = O(n)$. ◁