算法设计 HW01

Xun Ying

2024 年 7 月 8 日

Q1

证明. 根据分支算法的思想，整个n可以分为$\log_b n$层，最后一层是$a^{\log_b n}$个元素。因此，总的时间开销为:

$$
\begin{aligned}
sum &= \sum_{i=0}^{\log_b n} a^i \left(\frac{n}{b^i}\right)^d \left(\log \frac{n}{b^i}\right)^w \\
&= n^d \sum_{i=0}^{\log_b n} \left(\frac{a^i}{b^{id}}\right) \left(\log \frac{n}{b^i}\right)^w
\end{aligned}
$$

又不妨设$n = b^k$，可得:

$$
\begin{aligned}
sum &= n^d \sum_{i=0}^{\log_b n} \left(\frac{a^i}{b^{id}}\right) \left(\log b^{k-i}\right)^w \\
&= n^d \left(\log b\right)^w \sum_{i=0}^{\log_b n} \left(\frac{a}{b^d}\right)^i (k-i)^w
\end{aligned}
$$

我们取$c = 1/b^d$

1.若$a < b^d$

$$
\begin{aligned}
LHS &= n^d \left(\log b\right)^w \left(c^0 k^w + c^1 (k-1)^w + ... + c^k 0^w\right) \\
&< n^d \left(\log b\right)^w k^w \left(c^0 + c^1 + ... + c^k\right) \\
&= n^d \left(\log b\right)^w k^w \frac{1 - c^{k+1}}{1 - c} \\
&< n^d \left(\log n\right)^w
\end{aligned}
$$

所以$a < b^d$时，$sum = O\left(n^d \left(\log n\right)^w\right)$

2.若$a=b^d$

$$\begin{aligned}LHS&=n^d\left(\log b\right)^w\left(k^w+\left(k-1\right)^w+...+0^w\right)\\&<n^d\left(\log b\right)^w k^w\\&=n^d\left(\log n\right)^w\\&=n^d\left(\log n\right)^{w+1}\end{aligned}$$

所以$a=b^d$时，$sum=O\left(n^d\left(\log n\right)^{w+1}\right)$
3.若$a>b^d$

$$\begin{aligned}LHS&=n^d\left(\log b\right)^w\left(c^0k^w+c^1\left(k-1\right)^w+...+c^k0^w\right)\\&=n^d\left(\log b\right)^w c^k\left(\frac{1}{c^k}k^w+\frac{1}{c^{k-1}}\left(k-1\right)^w+...+\frac{1}{c^0}0^w\right)\\&=n^{\log_b a}\left(\log b\right)^w\left(\frac{1}{c^k}k^w+\frac{1}{c^{k-1}}\left(k-1\right)^w+...+\frac{1}{c^0}0^w\right)\end{aligned}$$

又由d'Alembert判别法得，括号中的项是一个收敛的级数，所以

$$LHS<C*n^{\log_b a}$$

所以$a>b^d$时，$sum=O\left(n^{\log_b a}\right)$

□

Q2 (a) 首先，先随机选择一个数，然后将数组分为两部分，一部分比它小，一部分比它大。从期望的意义下，我们将n分为至少1/3和2/3的两部分，因此根据主定理

$$T\left(n\right)<2T\left(\frac{n}{\frac{2}{3}}\right)+O\left(\frac{2}{3}n\right)$$

因此可得$T\left(n\right)=O\left(n^{\log_{\frac{3}{2}}2}\right)$

(b) 对于任意的i,j,我们考虑x_i,x_j两个数被比较到的概率，为$\frac{2}{j-i+1}$ 因

此，所有比较次数的期望为

$$
\begin{aligned}
E &= \sum_{i=1}^{n-1} \frac{2}{i+1} (n-i) \\
&= \frac{2}{2} (n-1) + \frac{2}{3} (n-2) \frac{2}{4} (n-3) + ... + \frac{2}{n} (n-(n-1)) \\
&= 2 \left(\sum_{i=1}^{n-1} \frac{n+1}{i+1} - n + 1 \right) \\
&= 2 (n+1) \sum_{i=1}^{n-1} \frac{1}{i+1} - 2n + 2 \\
&< 2 (n+1) \int_{1}^{n} \frac{1}{n} \\
&= 2 (n+1) \ln n \\
&= O (n \ln n)
\end{aligned}
$$

Q.E.d

Q3 首先不妨假设$n = 2^k$

我们考虑分治的做法，首先递归地将n分为等大的两部分，最终分到每组只有一个元素。然后在每次向上合并时，每次比较两组的值，记录下比较的结果，并将大的值向上传递，最终合并到最上层时即可得到最大的值。此时由于每层合并都要比较$\frac{n}{2}$次，所以找到最大值所需要的比较次数是n-1次。

接着，考虑找到第二大的值,我们根据之前找到的最大值，记为A，以及储存的比较结果，取最后一次和A比较的值，记为B，接着逐层向下将B和每个和A比较过的值进行比较，每次取大的结果作为新的B，最终得到第二大的值。由于每层比较都要比较一次，共需要比较$\log n - 1$次，因此总的比较次数为$n + \log n - 2$次

Q.E.D

Q4 (a) 考虑一个$n * 1$的矩阵，我们每次取其中第$\frac{n}{2}$个数据然后比较矩阵左右两端和这个数据的大小。

如果两个数中有一个数比它小，记为A，则我们取中间数到A，作为新的分治目标。同时考虑A的旁边值，记为A_1，如果$A_1 > A$，则已经找到局部最小值，否则，再去考虑A_2, A_3，重复以上过程，如果重复至中间值，且

可得序列中间值$> A > A_1 > A_2 ....$，因此可得最后的$A_{\frac{n-1}{2}}$即为局部最小值。因此可得，在新的分治目标中一定存在局部最小值。

如果两个数都比中间数要大，则先考虑中间值是是否为局部最小值，如果是，则找到，如果不是，则存在$B <$中间值,我们将中间值到B方向的端点作为新的分治目标，并同理A的过程，可得新的分治目标中也一定存在局部最小值。

因此综上，每一次的对半分支都可以确保新的分治目标中有局部最小值，且一定能够被找到，因此该算法正确。

又每次对半分治，该算法复杂度为$O(\log n)$

(b) 首先考虑最外侧两列，两行，中间行，中间列共$6n$个数据,找到其中的最小值，记为A，然后比较A的上下左右四个数，如果A是局部最小值，则找到，否则，如果A的上下左右四个数中有一个比A小，则取该数所在的$\frac{1}{4}$矩阵(包括边界)作为新的分治目标。

现在考虑这个新的矩阵中的最小值，如果该最小值在边界上，则与A以及边界的取法矛盾，如果在矩阵里面，则局部最小值在新的分治目标中。因此该算法可以确保新的分治目标中有局部最小值，且一定能够被找到，因此该算法正确。

考虑时间复杂度，第一次比较需要$6n$，接下去的每次比较的和$< 2n$，因此总的时间复杂度为$O(n)$。

(c) 若(b)中的情况在有任何一遍达到1时，改为(a)中的方法，不妨设$m < n$，则可得时间复杂度通解为$T = O(n \log m + \log n)$

Q5 (a) 如果x,y,z并不同奇同偶，不妨设x,y为奇数，z为偶数，则$y - x$为偶数，$z - x$为奇数，则x,y,z一定无法形成三元组，因此由逆否命题得，如果x,y,z形成三元组，则它们一定同奇同偶。

(b) $\{2, 1, 4, 3\}$,

$1 - 2 ! = 4 - 1, 1 - 2 ! = 3 - 1, 4 - 2 ! = 3 - 4, 4 - 1 ! = 3 - 4$

(c) 若x,y,z形成三元组，则$\frac{y+1}{2} - \frac{x+1}{2} = \frac{y-x}{2} = \frac{z-y}{2} = \frac{z+1}{2} - \frac{y+1}{2}$ 因此这三个数也构成三元组。若这三个数构成三元组, 则$\frac{y-x}{2} = \frac{y+1}{2} - \frac{x+1}{2} = \frac{z+1}{2} - \frac{y+1}{2} = \frac{z-y}{2}$ 因此x,y,z形成三元组。

Q.E.D

(d)对于$1 - n$，我们先将它们按照奇偶分治，对于同奇偶的一组，奇数我们就考虑它们(+1)/2的奇偶性，偶数我们就考虑它们/2的奇偶性并不断向下分治，直到每组只有一个数，然后再逐层向上合并，每次合并时，