Lab02-Divide and Conquer

CS214-Algorithm and Complexity, Xiaofeng Gao, Spring 2019.

∗ If there is any problem, please contact TA Jiahao Fan.

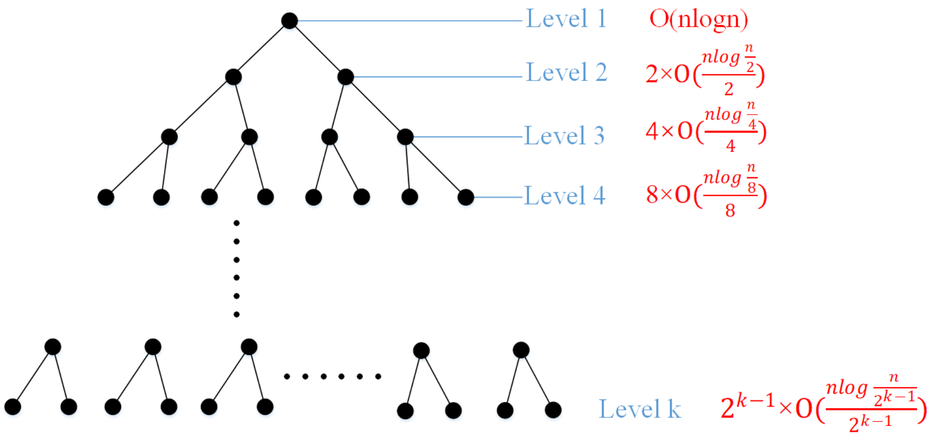
∗ Name: Yitian Ma   Student ID: 517021910467   Email: mkochab@sjtu.edu.cn

1. Consider the following recurrence:

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2T(n/2) + O(n \log n) & \text{if } n = 2^k \text{ and } k \geq 1 \end{cases}$$

(a) Solve $T(n)$ (in the form of O-notation) by recurrence tree. Detailed derivation is required.

Solution. Fig. 1 shows the recurrence tree.



Level 1
O(nlogn)
Level 2
$2\times O(\frac{nlog\frac{n}{2}}{2})$
Level 3
$4\times O(\frac{nlog\frac{n}{4}}{4})$
Level 4
$8\times O(\frac{nlog\frac{n}{8}}{8})$
Level k
$2^{k-1}\times O(\frac{nlog\frac{n}{2^{k-1}}}{2^{k-1}})$

图 1: Recurrence tree

According to the recurrence, the time complexity on level 1 is $O(nlogn)$. On level 2, the time complexity of each brance is $O\left(\frac{n \log \frac{n}{2}}{2}\right)$. As there are two branches, the total time complexity on this level is $2 \times O\left(\frac{n \log \frac{n}{2}}{2}\right)$. By that analogy, the total time complexity on level i is $2^{i-1} \times O\left(\frac{n \log \frac{n}{2^{i-1}}}{2^{i-1}}\right)$, $(i = 1, 2, 3, \ldots, k)$. In other words, the total complexity on level i is

$$\begin{aligned} 2^{i-1} \times O\left(\frac{n \log \frac{n}{2^{i-1}}}{2^{i-1}}\right) &= \left(\frac{2^{i-1}}{2^{i-1}}\right) \cdot O(n \log \frac{n}{2^{i-1}}) \\ &= O(n \log n - (i-1)n) \end{aligned}$$

When the branch reaches the base case on level k, then

$$\frac{n}{2^{k-1}} = 1 \Rightarrow k = \log n + 1$$

So there are altogether $k = \log n + 1$ levels.

The total work done is

$$\begin{aligned}T(n) &= O(n\log n) + O(n\log n - n) + O(n\log n - 2n)\\ &\quad + O(n\log n - 3n) + \cdots + O(n\log n - (k-1)n)\\ &= O(k\cdot n\log n - (1+2+3+\cdots+(k-1))n)\\ &= O((\log n + 1)n\log n - \frac{1}{2}(\log n + 1)n\log n)\\ &= O(\frac{1}{2}n\log^2 n + \frac{1}{2}n\log n)\end{aligned}$$

As a result, $T(n) = O(n\log^2 n)$.

□

(b) Can we use the Master Theorem to solve this recurrence? Please explain your answer.

Solution. We are not able to use the Master Theorem to solve this recurrence. For any $d > 1$, $\lim_{n\to\infty}\frac{n\log n}{n^d} = 0$. For any $0 < d \le 1$, $\lim_{n\to\infty}\frac{n\log n}{n^d} = \infty$. Therefore, $n\log n$ can not be writen in the form of n^d. We can not apply the Master Theorem. □

2. Given any array *num*, find the number of pairs (i, j) satisfying $i < j$ and $num[i] > 2\times num[j]$. For example, if $num = [1, 3, 2, 3, 1]$, the answer should be 2.

   (a) Design an algorithm to solve this problem using divide-and-conquer strategy and complete the implementation in the provided C/C++ source code. (The source code *Code-Pairs.cpp* is attached on the course webpage.)

   (b) Write a recurrence for the running time of your algorithm and solve it using the Master Theorem directly.

Solution. The algorithm divides the problem into halves each time. We count the number of pairs from the first half and the second half respectively. Then we try to find the number of pairs (i, j) with i in the first half and j in the second half. Or we can name these pais as **cross-middle** ones.

We apply the *MergeSort* to make the elements in halves in nondecreasing order, which makes the counting easier. The merging process takes $O(n)$ time. Also, we apply the idea of merging operation to count the number of **cross-middle** ones. The counting of **cross-middle** pairs takes $O(n)$ time as well.

The counting consists of the sum of the halves plus the counting of **cross-middle** pairs. The recurrence is thus $T(n) = 2T(n/2) + O(n)$. According to the Master Theorem, $T(n) = O(n\log n)$.

□

3. **Transposition Sorting Network**: A comparison network is a **transposition network** if each comparator connects adjacent lines, as in the network in Fig. 2.

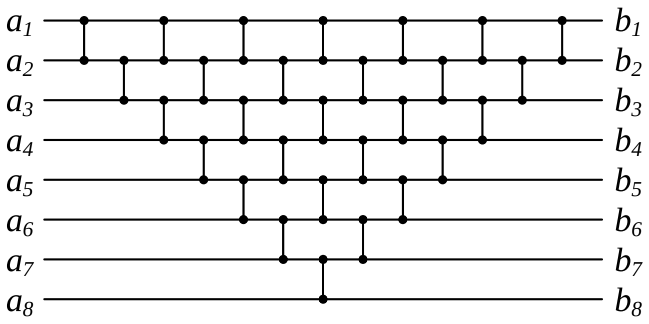


图 2: A Transposition Network Example

(a) Prove that a transposition network with n inputs is a sorting network if and only if it sorts the sequence $\langle n, n-1, \cdots, 1\rangle$. (Hint: Use an induction argument analogous to the *Domain Conversion Lemma*.)

Proof. If a transposition network with n inputs is a sorting network, it can surely transform the input sequence $\langle n, n-1, n-2, \cdots, 2, 1\rangle$ into output sequence $\langle 1, 2, 3, \cdots, n-1, n\rangle$.

Now we try to prove if a transposition network with n inputs can sort the input sequence $\langle n, n-1, n-2, \cdots, 2, 1\rangle$ correctly, it is a sorting network. Using *Domain Conversion Lemma*, obviously it can sort any decreasing sequences.

We represent arbitary sequence as $\langle a_1, a_2, a_3, \cdots, a_{n-1}, a_n\rangle$. We donate the number on line i after the k^{th} comparator by $a_{(k,i)}$ (for arbitary sequences) or $d_{(k,i)}$ (for decreasing sequences) respectively.

We assume that for any comparator and any i, j $(1 \le i < j \le n)$, if $d_{(k,i)} \le d_{(k,j)}$, then $a_{(k,i)} \le a_{(k,j)}$.

Basis step. Before any comparator, there is no $d_{(0,i)} \le d_{(0,j)}$, $(1 \le i < j \le n)$. So the assumption is always satisfied.

Induction. Assume that the claim holds for the k^{th} comparator. For the $(k+1)^{th}$ comparator, we try to prove with the condition $d_{(k+1,i)} \le d_{(k+1,j)}$ $(1 \le i < j \le n)$, $a_{(k+1,i)} \le a_{(k+1,j)}$ is true.

i. For any $i < j$, if the $(k+1)^{th}$ comparator involves both line i and line j, obviously, $a_{(k+1,i)} \le a_{(k+1,j)}$.

ii. If the $(k+1)^{th}$ comparator sorts neither line i nor line j, then we have $a_{(k+1,i)} = a_{(k,i)}$, $a_{(k+1,j)} = a_{(k,j)}$, and $d_{(k+1,i)} = d_{(k,i)} \le d_{(k+1,j)} = d_{(k,j)}$. Since $d_{(k,i)} \le d_{(k,j)}$, we have $a_{(k,i)} \le a_{(k,j)}$. That is, $a_{(k+1,i)} \le a_{(k+1,j)}$.

iii. If the $(k+1)^{th}$ comparator has line i as the upper line but doesn't involve line j, we have $a_{(k+1,i)} = \min\{a_{(k,i)}, a_{(k,i+1)}\}$, $a_{(k+1,j)} = a_{(k,j)}$, $d_{(k+1,i)} = \min\{d_{(k,i)}, d_{(k,i+1)}\}$ and $d_{(k+1,j)} = d_{(k,j)}$. As $d_{(k,j)} = d_{(k+1,j)} \ge d_{(k+1,i)} = \min\{d_{(k,i)}, d_{(k,i+1)}\}$, we get that $a_{(k,j)} \ge \min\{a_{(k,i)}, a_{(k,i+1)}\} = a_{(k+1,i)}$. In other words, $a_{(k+1,j)} \ge a_{(k+1,i)}$.

iv. If the $(k+1)^{th}$ comparator has line i as the lower line and doesn't involve line j, we have $a_{(k+1,i)} = \max\{a_{(k,i)}, a_{(k,i-1)}\}$, $a_{(k+1,j)} = a_{(k,j)}$, and $d_{(k+1,j)} = d_{(k,j)} \ge d_{(k+1,i)} = \max\{d_{(k,i)}, d_{(k,i-1)}\}$. With $d_{(k+1,j)} \ge \max\{d_{(k,i)}, d_{(k,i-1)}\}$, $a_{(k+1,j)} \ge \max\{a_{(k,i)}, a_{(k,i-1)}\}$ $= a_{(k+1,i)}$.

v. If the $(k+1)^{th}$ comparator has line j as the upper line and doesn't involve line i, we have $a_{(k+1,i)} = a_{(k,i)}$, $a_{(k+1,j)} = \min\{a_{(k,j)}, a_{(k,j+1)}\}$, $d_{(k+1,j)} = \min\{d_{(k,j)}, d_{(k,j+1)}\}$ and $d_{(k+1,i)} = d_{(k,i)}$. As we know $d_{(k,i)} = d_{(k+1,i)} \le d_{(k+1,j)} = \min\{d_{(k,j)}, d_{(k,j+1)}\}$, $a_{(k,i)} \le \min\{a_{(k,j)}, a_{(k,j+1)}\} = a_{(k+1,j)}$. Thus, $a_{(k+1,i)} = a_{(k,i)} \le a_{(k+1,j)}$.

vi. If the $(k+1)^{th}$ comparator has line j as the lower line but doesn't involve line i, we have $a_{(k+1,j)} = \max\{a_{(k,j)}, a_{(k,j-1)}\}$, $a_{(k+1,i)} = a_{(k,i)}$, and $d_{(k+1,i)} = d_{(k,i)} \leq d_{(k+1,j)} = \max\{d_{(k,j)}, d_{(k,j-1)}\}$. With $d_{(k,i)} \leq \max\{d_{(k,j)}, d_{(k,j-1)}\}$, we have $a_{(k+1,i)} = a_{(k,i)} \leq \max\{a_{(k,j)}, a_{(k,j-1)}\} = a_{(k+1,j)}$.

Therefore, since the transposition network can sort a decreasing sequence like $\langle n, n-1, n-2, \cdots, 2, 1\rangle$ into the increasing order after the last comparator, any arbitary sequence will follow the increasing order after the last comparator as well. $\square$

(b) (Bonus) Given any $n \in \mathbb{N}$, write a program using Tkinter in Python to draw a figure similar to Fig. 2 with n input wires.

See the source code in *Code-TranspositionSortingNetwork.py.*

Here are two examples.

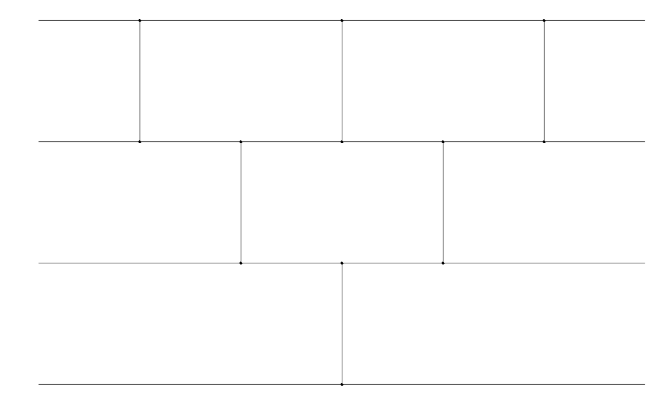


图 3: A Transposition Sorting Network Example with 4 inputs

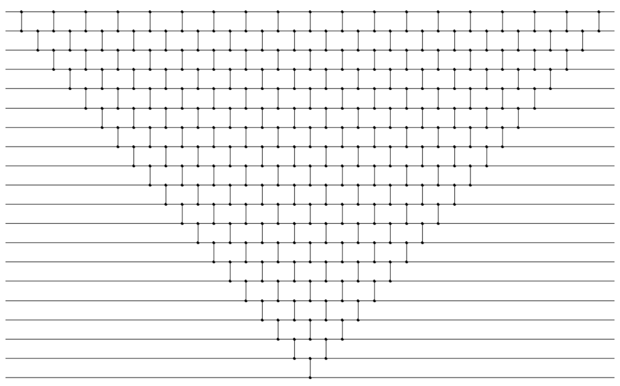


图 4: A Transposition Sorting Network Example with 20 inputs

Remark: include your .cpp, .py, .pdf and .tex files in your uploaded .rar or .zip file.