Lab02-Divide and Conquer

CS214-Algorithm and Complexity, Xiaofeng Gao, Spring 2021.

* If there is any problem, please contact TA Haolin Zhou.

* Name:BeichenYu   Student ID:519030910245   Email: polarisybc@sjtu.edu.cn

1. *Recurrence examples.* Give asymptotic upper and lower bounds for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for sufficiently small n. Make your bounds as tight as possible.

   (a) $T(n) = 4T(n/3) + n \log n$

   (b) $T(n) = 4T(n/2) + n^2\sqrt{n}$

   (c) $T(n) = T(n-1) + n$

   (d) $T(n) = 2T(\lfloor\sqrt{n}\rfloor) + \log n$

   Solution. First we have the master theorem:

$$\begin{aligned} T(n) &= aT(\lceil\frac{n}{b}\rceil) + O(n^d) \\ &= \begin{cases} O(n^d) & , b^d > a \\ O(n^d \log n) & , b^d = a \\ O(n^{\log_b a}) & , b^d < a \end{cases} \end{aligned}$$

   (a) $T(n) = 4T(n/3) + n \log n$
   As $T(n) \geqslant 0$ is always right, we know that $T(n) = \Omega(n \log n)$.
   Next we consider scaling from another direction. We know that $n \log n = O(n^{1+\epsilon})$, in which ϵ) is a really small number which is close to zero. So we can rewrite the recurrence into $T(n) = 4T(n/3) + O(n^{1+\epsilon})$. Then we can use the master theorem. Because $a = 4$, $b = 3$ and $d = 1 + \epsilon$, $b^d < a$, we know that $T(n) = O(n^{\log_3 4})$.
   So we conclude that $T(n) = \Omega(n \log n)$ and $T(n) = O(n^{\log_3 4})$.

   (b) $T(n) = 4T(n/2) + n^2\sqrt{n}$
   This time we can use the master theorem directly. $a = 4$, $b = 2$, and $d = \frac{5}{3}$, so $b^d > a$. According to the master theorem, we have $T(n) = O(n^{\frac{5}{2}})$. At the same time, we can make sure that $T(n) = \Omega(n^{\frac{5}{2}})$ as $T(n) \geqslant 0$, so $T(n) = \Theta(n^{\frac{5}{2}})$.

   (c) $T(n) = T(n-1) + n$
   We can use our knowledge of the Arithmetic sequence to solve it. Because that $T(n)$ is constant for sufficiently small n, we assume $T(1) = 1$. And $T(n) - T(n-1) = n$. So $T(n) = 1 + 2 + \cdots + n = \frac{1}{2}n(n+1)$, which means $T(n) = \Theta(n^2)$.

   (d) $T(n) = 2T(\lfloor\sqrt{n}\rfloor) + \log n$
   First let us do some calculations:

$$\begin{aligned} T(n) &\approx 2T(n^{\frac{1}{2}}) + \log n \\ &\approx 2(2T(n^{\frac{1}{4}}) + \log n^{\frac{1}{2}}) + \log n \\ &= 4T(n^{\frac{1}{4}}) + 2\log n \\ &\approx 8T(n^{\frac{1}{8}}) + 2\log n \\ &\approx \cdots \\ &\approx 2^k T(n^{\frac{1}{2^k}}) + k \log n \end{aligned}$$

Now we have to confirm the value of k.
When will $n^{\frac{1}{2^k}}$ be a constant? We let the expression be equal to 2:

$$\begin{aligned} n^{\frac{1}{2^k}} &= 2 \\ 2^{2^k} &= n \\ 2^k &= \log n \\ k &= \log\log n \end{aligned}$$

So when $T(n^{\frac{1}{2^k}}) = 1$, we have $k = \log\log n$. As $2^k = 2^{\log\log n} = \log n$, we know that $T(n) = \Theta(\log n \cdot \log\log n)$.

$\square$

2. *Divide-and-conquer.* Given an integer array $A[1..n]$ and two integers $lower \le upper$, design an algorithm using **divide-and-conquer** method to count the number of ranges (i, j) $(1 \le i \le j \le n)$ satisfying

$$lower \le \sum_{k=i}^{j} A[k] \le upper.$$

Example:

Given $A = [1, -1, 2]$, $lower = 1$, $upper = 2$, return 4.

The resulting four ranges are $(1, 1)$, $(3, 3)$, $(2, 3)$ and $(1, 3)$.

(a) Complete the implementation in the provided C/C++ source code.

(b) Write a recurrence for the running time of the algorithm and solve it by recurrence tree.

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

Solution. (a) I have finish the *Code-Range.cpp*, and my source code is included in the zip.

(b) The time complexity of the binary searching in the loop and the sorting are both $O(nlogn)$. Each time the question is divided into two parts, and doing the merging twice. So we can write down the recurrence:

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

And we can draw the recurrence tree as well.

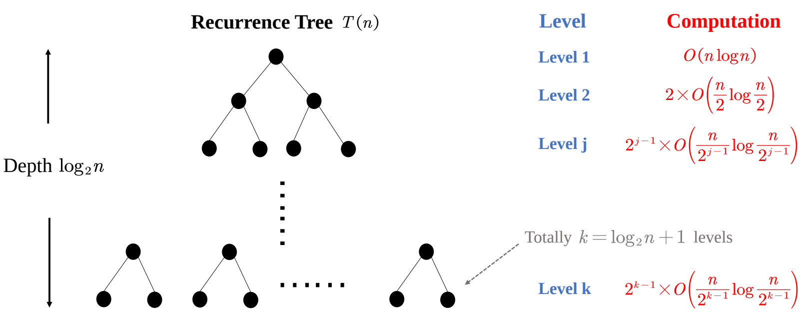

Figure 1: The recurrence tree

So we have:

$$T(n) = \sum_{j=1}^{1+\log_2 n} 2^{j-1} \times O(\frac{n}{2^{j-1}} \log \frac{n}{2^{j-1}}) = \sum_{j=0}^{\log_2 n} O(n \log \frac{n}{2^j})$$

$$= \sum_{j=0}^{\log_2 n} O(n \log n - jn) = n \log n(\log n + 1) - n\frac{\log n(\log n + 1)}{2} = \Theta(n \log^2 n)$$

(c) We can not use the Master Theorem to solve this problem. We know $n \log n = O(n^{1+\epsilon})$, in which ϵ is a small number which is close to zero. When comparing b^d with a, we will meet some problem. We know that $b = a = 2$, but it is wrong to say either $2^{1+\epsilon} > 2$ or $2^{1+\epsilon} = 2$. In fact the two expression cannot compare like this. That is to say, the Master Theorem has no effect in solving this problem.

□

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

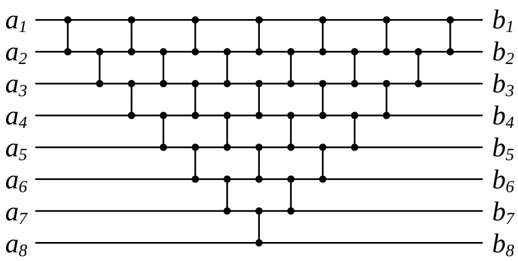

Figure 2: A Transposition Network Example

(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.*)

(b) (Optional Sub-question with 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.