Lab02-Divide and Conquer

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

∗ If there is any problem, please contact TA Haolin Zhou.
∗ Name:Renyang Guan Student ID:519021911058 Email: guanrenyang@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.

   (a) $T(n) = O(n^{\log_3 4})$
   Since $n \prec n \log n \prec n^{\log_3 4-\epsilon}$, where ϵ is any integer in $(0, \log_3 4 - 1)$, and the solutions of

   $$\begin{aligned} T(n) &= 4T(n/3) + O(n) \\ T(n) &= 4T(n/3) + O(n^{\log_3 4-\epsilon}) \end{aligned} \tag{1}$$

   are all $T(n) = O(n^{\log_3 4})$ according to the master theorem.
   Such that the solution of the original equation is $O(n^{\log_3 4})$.

   (b) $T(n) = O(n^2\sqrt{n})$
   According to the master theorem, $d = \frac{5}{2}, \log_b a = 2$ and $d > \log_b a$, so the solution $T(n) = O(n^2\sqrt{n})$ is obviously.

   (c) $T(n) = O(n^2)$
   Since $T(n)$ is constant for sufficiently small n, such that

   $$\begin{aligned} T(n) &= T(n-1) + n \\ &= T(1) - 1 + \sum_{i=1}^{n} i \\ &= \frac{n^2}{2} + \frac{n}{2} + T(1) - 1 \\ &= O(n^2) \end{aligned} \tag{2}$$

   (d) $T(n) = O(\log n \times \log(\log n))$
   Let $n = 2^k$, the equation is generated as

   $$T(2^k) = 2T(2^{\frac{k}{2}}) + O(k)$$

   Let $S(k) = T(2^k)$, the equation is generated as

   $$S(k) = 2S(\frac{k}{2}) + O(k)$$

   According to the master theorem, the solution of the equation is

   $$S(k) = O(k \log k)$$

   By putting in $k = \log n$, we could get the solution:

   $$T(n) = O(\log n \times \log \log n)$$

□

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

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

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 (The source code *Code-Range.cpp* is attached on the course webpage).

(b) Write a recurrence for the running time of the algorithm and solve it by recurrence tree (You can modify the figure sources *Fig-RecurrenceTree.vsdx* or *Fig-RecurrenceTree.pptx* to illustrate your derivation).

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

Solution.

(a) The code is shown in file *Code-Range.cpp*.

(b) We could get the recurrence easily from the source code:

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

The recurrence is shown below. For the convenience of analysis, assume that $n = 2^i$ to assure that the input size of each level is an integer.

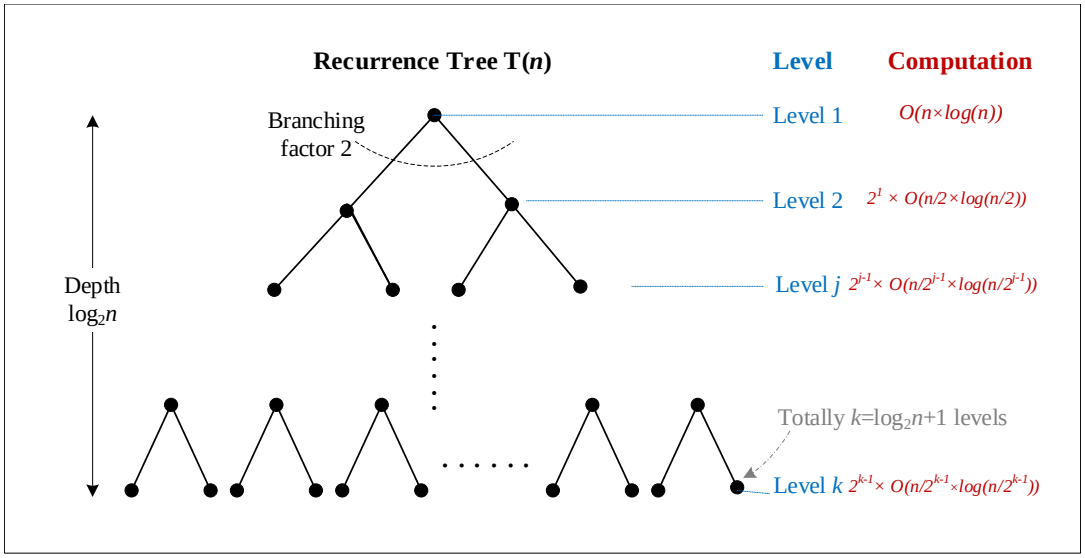

Figure 1: Recurrence Tree

The time complexity of the algorithm is the summation of each level's time complexity, which is

$$\begin{aligned} T(n) &= cn\log n + 2\times c\frac{n}{2}\log\frac{n}{2} + \cdots + 2^{k-1}\times c\frac{n}{2^{k-1}}\log\frac{n}{2^{k-1}} \\ &= cn\log(n\times\frac{n}{2}\times\frac{n}{2^2}\times\cdots\times\frac{n}{2^{k-1}}) \\ &= cn\log\frac{n^{\log_2 n+1}}{2^{\frac{(\log_2 n+1)(\log_2 n)}{2}}} \\ &= cn\log n^{\frac{\log_2 n+1}{2}} \\ &= cn\log n^{\frac{\log_2 n}{2}} + cn\log\sqrt{n} \\ &= \frac{c}{2\log 2}n\log^2 n + cn\log\sqrt{n} \\ &= O(n\log^2 n) \end{aligned} \tag{3}$$

Thus, the time complexity of the algorithm is $O(n\log^2 n)$.

(c) We *can not* use plain master theorem to solve this problem because the exact time complexity of the algorithm is not any format of the master theorem's possible results.

However, we could get the generalization of master theorem from the book *Induction to Algorithms*:
In the equation

$$T(n) = aT(n/b) + f(n)$$

satisfies

$$f(n) = \Theta(n^{\log_b a}\log^k n) \quad and \quad k \geq 0$$

the solution of the equation is

$$T(n) = \Theta(n^{\log_b a}\log^{k+1} n)$$

, which could be applied to the 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.

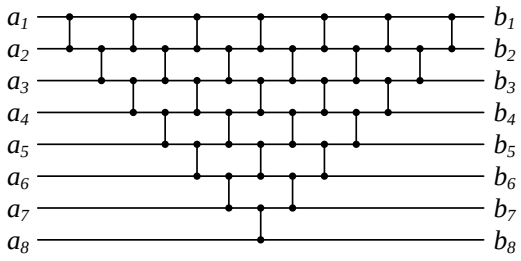

Figure 2: A Transposition Network Example

(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.

1. **Solution.**
Before the beginning of our proof of the problem, let we prove a lemma.

Lemma 1. *There is a transposition network with any input sequence $< a_1, a_2, \cdots, a_n >$, of which the maximum is denoted as max_a. If the comparators that make the max_a going down are all deleted, whichever wire the max_a is on initially, the rest of the network will remain the same.*
*(**Definition of the delete operation:**) As is shown in 3, if the input is $< x, y >$ where $x \leq y$, the comparator is deleted and the down-left node and the upper-right node is connected. At the same time, x is dropped and the output size is smaller than the input size by 1.)*

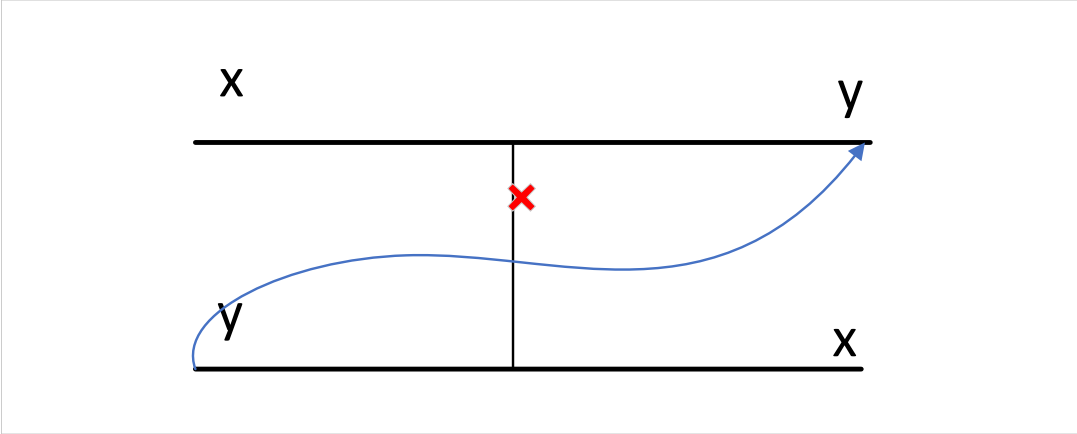

Figure 3: Delete Operation Example

Proof of the lemma 1.
Basis step: The $n = 2$ case is proved by the definition of the delete operation.
Induction hypothesis: Assume that for the input size of n, if the comparators that make the max_a going down are all deleted, whichever wire the max_a is on initially, the rest of the network will remain the same.
Proof of induction step: For the input size of $n + 1$, the rest of n wires above remains the same. The comparator between the n and $n + 1$ wires is the basis situation, so the lemma is proved yet.

□

Proof of the original problem:

Basis step: For $n = 2$, the transposition network composes only one comparator and it is obviously true.
Induction hypothesis: Assume that the transposition network which could sort the input sequence of $< n, n-1, \cdots, 1 >$ could sort any sequence.
Proof of the induction step: For the transposition network which could sort the input sequence of $< n+1, n, \cdots, 1 >$, if the wire the maximum of the input sequence $n + 1$ going down is deleted, the rest of the network could sort the input sequence of $< n, n-1, \cdots, 1 >$, which could sort any sequence according to the induction hypothesis, as is shown in the Fig. 4.

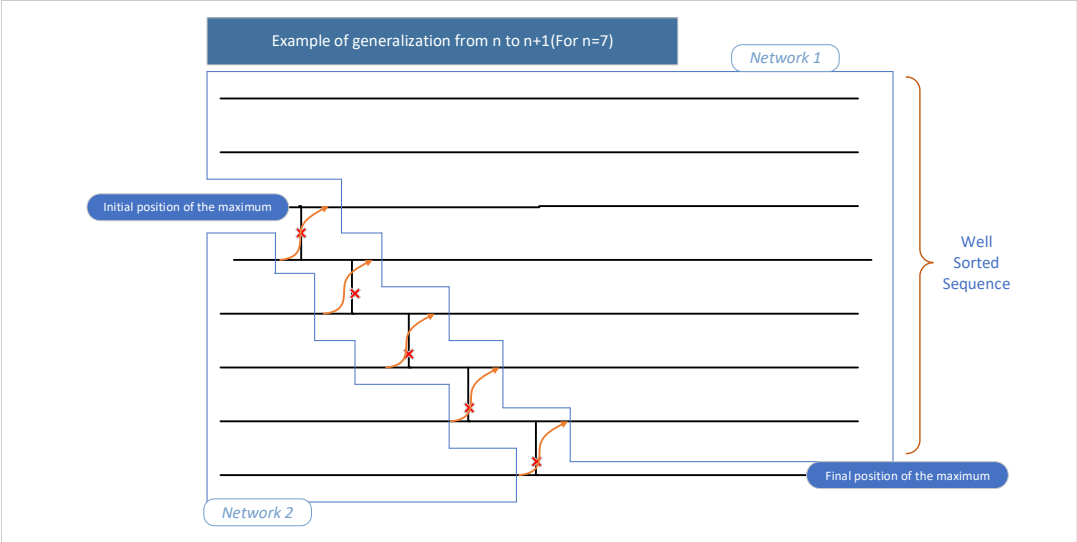

Figure 4: Example of generalization from n to n+1(For n=7)

(a) $n = 8$ (b) $n = 9$

Figure 5: Example of the transposition network.

Because the network could sort the sequence $< n+1, n, \cdots, 1 >$, there must be a comparator between every adjacent lines and the maximum of the input sequence will go down to the bottom line along these comparators.

According to the *Lemma 1*, if we delete the path the maximum of the input sequence going down, the rest of the network will remain the same, which is n case in the induction hypothesis and can sort the other numbers of the input except for the maximum. At the same time, the maximum is on the bottom line eventually, so the network could sort any input sequence with the size of $n + 1$. □

2. The source code is in file *Code.py* and you must execute it with **python3**. The content of the code is shown in Fig. Fig. 5 shows the output of the program when $n = 8$ and $n = 9$. However, you could input any integer as you like.

```
import sys
from tkinter import *
class MyCanvas(Canvas):
    def __init__(self, master, hLineWidth=1, vLineWidth=1, radius=2, **kwargs):
        Canvas.__init__(self, master, kwargs)
        self.hLineWidth = hLineWidth
        self.vLineWidth = vLineWidth
        self.radius = radius

    def create_segment_h(self, x, y, l):
        self.create_line(x, y, x + l, y, width=self.hLineWidth)
        self.create_oval(x - self.radius, y - self.radius, x + self.radius, y + self.radius, fill='black')
        self.create_oval(x + l - self.radius, y - self.radius, x + l - self.radius, y + self.radius, fill='black')

    def create_segment_v(self, x, y, l):#
        self.create_line(x, y, x, y + l, width=self.vLineWidth)
        self.create_oval(x - self.radius, y - self.radius, x + self.radius, y + self.radius, fill='black')
        self.create_oval(x - self.radius, y + l - self.radius, x + self.radius, y + l + self.radius, fill='black')

    def create_line_h(self, x, y, l):#
        self.create_line(x, y, x + l, y, width=self.hLineWidth)

    def create_line_v(self, x, y, l):
        self.create_line(x, y, x, y + l, width=self.vLineWidth)

if __name__ == '__main__':
    n = int(input('please input the number n: '))
    winW, winH = 2400 * 0.4, 1500 * 0.4
    hMargin, vMargin = winW // 20, winH // 20
    hScale, vScale = (winW - 2 * hMargin) // (2*n-2), (winH - 2 * vMargin) // (n - 1)
    root = Tk()
    root.title('A Typical Transposition Network with n=%d (Drawn by Python Tkinter)' % n)
    cvs = MyCanvas(root, bg='white', width=winW, height=winH)
    for i in range(n):
        cvs.create_line_h(hMargin,vMargin+i*vScale,(2*n-2)*hScale)
    for i in range(n-1):
        for j in range(i//2+1):
            cvs.create_segment_v(hMargin+(i+1)*hScale,vMargin+i*vScale-2*j*vScale,vScale)
            if i!=n-2:
                cvs.create_segment_v(winW-hMargin-(i+1)*hScale,vMargin+i*vScale-2*j*vScale,vScale)
    cvs.pack()
    root.mainloop()
```

Figure 6: Source code