Lab02-Divide and Conquer

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

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

∗ Name:Yanjie Ze Student ID:519021910706 Email: zeyanjie@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)

$$a = 4, b = 3, log_b a = log_3 4$$

Since $n < nlogn < n^2$, and based on Master Theorem, we have:

$$T(n) = O(n^2), T(n) = \Omega(n^{log_3 4})$$

(b)

$$T(n) = 4T(n/2) + n^2\sqrt{n} = 4T(n/2) + n^{\frac{5}{2}}$$

Based on Master Theorem, $a = 4, b = 2, log_b a = 2, d = \frac{5}{2}$,we have:

$$T(n) = \Theta(n^{\frac{5}{2}})$$

(c)

$$T(n) = \sum_{i=1}^{n} i + T(0) = \frac{n^2+n}{2} + T(0)$$

Thus:

$$T(n) = \Theta(n^2)$$

(d) Assume $n = 2^{2^k}$, then $k = log_2(log_2 n)$.

$$T(2^{2^k}) = 2T(2^{2^{k-1}}) + 2^k log2$$

$$\frac{T(2^{2^k})}{2^k} = \frac{T(2^{2^{k-1}})}{2^{k-1}} + log2$$

Thus we can sum them together and get:

$$\frac{T(2^{2^k})}{2^k} = (k-1)log2 + T(2)$$

Multiply both sides by 2^k and then replace k by n:

$$T(n) = log2 \cdot log_2 n \cdot log_2(log_2 n) + T(2) \cdot log_2 n$$

Therefore:

$$T(n) = \Theta(logn \cdot log(logn))$$

□

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 (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).

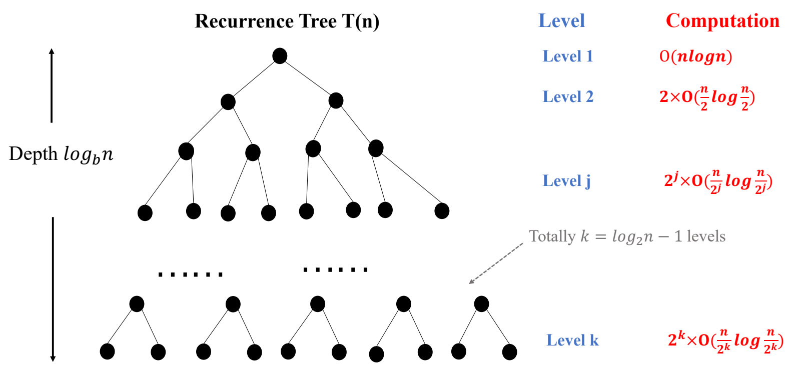
(c) Can we use the Master Theorem to solve the recurrence above? Please explain your answer.

Solution. (a) Code is in **Code-Range.cpp**.

(b) The recurrence is:

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

And the recurrence tree is:



Recurrence Tree T(n)
Level
Computation
Depth $log_b n$
Level 1
$O(nlogn)$
Level 2
$2\times O(\frac{n}{2} log \frac{n}{2})$
Level j
$2^j\times O(\frac{n}{2^j} log \frac{n}{2^j})$
Totally $k = log_2 n - 1$ levels
Level k
$2^k\times O(\frac{n}{2^k} log \frac{n}{2^k})$

Figure 1: A Recurrence Tree

Then we calculate the sum:

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

After the simplification we get:

$$T(n) = n \sum_{i=1}^{log_2 n-1} (O(logn) - ilog2)$$

Thus:

$$T(n) = n\left((log_2 n - 1)O(logn) - \frac{logn(log_2 n - 1)}{2}\right)$$

Which means:

$$T(n) = O(n\log^2 n)$$

(c) We denote:

$$a = 2, b = 2, \log_b a = 1$$

Since $nlogn = O(n^2)$, we can have the bound based on the master theorem:

$$T(n) = O(n^2)$$

Which is not tight.
However, from Wikipedia and CLRS we can know that there is a more advanced version of master theorem, which can explain this recurrence.

$$If\ f(n) = \Theta(n^{log_b a} log^k n), k \geq 0, then\ T(n) = \Theta(n^{log_b a} log^{k+1} n).$$

Thus:

$$T(n) = \Theta(n^{\log_b a} log^2 n) = \Theta(n \log^2 n)$$

$\square$

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

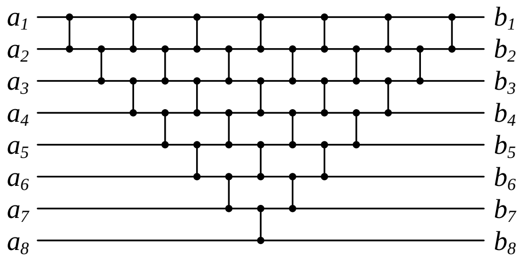


Figure 2: A Transposition Network Example

(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.

Solution. (a)
After I think through this problem, I try my best to figure out a proof, which is somewhat not rigorous. And after I discuss the problem with my classmates I also get a solution which is rigorous. Anyway, I consider my own solution is directer, and another solution is rigorous but not that direct. Finally I decide to just keep my original solution, and the rigorous one is referred in the appendix 8.

First, it is obvious that if a transposition network is a sorting network, it can sort the sequence $\langle n, n-1, \cdots, 1\rangle$.
Second, we will prove if a transposition network sorts the sequence $\langle n, n-1, \cdots, 1\rangle$, it is a sorting network.
Proof:

Basic step:

When n=3, the network can sort $\langle 3, 2, 1\rangle$. Since the biggest number is on the top, to transfer it to the bottom, we need at least 2 comparators. After transferring the top one, the rest of the numbers need one more comparator so that the smaller one can be transferred to the top. Therefore, we finally get the network which can sort this sequence in Fig. 3.

Though the more comparators can be added to this network without influencing the sort ability, the architecture in Fig. 3 is the simplest one, and more comparators are meaningless since they will do no exchange when comparing.

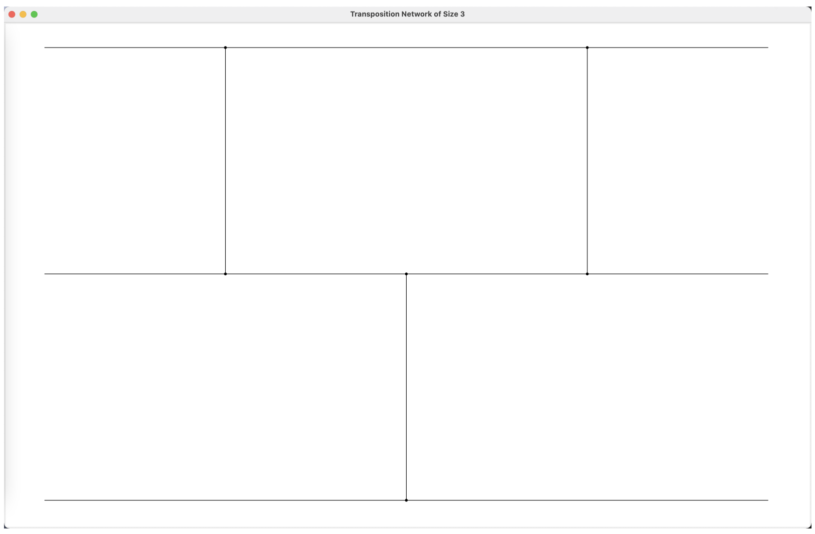


Figure 3: A Transposition Network of size 3

Now the problem is: Is this architecture a sorting network?
The answer is YES.
This network essentially can be seen as a **bubble sort**. Two adjacent elements are compared by a comparator and the smaller one is on top then, which is one bubble. Therefore, this network works the same as **bubble sort**, which proves it can function as a sorting network.
Therefore, we finish the basic step of proof, and here is the conclusion:
The network can sort $\langle 3, 2, 1\rangle \Rightarrow$ **The network's simplest architecture has the shape in Fig.** $3 \Rightarrow$ **The network is a sorting network.**

Induction Hypothesis:

Assume for $n \leq k$, if the network can sort $\langle k, k-1, ..., 1\rangle$, then the network's simplest architecture has the shape in Fig. 4.

(Because the network for $n = k$ is hard to draw, we use $n = 6$ for $n = k$ and $n = 7$ for $n = k + 1$.)

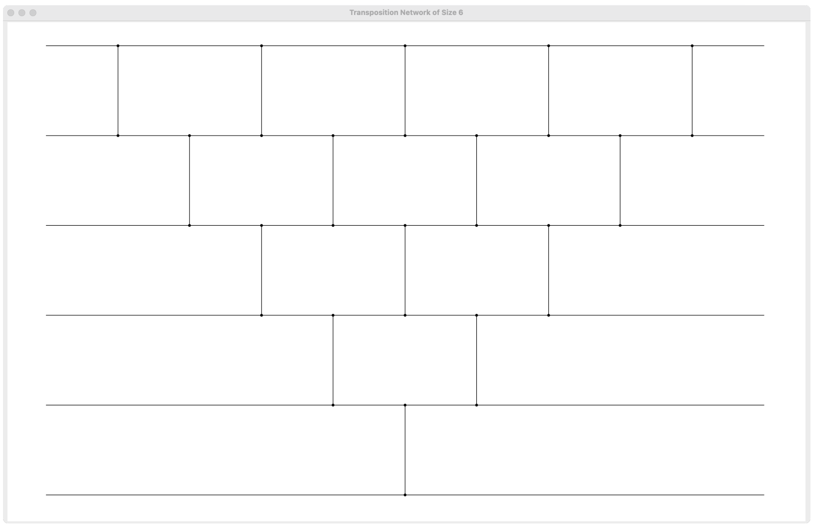


Figure 4: A Transposition Network of size 6

Proof of Induction Step: We prove: for $n = k+1$, if the network can sort $\langle k+1, k, ..., 1\rangle$, then the network's simplest architecture has the shape in Fig. 5.

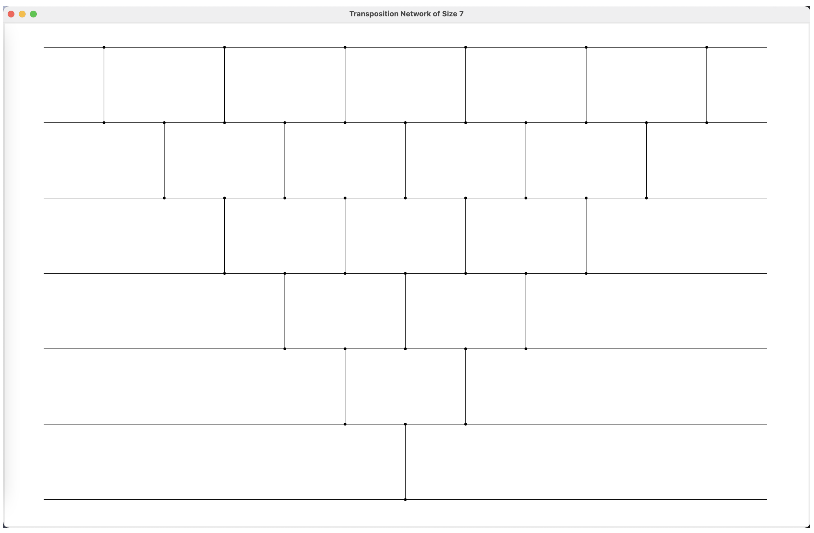


Figure 5: A Transposition Network of size 7

When $n \leq k$, the network has been a sorting network, which means for the input $\langle k+1, k, ..., 2, 1\rangle$, $\langle k+1, k, ..., 2\rangle$ can be sorted correctly.

Then for the element 1 at the bottom, to make it sorted correctly, the element should be transferred to the top, which means this element at least goes through k times of bubble, or k comparators.

If the comparators' number is less than k, then the element can not be transferred to the top.

If the comparators' number is more than k, then the element can be transferred to the top, but the extra comparators are useless.

Therefore, to make the bottom element sorted correctly, the network's simplest architecture has the shape in Fig. 5.

Finally, we prove that for $n = k + 1$, if the network can sort $\langle k+1, k, ..., 1\rangle$, then the network's simplest architecture has the shape in Fig. 5.

And this means if the network can sort $\langle k+1, k, ..., 1\rangle$, then the network is a sorting network, which is proved before.

(b) Code is in **sorting_network.py**.

Fig. 4, Fig. 5, Fig. 6, Fig. 7 are all drawn by the program.

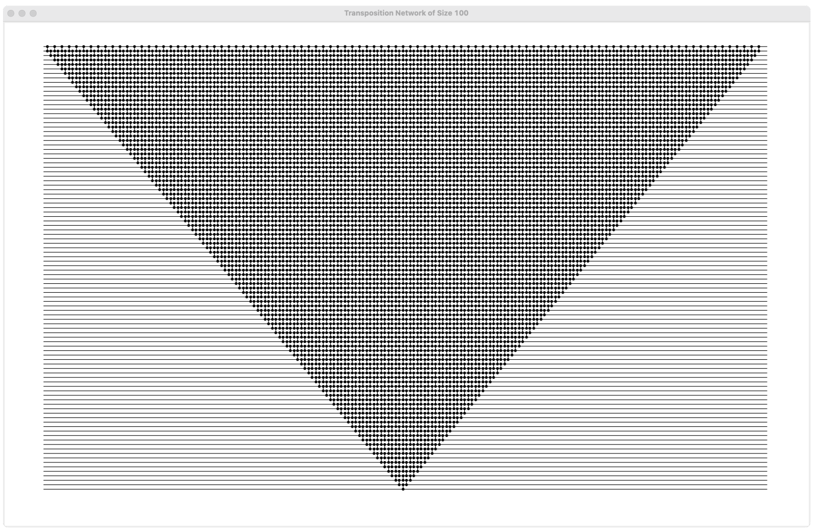


Figure 6: A Transposition Network of size 100

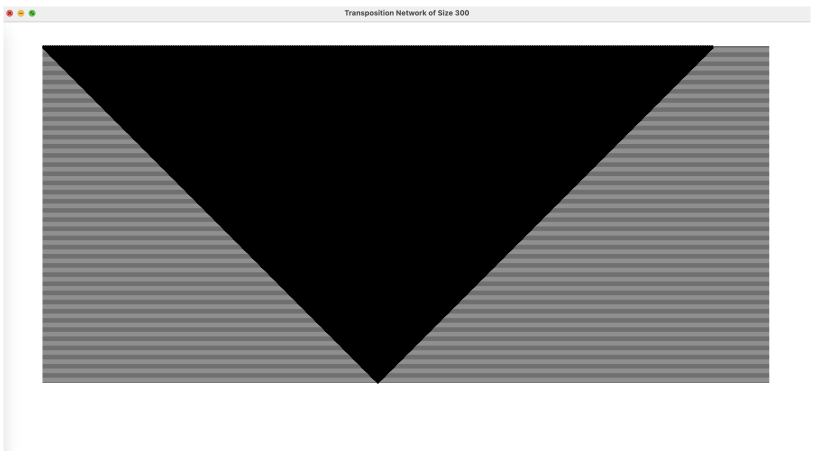
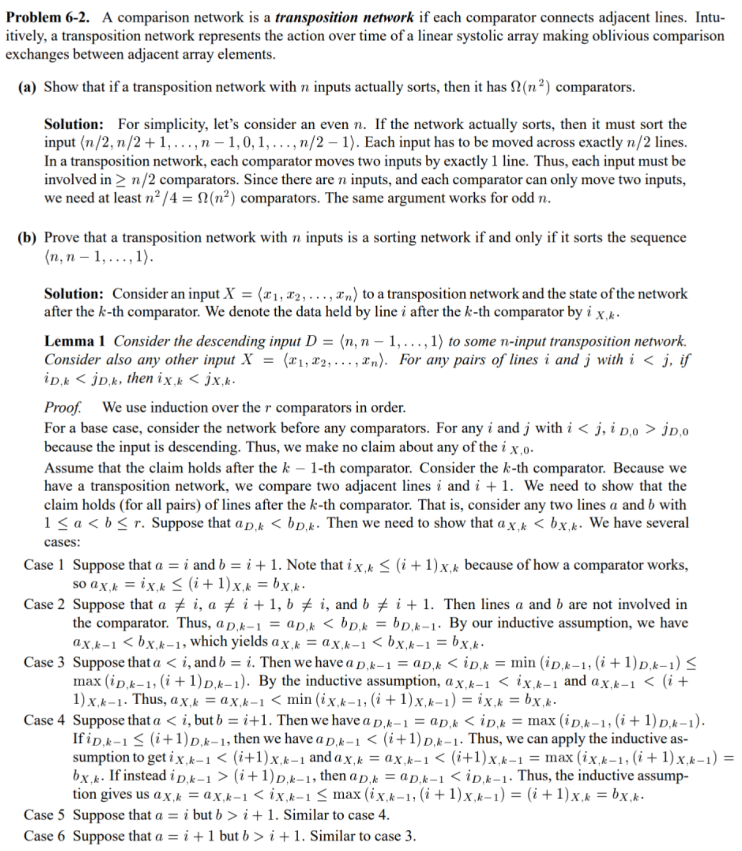


Figure 7: A Transposition Network of size 300

□

Appendix



Problem 6-2. A comparison network is a ***transposition network*** if each comparator connects adjacent lines. Intuitively, a transposition network represents the action over time of a linear systolic array making oblivious comparison exchanges between adjacent array elements.

(a) Show that if a transposition network with n inputs actually sorts, then it has $\Omega(n^2)$ comparators.

Solution: For simplicity, let's consider an even n. If the network actually sorts, then it must sort the input $\langle n/2, n/2+1, \ldots, n-1, 0, 1, \ldots, n/2-1\rangle$. Each input has to be moved across exactly $n/2$ lines. In a transposition network, each comparator moves two inputs by exactly 1 line. Thus, each input must be involved in $\geq n/2$ comparators. Since there are n inputs, and each comparator can only move two inputs, we need at least $n^2/4 = \Omega(n^2)$ comparators. The same argument works for odd n.

(b) Prove that a transposition network with n inputs is a sorting network if and only if it sorts the sequence $\langle n, n-1, \ldots, 1\rangle$.

Solution: Consider an input $X = \langle x_1, x_2, \ldots, x_n\rangle$ to a transposition network and the state of the network after the k-th comparator. We denote the data held by line i after the k-th comparator by $i_{X,k}$.

Lemma 1 *Consider the descending input $D = \langle n, n-1, \ldots, 1\rangle$ to some n-input transposition network. Consider also any other input $X = \langle x_1, x_2, \ldots, x_n\rangle$. For any pairs of lines i and j with $i < j$, if $i_{D,k} < j_{D,k}$, then $i_{X,k} < j_{X,k}$.*

Proof. We use induction over the r comparators in order.
For a base case, consider the network before any comparators. For any i and j with $i < j$, $i_{D,0} > j_{D,0}$ because the input is descending. Thus, we make no claim about any of the $i_{X,0}$.
Assume that the claim holds after the $k-1$-th comparator. Consider the k-th comparator. Because we have a transposition network, we compare two adjacent lines i and $i+1$. We need to show that the claim holds (for all pairs) of lines after the k-th comparator. That is, consider any two lines a and b with $1 \leq a < b \leq r$. Suppose that $a_{D,k} < b_{D,k}$. Then we need to show that $a_{X,k} < b_{X,k}$. We have several cases:

Case 1 Suppose that $a = i$ and $b = i+1$. Note that $i_{X,k} \leq (i+1)_{X,k}$ because of how a comparator works, so $a_{X,k} = i_{X,k} \leq (i+1)_{X,k} = b_{X,k}$.

Case 2 Suppose that $a \neq i$, $a \neq i+1$, $b \neq i$, and $b \neq i+1$. Then lines a and b are not involved in the comparator. Thus, $a_{D,k-1} = a_{D,k} < b_{D,k} = b_{D,k-1}$. By our inductive assumption, we have $a_{X,k-1} < b_{X,k-1}$, which yields $a_{X,k} = a_{X,k-1} < b_{X,k-1} = b_{X,k}$.

Case 3 Suppose that $a < i$, and $b = i$. Then we have $a_{D,k-1} = a_{D,k} < i_{D,k} = \min(i_{D,k-1}, (i+1)_{D,k-1}) \leq \max(i_{D,k-1}, (i+1)_{D,k-1})$. By the inductive assumption, $a_{X,k-1} < i_{X,k-1}$ and $a_{X,k-1} < (i+1)_{X,k-1}$. Thus, $a_{X,k} = a_{X,k-1} < \min(i_{X,k-1}, (i+1)_{X,k-1}) = i_{X,k} = b_{X,k}$.

Case 4 Suppose that $a < i$, but $b = i+1$. Then we have $a_{D,k-1} = a_{D,k} < i_{D,k} = \max(i_{D,k-1}, (i+1)_{D,k-1})$. If $i_{D,k-1} \leq (i+1)_{D,k-1}$, then we have $a_{D,k-1} < (i+1)_{D,k-1}$. Thus, we can apply the inductive assumption to get $i_{X,k-1} < (i+1)_{X,k-1}$ and $a_{X,k} = a_{X,k-1} < (i+1)_{X,k-1} = \max(i_{X,k-1}, (i+1)_{X,k-1}) = b_{X,k}$. If instead $i_{D,k-1} > (i+1)_{D,k-1}$, then $a_{D,k} = a_{D,k-1} < i_{D,k-1}$. Thus, the inductive assumption gives us $a_{X,k} = a_{X,k-1} < i_{X,k-1} \leq \max(i_{X,k-1}, (i+1)_{X,k-1}) = (i+1)_{X,k} = b_{X,k}$.

Case 5 Suppose that $a = i$ but $b > i+1$. Similar to case 4.

Case 6 Suppose that $a = i+1$ but $b > i+1$. Similar to case 3.

Figure 8: The Rigorous Proof for Problem 3