Lab02-Divide and Conquer

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

∗ If there is any problem, please contact TA Jiahao Fan.
∗ Name:Lynn Xiao   Student ID:________   Email: ____________

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.
   Consider the levels of recurrence tree:
   Level 1:O(nlogn)
   Level 2:2×$O(\frac{n}{2}log\frac{n}{2})$
   So we can get at the i-th level, the computation cost is $= O(\frac{n}{2^{i-1}}log\frac{n}{2^{i-1}}) = O(n(logn - i + 1))$.
   Notice that $(k+1) - th$ level are all leaf nodes, so we have $2^k = n$ and the depth of the recurrence tree is $logn$.
   So the total computation cost is $\sum_{i=1}^{k+1} n(logn - i + 1) = \sum_{i=0}^{k} n(logn - i) = \sum_{i=0}^{logn} n(logn - i) = O(n(logn)^2)$.

   □

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

   Solution.
   The recurrence relation is $2T(\frac{n}{2}) + O(nlogn)$. We notice that the merging cost is $O(nlogn)$ does not match the form of Master Theorem.
   Master Theorem has a promotion form:
   Assume the recurrence relationship is $T(n) = aT(\frac{n}{b}) + f(n)$, then
   (i) If $f(n) = O(n^{log_b a - \epsilon})$, $\epsilon > 0$, then $T(n) = \Theta(n^{log_b a}))$.
   (ii) If $f(n) = \Theta(n^{log_b a})$, then $T(n) = \Theta(n^{log_b a} logn)$
   (iii)If $f(n) = \Omega(n^{log_b a + \epsilon})$ and $\epsilon > 0$, and for a constant c and n which can be sufficiently large, we have $af(\frac{n}{b}) \leq cf(n)$, then $T(n) = \Theta(f(n))$.
   In our case, $nlog_b a = n^1$ and $f(n) = nlogn$, it is clear that it does not satisfy the second case in our theorem. And we also can not find a $\epsilon > 0$ so that $nlogn = O(n^{1-\epsilon})$ or $nlogn = \Omega(n^{1-\epsilon})$, so we can not use the Master Theorem to solve this recurrence.

   □

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.
(a)
We can easily come up with an algorithm of exhaustive search traversing all C_n^2 pairs to check if each satisfies the requirements.

A possible algorithm of exhaustive searching is shown in Alg.1.

Algorithm 1: ExhaustiveSearch

```
Input: An array $A[1, \cdots, n]$
Output: Number of inversing pairs in A

1 $i \leftarrow 1$;
2 $N \leftarrow 0$;
3 for $i \leftarrow 1$ to $n-1$ do
4     $max \leftarrow A[i]; pos \leftarrow i$;
5     for $j \leftarrow i+1$ to $n$ do
6         if $A[i] > 2 \times A[j]$ then
7             $N \leftarrow N+1$;
8     return N;
```

It is clear that the time complexity of this algorithm is $O(n^2)$.

To reduce the time complexity, we can come up with an algorithm with divide-and-conquer design. We divide the array into two parts. We first calculate the inversion pairs within each half, and then consider the inversion pairs between two parts. The algorithm is shown in Alg.2.

Algorithm 2: Divide-And-Conquer1

```
Input: An array $A[1, \cdots, n]$, starting point L, ending point R
Output: Number of inversing pairs in A

1 if $L = R$ then
2     return 0
3 $M \leftarrow \frac{L+R}{2}$;
4 $N \leftarrow$Divide-And-Conque1(arrwy,L,M)+Divide-And-Conque1(array,M,R);
5 for $i \leftarrow 1$ to $M$ do
6     for $j \leftarrow M+1$ to $R$ do
7         if $A[i] > 2 \times A[j]$ then
8             $N \leftarrow N+1$;
9     return N;
```

However, according to the Master Theorem, the time complexity of Alg.2 is still $O(n^2)$. Because the merging part is still exhaustive search, which traverse all pairs between the two parts.

We can derive a new algorithm as is shown in Alg.3.

Algorithm 3: Divide-And-Conquer2

```
Input: An array $A[1, \cdots, n]$, starting point L, ending point R
Output: Number of inversing pairs in A

if $L = R$ then
    return 0
$M \leftarrow \frac{L+R}{2}$;
$N \leftarrow$Divide-And-Conque1(arrwy,L,M)+Divide-And-Conque1(array,M,R);
for $j \leftarrow M + 1$ to $R$ do
    while $A[i] \leq 2 \times A[j]$ and $i \leq M$ do
        $i \leftarrow i + 1$
    $N \leftarrow N + (M - i + 1)$
return N;
```

The time complexity of Alg.3 is $O(n)$ according to the Master Algorithm.

(b)

The following recurrence relation can be derived from the above pseudo code:

$$T(n) = \begin{cases} 0 & n = 1 \\ 2T(\frac{n}{2}) + O(n) & \text{otherwise} \end{cases}$$

According to Main Theorem, the complexity of this algorithm is $O(nlogn)$. □

3. **Transposition Sorting Network**: A comparison network is a **transposition network** if each comparator connects adjacent lines, as in the network in Fig. 1.

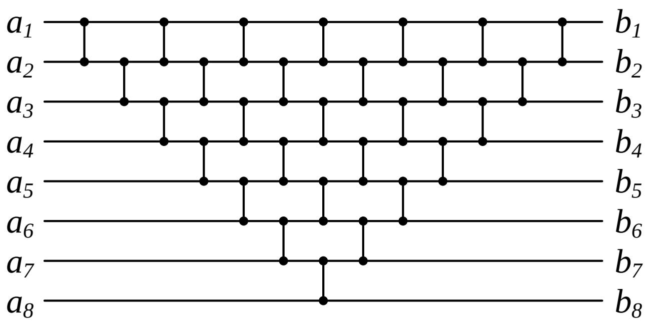

图 1: A Transposition Network Example

(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. The descending sequence has the largest number of inversion pairs, $Num_i = C_n^2$. And each comparator can cut down Num_i by one. So according to the Fig 1, transposition network with n lines has C_n^2 comparators, which can make Num_i be zero. So a transposition network with n inputs is a sorting network if and only if it sorts the sequence $\langle n, n-1, ..., 1\rangle$. Otherwise, the number of inversion pairs of the sequence can not be zero so that the output will not be in order. □

(b) (Bonus) Given any $n \in \mathbb{N}$, write a program using Tkinter in Python to draw a figure similar to Fig. 1 with n input wires.

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