Lab02-Divide and Conquer

CS214-Algorithm and Complexity, Xiaofeng Gao, Spring 2018.

∗ Name: Hongyi Guo Student ID: 516030910306 Email: guohongyi@sjtu.edu.cn

1. Assume that all elements of $A[1,\ldots,n]$ are distinct and sorted, and x is in A. Each element of A is equally likely to be in any position in the array. Please give the average case analysis of Algorithm BinarySearch ($n = 2^k$, $k \in \mathbb{N}$). The exact expression for the average number of comparisons $T(n)$ is required.

   Solution. Analyze.

   (a) The probability that we only need to compare **One** time is $\frac{1}{n}$. It's when x is the middle element of the sequence A.

   $$(\cdots \underline{x_1} | \cdots)$$

   (b) The probability that we need to compare **Two** times is $\frac{2}{n}$. It's when x is the middle element of the right or left half of A.

   $$(\cdots \underline{x_2} | \cdots x_1 | \cdots \underline{x_2} | \cdots)$$

   (c) The probability that we need to compare **Three** times is $\frac{4}{n}$. It's when x is in the position shown below.

   $$(\cdots \underline{x_3} | \cdots x_2 | \cdots \underline{x_3} | \cdots x_1 | \cdots \underline{x_3} | \cdots x_2 | \cdots \underline{x_3} | \cdots)$$

   (d) $\cdots$

   So, the probability that we need to compare **i** times is $\frac{2^{i-1}}{n}$, $i \le \log n = k$

   Because the first x_1 actually belongs to the left half of the sequence A, there will be an additional element A_n in the last of the right half. Thus, there is a probability of $1/n$ that we need to compare $k+1$ times.

   The exact expression for the average number of comparisons $T(n)$ is:

   $$T(n) = \sum_{i=1}^{k} i \times \frac{2^{i-1}}{n} + \frac{k+1}{n} = 1 \times \frac{1}{n} + 2 \times \frac{2}{n} + \cdots + k \times \frac{2^{k-1}}{n} + \frac{k+1}{n}$$

   .

   $$2 \times T(n) = 1 \times \frac{2}{n} + 2 \times \frac{4}{n} + \cdots + k \times \frac{2^k}{n} + \frac{2(k+1)}{n}$$

   $$T(n) = 2 \times T(n) - T(n) = -1 \times \frac{1}{n} - 1 \times \frac{2}{n} - 1 \times \frac{4}{n} - \cdots - 1 \times \frac{2^{k-1}}{n} + k \times \frac{2^k}{n} + \frac{k+1}{n}$$

   $$= \frac{1-2^k}{n} + k \times \frac{2^k}{n} + \frac{k+1}{n} = \frac{(k-1) * 2^k + 1}{n} + \frac{k+1}{n} = \log n - 1 + \frac{\log n + 2}{n}$$

   □

2. 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 source *Fig-RecurrenceTree.vsdx* to illustrate your derivation).

Solution.

$$T(n) = 2T(n/2) + O(n \log n)$$

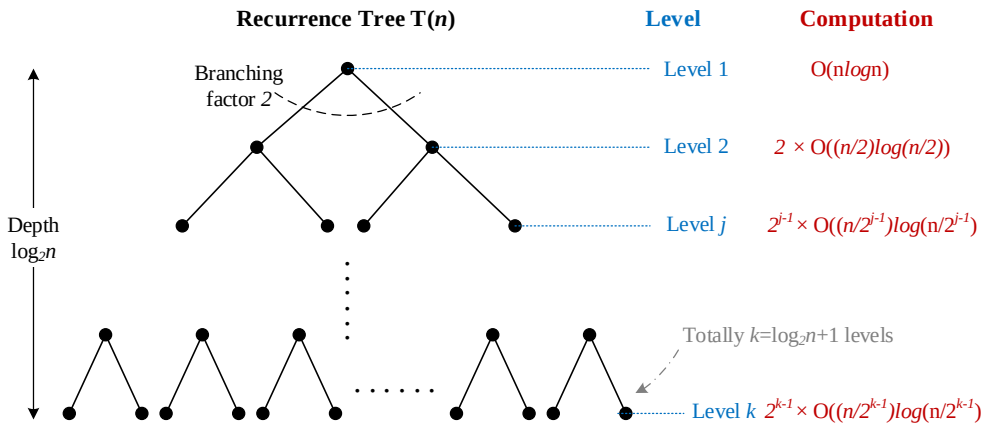

Figure 1: recurrence tree

Complexity of $T(n)$=Sum up all computations at each level.
The total work done at the j-th level is

$$2^{j-1} \times O\left(\frac{n}{2^{j-1}} \log \frac{n}{2^{j-1}}\right)$$

The total work done is

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

$$= (\log n + 1) \times n \log n - n \times \frac{(\log n + 1) \log n}{2} = \frac{n(\log n + 1) \log n}{2} = O(n(\log n)^2)$$

So, $T(n) = O(n(\log n)^2)$. □

(c) (Optional Sub-question with Bonus) Can we use the Master Theorem to solve the recurrence above? Please explain your answer.

Solution. No, we cannot use the Master Theorem to solve the recurrence above.
We can rewrite $T(n)$ as following

$$T(n) = 2T(n/2) + O(n \log n) = 2(T/2) + O(n * n^{\log_n(\log n)}) = 2(T/2) + O(n^{\log_n(\log n)+1})$$

Then we use Master Theorem.$a = 2$, $b = 2$, $d = \log_n(\log n) + 1 > 1$.
As we can see, $a < b^d$. So, $T(n) = O(n^d) = O(n^{\log_n(\log n)+1}) = O(n \log n) \neq O(n(\log n)^2)$.
The answer we get via Master Theorem is wrong. Thus, we cannot use Master Theorem in this case. □

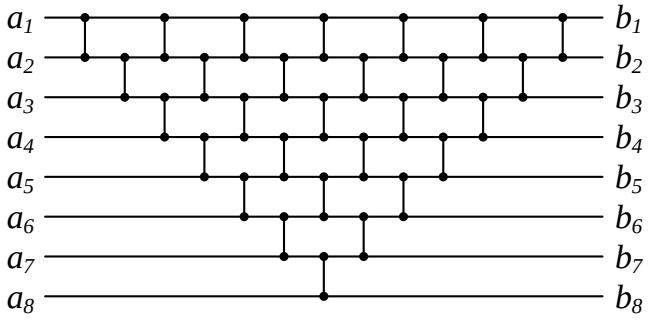

Figure 2: A Transposition Network Example

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

   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. We prove the 'if' part first.

   Define A_a^i is the element in the a line after the i_{th} comparator when the input is sequence A.

   We have the original sequence $D = \langle n, n-1, \ldots, 1\rangle$ and the normal sequence $A = \langle x_1, x_2, \ldots, x_n\rangle$.

   We can solve the problem by solving the claim that for any pairs of line i and line j, if $i < j, D_i^k < D_j^k$ then $A_i^k \leq A_j^k$.

   When $k = 0$, it's obvious that $D_i^0 > D_j^0$. The claim is true.

   Now we assume the claim is true after the $k-1_{th}$ comparator and the k_{th} comparator is between line i and $i+1$. We should prove all of the pairs (a, b) meets the claim, $a < b$.

   Case 1: $a = i$ and $b = i+1$. It's obvious that $A_a^k \leq A_b^k$ because they are just compared and sorted.

   Case 2: $a < i$ and $b > i+1$. Then $A_a^k = A_a^{k-1}$, $A_b^k = A_b^{k-1}$. The induction assumes that the $k-1_{th}$ comparator meets the claim, so does the k_{th} comparator since the input elements are the same.

   Case 3: $a < i$ and $b = i$. Then $D_a^k = D_a^{k-1}$, $A_a^k = A_a^{k-1}$, $D_b^k = \min\{D_i^{k-1}, D_{i+1}^{k-1}\}$, $A_b^k = \min\{A_i^{k-1}, A_{i+1}^{k-1}\}$. If $D_a^k < D_b^k$, then $D_a^{k-1} < \min\{D_i^{k-1}, D_{i+1}^{k-1}\}$. Because of the induction assume, $A_a^{k-1} \leq \min\{A_i^{k-1}, A_{i+1}^{k-1}\} = A_b^k$. Thus, $A_a^k = A_a^{k-1} \leq A_b^k$.

   Case 4: $a < i$ and $b = i+1$. Then $D_a^k = D_a^{k-1}$, $A_a^k = A_a^{k-1}$, $D_b^k = \max\{D_i^{k-1}, D_{i+1}^{k-1}\}$, $A_b^k = \max\{A_i^{k-1}, A_{i+1}^{k-1}\}$. If $D_a^k < D_b^k$, then $D_a^{k-1} < \max\{D_i^{k-1}, D_{i+1}^{k-1}\}$. Because of the induction assume, $A_a^{k-1} \leq \max\{A_i^{k-1}, A_{i+1}^{k-1}\} = A_b^k$. Thus, $A_a^k = A_a^{k-1} \leq A_b^k$.

   Case 5: $a = i$ and $b > i+1$. Then $D_a^k = \min\{D_i^{k-1}, D_{i+1}^{k-1}\}$, $A_a^k = \min\{A_i^{k-1}, A_{i+1}^{k-1}\}$, $D_b^k = D_b^{k-1}$, $A_b^k = A_b^{k-1}$. If $D_a^k < D_b^k$, then $\min\{D_i^{k-1}, D_{i+1}^{k-1}\} < D_b^{k-1}$. Because of the induction assume, $\min\{A_i^{k-1}, A_{i+1}^{k-1}\} \leq A_b^{k-1} = A_b^k$. Thus, $A_a^k = \min\{A_i^{k-1}, A_{i+1}^{k-1}\} \leq A_b^k$

   Case 6: $a = i+1$ and $b > i+1$. Then $D_a^k = \max\{D_i^{k-1}, D_{i+1}^{k-1}\}$, $A_a^k = \max\{A_i^{k-1}, A_{i+1}^{k-1}\}$, $D_b^k = D_b^{k-1}$, $A_b^k = A_b^{k-1}$. If $D_a^k < D_b^k$, then $\max\{D_i^{k-1}, D_{i+1}^{k-1}\} < D_b^{k-1}$. Because of the induction assume, $\max\{A_i^{k-1}, A_{i+1}^{k-1}\} \leq A_b^{k-1} = A_b^k$. Thus, $A_a^k = \max\{A_i^{k-1}, A_{i+1}^{k-1}\} \leq A_b^k$

   Thus we've prove our claim, so it's true that if a transposition network could sort $\langle n, n-1, \ldots, 1\rangle$, it's a sorting network.

Then we prove the 'only if' part. It's obvious that if it cannot sort $\langle n, n-1, \ldots, 1 \rangle$, then it's not a sorting network.

Together, a transposition network with n inputs is a sorting network if and only if it sorts the sequence $\langle n, n-1, \cdots, 1 \rangle$.

□