Lab02-Divide and Conquer

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

$*$ If there is any problem, please contact TA Jiahao Fan.

$*$ Name:Juncheng Wan Student ID:516021910620 Email: 578177149@qq.com

1. Assume that all elements of $A[1..n]$ are distinct 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. The pseudocode of Binary Search is in algorithm 1. I set mid equal $\lfloor (left + right)/2 \rfloor$ in this algorithm.

Algorithm 1: Binary Search

Input: An array $A[1, \cdots, n]$ ($n = 2^k$, $k \in \mathbb{N}$), x which is in A
Output: the position pos of x in A

$left \leftarrow 1$;
$right \leftarrow n$;
$pos \leftarrow -1$;
while *true* **do**
  $mid \leftarrow \lfloor (left + right)/2 \rfloor$;
  if $x < A[mid]$ **then**
    $right \leftarrow mid - 1$; continue;
  if $x > A[mid]$ **then**
    $left \leftarrow mid + 1$; continue;
  if $x = A[mid]$ **then**
    $pos \leftarrow mid$; break;
return pos

Assuming that each element of A is equally likely to be in any position in the array, the probability of x being in any position in A is $\frac{1}{n}$. For the 1st loop in **while**, Binary Search is able to find one element (position is 2^{k-1}). For the 2nd loop in **while**, Binary Search is able to find two elements (positions are 2^{k-2} and $2^{k-2} + 2^{k-1}$). For the 3rd loop in **while**, Binary Search is able to find four elements (positions are 2^{k-3}, $2^{k-3} + 2^{k-2}$, $2^{k-3} + 2^{k-1}$, $2^{k-3} + 2^{k-2} + 2^{k-1}$).

It is obvious to conclude that for the tth loop in **while**, where $t \leq k - 1$, Binary Search is able to find 2^{t-1} elements.

There will remain one element at last. Because $n - (2^0 + 2^1 + \cdots + 2^{k-1}) = 2^k - (2^0 + 2^1 + \cdots + 2^{k-1}) = 1$. The last element needs $k + 1$ comparisons. Set $B(i)$ the number of comparisons of x in Binary Search when the exact position of x in A is i.

$$
\begin{aligned}
T(n) &= \frac{1}{n}\sum_{i=1}^{n} B(i) \\
&= \frac{1}{n}[\sum_{i=1}^{k} i2^{i-1} + (k+1)] \\
&= \frac{1}{n}[(k-1)2^k + 1 + (k+1)] \\
&= \frac{1}{2^k}[(k-1)2^k + k + 2] \\
&= (k-1) + \frac{k+2}{2^k} \\
&= logn + \frac{logn+2}{n} - 1
\end{aligned}
$$

□

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).

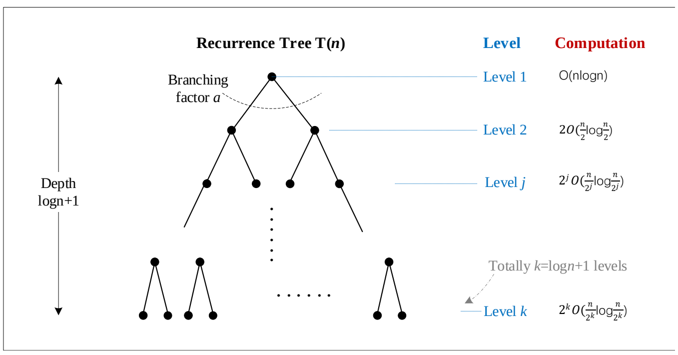

图 1: A Recurrence Tree

Solution. Set $T(n)$ the time complexity of this algorithm.

$$
\begin{aligned}
T(n) &= 2T(\frac{n}{2}) + \frac{n}{2}(O(log\frac{n}{2}) + O(log\frac{n}{2})) + O(nlogn) \\
&= 2T(\frac{n}{2}) + O(nlogn) \\
&= O(\sum_{i=0}^{\lfloor logn \rfloor} nlog\frac{n}{2^i}) \\
&= O(n(\lfloor logn \rfloor + 1)logn - \frac{\lfloor logn \rfloor(\lfloor logn \rfloor + 1)}{2}) \\
&= O(n(logn)^2)
\end{aligned}
$$

□

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

Solution. If we use the Master Theorem, $a = 2, b = 2, d = log_n nlogn = 1 + log_n logn$. We get $log_b a = 1 < d$. Using the Master Theorem, $T(n) = n^d = nlogn$. The answer is not in accord with the answer in (b). Therefore, I think that if we can just regard the d in the Master Theorem a constant. When d is a function of n, the Master Theorem will fail. However, the proof is waitting to done. □

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

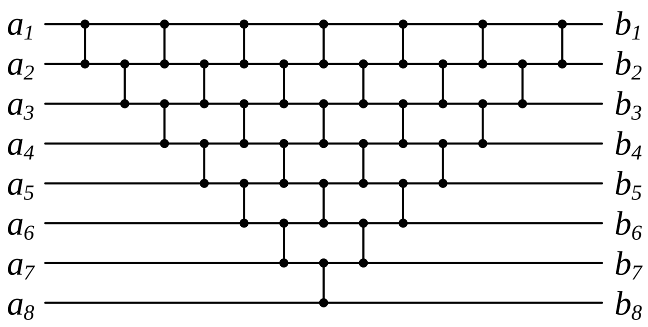

图 2: A Transposition Network Example

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. First Step: When $n = 2$, if the transposition network can sort the sequence $\langle 2, 1\rangle$ and output 1 in the upper and 2 in the lower, it will mean that this comparator output the smaller one in the upper and the larger one in the lower. For input a_1, a_2, the upper output is $min\{a_1, a_2\}$ and the lower output is $max\{a_1, a_2\}$.

Thus the proof of the claim as the base case is completed.

Second Step: Assume that when $n = k-1$, the ability of sorting the sequences $\langle k-1, k-2, \cdots, 1\rangle$ in the transposition network means the ability of sorting any sequences $\langle a_1, a_2, \cdots, a_{k-1}\rangle$ in the transposition network.

Third Step: When $n = k$, divide the transposition network into two parts. The first part is the triangle on the top left which is the same as the transposition network when $n = k - 1$. The second park is the rightmost line.

Since in the **Second Step** assuming that the $(k-1)$th line (before the rightmost comparator) assumes the largest element in the sequences $\langle a_1, a_2, \cdots, a_{k-1} \rangle$. Set it b_{k-1}. After the rightmost comparator between the $(k-1)$th line and the kth line, the network comparas b_{k-1} and a_k. Thus the last line assumes the largest element of the sequences $\langle a_1, a_2, \cdots, a_{k-1}, a_k \rangle$.

Since in the **Second Step** assuming that the $(k-2)$th line (before the rightmost comparator) assumes the second largest element in the sequences $\langle a_1, a_2, \cdots, a_{k-1} \rangle$. After the rightmost comparator between the $(k-2)$th line and the $(k-1)$th line, the $(k-1)$th line assumes the second largest element of the sequences $\langle a_1, a_2, \cdots, a_{k-1}, a_k \rangle$.

After $k-1$ comparators, the rightmost sequences is the sorted sequences of $\langle a_1, a_2, \cdots, a_{k-1}, a_k \rangle$.

$\square$