Homework #8

You should try to solve these problems by yourself. I recommend that you start early and get help in office hours if needed. If you find it helpful to discuss problems with other students, go for it. **The goal is to be ready for the in class quiz (in this case the first exam) that will cover the same or similar problems.**

Problem 1: Master Method

Use the master method to give a tight asymptotic bound for each of the following recurrences.

1. $T(n) = 8T(n/2) + \Theta(n^3 \lg n)$

   Solution

   $a = 8$, $b = 2$, $l = 3$, $k = 1$. Then $\log_2 8 = 3 = l$. So Case 2.
   $\Theta(n^3(\log n)^2)$

2. $T(n) = 3T(n/2) + \Theta(n)$

   Solution

   $a = 3$, $b = 2$, $l = 1$, $k = 0$. Then $\log_2 3 = 1.58 > l$. So Case 1.
   $\Theta(n^{\log_2 3})$

3. $T(n) = 3T(n/2) + \Theta(n^2)$

   Solution

   $a = 3$, $b = 2$, $l = 2$, $k = 0$. Then $\log_2 3 = 1.58 < l$. So Case 3.
   $\Theta(n^2)$

4. $T(n) = 16T(n/2) + \Theta(n^3 \lg n)$

   Solution

   $a = 16$, $b = 2$, $l = 3$, $k = 1$. Then $\log_2 16 = 4 > 2$. So Case 1.
   $\Theta(n^4)$

5. $T(n) = T(9n/10) + \Theta(n)$

   Solution

   $a = 1$, $b = 10/9$, $l = 1$, $k = 3$. Then $\log_{10/9} 1 = 0 < 1$. So Case 3.
   $\Theta(n)$

Problem 2: Recursion Trees

Determine a good asymptotic upper bound for the following recurrence using a recursion tree: $T(n) = T(n/2) + n^2$. Verify your answer using the substitution method.

Solution

$$
\begin{aligned}
T(n) &= \left[\sum_{i=0}^{(\log_2 n)-1} 1^i \times \Theta((n/2^i)^2)\right] + \Theta(n^{\log_2 1}) \\
&= \left[cn^2 \sum_{i=0}^{(\log_2 n)-1} (1/4)^i\right] + 0 \\
&< \left[cn^2 \sum_{i=0}^{\infty} (1/4)^i\right] \\
&= \frac{4}{3} cn^2 \\
&= \Theta(n^2)
\end{aligned}
$$

Problem 3: Divide and Conquer

Suppose you are given a sorted sequence of *distinct* integers $\{a_1, a_2, \ldots a_n\}$. Give an $O(\log n)$ algorithm to determine whether there exists an index i such that $a_i = i$. For example, in $\{-10, -3, 3, 5, 7\}$, $a_3 = 3$; there is no such i in $\{2, 3, 4, 5, 6, 7\}$. Write the recurrence for your algorithm and show that its recurrence solves to $O(\log n)$ (e.g., using the Master Method, a recursion tree, or the substitution method).

Solution

Let MATCH(i, j) return true if $a_i = i$ or if $a_{i+1} = i + 1 \ldots$ or if $a_j = j$. We call this function at the beginning with MATCH$(1, n)$.

MATCH(i, j)

if $i > j$
  then return false
if $i = j$
  then if $a_i = i$ **return** true
  else **return** false
if $i < j$
  then $m \leftarrow \lfloor \frac{i+j}{2} \rfloor$
  if $a_m = m$ **then return** true
  if $a_m > m$ **then return** MATCH$(i, m - 1)$
  if $a_m < m$ **then return** MATCH$(m + 1, j)$

The recurrence for the above is derived as follows. It generates one subproblem of size $n/2$. The other work (outside of the recursive call) takes $O(1)$ time. So the recurrence is $T(n) = T(n/2) + O(1)$. Using the master method, $a = 1$, $b = 2$, $l = 0$, $k = 0$. This is case 2 ($\log_b(a) = 0 = l$). So the solution is $O(n^l (\log n)^{k+1}) = O(\log n)$.

Problem 4: Divide and Conquer, Take 2

Suppose you are given an array A of n sorted numbers that has been *circularly shifted* to the right by k positions, where k is unknown to you. For example $\{35, 42, 5, 15, 27, 29\}$ is a sorted array that has been circularly shifted $k = 2$ positions, while $\{27, 29, 35, 42, 4, 15\}$ has been shifted $k = 4$ positions. Give an $O(\log n)$ algorithm to find the largest number in A. You may assume the elements of A are distinct. Write the recurrence for your algorithm and show that its recurrence solves to $O(\log n)$ (e.g., using the Master Method, a recursion tree, or an inductive proof).

Solution

Let MAX-CIRC$(i.j)$ return x if a_x is the max element in the array and 0 if no x in the range i, j is the max element. We call this function at the beginning with MAX-CIRC$(1, n)$.

MAX-CIRC(i, j)

$m \leftarrow \lfloor \frac{i+j}{2} \rfloor$
if $A[m] > A[m-1]$ **and** $A[m] > A[m+1]$ **then return** m
if $A[m] > A[1]$ **then return** MAX-CIRC$(m+1, j)$
if $A[m] < A[1]$ **then return** MAX-CIRC$(i, m-1)$

I'm assuming the elements are distinct; there would be a couple of extra checks otherwise. The recurrence for the above is derived as follows. It generates one subproblem of size $n/2$. The other work (outside of the recursive call) takes $O(1)$ time. So the recurrence is $T(n) = T(n/2) + O(1)$. Using the master method, $a = 1$, $b = 2$, $l = 0$, $k = 0$. This is case 2 ($\log_b(a) = 0 = l$). So the solution is $O(n^l (\log n)^{k+1}) = O(\log n)$.