CS 375: Theory Assignment #2

Due on February 26, 2016 at 2:20pm

Professor Lei Yu Section B1

I have done this assignment completely on my own. I have not copied it, nor have I given my solution to anyone else. I understand that if I am involved in plagiarism or cheating I will have to sign an official form that I have cheated and that this form will be stored in my official university record. I also understand that I will receive a grade of 0 for the involved assignment for my first offense and that I will receive a grade of F for the course for any additional offense.

Tim Hung

Problem 1

(24 points) Use the Master theorem to solve the following recurrences (show necessary steps to justify your answer).

a) $T(n) = 3T(\frac{n}{4}) + n$

Solution
Consider $T(n) = 3T(\frac{n}{4}) + n$.
We have $a = 3$, $b = 4$, and $f(n) = n$
$\log_b a = \log_4 3$. Since $1 = \log_4 4$, $\varepsilon = \log_4 4 - \log_4 3 > 0$.
$\rightarrow f(n) = n = n^{\log_4 3+\varepsilon} \in \Omega(n^{\log_4 3+\varepsilon})$
$af(\frac{n}{b}) = 3f(\frac{n}{4}) = 3(\frac{n}{4}) = 3\frac{n}{4} \leq cn$ for $c = \frac{3}{4} < 1$
According to Case 3 of the Master Theorem
$\rightarrow T(n) = \Theta(n)$

b) $T(n) = 2T(\frac{n}{4}) + \sqrt{n}\lg n$

Solution
Consider $T(n) = 2T(\frac{n}{4}) + \sqrt{n}\lg n$.
We have $a = 5$, $b = 2$, and $f(n) = \sqrt{n}\log(n)$
Since $n^{\log_b a} = n^{\log_4 2} = n^{\frac{1}{2}} = \sqrt{n}$, $f(n) = n^{\log_4 2}\log(n)$.
We use Case 2 of the Master Theorem with $k = 1$
$\rightarrow T(n) = \Theta(\sqrt{n}\log^{1+1} n) = \Theta(\sqrt{n}\log^2 n)$

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

Solution
Consider $T(n) = 5T(\frac{n}{2}) + n^2$.
We have $a = 5$, $b = 2$.
Since $\log_b a = \log_2 5 > \log_2 4, \varepsilon = \log_2 5 - \log_2 4 > 0$.
Since $f(n) = n^2 = n^{\log_2 5-\varepsilon} \in O(n^{\log_2 5-\varepsilon})$,
according to Case 1 of the Master Theorem
$\rightarrow T(n) = \Theta(n^{\log_2 5})$

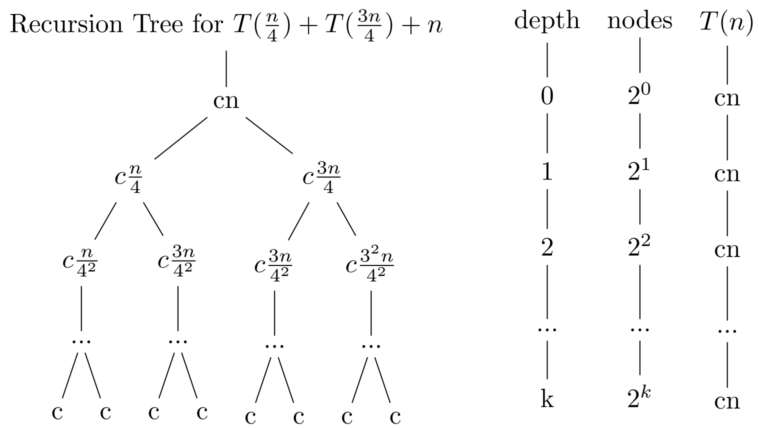
Problem 2

(20 points) Solve the recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{for } n \leq 1 \\ T(\frac{n}{4}) + T(\frac{3n}{4}) + n & \text{otherwise} \end{cases} \tag{1}$$

using the recursion tree method. Draw the recursion tree and show the aggregate instruction counts for the following levels (0th, 1st, and last levels), and derive the Θ growth class for $T(n)$ with justifications.

Solution



$$T(n) = (k+1)(cn) = (\log(n)+1)(cn) = \Theta(n\log(n))$$

Problem 3

(10 points) Use the substitution method to prove that $T(n) = T(n-1) + n \in O(n^2)$. You can assume $T(1) = 1$.

Solution

We want to show that $T(n) \leq cn^2 \forall n \geq n_0$.

We assume $T(k) \leq ck^2 \forall k < n$.

$$\begin{aligned} T(n) = (n-1) + n \quad & \leq c(n-1)^2 + n \\ & = cn^2 - 2cn + c + n \\ & = cn^2 - 2cn + n + c \\ & = cn^2 - 2cn + n + c \qquad \leq cn^2 \\ & \text{provided that } -2cn + n + c \leq 0. \end{aligned} \tag{2}$$

For $-2cn + n + c \leq 0$ to hold, we need $c(2n-1) \geq n$
which can be satisfied when $c \geq \frac{n}{2n-1}$ for all $n \geq 2$.
$\rightarrow c \geq 1$ and $n \geq 2$.

Problem 4

(21 points) Assume that you are given an array of n ($n \geq 1$) elements sorted in non-descending order. Design a ternary search function that searches the array for a given element x by applying the divide and conquer strategy. Hint: extend the binary search example introduced in the class - divide the array into three subarrarys where each subarray has $\frac{n}{3}$ (or almost $\frac{n}{3}$) elements).

Your answer should contain four parts:

a) Briefly describe the divide, conquer, and combine steps

b) Clearly define the recursive function ternarySearch(x, A, left, right), where x is the element to search for in the array A with starting index left and ending index right

c) Clearly define the recursive time complexity function T(n) for ternarySearch(x, A, left, right)

d) Solve the recursive T(n) by the master theorem

Solution

a) Divide: Check element at the $\frac{1}{3}^{\text{rd}}$ position. If needed, also check element at the $\frac{2}{3}^{\text{rd}}$ position.
Conquer: Recursively search one subarray of size approximately $\frac{1}{3}n$.
Combine: Trivial.

b)
```
ternarySearch(x, A, left, right)
    part1 = left + (right - left) / 3
    part2 = left + 2*(right - left) / 3
    if A[part1] == x
        return true
    if A[part2] == x
        return true
    if x < A[part1]
        return ternarySearch (x, A, left, part1 - 1)
    if x < A[part2]
        return ternarySearch (x, A, part1, part2)
    if x > A[part2]
        return ternarySearch (x, A, part2 + 1, right)
```

c) $T(n) = T(\frac{n}{3}) + \Theta(1)$
Reasoning: 1 subproblem chosen, each subarray of size $\frac{1}{3}$, and trivial constant time for dividing and combining.

d) Consider $T(n) = T(\frac{n}{3}) + \Theta(1)$
We have $a = 1$, $b = 3$, and $f(n) = 1$
Since $n^{\log_b a} = n^{\log_3 1} = n^0$, $f(n) = n^0 \log^0(n)$.
We use Case 2 of the Master Theorem with $k = 0$
$\rightarrow T(n) = \Theta(n^{\log_3 1} \log^{(0+1)} n) = \Theta(n^0 \log^1 n) = \Theta(\log n)$

Problem 5

(25 points) Computing the median of n numbers is easy: just sort them. The drawback of this approach is that this takes O(n log n) time, whereas we would ideally like something linear. We have reason to be hopeful, because sorting is doing far more work than what we really need - we just want the middle element and don't care about the relative ordering of the rest of them. Can we develop a recursive solution for deciding the median of a list of numbers?
When looking for a recursive solution, it is paradoxically often easier to work with a more general version of the problem. In our case, the generalization we will consider is selection.

SELECTION
Input: A list of numbers S; an integer k
Output: The kth smallest element of S
For instance, if k = 1, the minimum of S is sought, whereas if k = ceiling($\frac{|S|}{2}$), it is the median.

Develop a divide-and-conquer approach to selection (and hence a solution for the finding median problem). Hint: for any number v, imagine splitting list S into three categories: elements smaller than v, those equal to v (there might be duplicates), and those greater than v.

In your answer, show the following:

a) Briefly describe the divide, conquer, and combine steps
b) Clearly define the recursive function for selection(S, k); (note: this is not the function for the time complexity of the selection function.)
c) Analyze the best case and worst case time complexity of this approach given input size n.

Solution

a) Divide: Use the partition method from QUICKSORT about a randomly chosen pivot point to divide into two subarrays.
Conquer: Select from the one chosen subarray recursively.
Combine: Trivial.

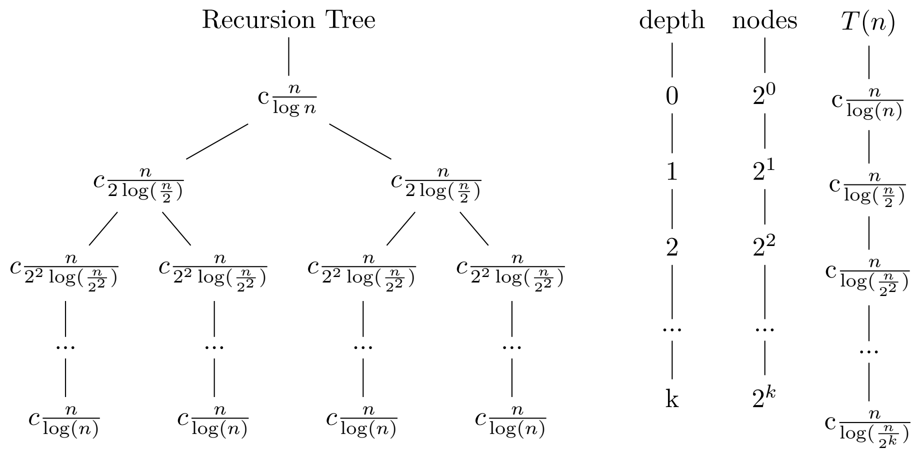
b)
```
SELECT(A, k, left, right)
    if left < right
        pivot = PARTITION(A, left, right)
        if k < pivot - left
            SELECT(A, k, left, pivot)
        else if k > pivot
            SELECT(A, k, pivot, right)
        else
            return A[0]
```

c) Best case: $O(n)$ (Only have to select from one subarray, as opposed to quicksort which has to subsequently sort each subarray.)
Worst case: $O(n^2)$

Problem 6

Bonus Question (20 points):
We know that the master theorem does not apply to the recursive function $T(n) = 2T(\frac{n}{2}) + \frac{n}{\log n}$. Use the recursion tree method to solve this recursion. Draw the recursion tree and show the aggregate instruction counts for the following levels (0^{th}, 1^{st}, and last levels), and derive the growth class for T(n) with justifications.

Solution



$$T(n) = \sum_{i=0}^{k} \frac{n}{\log(\frac{n}{2^i})} = \sum_{i=0}^{k} \frac{n}{\log(n) - \log(2^i)} = \sum_{i=0}^{k} \frac{n}{\log(n) - i\log(2)} = \sum_{i=0}^{k} \frac{n}{\log(n) - i} = ?$$