CPSC 320 2024W1: Divide & Conquer Tutorial Problems

1 Practice at Solving Recurrences

A student in the class has proposed a sophisticated algorithm to predict whether or not there will be a snow storm during the final, based on data from n previous years. The recurrence of the student's algorithm is as follows:

$$T(n) = \begin{cases} 2T(n/4) + T(3n/4) + cn^2, & \text{if } n \geq 2 \\ c, & \text{if } n = 1. \end{cases} \tag{1}$$

1. You want to get an upper bound on $T(n)$. Since the Master Theorem does not handle recurrences like this, with two terms involving $T()$ on the right hand side, you decide to work with the following recurrence:

$$T(n) \leq \begin{cases} 3T(3n/4) + cn^2, & \text{if } n \geq 2 \\ c, & \text{if } n = 1. \end{cases} \tag{2}$$

Apply the Master Theorem to solve recurrence (2). Here is a statement of the Master Theorem:

Theorem: Let $T : \mathbb{N} \to \mathbb{R}_{\geq 0}$ be defined by

$$T(n) = \begin{cases} aT(n/b) + cn^k, & \text{for } n \geq n_0, \\ c, & \text{for } n < n_0, \end{cases}$$

where $a > 0$, $b > 1$, $c > 0$, and $k \geq 0$ are constants.

- If $a > b^k$, then $T(n) = \Theta(n^{\log_b a})$. → leaf heavy
- If $a = b^k$, then $T(n) = \Theta(n^k \log n)$.
- If $a < b^k$, then $T(n) = \Theta(n^k)$. → root heavy

subproblems

work to split or combine subproblems

$a = 3$

$b = \frac{4}{3}$

$k = 2$

$a > b^k \Rightarrow$

$T(n) = \Theta\left(n^{\log_{\frac{4}{3}}(3)}\right)$

$b^k = \left(\frac{4}{3}\right)^2 = \frac{16}{9}$

2. Here again is the recurrence from the previous page:

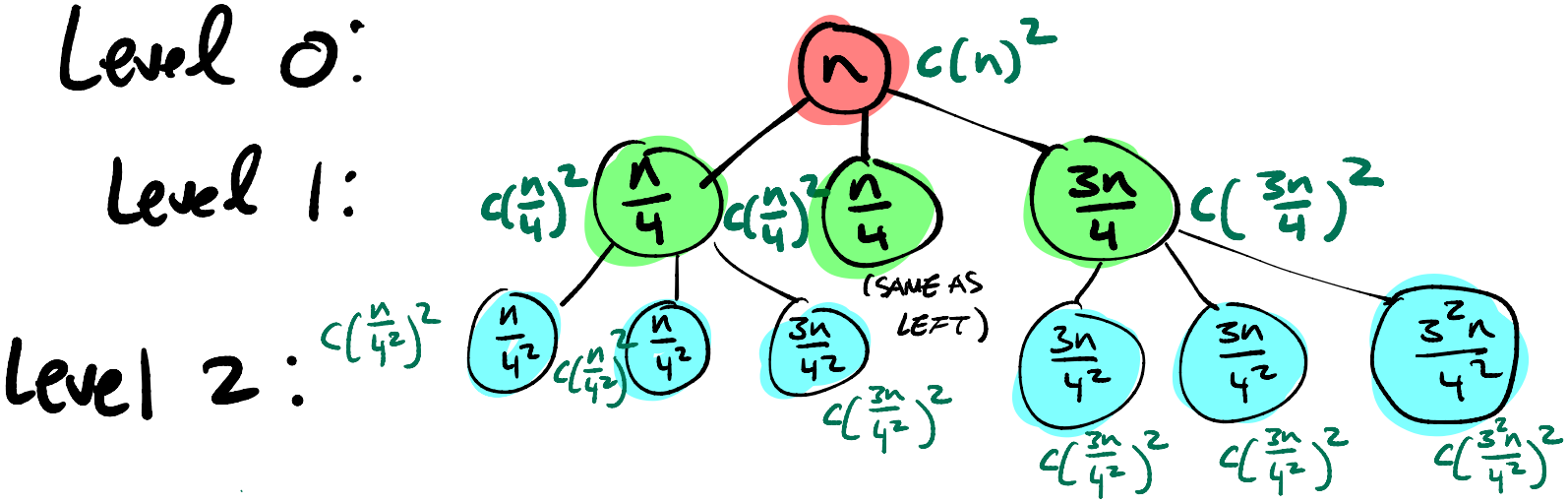
$$T(n) = \begin{cases} 2T(n/4) + T(3n/4) + cn^2, & \text{if } n \geq 2 \\ c, & \text{if } n = 1. \end{cases} \tag{3}$$

Draw level 0 (the root), level 1 and level 2 of the recursion tree for the original recurrence (1). Within each node, write (i) the size of the subproblem at this node and (ii) the time needed for the subproblem at this node (not counting times at deeper levels of recursion).

Level 0:

Level 1:

Level 2:



5^{th}: $cn^2\left(\frac{11}{16}\right)^5$

$\left(\frac{1}{4}\right)^2 + \left(\frac{1}{4}\right)^2 + \left(\frac{3}{4}\right)^2$

3. As a function of n, what is the total time needed at levels 0, 1 and 2 of the tree?

Level 0: cn^2 Level 1: $cn^2\left(\frac{11}{16}\right)$ Level 2: $cn^2\left(\frac{11}{16}\right)^2$

4. Generalizing the pattern from the first three levels of the tree, write down the total time needed at level i of the tree.

Level i: $cn^2\left(\frac{11}{16}\right)^i$

5. Write down a sum that upper bounds the total time over all levels of the tree.

$$\sum_{i=0}^{\infty} cn^2\left(\frac{11}{16}\right)^i$$

3. As a function of n, what is the total time needed at levels 0, 1 and 2 of the tree?

Level 0: [ ] Level 1: [ ] Level 2: [ ]

4. Generalizing the pattern from the first three levels of the tree, write down the total time needed at level i of the tree.

Level i: [ ]

5. Write down a sum that upper bounds the total time over all levels of the tree.

6. Use part 5 to provide a big-O bound on the running time of the algorithm as a function of n.

$$\sum_{i=0}^{\infty} cn^2 \left(\frac{11}{16}\right)^i = cn^2 \sum_{i=0}^{\infty} \left(\frac{11}{16}\right)^i = c'n^2 \in O(n^2)$$

7. Which method leads to a better bound, the Master Theorem or the recursion tree?

Tree: $O(n^2)$

MT: $O(n^{\log_{\frac{4}{3}}(3)})$

$n^2 \in o(n^{\log_{\frac{4}{3}}(3)})$

2 Divide & conquer

Consider the problem of taking a **sorted** array A containing **distinct** (and not necessarily positive) integers, and determining whether or not there is a position i such that $A[i] = i$.

1. Describe a divide-and-conquer algorithm to solve this problem. Your algorithm should return such a position if it exists, or *nil* otherwise. If $A[i] = i$ for several different integers i, then you may return any one of them.

$A = [A_1 \quad A_2 \quad A_3 \quad \cdots \quad A_n]$, $A_i > A_{i-1}$ for all $i \leq n$

e.g. if $A = [-5 \quad -1 \quad 0 \quad 4 \quad 9]$, 4 should be returned since $A[4] = 4$
(1 2 3 4 5)

Claim: If $A[mid] < mid$, then for every non-negative integer $j \leq mid$, $A[mid - j] < mid - j$

We have the following algorithm:

```
function FINDPOSITION(A, first, last)
    if first > last then
        return nil
    end if
    if first = last then
        if A[first] = first then
            return first
        end if
        return nil
    end if
    mid ← ⌊(first + last)/2⌋
    if A[mid] = mid then
        return mid
    end if
    if A[mid] < mid then
        return FINDPOSITION(A, mid + 1, last)
    else
        return FINDPOSITION(A, first, mid − 1)
    end if
end function
```

2. Analyze the running time of your algorithm as a function of the number of elements of A.

$$T(n) \leq \begin{cases} T(\lfloor \frac{n}{2} \rfloor) + \Theta(1) & \text{if } n > 1 \\ \Theta(1) & n = 1 \end{cases}$$

MT CASES:

If $a > b^k$, then $T(n) = \Theta(n^{\log_b a})$.
If $a = b^k$, then $T(n) = \Theta(n^k \log n)$.
If $a < b^k$, then $T(n) = \Theta(n^k)$.

$a = 1$
$b = 2$
$k = 0$

$b^k = 2^0 = 1 \quad a = b^k \quad T(n) = \Theta(n^k \log n)$

$= \Theta(\log n)$