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.

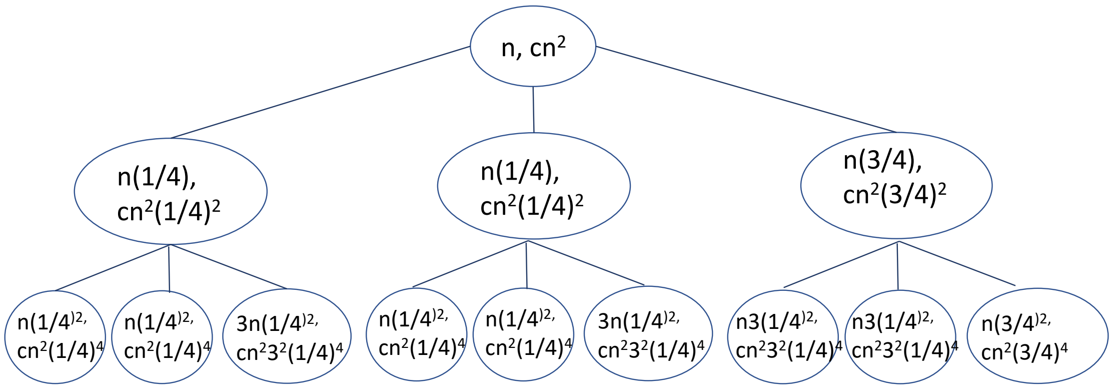
- If $a > b^k$, then $T(n) = \Theta(n^{\log_b a})$.
- If $a = b^k$, then $T(n) = \Theta(n^k \log n)$.
- If $a < b^k$, then $T(n) = \Theta(n^k)$.

Applying the notation of the Master theorem to this recurrence, we have that $a = 3$, $b = 4/3$ and $f(n) = cn^2$. Since $\log_b a = \log_{4/3} 3 > \log_{4/3}(4/3)^2 = 2$, so we see that $f(n) = cn^2 = O(n^{(\log_b a - \epsilon)})$ for some $\epsilon > 0$. So case 1 of the theorem applies and we conclude that $T(n) = O(n^{\log_{4/3} 3})$.

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).



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

Level 0: $\boxed{cn^2}$ Level 1: $\boxed{cn^2(11/16)}$ Level 2: $\boxed{cn^2(11/16)^2}$

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: $\boxed{cn^2(11/16)^i}$

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

$cn^2 \sum_{i=0}^{\infty}(11/16)^i$.

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

The sum in the expression of part 5 is a geometric sum that converges to a constant. So the running time of the algorithm is $O(n^2)$.

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

The recursion tree provides a better bound because $n^2 = o(n^{\log_{4/3} 3})$.

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.

   The algorithm described here will find if such a position exists between two positions $first$ and $last$ of the array, including the endpoints. The idea is simple: we look at the middle position, and then recurse on the either the first half or the second half of the array depending on the result of the comparison (similarly as the binary search does).

   For instance, assume that $A[mid] < mid$. Then the value of $A[m-1]$ is at least by one less than $A[mid]$, so it will be still less than $m-1$. More formally, we can prove the following claim, which justifies, why we can discard the left subarray when $A[mid] < mid$.

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

   Proof. By induction on j. When $j = 0$, we are comparing $A[mid]$ to mid, which is true since this is the case we are examining. Suppose now that the claim holds for j. Because A contains elements that are distinct and sorted, $A[mid-(j+1)] \leq A[mid-j]-1 < (mid-j)-1 = mid-(j+1)$. □

   Similarly, if $A[mid] > mid$, then it's possible to prove that $A[mid+j] > mid+j$ for every non-negative integer 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
    $mid \leftarrow \lfloor (first + last)/2 \rfloor$
    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.

   The running time of the algorithm satisfies the recurrence relation $T(n) \leq T(\lfloor n/2 \rfloor) + \Theta(1)$ with $T(1) \in \Theta(1)$ and so by Case 2 of the Master theorem, $T(n) \in \Theta(\log n)$.