Master Theorem

CS 5633 Analysis of Algorithms

Computer Science
University of Texas at San Antonio

September 16, 2024

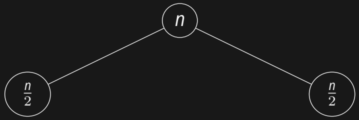
Recursive Trees

Recursive Trees

- You probably know that analyzing the run-time of divide-and-conquer algorithms is very hard.
- Although we know the induction-based method, it is still quite challenging to make a good guess of the run-time.
- Recursive Tree is another way of guessing the run-time of a divide-and-conquer algorithm.

An Example: Merge Sort Recursive Tree

- Let's consider the run-time of merge sort, which is $T(n) = 2 \cdot T(\frac{n}{2}) + O(n)$.
- At the top level, the time complexity can be draw as a tree.



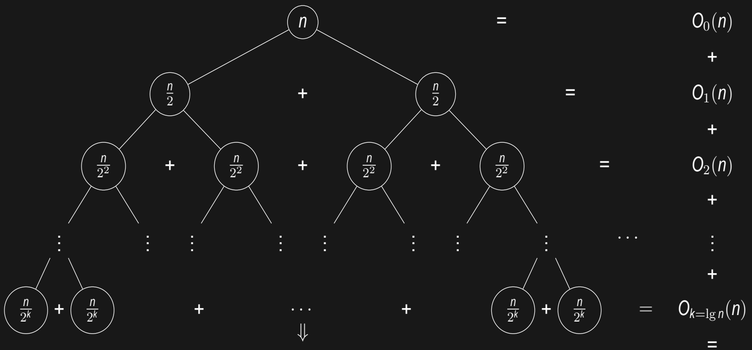
$= O(n)$

$= O(n)$

- The root represents the whole merge sort, with an input size of n.
  - The basic cost at the root is just the merge step, which is $O(n)$.
- The two children are represents two recursive calls on the sub arrays, each with an input size of $\frac{n}{2}$.
  - The basic cost at this level is the two merges, which is $O(\frac{n}{2}) + O(\frac{n}{2}) = O(n)$.

An Example: Merge Sort Recursive Tree cont.

- If we further expand the tree:



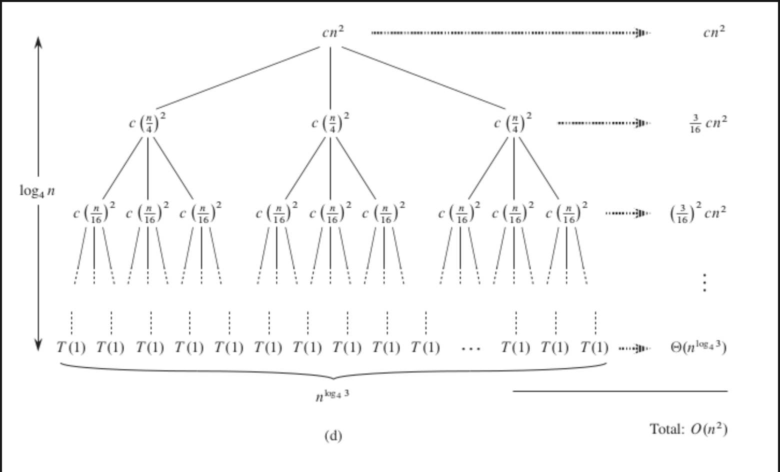
$$O\left(\sum_{i=0}^{k} 2^i \cdot \frac{n}{2^i}\right) = O\left(\sum_{i=0}^{k} n\right) = O(k \cdot n) \Leftrightarrow O(n \cdot \lg n)$$

An Example: Merge Sort Recursive Tree cont.

- The height of the whole tree is $\lg n$.
  - The tree stops when $\frac{n}{2^k} = 1$, or when $k = \lg n$.
- On each level, the run time is $O(n)$.
- If we sum the the run times of all levels together, we have the total run time, which is $O(k \cdot n) = O(n \lg n)$.

Another Example: $T(n) = 3T(\frac{n}{4}) + c \cdot n^2$

- Lets consider another example from CLRS Ch 4.4, $T(n) = 3T(\frac{n}{4}) + c \cdot n^2$



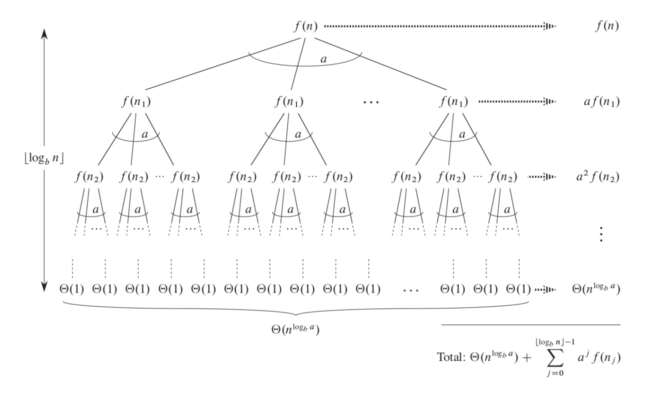
Another Example: $T(n) = 3T(\frac{n}{4}) + c \cdot n^2$ cont.

- The height of the whole tree is $\log_4 n$.
  - The tree stops when $\frac{n}{4^k} = 1$, or when $k = \log_4 n$.
- The bottom level of the tree has $n^{\log_4 3}$ nodes.
  - At a level l, there are 3^l nodes.
  - At the bottom level, there are $3^k = 3^{\log_4 n} = n^{\log_4 3}$ nodes.
- The overall cost of this recursive function is, by summing the costs at each level,

$$
\begin{aligned}
T(n) &= cn^2 + \frac{3}{16}cn^2 + (\frac{3}{16})^2 cn^2 + \ldots + (\frac{3}{16})^{\log 4n-1} cn^2 + \\
&\quad \Theta(n^{\log_4 3}) \\
&= \sum_{i=0}^{\log_4 n-1} (\frac{3}{16})^i cn^2 + \Theta(n^{\log_4 3}) \\
&< \sum_{i=0}^{\infty} (\frac{3}{16})^i cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{3}{13}cn^2 + \Theta(n^{\log_4 3}), (\text{ by eq A.5 in CLRS}) \\
&= O(n^2)
\end{aligned}
\qquad (1)
$$

The Generic Divide-and-Conquer Recursive Tree

- Consider the generic divide-and-conquer run time $T(n) = aT(\frac{n}{b}) + f(n)$
- The recursive tree for this function is,



The Generic Divide-and-Conquer Recursive Tree cont.

- The height of the whole tree is $\log_b n$.
  - The tree stops when $\frac{n}{b^k} = 1$, or when $k = \log_b n$.
- The bottom level of the tree has $n^{\log_b a}$ nodes.
  - At a level l, there are a^l nodes.
  - At the bottom level, there are $a^k = a^{\log_b n} = n^{\log_b a}$ nodes.
- The cost at level l is $a^l f(n_l)$.
- The overall cost is, by summing the cost at each level is $\Theta(n^{\log_b a}) + \sum_{l=0}^{\log_b n - 1} a^l f(n_l)$.
  - The actual time complexity will be determined by the larger one of the two terms, $\Theta(n^{\log_b a})$ and $\sum_{l=0}^{\log_b n - 1} a^l f(n_l)$

Master Theorem

The Master Theorem

- Directly derived from the analysis of slide 10, of comparing the two terms.
- The Master Theorem: For any recursive cost function in the form of $T(n) = aT(\frac{n}{b}) + f(n)$,
  1. if $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
  2. if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
  3. if $f(n) = \Omega(n^{\log_b a + \epsilon})$, for some constant $\epsilon > 0$, and if $af(\frac{n}{b}) \leqslant cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Using Master Theorem: Merge Sort

- Merge sort cost function: $T(n) = 2T(\frac{n}{2}) + \Theta(n)$
- First term: $n^{\log_b a} = n^{log_2 2} = n$.
- Second term: $f(n) = \Theta(n)$
- Apparently $f(n) = \Theta(n^{\log_b a})$, so the run time is $T(n) = \Theta(n \lg n)$.

Using Master Theorem: Divide-n-Conquer Matrix Multiplication

- DnC Matrix multiplication cost function: $T(n) = 8T(\frac{n}{2}) + \Theta(n^2)$
- First term: $n^{\log_b a} = n^{log_2 8} = n^3$.
- Second term: $f(n) = \Theta(n^2)$
- Apparently $f(n) = O(n^{\log_b a - \epsilon})$, for $\epsilon = 0.1$ so the run time is $T(n) = \Theta(n^3)$.

Using Master Theorem: Strassen's Algorithm

- Consider cost function: $T(n) = 7T(\frac{n}{2}) + \Theta(n^2)$
- First term: $n^{\log_b a} = n^{log_2 7} = n^{2.81}$.
- Second term: $f(n) = \Theta(n^2)$
- Apparently $f(n) = O(n^{\log_b a - \epsilon})$, for $\epsilon = 0.1$ so the run time is $T(n) = \Theta(n^{2.81})$.

Using Master Theorem: Cases Not Applicable

- b is not a constant: $T(n) = sin(n)$ or $T(n) = \sqrt{n}$
- Consider cost function: $T(n) = 4T(\frac{n}{2}) + \frac{n^2}{\log n}$
- First term: $n^{\log_b a} = n^{log_2 4} = n^2$.
- Second term: $f(n) = \frac{n^2}{\log n}$
- However, $\frac{n^2}{\log n} \neq O(n^{2-\epsilon})$
  - Suppose there is an ϵ such that $\frac{n^2}{\log n} \leqslant c \cdot n^{2-\epsilon}$,
  - Then we have,
    $if \frac{n^2}{\log n} \leqslant c \cdot n^{2-\epsilon} \implies \frac{n^2}{\log n} \leqslant c \cdot \frac{n^2}{n^\epsilon} \implies \frac{1}{\log n} \leqslant c \cdot \frac{1}{n^\epsilon}$
  - The above inequality is false for every $\epsilon > 0$

Using Master Theorem: Case 3 Example

- Consider cost function: $T(n) = 4T(\frac{n}{2}) + n^3$
- First term: $n^{\log_b a} = n^{log_2 4} = n^2$.
- Second term: $f(n) = n^3$
- Apparently $f(n) = \Omega(n^{\log_b a + \epsilon})$, for $\epsilon = 0.1$.
- Also $af(\frac{n}{b}) = 4(\frac{n}{2})^3 = \frac{1}{2}n^3$. Apparently, $af(\frac{n}{b}) \leqslant c \cdot f(n)$, for $c = \frac{1}{2}$.
- This example falls in case 3, so the run time is $T(n) = \Theta(n^3)$.