Tree Method (Master Method)

Can be used to solve recurrences of the form:

\[ T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d) \quad \text{for } n > C \]

\[ T(n) = O(1) \quad \text{for } n < C \]

Q: If \( T(n) \) is runtime of an algorithm, what are \( a, b, d \) in words?

A: 
\( a \): number of recursive calls
\( b \): factor by which problem shrinks in recursive call
\( d \): characterizes extra work outside recursive call

Ex: \( a = 3 \), \( b = 5 \), \( d = 4 \)

\[ O\left(\frac{n^4}{5}\right) \quad O\left(\frac{n^4}{25}\right) \quad O(1) \]

Input size: \( n \)

Proof of Master Method Page 1
Proof of Tree Method

Problem Size $n$

Level 0

Level 1

Level 2

Level $F$ $b$ $b$ $b$ $b$ $b$ $b$ $b$ $b$ $b$ $b$ $b$ $b$ $b$ $b$ $b$ $b$

Constant $= \frac{n}{b^F}$

Q. What is $F$ (in terms of $a$, $b$, $d$)?

A) $O(\log_b n)$  B) $O(\log_a n)$  C) $O(n^{\log_b d})$  D) $O(b^{\log_a n})$
Proof of Tree Method

Problem Size $n$

- Level 0: Problem size $n$
- Level 1: $\frac{n}{b}$
- Level 2: $\frac{n}{b^2}$
- Level $F$: $\frac{n}{b^F}$

Q. What is $F$ (in terms of $a$, $b$, $d$)?

A) $O(\log_b n)$  B) $O(\log_d n)$  C) $O(n^{\log_b d})$  D) $O(b^{\log_a n})$

Because at each level, problem size is divided by $b$. $\log_b n$ is the number of times $n$ can be divided by $b$ before reaching a constant.

$C = \frac{n}{b^F}$, so $b^F = \frac{n}{C}$, so $F = \log_b \left(\frac{n}{C}\right)$ constant

$= \log_b n - \log_b C$

$= O(\log_b n)$
Q. What is the work done just at level $K$, not at other levels?

1. $a^k$ subproblems at level $K$.
2. Level $K$ subproblem size: \( \frac{n}{b^k} \)
3. Work outside of recursive call required to solve 1 subproblem

\[ \Rightarrow \text{Total work} \quad a^k \left( \frac{n}{b^k} \right)^d = \left( \frac{a}{b^d} \right)^k n^d \]

Now we add up work done at all levels:

\[
T(n) = n^d \left( \sum_{k=0}^{\log_b n} \left( \frac{a}{b^d} \right)^k \right)
\]

Geometric Series:

\[
\sum_{k=0}^{F} r^k = \begin{cases} 
F+1 & \text{if } r = 1 \\
\frac{1-r^{F+1}}{1-r} & \text{otherwise}
\end{cases}
\]
Geometric Series:

\[
\sum_{k=0}^{F} r^k = \begin{cases} 
F+1 & \text{if } r = 1 \\
\frac{1-r^{F+1}}{1-r} & \text{otherwise}
\end{cases}
\]
PSet:

\[ T(n) = \begin{cases} 
O(n^d \log n) & \text{if } a = b^d \\
O(n^d) & \text{if } a < b^d \\
O(n^\log_b a) & \text{if } a > b^d 
\end{cases} \]

This is usually called "master method" "master theorem"

Master has pretty unpleasant connotations. Also it is not descriptive

My term: "Tree method"

ex: Binary Search:

\[ T(n) = T\left(\frac{n}{2}\right) + O(1) \quad a = 1 \quad b = 2 \quad d = 0 \]

\[ T(1) = O(1) \]

\[ T(n) = O(n^d \log n) = O(n^d \log n) \]

\[ = O(n^d \log n) \]
We'll do one of the 3 cases here:

Case: \( a < b^d \)

\[
T(n) = n^d \left( \sum_{k=0}^{\log_b n} \left( \frac{a}{b^d} \right)^k \right) = n^d \left( \frac{1 - \left( \frac{a}{b^d} \right)^{\log_b n + 1}}{1 - \left( \frac{a}{b^d} \right)} \right)
\]

Constant. Can ignore for big-O

So as \( n \) gets big, \( r^{\log_b n + 1} \to 0 \)

\[
\text{ex: } (\frac{1}{3})^{10} = \frac{1}{3^{10}} \approx 0
\]

\[
1 - r^{\log_b n + 1} \to 1 \quad \text{for large } n.
\]

\( = O(n^d) \)
Big-O only tells you about relative scaling at large input sizes.

Even if you know $C,k$, still don't know anything. Infinitely many $C,k$ work. One pair is not useful.

To know how algorithm will do on specific input, need to know actual time complexity function. Not just big-O or big-$\Theta$, or big-$\Omega$. 