IT501: Algorithms
Lecture 4: Recursion-tree method, Master theorem
Recursion-tree method

- Convert the recurrence into a tree where each node represents the cost of a single subproblem in a recursive call
- Let’s draw the tree for the recurrence $T(n) = 2T(n/2) + n$
Recursion-tree method

- Convert the recurrence into a tree where each node represents the cost of a single subproblem in a recursive call
- Let’s draw the tree for the recurrence \( T(n) = 2T(n/2) + n \)
Definitions

- Height of a node = the number of edges on the longest path from the node down to a leaf
- Level of a node = the length of a path from the root to the node
- Height of tree = height of root node
Definitions

- Height of a node = the number of edges on the longest path from the node down to a leaf
- Level of a node = the length of a path from the root to the node
- Height of tree = height of root node

![Diagram showing height and level of nodes in a tree](image-url)
Example 1

Consider the recursion tree for \( T(n) = 2T(n/2) + 1 \)
Example 1

- Consider the recursion tree for $T(n) = 2T(n/2) + 1$

- We can see that each level of the tree sums to $n$
Example 1

- Consider the recursion tree for $T(n) = 2T(n/2) + 1$

- We can see that each level of the tree sums to $n$

- Further the depth of the tree is $\log n$ ($n/2^d = 1$ implies that $d = \log n$).
Example 1

- Consider the recursion tree for $T(n) = 2T(n/2) + 1$

![Recursion Tree Diagram]

- We can see that each level of the tree sums to $n$
- Further the depth of the tree is $\log n$ ($n/2^d = 1$ implies that $d = \log n$).
- Thus there are $\log n + 1$ levels each of which sums to $n$
Example 1

- Consider the recursion tree for $T(n) = 2T(n/2) + 1$

We can see that each level of the tree sums to $n$

Further the depth of the tree is $\log n$ ($n/2^d = 1$ implies that $d = \log n$).

Thus there are $\log n + 1$ levels each of which sums to $n$

Hence, $T(n) = \Theta(n \log n)$. 
Example 2

- Solve the recurrence: \( T(n) = 3T(n/4) + n^2 \)
- For simplicity, we assume that \( T(i) = \Theta(1) \) for small constants \( i \).
- We start by drawing the recursion tree
Example 2

- We can see that $i$th level of the tree sums to $(3/16)^i n^2$
- Further the depth of the tree is $\log_4 n$ ($n/4^d = 1$ implies that $d = \log_4 n$).
- So we can see that $T(n) = \sum_{i=0}^{\log_4 n} (3/16)^i n^2$
Solution

\[ T(n) = \sum_{i=0}^{\log_4 n} (3/16)^i n^2 \]
Solution

\[ T(n) = \sum_{i=0}^{\log_4 n} (3/16)^i n^2 \]

\[ < n^2 \sum_{i=0}^{\infty} (3/16)^i \]
Solution

\[ T(n) = \sum_{i=0}^{\log_4 n} (3/16)^i n^2 \]

\[ < n^2 \sum_{i=0}^{\infty} (3/16)^i \]

\[ = \frac{1}{1 - (3/16)} n^2 \]
Solution

\[ T(n) = \sum_{i=0}^{\log_4 n} (3/16)^i n^2 \]

\[ < n^2 \sum_{i=0}^{\infty} (3/16)^i \]

\[ = \frac{1}{1 - (3/16)} n^2 \]

\[ = O(n^2) \]
Master Theorem

- Divide and conquer algorithms often give us running-time recurrences of the form
  \[ T(n) = aT(n/b) + f(n) \]
- Where \( a \geq 1 \) and \( b > 1 \) are constants and \( f(n) \) is some other function
- The so-called “Master theorem” gives us a general method for solving such recurrences when \( f(n) \) is a simple polynomial
Unfortunately, the Master Theorem doesn’t work for all functions $f(n)$

Further, many useful recurrences don’t look like $T(n)$

However, the theorem allows for very fast solution of recurrences when it applies
Master Theorem

- Master Theorem is just a special case of the use of the recursion trees
- Consider equation $T(n) = aT(n/b) + f(n)$
- We start by drawing a recursion tree
The Recursion Tree

- The root contains the value $f(n)$
- It has $a$ children, each of which contains the value $f(n/b)$
- Each of these nodes has $a$ children, containing the value $f(n/b^2)$
- In general, level $i$ contains $a^i$ nodes with values $f(n/b^i)$
- Hence the sum of the nodes at the $i$-the level is $a^i f(n/b^i)$
The Recursion Tree

- The tree stops when we get to the base case for the recurrence
- We’ll assume $T(1) = f(1) = \Theta(1)$ is the base case
- Thus the depth of the tree is $\log_b(n)$ and there are $\log_b n + 1$ levels
The Recursion Tree

- We can see that $T(n)$ is the sum of all values stored in all levels of the tree.

\[
T(n) = f(n) + af(n/b) + a^2f(n/b^2) + \cdots + a^i f(n/b^i) + \cdots + a^L f(n/b^L)
\]

- Where $L = \log_b n$ is the depth of the tree

- Since $f(1) = \Theta(1)$, the last term of this summation is $\Theta(a^L) = \Theta(a^{\log_b n}) = \Theta(n^{\log_b a})$
Master Theorem

- We can now state the Master Theorem
- We will state it in a way slightly different from the book
- Note: The Master method is just a “short cut” for the recursion tree method. It is less powerful than recursion trees.
Master Method

The recurrence $T(n) = aT(n/b) + f(n)$ can be solved as follows:

- If $af(n/b) \leq Kf(n)$ for some constant $K < 1$, then $T(n) = \Theta(f(n))$.
- If $af(n/b) \geq Kf(n)$ for some constant $K > 1$, then $T(n) = \Theta(n\log_b a)$
- If $af(n/b) = f(n)$, then $T(n) = \Theta(f(n) \log_b n)$. 
Proof

Case 1: If $f(n)$ is a constant factor smaller than $af(n) = b)$, then the sum is an ascending geometric series. The sum of any geometric series is a constant times its largest term. In this case, this is the last term, which by our earlier argument is $(n \log b a)$.

Case 2: Finally, if $af(n) = f(n)$, then each of the $L+1$ terms in the summation is equal to $f(n)$.

Case 3: If $f(n)$ is a constant factor larger than $af(n) = b)$, then the sum is a descending geometric series. The sum of any geometric series is a constant times its largest term. In this case, the largest term is the first term $f(n)$.
Proof

- **Case 1**: If $f(n)$ is a *constant factor smaller* than $af(n/b)$, then the sum is an ascending geometric series. The sum of any geometric series is a constant times its largest term. In this case, this is the last term, which by our earlier argument is $\Theta(n^{\log_b a})$. 

- **Case 2**: Finally, if $af(n/b) = f(n)$, then each of the $L+1$ terms in the summation is equal to $f(n)$.

- **Case 3**: If $f(n)$ is a *constant factor larger* than $af(n/b)$, then the sum is a descending geometric series. The sum of any geometric series is a constant times its largest term. In this case, the largest term is the first term $f(n)$. 


Proof

- Case 1: If \( f(n) \) is a \textit{constant factor smaller} than \( af(n/b) \), then the sum is an ascending geometric series. The sum of any geometric series is a constant times its largest term. In this case, this is the last term, which by our earlier argument is \( \Theta(n^{\log_b a}) \).

- Case 2: Finally, if \( af(n/b) = f(n) \), then each of the \( L + 1 \) terms in the summation is equal to \( f(n) \).
Case 1: If $f(n)$ is a constant factor smaller than $af(n/b)$, then the sum is an ascending geometric series. The sum of any geometric series is a constant times its largest term. In this case, this is the last term, which by our earlier argument is $\Theta(n^{\log_b a})$.

Case 2: Finally, if $af(n/b) = f(n)$, then each of the $L + 1$ terms in the summation is equal to $f(n)$.

Case 3: If $f(n)$ is a constant factor larger than $af(n/b)$, then the sum is a descending geometric series. The sum of any geometric series is a constant times its largest term. In this case, the largest term is the first term $f(n)$. 

Example

- Solve $T(n) = T(3n/4) + n$
Example

- Solve $T(n) = T(3n/4) + n$
- If we write this as $T(n) = aT(n/b) + f(n)$, then $a = 1, \ b = 4/3, \ f(n) = n$. 
Example

- Solve $T(n) = T(3n/4) + n$
- If we write this as $T(n) = aT(n/b) + f(n)$, then $a = 1$, $b = 4/3$, $f(n) = n$.
- Here, $af(n/b) = 3n/4$ is smaller than $f(n) = n$ by a factor of $4/3$, so $T(n) = \Theta(n)$. 
Example

- Solve $T(n) = 3T(n/2) + n$
Example

- Solve $T(n) = 3T(n/2) + n$
- If we write this as $T(n) = aT(n/b) + f(n)$, then $a = 3$, $b = 2$, $f(n) = n$. 
Example

- Solve $T(n) = 3T(n/2) + n$
- If we write this as $T(n) = aT(n/b) + f(n)$, then $a = 3$, $b = 2$, $f(n) = n$.
- Here, $af(n/b) = 3n/2$ is bigger than $f(n) = n$ by a factor of $3/2$, so $T(n) = \Theta(n^{\log_2 3})$. 

Example

- Solve $T(n) = 2T(n/2) + n$
Example

- Solve $T(n) = 2T(n/2) + n$
- If we write this as $T(n) = aT(n/b) + f(n)$, then $a = 2$, $b = 2$, $f(n) = n$. 
Example

- Solve $T(n) = 2T(n/2) + n$
- If we write this as $T(n) = aT(n/b) + f(n)$, then $a = 2, b = 2, f(n) = n$.
- Here, $af(n/b) = f(n)$, so $T(n) = \Theta(n \log n)$. 