Recurrences and Running Time

Recurrences and Running Time

- **An equation or inequality that describes a function in terms of its value on smaller inputs.**

$$T(n) = T(n-1) + n$$

- **Recurrences arise when an algorithm contains recursive calls to itself**

- **What is the actual running time of the algorithm?**

- **Need to solve the recurrence**
  - **Find an explicit formula of the expression**
  - **Bound the recurrence by an expression that involves n**

Example Recurrences

- $T(n) = T(n-1) + n$ $\Theta(n^2)$
  - Recursive algorithm that loops through the input to eliminate one item
- $T(n) = T(n/2) + c$ $\Theta(\lg n)$
  - Recursive algorithm that halves the input in one step
- $T(n) = T(n/2) + n$ $\Theta(n)$
  - Recursive algorithm that halves the input but must examine every item in the input
- $T(n) = 2T(n/2) + 1$ $\Theta(n)$
  - Recursive algorithm that splits the input into 2 halves and does a constant amount of other work

Methods for Solving Recurrences

- **Iteration method**
- **Substitution method**
- **Recursion tree method**
- **Master method**

The Iteration Method

- **Convert the recurrence into a summation and try to bound it using known series**
  - **Iterate the recurrence until the initial condition is reached.**
  - **Use back-substitution to express the recurrence in terms of *n* and the initial (boundary) condition.**

The Iteration Method

$$T(n) = c + T(n/2)$$

$$T(n) = c + T(n/2)$$
$$= c + c + T(n/4)$$
$$= c + c + c + T(n/8)$$

$$T(n/2) = c + T(n/4)$$
$$T(n/4) = c + T(n/8)$$

Assume $n = 2^k$

$$T(n) = c + c \underbrace{+ \ldots + c}_{k \text{ times}} + T(1)$$
$$= c\lg n + T(1)$$
$$= \Theta(\lg n)$$

Iteration Method – Example

$$T(n) = n + 2T(n/2) \qquad \text{Assume: } n = 2^k$$

$$T(n/2) = n/2 + 2T(n/4)$$

$$\begin{aligned}
T(n) &= n + 2T(n/2) \\
&= n + 2(n/2 + 2T(n/4)) \\
&= n + n + 4T(n/4) \\
&= n + n + 4(n/4 + 2T(n/8)) \\
&= n + n + n + 8T(n/8) \\
\ldots \quad &= kn + 2^k T(n/2^k) \\
&= kn + 2^k T(1) \\
&= n\lg n + nT(1) = \Theta(n\lg n)
\end{aligned}$$

The substitution method

1. **Guess a solution**

2. **Use induction to prove that the solution works**

Substitution method

- **Guess a solution**
  - $T(n) = O(g(n))$
  - **Induction goal: apply the definition of the asymptotic notation**
    - $T(n) \leq d\ g(n)$, for some $d > 0$ and $n \geq n_0$ (strong induction)
  - **Induction hypothesis:** $T(k) \leq d\ g(k)$ for all $k < n$
- **Prove the induction goal**
  - **Use the induction hypothesis to find some values of the constants d and n_0 for which the induction goal holds**

Example: Binary Search

$$T(n) = c + T(n/2)$$

- Guess: $T(n) = O(\lg n)$
  - Induction goal: $T(n) \leq d \lg n$, for some d and $n \geq n_0$
  - Induction hypothesis: $T(n/2) \leq d \lg(n/2)$
- Proof of induction goal:

$$T(n) = T(n/2) + c \leq d \lg(n/2) + c$$

$$= d \lg n - d + c \leq d \lg n$$

$$\text{if: } -d + c \leq 0,\ d \geq c$$

The recursion-tree method

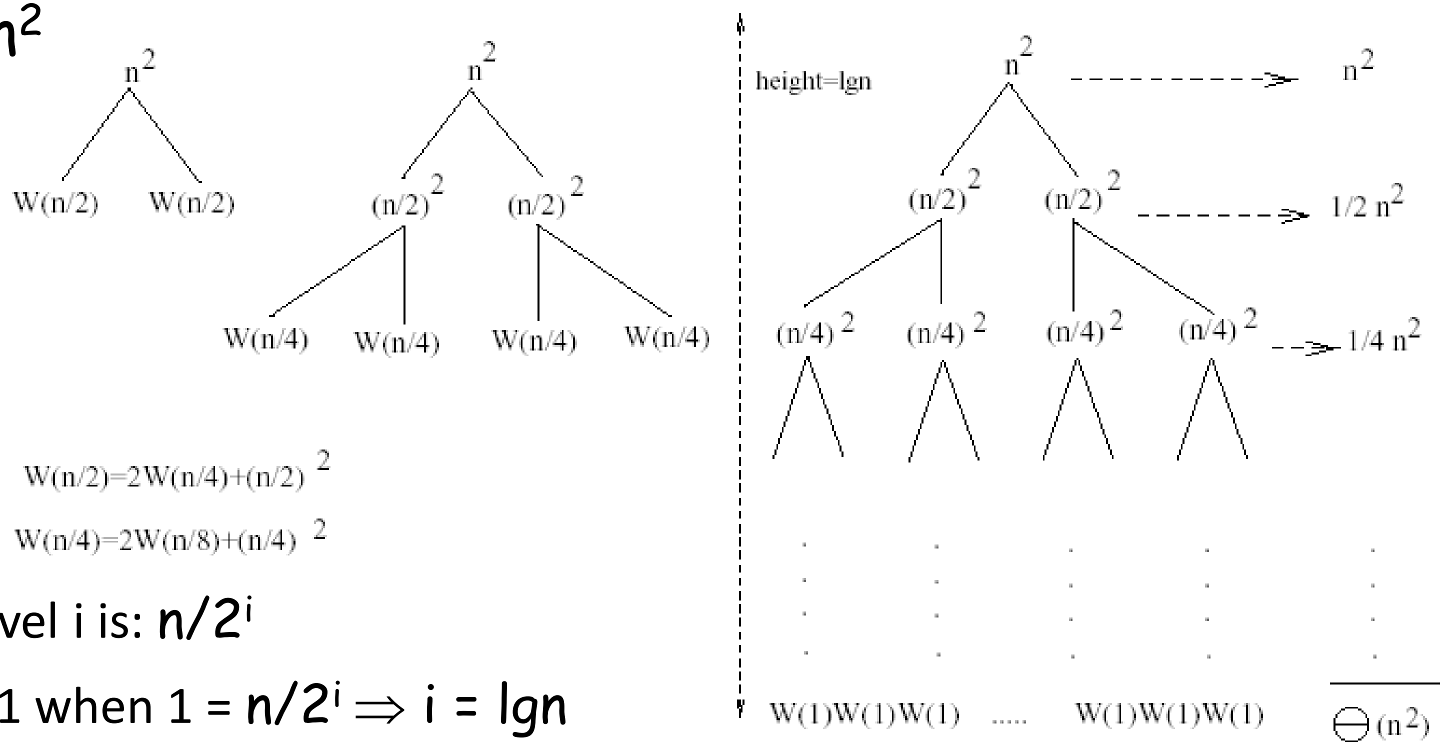
Convert the recurrence into a tree:

- **Each node represents the cost incurred at various levels of recursion**
- **Sum up the costs of all levels**

Used to "guess" a solution for the recurrence

Example 1

$W(n) = 2W(n/2) + n^2$



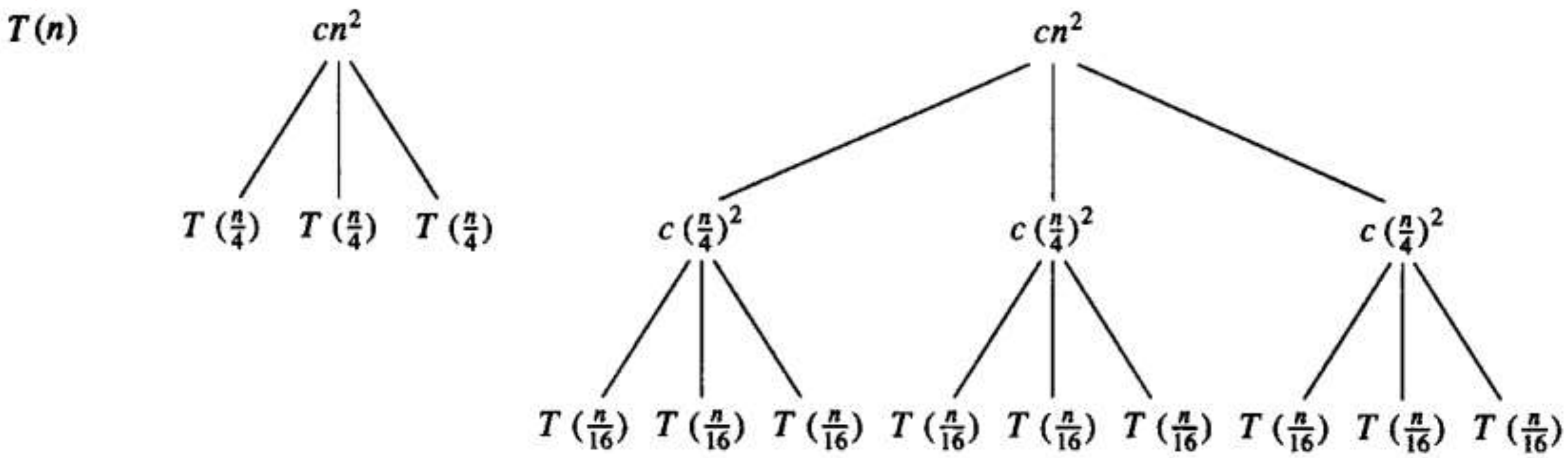
- Subproblem size at level i is: $n/2^i$
- Subproblem size hits 1 when $1 = n/2^i \Rightarrow i = \lg n$
- Cost of the problem at level i = $(n/2^i)^2$ No. of nodes at level $i = 2^i$
- Total cost:

$$W(n) = \sum_{i=0}^{\lg n - 1} \frac{n^2}{2^i} + 2^{\lg n} W(1) = n^2 \sum_{i=0}^{\lg n - 1} \left(\frac{1}{2}\right)^i + n \le n^2 \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i + O(n) = n^2 \frac{1}{1 - \frac{1}{2}} + O(n) = 2n^2$$

$\Rightarrow W(n) = O(n^2)$

Example 2

E.g.: $T(n) = 3T(n/4) + cn^2$



- Subproblem size at level i is: $n/4^i$
- Subproblem size hits 1 when $1 = n/4^i \Rightarrow i = \log_4 n$
- Cost of a node at level i = $c(n/4^i)^2$
- Number of nodes at level i = $3^i \Rightarrow$ last level has $3^{\log_4 n} = n^{\log_4 3}$ nodes
- Total cost:

$$T(n) = \sum_{i=0}^{\log_4 n-1}\left(\frac{3}{16}\right)^i cn^2 + \Theta\left(n^{\log_4 3}\right) \le \sum_{i=0}^{\infty}\left(\frac{3}{16}\right)^i cn^2 + \Theta\left(n^{\log_4 3}\right) = \frac{1}{1-\frac{3}{16}} cn^2 + \Theta\left(n^{\log_4 3}\right) = O(n^2)$$

$\Rightarrow T(n) = O(n^2)$

Example 2 - Substitution

$$T(n) = 3T(n/4) + cn^2$$

- Guess: $T(n) = O(n^2)$
  - Induction goal: $T(n) \le dn^2$, for some d and $n \ge n_0$
  - Induction hypothesis: $T(n/4) \le d\,(n/4)^2$
- Proof of induction goal:

$$\begin{aligned} T(n) &= 3T(n/4) + cn^2 \\ &\le 3d\,(n/4)^2 + cn^2 \\ &= (3/16)\,d\,n^2 + cn^2 \\ &\le d\,n^2 \qquad \text{if: } d \ge (16/13)c \end{aligned}$$

- Therefore: $T(n) = O(n^2)$

Master's method

- "Cookbook" for solving recurrences of the form:.

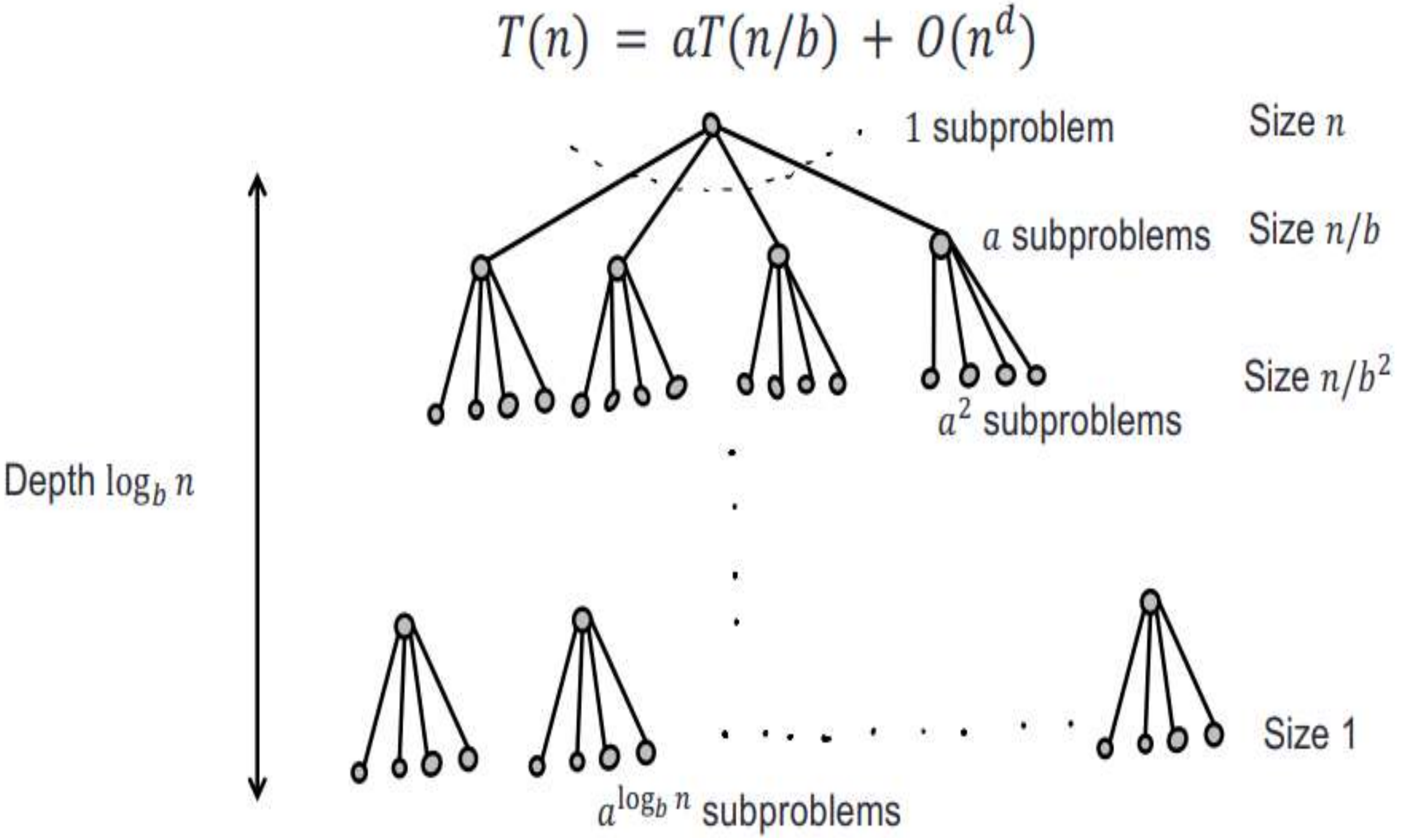
$$T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$$

for some constants, a ≥ 1, b > 1, and $d \geq 0$

- Then

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

Master Theorem: Solving the recurrence

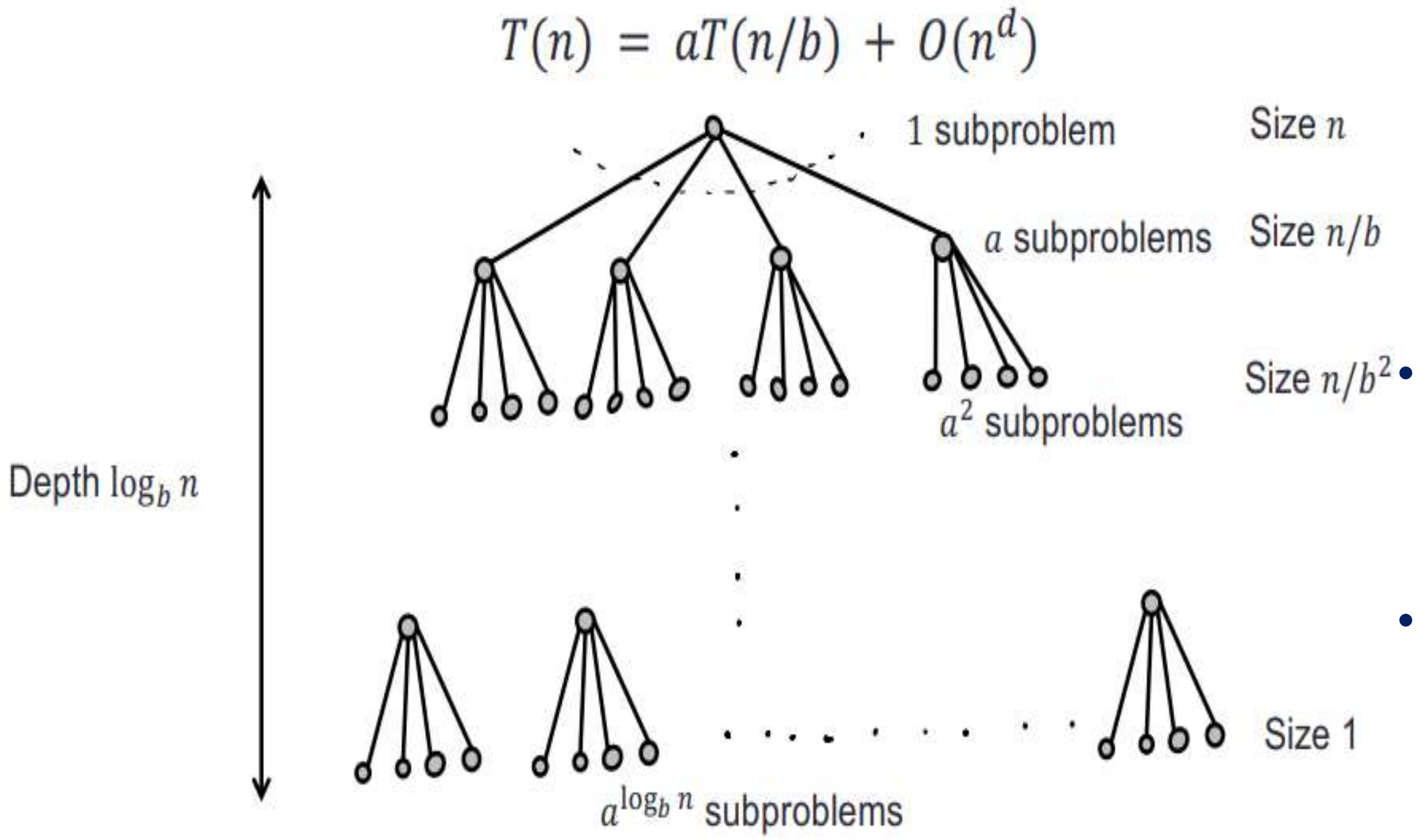


- **After k levels, there are a^k subproblems, each of size $\frac{n}{b^k}$**
- **So, during the kth level of recursion, the time complexity is**

$$O\left(\left(\frac{n}{b^k}\right)^d\right)a^k = \left(a^k\left(\frac{n}{b^k}\right)^d\right)$$

$$= O\left(n^d\left(\frac{a}{b^d}\right)^k\right)$$

Master Theorem: Solving the recurrence



- **So, during the kth level of recursion, the time complexity is**

$$O\left(\left(\frac{n}{b^k}\right)^d\right)a^k = \left(a^k\left(\frac{n}{b^k}\right)^d\right) = O\left(n^d\left(\frac{a}{b^d}\right)^k\right)$$

- **After $\log_b a$ levels, the subproblem size is reduced to 1, which usually is the size of the base case.**

- **So, the entire algorithm is a sum of each level.**

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

Master Theorem: Solving the recurrence

- **So, the entire algorithm is a sum of each level.**

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

- **Case 1: $a < b^d$**
  - **Then we have that $\frac{a}{b^d} < 1$ and the series converges to a constant so**

$$T(n) = O(n^d)$$

Master Theorem: Solving the recurrence

- **So, the entire algorithm is a sum of each level.**

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

- **Case 1: $a = b^d$**
  - **Then we have that $\frac{a}{b^d} = 1$ and so each term is equal to 1 so**

$$T(n) = O(n^d \log_b n)$$

Master Theorem: Solving the recurrence

- **So, the entire algorithm is a sum of each level.**

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

- **Case 1: $a > b^d$**
  - **Then the summation is exponential and grows proportional to its last term $\left(\frac{a}{b^d}\right)^{\log_b n}$ so**

$$T(n) = O\left(n^d \left(\frac{a}{b^d}\right)^{\log_b n}\right) = O\left(n^{\log_b a}\right)$$

Examples

- $T(n) = 2T(n/2) + n$
- $T(n) = 2T(n/2) + n^2$

Master Theorem

- **You cannot use the Master Theorem if**
  - **T(n) is not monotone, ex:** $T(n) = \sin \boldsymbol{n}$
  - **f(n) is not a polynomial, ex:** $T(n) = 2T(n/2) + \mathbf{2}^{\boldsymbol{n}}$
  - **b cannot be expressed as a constant, ex: T(n) = T(√ n)**

References

- **Data Structure and Algorithms for Electrical Engineering by Yung Yi**