CSC 344 – Algorithms and Complexity

Lecture #10 – Recurrences

Recurrence Relations

- Overview
  - Connection to recursive algorithms
- Techniques for solving them
  - Methods for generating a guess
  - Induction proofs
  - Master Theorem

Recursion *and* Mathematical Induction

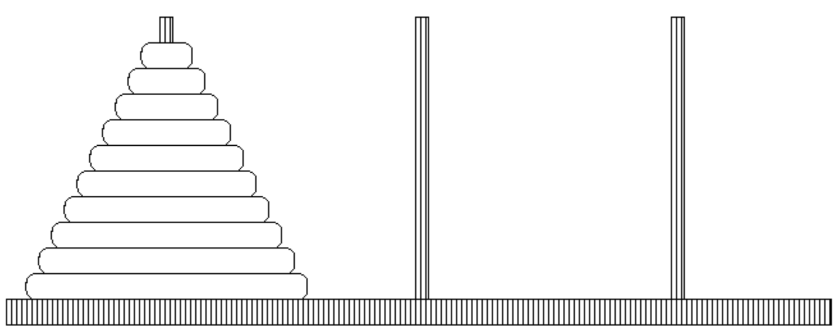
In both, we have general and boundary conditions:

The **general** conditions break the problem into smaller and smaller pieces.

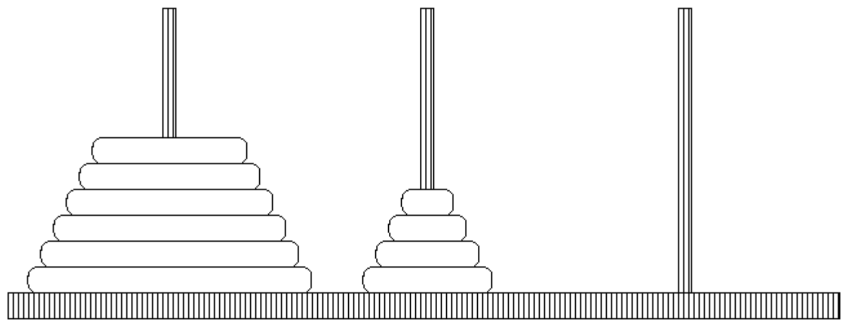
The **initial** or **boundary** condition(s) terminate the recursion.

Both take a **Divide and Conquer** approach to solving mathematical problems.

The Towers of Hanoi

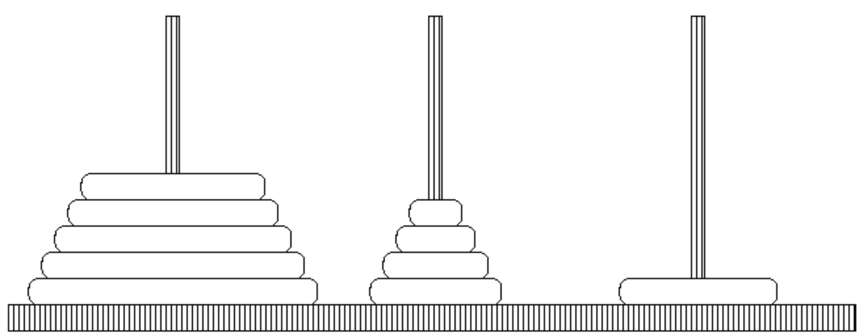


What if we knew we could solve *part* of the problem?

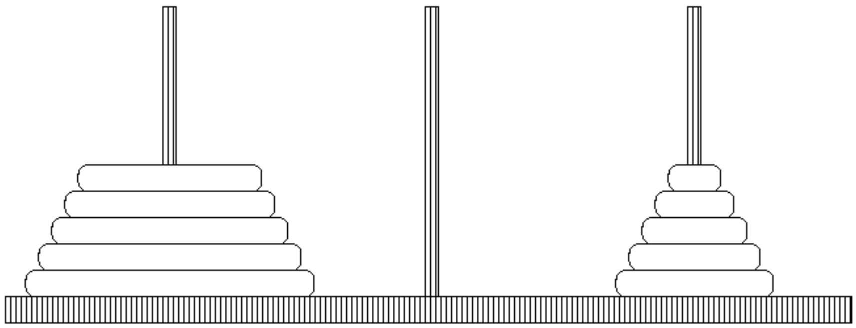


Assume we can move k (in this case, 4) different rings

Can we do one better?



Solved for one more!



Where do recurrence relations come from?

- Analysis of a divide and conquer algorithm
  - Towers of Hanoi, Merge Sort, Binary Search
- Analysis of a combinatorial object
- **This is the key analysis step I want you to master**
- Use small cases to check correctness of your recurrence relation

Recurrence Relations

- Overview
  - Connection to recursive algorithms
- Techniques for solving them
  - Methods for generating a guess
  - Induction proofs
  - Master Theorem

Solving Recurrence Relations

- No general, automatic procedure for solving recurrence relations is known.
- There are methods for solving specific forms of recurrences

Some Solution Techniques

- **Guess** a solution and prove by induction.
  - Extrapolate from small values
  - Try back-substituting
  - Draw a recursion tree
- Master Theorem
  - Quick solutions for many simple recurrences

Extrapolate from small values

Example: $T_n = 2T_{n-1} + 1$; $T_0 = 0$

n = 0 1 2 3 4 5 6 7

T_n=

- Guess:

Back-substitution or Unrolling

$T_n = 2T_{n-1} + 1$; $T_0 = 0$

$T_n = 2(2T_{n-2} + 1) + 1$

$= 4T_{n-2} + 2 + 1$

$T_n = 4(2T_{n-3} + 1) + 2 + 1$

$= 8T_{n-3} + 4 + 2 + 1$

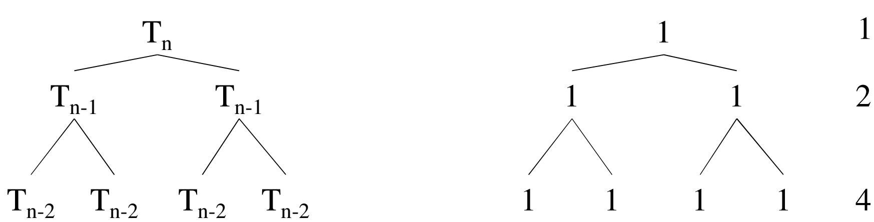
$T_n = 8(2T_{n-4} + 1) + 4 + 2 + 1$

$= 16T_{n-4} + 8 + 4 + 2 + 1$

Guess:

Recursion Trees

$T_n = 2\ T_{n-1} + 1\ ,\ T_0 = 0$



Guess:

Extrapolate from small values

Example: $T(n) = 3T(\lfloor n/4 \rfloor) + n$, $T(0) = 0$

```
  n = 0 1 2 3 4 5 6 7 8 9 10 11 12 13
T(n)=

  n = 0 1 4 16 64 256 1024
T(n)=
```

Guess:

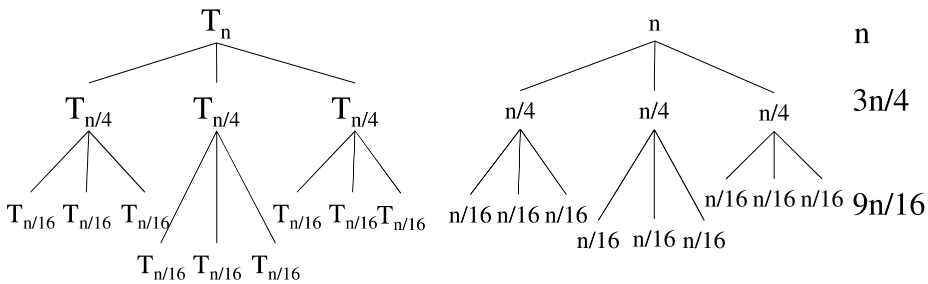
Back-substitution or unrolling

$$
\begin{aligned}
T(n) &= 3T(\lfloor n/4 \rfloor) + n, \quad T(0) = 0 \\
&\le 3(3T(n/16) + n/4) + n \\
&= 9T(n/16) + 3n/4 + n \\
&= 9(3T(n/64) + n/16) + 3n/4 + n \\
&= 27T(n/64) + 9n/16 + 3n/4 + n
\end{aligned}
$$

Guess:

Recursion Trees

$T_n = 3T_{\lfloor n/4 \rfloor} + n, \; T_0 = 0$



Guess:

Third Example

Example: $T_n = 2\,T_{n/2} + n^2, \; T_0 = 0$

- Generate a potential solution using the 3 different techniques

Extrapolate from small values

Example: $T_n = 2\,T_{n/2} + n^2$, $T_0 = 0$

```
  n = 0 1 2  4   8  16   32
T(n)= 0 1 6 28 120 496 2016

  n =   64   128    256
T(n)= 8128 32640 130816
```

Guess: ??

Extrapolate from small values

n	T_n	T_n (factored)	
0	0	0·0	
1	1	1·1	
2	6	3·2	
4	28	7·4	
8	120	15·8	
16	496	31·16	
32	2016	63·32	
64	8128	127·64	
128	32640	255·128	

Guess: $(2n-1)n$

Third Example from Substitution

- $T_1 = 2T_0 + 1^2 = 2(0) + 1 = 1 = 1 \cdot 1$
- $T_2 = 2(T_1) + 2^2 = 2(1) + 4 = 6 = 3 \cdot 2$
- $T_4 = 2(6) + 4^2 = 12 + 16 = 28 = 7 \cdot 4$
- $T_8 = 2(T_4) + 8^2 = 2(28) + 64 = 120 = 15 \cdot 8$
- $T_{16} = 2(T_8) + 16^2 = 2(120) + 256$
  $= 496 = 31 \cdot 16$
- $T_{32} = 2(T_{16}) + 32^2 = 2(496) + 1024$
  $= 2016 = 63 \cdot 32$
- $T_{64} = 2(T_{32}) + 64^2 = 2(2016) + 4096$
  $= 8128 = 127 \cdot 64$

Example: D&C into variable sized pieces

$T(n) = T(n/3) + T(2n/3) + n$
$T(1) = 1$

Generate a potential solution for T(n) using the methods we have discussed.

Recurrence Relations

- Overview
  - Connection to recursive algorithms
- Techniques for solving them
  - Methods for generating a guess
  - Induction proofs
  - Master Theorem

Induction Proof

$T_n = 2T_{n-1} + 1$; $T_0 = 0$

Prove: $T_n = 2^n - 1$ by induction:
1. Base Case: n=0: $T_0 = 2^0 - 1 = 0$
2. Inductive Hypothesis (IH): $T_n = 2^n - 1$ for $n \geq 0$
3. Inductive Step: Show $T_{n+1} = 2^{n+1} - 1$ for $n \geq 0$

$$
\begin{aligned}
T_{n+1} &= 2T_n + 1 \\
&= 2\,(2^n - 1) + 1 \quad \text{(applying IH)} \\
&= 2^{n+1} - 1
\end{aligned}
$$

Merge sort analysis

```
Mergesort(array)
        n = size(array)
        if ( n = = 1) return array
        array1 = Mergesort(array[1 .. n/2])
        array2 = Mergesort(array[n/2 + 1 .. n])
        return Merge(array1, array2)
```

Develop a recurrence relation:

Problem: Merge Sort

- Merge sort breaking array into 3 pieces
  - What is a recurrence relation?
  - What is a solution?
  - How does this compare to breaking into 2 pieces?

Merge Sort Induction Proof

Prove that $T(n) = 2T(\lfloor n/2 \rfloor) + n$, $T(1) = 1$ is $O(n \lg n)$.

Prove that $T(n) \leq c\, n \lg n$, for all n greater than some value.

Base cases: why not T(1)?

$T(2) = 4 \leq c\ 2 \lg 2$

$T(3) = 5 \leq c\ 3 \lg 3$

$c \geq 2$ suffices

Inductive Hypothesis: $T(\lfloor n/2 \rfloor) \leq c\, (\lfloor n/2 \rfloor) \lg (\lfloor n/2 \rfloor)$ for $n \geq$?

Inductive Step: Show that $T(n) \leq c\, n \lg n$ for $n \geq$?

Induction Step

Given : $T(\lfloor n/2 \rfloor) \leq c\, (\lfloor n/2 \rfloor) \lg (\lfloor n/2 \rfloor)$

$$\begin{aligned}
T(n) &= 2T(\lfloor n/2 \rfloor) + n \\
&\leq 2(\ c(\lfloor n/2 \rfloor) \log (\lfloor n/2 \rfloor)\ ) + n \quad \text{(applying IH)} \\
&\leq 2(\ c(n/2) \log (n/2)\ ) + n \quad \text{(dropping floors makes it bigger!)} \\
&= c\, n \lg(n/2) + n \\
&= c\, n\, (\ \lg(n) - \lg(2)\ ) + n \\
&= c\, n \lg(n) - c\, n + n \quad (\lg 2 = 1) \\
&= c\, n \lg(n) - (c - 1)\, n \\
&< c\, n \lg(n) \quad (c > 1)
\end{aligned}$$

Recurrence Relations

- Overview
  - Connection to recursive algorithms
- Techniques for solving them
  - Methods for generating a guess
  - Induction proofs
  - Master Theorem

Master Theorem

- $T(n) = a\,T(n/b) + f(n)$
  - Ignore floors and ceilings for n/b
  - constants $a \geq 1$ and $b > 1$
  - f(n) any function
- If $f(n) = O(n^{\log\_b\, a-\varepsilon})$ for constant $\varepsilon>0$, $T(n) = \Theta(n^{\log\_b\, a})$
- If $f(n) = \Theta(n^{\log\_b\, a})$, $T(n) = \Theta(n^{\log\_b\, a} \lg n)$
- If $f(n) = \Omega(n^{\log\_b\, a+\varepsilon})$ for some constant $\varepsilon > 0$, and if $a\,f(n/b) \leq c\,f(n)$ for some constant $c < 1$ and all sufficiently large n, $T(n) = \Theta(f(n))$.
- Key idea: Compare $n^{\log\_b\, a}$ with f(n)

Master Theorem

- The master theorem concerns recurrence relations of the form:
  - $T(n) = aT(n/b) + f(n)$, where $a \geq 1$, $b > 1$
  - In the application to the analysis of a recursive algorithm, the constants and function take on the following significance:
    - n is the size of the problem.
    - a is the number of subproblems in the recursion.
    - n/b is the size of each subproblem. (Here it is assumed that all subproblems are essentially the same size.)
    - f (n) is the cost of the work done outside the recursive calls, which includes the cost of dividing the problem and the cost of merging the solutions to the subproblems.

$f(n) \in \Theta\left(n^{\log_b a}\right)$

Applying Master Theorem

- Case 1 – Generic form

  If $f(n) \in O(n^c)$, where $c < log_b\, n$

  $T(n) \in \Theta(n^{log_b a})$

Master Theorem Case 1 - Example

- $T(n) = 8T(n/2) + 1000n^2$

  $a = 8$, $b = 2$, $f(n) = 1000n^2$
- so

  $T(n) \in \Theta(n^c)$, where $c = 2$
- Do we satisfy the condition of case 1?

  $log_b a = log_2 8 = 3 > c$? We do!!
- $T(n) \in \Theta(n^{log_b a}) = \Theta(n^3)$
- We find that $T(n) = 1001n^3 - 1000n^2$, if $T(1) = 1$

Applying Master Theorem

- Case 2 - Generic Form
- If for $k \geq 0$:

  $f(n) \in \Theta(n^c log^k n)$ where $c = log_b a$

  then

  $T(n) \in \Theta(n^c \, log^{k+1} n)$
- $T(n) \in \Theta(n^{log_b a} \, log^{k+1} n) = \Theta(n^1 log^1 n) = \Theta(n \log n)$

Master Theorem Case 2 - Example

- $T(n) = 2T(n/2) + 10n$
- We find that

  $a = 2$, $b = 2$, $c = 1$, $f(n) = 10n$

  so

  $f(n) = \Theta(n^c \log^k n)$, where $c = 1$, $k = 0$
- Do we satisfy the Case 2 condition?

  $\log_b a = \log_2 2 = 1$, and therefore $c = \log_b a$ Yes!!
- $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$

  $= \Theta(n^1 \log^1 n) = \Theta(n \log n)$

Master Theorem Case 2 - Example

- $T(n)$ is in $\Theta(n \log n)$

  and we know that $T(1) = 1$, therefore
- $T(n) = n + 10n \log_2 n$

Applying Master Theorem

- Case 3 - Generic Form
- If $f(n) \in \Omega(n^c)$ where $c > log_b a$

  and if

  $af(n/b) \leq k\, f(n)$

  where $k < 1$ and n is sufficiently large

  (called the ***regularity condition***)

  then

  $T(n) \in \Theta(f(n))$

Master Theorem Case 3 - Example

- $T(n) = 2T(n/2) + n^2$
- We find that

  $a = 2,\ b = 2,\ f(n) = n^2$

  so

  $f(\text{n}) = \Omega(n^c)$, where $c = 2$
- Do we satisfy the Case 3 condition?

  $\log_b a = log_2 2 = 1$, and therefore $c > log_b a$ Yes!!
- The regularity condition is met:

  $2(n^2/4)) \leq k\, n^2$

Master Theorem Case 2 - Example

- $T(n) = \Theta(f(n)) = \Theta(n^2)$
  and we know that $T(1) = 1$, therefore
- $T(n) = 2n^2 - n$

Inadmissible Equations

- $T(n) = 2^n T(n/2) + n^n$
  - a is not constant
- $T(n) = 2T(n/2) + n/\log n$
  - $f(n)/n^{\log_b a}$ must be a polynomial
- $T(n) = 0.5T(n/2) + n$
  - a cannot be less than one
- $T(n) = 64T(n/8) - n^2 \log n$
  - *$f(n)$ must be positive*
- $T(n) = T(n/2) + n(2 - \cos n)$
  - *It's case 3 but there is a regularity violation*