CS 341: ALGORITHMS

Lecture 4: divide & conquer I
Readings: see website

Trevor Brown (co-taught with Anna Lubiw)

https://www.student.cs.uwaterloo.ca/~cs341
trevor.brown@uwaterloo.ca
ONE DOES NOT SIMPLY UNDERSTAND RECURSION WITHOUT UNDERSTANDING RECURSION

DIVIDE AND CONQUER

Notable algorithms: mergesort, quicksort, binary search, ...
DIVIDE-AND-CONQUER DESIGN STRATEGY

- **divide**: Given a problem instance \( I \), construct one or more smaller problem instances \( I_1, \ldots, I_a \)
  - These are called **subproblems**
  - Usually, want subproblems to be small compared to the size of \( I \) (e.g., half the size)
- **conquer**: For \( 1 \leq j \leq a \), solve instance \( I_j \) **recursively**, obtaining solutions \( S_1, \ldots, S_a \)
- **combine**: Given solutions \( S_1, \ldots, S_a \), use an appropriate combining function to find the solution \( S \) to the problem instance \( I \)
  - i.e., \( S = \text{Combine}(S_1, \ldots, S_a) \).
D&C PROTO-ALGORITHM

1. DnC template(I)
2.   if BaseCase(I) return Result(I)
3.   subproblems = [I_1, I_2, ..., I_a]
4.   subsolutions = []
5.   for j = 1..a
6.     subsolutions[j] = DnC_template(I_j)
7.   return Combine(subsolutions)
CORRECTNESS

```python
DnC_template(I):
    if BaseCase(I) return Result(I)
    subproblems = [I_1, I_2, ..., I_a]
    subsolutions = []
    for j = 1..a
        subsolutions[j] = DnC_template(I_j)
    return Combine(subsolutions)
```

• Prove base cases are correct
• Inductively assume subproblems are solved correctly
• Show they are correctly assembled into a solution
RUNTIME/SPACE COMPLEXITY?

1. DnC_template(I)
   2. if BaseCase(I) return Result(I)
   3. subproblems = [I_1, I_2, ..., I_a]
   4. subsolutions = []
   5. for j = 1..a
      6.     subsolutions[j] = DnC_template(I_j)
   7. return Combine(subsolutions)

- Techniques covered in this lecture
  - Model complexities using recurrence relations
  - Solve with substitution, master theorem, etc.
WORKED EXAMPLE: DESIGN OF MERGESORT

Here, a problem instance consists of an array $A$ of $n$ integers, which we want to sort in increasing order. The size of the problem instance is $n$.

divide: Split $A$ into two subarrays: $A_L$ consists of the first $\left\lfloor \frac{n}{2} \right\rfloor$ elements in $A$ and $A_R$ consists of the last $\left\lfloor \frac{n}{2} \right\rfloor$ elements in $A$.

conquer: Run Mergesort on $A_L$ and $A_R$.

combine: After $A_L$ and $A_R$ have been sorted, use a function $\text{Merge}$ to merge $A_L$ and $A_R$ into a single sorted array. Recall that this can be done in time $\Theta(n)$ with a single pass through $A_L$ and $A_R$. We simply keep track of the “current” element of $A_L$ and $A_R$, always copying the smaller one into the sorted array.
DIVIDE
MERGE: CONQUER AND COMBINE
MERGE SIMULATION

L
4 10 96 98
↑↑↑↑
R
5 12 21 31
↑↑↑↑

O
4 5 10 12 21 31 96 98
Mergesort(A[1..n])

if n == 1 then return A

nL = ceil(n/2)
aL = A[1..nL]
aR = A[(nL+1)..n]
sL = Mergesort(aL)
sR = Mergesort(aR)

return Merge(sL, sR)
PSEUDOCODE FOR MERGE

Merge(aL[1..nL], aR[1..nR])

aOut[1..(nL+nR)] = empty array
il = 1 ; iR = 1 ; iOut = 1

while il < nL and iR < nR
  if aL[il] < aR[iR]
    aOut[iOut] = aL[il]
    il++ ; iOut++
  else
    aOut[iOut] = aR[iR]
    iR++ ; iOut++

while il < nL
  aOut[iOut] = aL[il]
  il++ ; iOut++

while iR < nR
  aOut[iOut] = aR[iR]
  iR++ ; iOut++

return aOut
So, MergeSort(A) takes $O(n)$ time, plus the time for its two recursive calls!

How can we analyze this recursive program structure?
Hulk(n) = Face - Chin + Hulk(n - 1)

RECURRANCE RELATIONS
A crucial analysis tool for recursive algorithms
Suppose $a_1, a_2, \ldots$, is an infinite sequence of real numbers.

A recurrence relation is a formula that expresses a general term $a_n$ in terms of one or more previous terms $a_1, \ldots, a_{n-1}$.

A recurrence relation will also specify one or more initial values starting at $a_1$.

Solving a recurrence relation means finding a formula for $a_n$ that does not involve any previous terms $a_1, \ldots, a_{n-1}$.

There are many methods of solving recurrence relations. Two important methods are guess-and-check and the recursion tree method.
Let $T(n)$ denote the time to run Mergesort on an array of length $n$.

- **divide** takes time $\Theta(n)$
- **conquer** takes time $T\left(\lceil \frac{n}{2}\rceil\right) + T\left(\lfloor \frac{n}{2}\rfloor\right)$
- **combine** takes time $\Theta(n)$

Recurrence relation:

$$T(n) = \begin{cases} 
T\left(\lceil \frac{n}{2}\rceil\right) + T\left(\lfloor \frac{n}{2}\rfloor\right) + \Theta(n) & \text{if } n > 1 \\
\Theta(1) & \text{if } n = 1.
\end{cases}$$

$T(n)$ is a function of $T(\ldots)$ so $T$ is a recurrence relation.

How can we compute/solve for $T(n)$?

To make this easier, assume $n = 2^k$, which lets us ignore floors/ceilings.
If pants wore pants, would it wear them like this? or like this?

Compare vs:

\[ T(n) \]
\[ T(n - 1) \]
\[ T(n - 2) \]
... 

Recursion tree

\[ T(n) \]
\[ T(n/2) \]
\[ T(n/2) \]
... 
\[ T(n/4) \]
\[ T(n/4) \]
... 
\[ T(n/8) \]
\[ T(n/8) \]

**Recursion Tree Method**

Evaluating recurrences with \( T(n/c) \) terms
**RECURSION TREE METHOD**

```
msort(n)  \rightarrow  cn = cn
msort(n/2)  \rightarrow  2(cn/2) = cn
msort(n/4)  \rightarrow  4(cn/4) = cn
msort(1)  \rightarrow  n(c) = cn
```

<table>
<thead>
<tr>
<th>Level</th>
<th># of nodes</th>
<th>runtime per node</th>
<th>total runtime for level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>( cn )</td>
<td>( cn )</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>( c(n/2) )</td>
<td>( 2c(n/2) = cn )</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>( c(n/4) )</td>
<td>( 4c(n/4) = cn )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>( \log n )</td>
<td>( n )</td>
<td>( c(n/n) = c )</td>
<td>( nc(n/n) = cn )</td>
</tr>
</tbody>
</table>

Total = \( cn \times \# \text{levels} \)

Total = \( cn \log_2(n) \)

So, mergesort has runtime \( O(n \log n) \)

Can also compute using a table...
Sample recurrence for two recursive calls on problem size \( n/2 \)

\[
T(n) = \begin{cases}
2T\left(\frac{n}{2}\right) + cn & \text{if } n > 1 \text{ is a power of } 2 \\
d & \text{if } n = 1,
\end{cases}
\]

where \( c \) and \( d \) are constants.

We can solve this recurrence relation when \( n \) is a power of two, by constructing a recursion tree, as follows:

- **Step 1**: Start with a one-node tree, say \( N \), having the value \( T(n) \).
- **Step 2**: Grow two children of \( N \). These children, say \( N_1 \) and \( N_2 \), have the value \( T(n/2) \), and the value of \( N \) is replaced by \( cn \).
- **Step 3**: Repeat this process recursively, terminating when a node receives the value \( T(1) = d \).
- **Step 4**: Sum the values on each level of the tree, and then compute the sum of all these sums; the result is \( T(n) \).
GUESS-AND-CHECK METHOD

- Suppose we have the following recurrence:
  \[ T(0) = 4 ; \quad T(n) = T(n - 1) + 6n - 5 \]
- **Guess** the form of the solution **any** way you like
- My approach: the substitution method
  - Recursively substitute the formula into itself
  - Try to identify patterns to **guess** the final closed form
- **Prove** that the guess was correct
Recurrence: \( T(0) = 4; \quad T(n) = T(n-1) + 6n - 5 \)

- \( T(n) = (T(n-2) + 6(n-1) - 5) + 6n - 5 \) (substitute)
- \( = T(n-2) + 6n - 6 - 5 + 6n - 5 \)
- \( = T(n-2) + 2(6n - 5) - 6 \)
- \( = (T(n-3) + 6(n-2) - 5) + 2(6n - 5) - 6 \) (substitute)
- \( = T(n-3) + 6n - 2(6) - 5 + 2(6n - 5) - 6 \)
- \( = T(n-3) + 3(6n - 5) - 6(1 + 2) \)

... identify patterns and **guess** what happens in the limit

\[ = T(0) + n(6n - 5) - 6(1 + 2 + 3 + \cdots + (n - 1)) = \text{guess}(n) \]
• $\text{guess}(n) = T(0) + n(6n - 5) - 6(1 + 2 + 3 + \cdots + (n - 1))$

• $= 4 + 6n^2 - 5n - 6n(n - 1)/2$ \quad \text{(simplify)}

• $= 3n^2 - 2n + 4$

• Are we done?

• The form of $\text{guess}(n)$ was an \textit{educated guess}.

• To be formal, we must \textit{prove} it correct using \textit{induction}.
• Recall: \( T(0) = 4 \); \( T(n) = T(n-1) + 6n - 5 \); \( guess(n) = 3n^2 - 2n + 4 \)

• Want to prove: \( guess(n) = T(n) \) for all \( n \)

• Base case: \( guess(0) = 3(0)^2 - 2(0) + 4 = T(0) \)

• Inductive case: suppose \( guess(n) = T(n) \) for \( n \geq 0 \), show \( guess(n + 1) = T(n + 1) \).

\[
T(n + 1) = T(n) + 6(n + 1) - 5 \quad \text{(by definition)}
\]

\[
= guess(n) + 6(n + 1) - 5 \quad \text{(by inductive hypothesis)}
\]

\[
= 3n^2 + 4n + 5 \quad \text{(substitute & simplify)}
\]

\[
guess(n + 1) = 3(n + 1)^2 - 2(n + 1) + 4 \quad \text{(by definition)}
\]

\[
= 3n^2 + 4n + 5 = T(n + 1) \quad \text{(simplify)}
\]
ANOTHER APPROACH

• Suppose you look for a while at the previous recurrence:
  • \( T(0) = 4; T(n) = T(n - 1) + 6n - 5 \)
• With some experience, you might just guess it’s quadratic
• If you’re right, it should have the form:
  • \( an^2 + bn + c \) for some unknown constants \( a, b, c \)
• So, just carry the unknown constants into the proof!
  • You can then determine what the constants must be for the proof to work out
• $T(0) = 4 \ ; T(n) = T(n - 1) + 6n - 5 \ ; \text{guess}(n) = an^2 + bn + c$

• Want to prove: $\text{guess}(n) = T(n)$ for all $n$

• Base case: $\text{guess}(0) = a(0)^2 + b(0) + c = T(0) = 4$

  this holds if $c = 4$ \quad ($a, b$ are not constrained)

• Inductive case: suppose $\text{guess}(n) = T(n)$ for $n \geq 0$, show $\text{guess}(n + 1) = T(n + 1)$.

  $T(n + 1) = T(n) + 6(n + 1) - 5$ \quad (by definition)

  $\quad = \text{guess}(n) + 6(n + 1) - 5$ \quad (by inductive hypothesis)

  $\quad = an^2 + bn + 4 + 6(n + 1) - 5$ \quad (substitute)

  $\quad = an^2 + (b + 6)n + 5$ \quad (simplify)
Recall: $\text{guess}(n) = an^2 + bn + c$ where $c = 4$

Inductive case: suppose $\text{guess}(n) = T(n)$ for $n \geq 0$, show $\text{guess}(n + 1) = T(n + 1)$.

$T(n + 1) = an^2 + (b + 6)n + 5$ (continue previous slide)

$\text{guess}(n + 1) = a(n + 1)^2 + b(n + 1) + 4$ (by definition)

$= a(n^2 + 2n + 1) + bn + b + 4$ (simplify, and...)

$= an^2 + (2a + b)n + (a + b + 4)$ (rearrange polynomial)

We want this to be equal to $T(n + 1)$

$an^2 + (2a + b)n + (a + b + 4) = an^2 + (b + 6)n + 5$

equivalent to $(2a + b) = (b + 6)$ and $(a + b + 4) = 5$

first implies $a = 3$ plug $a$ into second to get $b = 5 - 4 - 3 = -2$

So, inductive hypothesis is correct for $a = 3, b = -2, c = 4$
MASTER THEOREM FOR RECURRENCES

- Provides a formula for solving many recurrence relations
- We start with a simplified version
- Consider recurrence:  \( T(1) = d \);  \( T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^y) \)

  where \( a \geq 1, b \geq 2 \) and \( n \) is a power of \( b \) (i.e., \( n = b^j \) for integer \( j \))

Example corresponding algorithm

```plaintext
2  if BaseCase(I) return Result(I)
3  
4  subsolutions = []
5  for j = 1..a
6      let s = subproblem of size n/b
7      subsolutions[j] = DnC_algo(s)
8  
9  solution = combine in \( n^y \) time
10  return solution
```
**MASTER THEOREM FOR RECURRENCES**

\[ T(1) = d ; \ T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^y) \text{ where } a \geq 1, \ b \geq 2 \text{ and } n = b^j \]

<table>
<thead>
<tr>
<th>1 node</th>
<th>Problem size ( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a ) nodes</td>
<td>problem size ( \frac{n}{b} )</td>
</tr>
<tr>
<td>( a^2 ) nodes</td>
<td>Problem size ( \frac{n}{b^2} )</td>
</tr>
<tr>
<td>( a^j ) nodes</td>
<td>prob size ( \frac{n}{b^j} = 1 )</td>
</tr>
</tbody>
</table>

Sum over all levels we get

\[ T(n) = da^j + \sum_{i=0}^{j-1} ca^i \left(\frac{n}{b^i}\right)^y \]

Let’s rearrange this into a **geometric sequence** and solve
REARRANGING

\[ T(n) = da^j + \sum_{i=0}^{j-1} ca^i \left( \frac{n}{b^i} \right)^y \]

\[ = da^j + \sum_{i=0}^{j-1} ca^i \frac{n^y}{(b^i)^y} \]

\[ = da^j + \sum_{i=0}^{j-1} ca^i \frac{n^y}{(by)^i} \]

\[ = da^j + \sum_{i=0}^{j-1} cn^y \frac{a^i}{(by)^i} \]

\[ = da^j + c n^y \sum_{i=0}^{j-1} \left( \frac{a}{by} \right)^i \]

\[ = da^j + c n^y \sum_{i=0}^{j-1} (b^{x-y})^i \]

\[ = da^j + c n^y \sum_{i=0}^{j-1} (b^{x-y})^i \]

\[ = \frac{x}{\log_b a} \]

\[ x \text{ relates # of subproblems to their size} \]

\[ \text{Rearranging we have } b^x = a \]

\[ \text{So } T(n) = da^j + cn^y \sum_{i=0}^{j-1} \left( \frac{b^x}{by} \right)^i \]

\[ = da^j + c n^y \sum_{i=0}^{j-1} (b^{x-y})^i \]

\[ \text{Also } da^j = d(b^x)^j = d(b^j)^x \]

\[ \text{Since } n = b^j \text{ this is just } dn^x \]

\[ \text{So } T(n) = dn^x + c n^y \sum_{i=0}^{j-1} r^i \]

where \( r = b^{x-y} \)
SOLVING THE GEOMETRIC SEQ

• \( T(n) = dn^x + cn^y \sum_{i=0}^{i-1} r^i \) where \( r = b^{x-y} \)

\[
\sum_{i=0}^{n-1} ar^i = \begin{cases} 
  a \frac{r^n - 1}{r-1} \in \Theta(n^r) & \text{if } r > 1 \\
  na \in \Theta(n) & \text{if } r = 1 \\
  a \frac{1-r^n}{1-r} \in \Theta(1) & \text{if } 0 < r < 1
\end{cases}
\]

• Recall formula: \( \sum_{i=0}^{n-1} ar^i = \begin{cases} 
  a \frac{r^n - 1}{r-1} \in \Theta(n^r) & \text{if } r > 1 \\
  na \in \Theta(n) & \text{if } r = 1 \\
  a \frac{1-r^n}{1-r} \in \Theta(1) & \text{if } 0 < r < 1
\end{cases} \)

• So different solutions depending on \( r \)
  • Case 1: \( r = b^{x-y} > 1 \) \( \Leftrightarrow \) \( x - y > 0 \) \( \Leftrightarrow \) \( x > y \)
  • Case 2: \( r = b^{x-y} = 1 \) \( \Leftrightarrow \) \( x - y = 0 \) \( \Leftrightarrow \) \( x = y \)
  • Case 3: \( 0 < r = b^{x-y} < 1 \) \( \Leftrightarrow \) \( x - y < 0 \) \( \Leftrightarrow \) \( x < y \)
SOLVING THE GEOMETRIC SEQ

- Formula: \( \sum_{i=0}^{n-1} ar^i = \begin{cases} a \frac{r^n-1}{r-1} \in \Theta(r^n) & \text{if } r > 1 \\ na \in \Theta(n) & \text{if } r = 1 \\ a \frac{1-r^n}{1-r} \in \Theta(1) & \text{if } 0 < r < 1 \end{cases} \)

- **Case 1:** \( r = b^{x-y} > 1 \iff x - y > 0 \iff x > y \)

- \( T(n) = dn^x + cn^y \sum_{i=0}^{j-1} r^i \in dn^x + cn^y \Theta(r^j) \)

- \( T(n) \in \Theta(n^x + n^y r^j) = \Theta(n^x + n^y (b^{x-y})^j) = \Theta(n^x + n^y (b^j)^{x-y}) \)

- Recall \( b^j = n \), so \( T(n) \in \Theta(n^x + n^y n^{x-y}) = \Theta(n^x + n^{y+x-y}) \)

- So \( T(n) \in \Theta(n^x) \)
SOLVING THE GEOMETRIC SEQ

- Formula: \( \sum_{i=0}^{n-1} ar^i = \begin{cases} 
  a \frac{r^n - 1}{r-1} \in \Theta(r^n) & \text{if } r > 1 \\
  na \in \Theta(n) & \text{if } r = 1 \\
  a \frac{1-r^n}{1-r} \in \Theta(1) & \text{if } 0 < r < 1 
\end{cases} \)

- **Case 2:** \( r = b^{x-y} = 1 \iff x - y = 0 \iff x = y \)

- \( T(n) = dn^x + cn^y \sum_{i=0}^{j-1} r^i \in dn^x + cn^y \Theta(j) \)

- \( T(n) \in \Theta(n^x + jn^y) = \Theta(n^x + jn^x) \) since \( x = y \)

- Recall \( b^j = n \), so \( \log_b b^j = \log_b n \). This means \( j \in \Theta(\log n) \).

- So \( T(n) = \Theta(n^x + n^x \log n) = \Theta(n^x \log n) \)
SOLVING THE GEOMETRIC SEQ

- Formula: $\sum_{i=0}^{n-1} ar^i = \begin{cases} a \frac{r^n-1}{r-1} \in \Theta(r^n) & \text{if } r > 1 \\ na \in \Theta(n) & \text{if } r = 1 \\ a \frac{1-r^n}{1-r} \in \Theta(1) & \text{if } 0 < r < 1 \end{cases}$

- Case 3: $0 < r = b^{x-y} < 1 \iff x - y < 0 \iff x < y$

- $T(n) = dn^x + cn^y \sum_{i=0}^{j-1} r^i \in dn^x + cn^y \Theta(1)$

- $T(n) \in \Theta(n^x + n^y)$

- Since $x < y$, we simply have $T(n) \in \Theta(n^y)$

Note that the base case constant $d$ is not present in any of these complexities!
Recall: $T(n) = d n^x + c n^y \sum_{i=0}^{j-1} r^i$ where $r = b^{x-y}$

$x = \log_b a$ i.e. $\log_{\text{subproblem size}} |\text{subproblems}|$

<table>
<thead>
<tr>
<th>case</th>
<th>$r$</th>
<th>$y$, $x$</th>
<th>complexity of $T(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>heavy leaves</td>
<td>$r &gt; 1$</td>
<td>$y &lt; x$</td>
<td>$T(n) \in \Theta(n^x)$</td>
</tr>
<tr>
<td>balanced</td>
<td>$r = 1$</td>
<td>$y = x$</td>
<td>$T(n) \in \Theta(n^x \log n)$</td>
</tr>
<tr>
<td>heavy top</td>
<td>$r &lt; 1$</td>
<td>$y &gt; x$</td>
<td>$T(n) \in \Theta(n^y)$</td>
</tr>
</tbody>
</table>

**heavy leaves** means that the value of the recursion tree is dominated by the values of the leaf nodes.

**balanced** means that the values of the levels of the recursion tree are constant (except for the last level).

**heavy top** means that the value of the recursion tree is dominated by the value of the root node.
MASTER THEOREM FOR RECURRENCES

• **Simplified version**

Consider recurrence:
\[ T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^y) \] where \( a \geq 1, b \geq 2 \) and \( n = b^j \)
And let \( x = \log_b a \).

\[
T(n) \in \begin{cases} 
\Theta(n^x) & \text{if } y < x \\
\Theta(n^x \log n) & \text{if } y = x \\
\Theta(n^y) & \text{if } y > x.
\end{cases}
\]
Recall: simplified master theorem

Suppose that \( a \geq 1 \) and \( b > 1 \). Consider the recurrence

\[
T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^y), \text{ where } n \text{ is a power of } b.
\]

Denote \( x = \log_b a \). Then

\[
T(n) \in \begin{cases} 
\Theta(n^x) & \text{if } y < x \\
\Theta(n^x \log n) & \text{if } y = x \\
\Theta(n^y) & \text{if } y > x.
\end{cases}
\]

Questions: \( a = ? \) \( b = ? \) \( y = ? \) \( x = ? \)

which \( \Theta \) function?

\[
T(n) = 2T(n/2) + cn. \\
\begin{align*}
\text{a=2; } & b=2; \ y=1; \ x=1 \\
\Theta(n^x \log n) = & \Theta(n \log n)
\end{align*}
\]

\[
T(n) = 3T(n/2) + cn. \\
\begin{align*}
\text{a=3; } & b=2; \ y=1; \ x=\log_2 3 \\
\Theta(n^x) = & \Theta(n^{\log_2 3})
\end{align*}
\]

\[
T(n) = 4T(n/2) + cn. \\
\begin{align*}
\text{a=4; } & b=2; \ y=1; \ x=\log_2 4 \\
\Theta(n^x) = & \Theta(n^2)
\end{align*}
\]

\[
T(n) = 2T(n/2) + cn^{3/2}. \\
\begin{align*}
\text{a=2; } & b=2; \ y=3/2; \ x=1 \\
\Theta(n^y) = & \Theta(n^{3/2})
\end{align*}
\]
**GENERAL MASTER THEOREM**

Suppose that $a \geq 1$ and $b > 1$. Consider the recurrence

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

where $n$ is a power of $b$. Denote $x = \log_b a$. Then

$$T(n) \in \begin{cases} 
\Theta(n^x) & \text{if } f(n) \in O(n^{x-\epsilon}) \text{ for some } \epsilon > 0 \\
\Theta(n^x \log n) & \text{if } f(n) \in \Theta(n^x) \\
\Theta(f(n)) & \text{if } f(n)/n^{x+\epsilon} \text{ is an increasing function of } n \\
\end{cases}$$

for some $\epsilon > 0$. 

Example recurrence:

$$T(n) = 3T(n/4) + n \log n$$

Arbitrary function of $n$ (not just $cn^y$)

Must reason about relationship between $f(n)$ and $n^x$
REVISITING THE RECURSION TREE METHOD

- Some recurrences with complex \( f(n) \) functions (such as \( f(n) = \log n \)) can still be solved “by hand”

- Example: Let \( n = 2^j \); \( T(1) = 1 \); \( T(n) = 2T \left( \frac{n}{2} \right) + n \log n \)

<table>
<thead>
<tr>
<th>level</th>
<th># nodes</th>
<th>value at each node</th>
<th>value of the level</th>
</tr>
</thead>
<tbody>
<tr>
<td>( j )</td>
<td>1</td>
<td>( j2^j )</td>
<td>( j2^j )</td>
</tr>
<tr>
<td>( j - 1 )</td>
<td>2</td>
<td>( (j - 1)2^{j-1} )</td>
<td>( (j - 1)2^j )</td>
</tr>
<tr>
<td>( j - 2 )</td>
<td>( 2^2 )</td>
<td>( (j - 2)2^{j-2} )</td>
<td>( (j - 2)2^j )</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>1</td>
<td>( 2^{j-1} )</td>
<td>( 2^1 )</td>
<td>( 2^j )</td>
</tr>
<tr>
<td>0</td>
<td>( 2^j )</td>
<td>1</td>
<td>( 2^j )</td>
</tr>
</tbody>
</table>

Note

\( \log_2 n = j \)

So

\( j2^j = n \log_2 n \)

And

\( (j - 1)2^{j-1} = \frac{n}{2} \log_2 \frac{n}{2} \)
REVISITING THE RECURSION TREE METHOD

- Recall: \( n = 2^j \); \( T(1) = 1 \); \( T(n) = 2T\left(\frac{n}{2}\right) + n \log n \)

Summing the values at all levels of the recursion tree, we have

\[
T(n) = 2^j \left( 1 + \sum_{i=1}^{j} i \right) = 2^j \left( 1 + \frac{j(j + 1)}{2} \right)
\]

Since \( n = 2^j \), we have \( j = \log_2 n \) and \( T(n) \in \Theta(n(\log n)^2) \).
MASTER THEOREM WHEN $b^{j-1} < n < b^j$

- $n/b$ is **not always an integer**!
  - floors/ceilings are hard
  - not a geometric sequence
- Suppose we get a big-O bound for $b^{j-1} < n < b^j$ by instead considering the **larger problem size** $b^j$

\[
T(n) \leq T(b^j) \in \begin{cases} 
\Theta \left( (b^j)^x \right) & \text{if } y < x \\
\Theta \left( (b^j)^x \log b^j \right) & \text{if } y = x \\
\Theta \left( (b^j)^y \right) & \text{if } y > x 
\end{cases}
\]
MASTER THEOREM WHEN $b^{j-1} < n < b^j$

\[
T(n) \leq T(b^j) \in \begin{cases} 
\Theta\left((b^j)^x\right) & \text{if } y < x \\
\Theta\left((b^j)^x \log b^j\right) & \text{if } y = x \\
\Theta\left((b^j)^y\right) & \text{if } y > x 
\end{cases}
\]

- **Observation:** $b^j < bn$ since $n$ is between $b^{j-1}$ and $b^j$

\[
T(n) \leq T(b^j) \in \begin{cases} 
\Theta\left((bn)^x\right) & \text{if } y < x \\
\Theta\left((bn)^x \log bn\right) & \text{if } y = x \\
\Theta\left((bn)^y\right) & \text{if } y > x 
\end{cases}
\]
MASTER THEOREM WHEN $b^{j-1} < n < b^j$

\[
T(n) = \begin{cases} 
\Theta((bn)^x) & \text{if } y < x \\
\Theta((bn)^x \log bn) & \text{if } y = x \\
\Theta((bn)^y) & \text{if } y > x
\end{cases}
\]

• **Case 1 ($y < x$):** $(bn)^x = b^x n^x$ and $b^x$ is a constant
  • So $T(n) \in O(n^x)$

• **Case 2 ($y = x$):** $(bn)^x \log bn = b^x n^x (\log b + \log n)$
  • $T(bn) \in \Theta(b^x n^x \log b + b^x n^x \log n) = \Theta(n^x + n^x \log n)$
  • So $T(n) \in O(n^x \log n)$

• **Case 3 ($y > x$):** $(bn)^y = b^y n^y$
  • So $T(n) \in O(n^y)$

Can tackle $\Omega$ similarly to get $\Theta$