**DIVIDE-AND-CONQUER DESIGN STRATEGY**

- **divide:** Given a problem instance $I$, construct one or more smaller problem instances $I_1, ..., I_a$
  - 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 $S_{I_1}, ..., S_{I_a}$
- **combine:** Given solutions $S_{I_1}, ..., S_{I_a}$, use an appropriate combining function to find the solution $S$ to the problem instance $I$
  - i.e., $S = \text{Combine}(S_{I_1}, ..., S_{I_a})$.

**D&C PROTO-ALGORITHM**

```
DnC_template(I)

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

**CORRECTNESS**

- Prove base cases are correct
- Inductively assume subproblems are solved correctly
- Show they are correctly assembled into a solution

**RUNTIME/SPACE COMPLEXITY?**

- 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 [$\lceil n/2 \rceil$] elements in $A$ and $A_R$ consists of the last [$\lceil n/2 \rceil$] 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 $Merge$ 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.

MERGE: CONQUER AND COMBINE

PSEUDOCODE FOR MERGESORT

```
1. Mergesort(A[1..n])
2. if n == 1 then return A
3. nL = ceil(n/2)
4. aL = A[1..nL]
5. aR = A[(nL+1) .. n]
6. sL = Mergesort(aL)
7. aR = Mergesort(aR)
8. return Merge(sL, aR)
```

PSEUDOCODE FOR MERGE

```
1. Merge(A[1..nL], aR[1..nr]):
2. Out[1..(nL+nr)] = empty array
3. if A[1L] < aR[1R]
4. while iL < nL and iR < nr
5. if A[1L] < aR[1R]
7. iL++
8. else
9. Out[iOut++] = aR[1R]
10. iR++
11. while iL < nL
13. iL++
14. while iR < nr
15. Out[iOut++] = aR[1R]
16. iR++
17. out = Out
```

There are still elements left in both arrays

Right array is out of elements

Left array is out of elements
ANALYSIS OF MERGESORT

1. MergeSort(A[1..n])
2. if n == 1 then return A
3. nl = cell(n/2) <= O(1)
4. ar = A[(nL+1)\ldots n] <= O(n)
5. sl = MergeSort(nl)
6. sr = MergeSort(ar)
7. return Merge(sl, sr) <= O(n)

So, MergeSort(A) takes O(n) time, plus the time for its two recursive calls.

How can we analyze this recursive program structure?

RECURSIVE RELATIONS

A crucial analysis tool for recursive algorithms

RECURSIVE RELATIONS

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.

MATHEMATICALLY EXPRESSING THE COMPLEXITY OF MERGESORT

Let $T(n)$ denote the time to run MergeSort on an array of length $n$.

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

Recurrence relation:

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

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

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

Can also compute using a table...

RECURSION TREE METHOD

Evaluating recurrences with $T(n/4)$ terms

Recursion tree

<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>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>$\log n$</td>
<td>$n$</td>
<td>$c(n) = c$</td>
<td>$nc(n/n) = cn$</td>
</tr>
</tbody>
</table>

Total = $cn \times \#\text{levels}$

So, MergeSort has runtime $O(n \log n)$

Can also compute using a table...
**RECURSION TREE METHOD FORMALIZED**

Sample recurrence for two recursive calls on problem size $n/2$:  

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

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 root node, 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 that the result is $T(n)$.

---

**GUESS-AND-CHECK METHOD**

In Math, I use the **Guess & Hope 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

---

**SUBSTITUTION METHOD: WORKED EXAMPLE**

Recurrence:  

\[
T(0) = 4; \quad T(n) = T(n-1) + 6n - 5
\]

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

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

• \(T(0) + n(6n - 5) - 6(1 + 2 + 3 + \cdots + (n - 1)) = guess(n)\)

\[\begin{array}{c}
guess(n) = T(0) + n(6n - 5) - 6(1 + 2 + 3 + \cdots + (n - 1)) \\
= 4 + 6n^2 - 5n - 6n(n - 1)/2 \quad \text{(simplify)}
\end{array}\]

• Are we done?

- The form of guess(n) was an educated guess.
- To be formal, we must prove it correct using induction

---

**PROOF**

- **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 \(guess(n+1) = T(n+1)\).

\[
\begin{align*}
T(n+1) &= T(n) + 6(n + 1) - 5 \\
&= guess(n) + 6(n + 1) - 5 \\
&= 3n^2 + 4n + 5 \\
\text{(by definition)}
&= 3(n + 1)^2 - 2(n + 1) + 4 \quad \text{(by inductive hypothesis)}
&= 3(n + 1)^2 + 4 + 5 \\
\text{(substitute & simplify)}
&= T(n + 1) \quad \text{(by definition)}
\end{align*}
\]

**ANOTHER APPROACH**

- **Suppose** you look for a while at the previous recurrence:

  \[
  T(0) = 4; \quad T(n) = T(n-1) + 6n - 5
  \]

- **With some experience, you might just guess** it’s quadratic

  If you’re right, it should have the form:

  \[
  an^2 + bn + c \quad \text{for some unknown constants } a, b, c
  \]

  So, just carry the unknown constants into the proof!

- You can then determine what the constants must be for the proof to work out
• \( T(0) = 4; T(n) = T(n-1) + 6n - 5 \); 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 \( c = 4 \) \( (a, b) \) are not constrained
• Inductive case: \( \text{suppose} \) \( \text{guess}(n) = T(n) \) for \( n \geq 0 \),
  \( \text{show} \) \( \text{guess}(n+1) = T(n+1) \).

\( T(n+1) = T(n) + 6(n+1) - 5 \) \( \text{(by definition)} \)
\( = \text{guess}(n) + 6(n+1) - 5 \) \( \text{(by inductive hypothesis)} \)
\( = an^2 + bn + 4 + 6(n+1) - 5 \) \( \text{(substitute)} \)
\( = an^2 + (b+6)n + 5 \) \( \text{(simplify)} \)

- Recall: \( \text{guess}(n) = an^2 + bn + c \) where \( c = 4 \)
- Inductive case: \( \text{suppose} \) \( \text{guess}(n) = T(n) \) for \( n \geq 0 \),
  \( \text{show} \) \( \text{guess}(n+1) = T(n+1) \).

\( T(n+1) = an^2 + (b+6)n + 5 \) \( \text{(continue previous slide)} \)
\( \text{guess}(n+1) = a(n+1)^2 + b(n+1) + 4 \) \( \text{(by definition)} \)
\( = an^2 + 2bn + 1 + bn + b + 4 \) \( \text{(simplify, and...)} \)
\( = an^2 + (2a + b)n + (a + b + 4) \) \( \text{(rearrange polynomial)} \)

We want this to be equal to \( T(n+1) \)
\( an^2 + (2a + b)n + (a + b) + 4 = an^2 + (b + 6)n + 5 \)
\( \text{equivalent to} \ (2a + b) = (b + 6) \) and \( (a + b + 4) = 5 \)
\( \text{first implies} \ a = 3 \) \( \text{plug a into second to get} \ b = 5 - 4 - 3 = -2 \)

**MASTER THEOREM FOR RECURRENCES**

- Provides a formula for solving many recurrence relations
- We start with a simplified version
- Consider recurrence: \( T(1) = d; T(n) = aT\left(\frac{n}{b}\right) + \Theta(n^p) \)
  where \( a \geq 1, b \geq 2 \) and \( n \) is a power of \( b \) \( \text{(i.e.,} \ n = b^j \text{for integer} j \text{)} \)

**REARRANGING**

- \( T(n) = da + \sum_{i=0}^{\log_b n} ca^i \left(\frac{n}{b}\right)^i \)
- \( = da + \sum_{i=0}^{\log_b n} ca^i \left(\frac{a^i}{a^i}\right) \)
- \( = da + \sum_{i=0}^{\log_b n} ca^i \left(\frac{a}{a}\right)^i \)
- \( = da + \sum_{i=0}^{\log_b n} ca^i a^i \left(\frac{a}{1}\right)^i \)
- \( = da + \sum_{i=0}^{\log_b n} ca^i \left(\frac{a}{1}\right)^i \)
- \( = da + \sum_{i=0}^{\log_b n} c \left(\frac{a}{1}\right)^i \)
- \( = da + cn \sum_{i=0}^{\log_b n} \left(\frac{a}{1}\right)^i \)
- \( \text{Let} \ x = \log_b a \)
- \( x \text{relates} \# \text{of subproblems to their size} \)
- Rearranging we have \( b^x = a \)
- \( \text{So} \ T(n) = da + cn \sum_{i=0}^{x} \left(\frac{a}{1}\right)^i \)
- \( = da + cn \sum_{i=0}^{x} \left(\frac{a}{1}\right)^i \)
- \( = da + cn \sum_{i=0}^{x} \left(\frac{a}{1}\right)^i \)
- \( = da + cn \sum_{i=0}^{x} \left(\frac{a}{1}\right)^i \)
- \( = da + cn \sum_{i=0}^{x} \left(\frac{a}{1}\right)^i \)
- \( \text{So} \ T(n) = da + cn \sum_{i=0}^{x} \left(\frac{a}{1}\right)^i \)
  where \( r = b^{x-y} \)

**SOLVING THE GEOMETRIC SEQ**

- \( T(n) = dn^x + cn^y \sum_{i=0}^{x-1} \left(\frac{a}{1}\right)^i \text{where} \ r = b^{x-y} \)
- \( \text{Recall formula:} \ 
  \sum_{i=0}^{x-1} \left(\frac{a}{1}\right)^i = \frac{\left(\frac{a}{1}\right)^x - 1}{\frac{a}{1} - 1} \text{if} \ r > 1 \)
  \( = \frac{\left(\frac{a}{1}\right)^x - 1}{\frac{a}{1} - 1} \text{if} \ r = 1 \)
  \( = \frac{\left(\frac{a}{1}\right)^x - 1}{\frac{a}{1} - 1} \text{if} \ 0 < r < 1 \)
- \( \text{So different solutions depending on} \ r \)
  \( \text{Case 1:} \ r = b^{x-y} > 1 \ \iff \ x - y > 0 \ \iff \ x > y \)
  \( \text{Case 2:} \ r = b^{x-y} = 1 \ \iff \ x - y = 0 \ \iff \ x = y \)
  \( \text{Case 3:} \ 0 < r = b^{x-y} < 1 \ \iff \ x - y < 0 \ \iff \ x < y \)

Let’s rearrange this into a geometric sequence and solve

**Example corresponding algorithm**

```
1 node
Problem size
1
subproblems = []
for i = 1..x:
  let x = subproblem of size n/b
  return combine in n^y time
return solution
```
SOLVING THE GEOMETRIC SEQ

- Formula: \[ \sum_{i=0}^{\infty} ar^i = \begin{cases} \frac{a}{1-r} & \text{if } r > 1 \\ na & \text{if } r = 1 \\ a \frac{1-r^n}{1-r} & \text{if } 0 < r < 1 \end{cases} \]

- Case 1: \( r = b^{x-y} > 1 \) \( \iff \) \( x - y > 0 \) \( \iff \) \( x > y \)
  - \( T(n) = dn^2 + cn^2 \sum_{i=0}^{n-1} r^i \in dn^2 + cn^2 \Theta(r^n) \)
  - \( T(n) \in \Theta(n^2 + n^2, (b^{x-y})) = \Theta(n^2 + n^2, (b^{x-y})) \)
  - Recall \( b^i = n \), so \( T(n) \in \Theta(n^2 + n^2 \times b^{x-y}) \)
  - So \( T(n) \in \Theta(n^2) \)

- Case 2: \( r = b^{x-y} = 1 \) \( \iff \) \( x - y = 0 \) \( \iff \) \( x = y \)
  - \( T(n) = dn^2 + cn^2 \sum_{i=0}^{n-1} r^i \in dn^2 + cn^2 \Theta(1) \)
  - \( T(n) \in \Theta(n^2 + n^2) \)
  - Recall \( b^i = n \), so \( T(n) = \Theta(n^2 + n^2 \log n) = A(n^2 \log n) \)
  - So \( T(n) = \Theta(n^2 + n^2 \log n) = \Theta(n^2 \log n) \)

SOLVING THE GEOMETRIC SEQ

- Formula: \[ \sum_{i=0}^{\infty} ar^i = \begin{cases} \frac{a}{1-r} & \text{if } r > 1 \\ na & \text{if } r = 1 \\ a \frac{1-r^n}{1-r} & \text{if } 0 < r < 1 \end{cases} \]

- Case 3: \( 0 < r = b^{x-y} < 1 \) \( \iff \) \( x - y < 0 \) \( \iff \) \( x < y \)
  - \( T(n) = dn^2 + cn^2 \sum_{i=0}^{n-1} r^i \in dn^2 + cn^2 \Theta(1) \)
  - \( T(n) \in \Theta(n^2 + n^2) \)
  - Since \( x < y \), we simply have \( T(n) \in \Theta(n^2) \)

Note that the base case constant \( d \) is not present in any of these complexities!

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 \( n = b^i \)
  And let \( x = \log_b a \).

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

WORKED EXAMPLES

Recall: simplified master theorem
Suppose that \( a \geq 1 \) and \( b > 1 \). Consider the recurrence
\[ T(n) = aT \left( \frac{n}{b} \right) + \Theta(n^y) \] where \( n \) is a power of \( b \).
Denote \( x = \log_b a \). Then
\[ T(n) = \begin{cases} \Theta(n^x) & \text{if } y < x \\ \Theta(n^x \log n) & \text{if } y = x \\ \Theta(n^y) & \text{if } y > x \end{cases} \]

Questions:
- \( a = \_ \) \( b = \_ \) \( y = \_ \) \( x = \_ \)
  which \( \Theta \) function?

\[ T(n) = 2T(n/2) + cn. \]
\[ T(n) = 3T(n/2) + cn. \]
\[ T(n) = 4T(n/2) + cn. \]
\[ T(n) = 5T(n/2) + cn. \]
\[ T(n) = 6T(n/2) + cn. \]
**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 \( \alpha = \log_b a \). Then,

\[
T(n) \leq \begin{cases} 
O(n^\alpha) & \text{if } f(n) = O(n^{\alpha}) \text{ for some } \alpha > 0 \\
O(n^{\alpha} \log n) & \text{if } f(n) = \Theta(n^\alpha) \\
O(f(n)) & \text{if } f(n) \text{ is an increasing function of } n \\
\end{cases}
\]

must reason about relationship between \( f(n) \) and \( n^\alpha \).

---

**EXAMPLE RECURRENCE:**

\[
T(n) = 3T\left(\frac{n}{4}\right) + n \log n
\]

---

**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>( j )</th>
<th>( j-1 )</th>
<th>( j-2 )</th>
<th>( j-3 )</th>
<th>( \ldots )</th>
<th>( 2^j )</th>
<th>( 2^{j-1} )</th>
<th>( 2^1 )</th>
<th>( 2^0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>value at each node</td>
<td>( \frac{2^j}{j-1} )</td>
<td>( \frac{2^j}{j-2} )</td>
<td>( \frac{2^j}{j-3} )</td>
<td>( \frac{2^j}{j-4} )</td>
<td>( \ldots )</td>
<td>( \frac{2^j}{2^1} )</td>
<td>( \frac{2^j}{2^0} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>value of the level</td>
<td>( \frac{2^j}{j-1} )</td>
<td>( \frac{2^j}{j-2} )</td>
<td>( \frac{2^j}{j-3} )</td>
<td>( \frac{2^j}{j-4} )</td>
<td>( \ldots )</td>
<td>( \frac{2^j}{2^1} )</td>
<td>( \frac{2^j}{2^0} )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note \( \log_2 n = j \)

**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 \( \Theta \)-bound for \( b^{j-1} < n < b^j \)
  - by instead considering the larger problem size \( b^j \)
  - \( n \log \log n \)
  - \( \Theta \log \log n \)
  - \( \Theta \log n \)

**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 \( \Theta \)-bound for \( b^{j-1} < n < b^j \)

**Master Theorem**

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

- Case 1 (\( y < x \)): \( (bn)^x = b^x n^x \) and \( b^x \) is a constant
  - So \( T(n) \in \Theta(n^x) \)
- 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 \( T(n) \in \Theta(n^x \log n) \)
- Case 3 (\( y > x \)): \( (bn)^y = b^y n^y \)
  - So \( T(n) \in \Theta(n^y) \)

**Master Theorem**

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

- Case 1 (\( y < x \)): \( (bn)^x = b^x n^x \) and \( b^x \) is a constant
  - So \( T(n) \in \Theta(n^x) \)
- 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 \( T(n) \in \Theta(n^x \log n) \)
- Case 3 (\( y > x \)): \( (bn)^y = b^y n^y \)
  - So \( T(n) \in \Theta(n^y) \)