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

DIVIDE AND CONQUER

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 $S_1, ..., S_a$
- combine: Given solutions $S_1, ..., S_a$, use an appropriate combining function to find the solution $S$ to the problem instance $I$
  - i.e., $S = \text{Combine}(S_1, ..., S_a)$.

D&C PROTO-ALGORITHM

```python
DnC_template(I):
  if BaseCase(I) return Result(I)
  subproblems = [I_1, I_2, ..., I_a]
  subproblems = []
  for j = 1 to 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 $\lfloor n/2 \rfloor$ elements in $A$.

conquer: Run $\text{Mergesort}$ 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 $O(n)$ 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

PSEUDOCODE FOR MERGESORT

1. Mergesort($A[1..n]$)
2. if $n = 1$ then return $A$
3. $nL = \lceil n/2 \rceil$
4. $aL = A[1..nL]$
5. $nR = n - nL$
6. $aR = A[nL+1..n]$
7. $aOut = \text{Mergesort}(aL)$
8. $aOut = \text{Mergesort}(aR)$
9. return $\text{Merge}(aL, aR)$

PSEUDOCODE FOR MERGE

1. Merge($aL[1..nL], aR[1..nR]$)
2. $aOut[1..(nL+nR)] = \text{empty array}$
3. $iL = 1$, $iR = 1$, $iOut = 1$
4. while $iL < nL$ and $iR < nR$
5. if $aL[iL] < aR[iR]$
6. $aOut[iOut] = aL[iL]$
7. $iL = iL + 1$; $iOut = iOut + 1$
8. else
9. $aOut[iOut] = aR[iR]$
10. $iR = iR + 1$; $iOut = iOut + 1$
11. while $iL < nL$
12. $aOut[iOut] = aL[iL]$
13. $iL = iL + 1$; $iOut = iOut + 1$
14. while $iR < nR$
15. $aOut[iOut] = aR[iR]$
16. $iR = iR + 1$; $iOut = iOut + 1$
17. return $aOut$
ANALYSIS OF MERGESORT

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

How can we analyze this recursive program structure?

RECURSION RELATIONS
A crucial analysis tool for recursive algorithms

MATHEMICALLY EXPRESSING THE COMPLEXITY OF MERGESORT

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

divide takes time $\Theta(n)$

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

Recurrence relation:

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

To make this easier, assume $n = 2^k$, which lets us ignore floors/ceilings.

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

So, mergesort has runtime $O(n \log n)$.
RECURSION TREE METHOD FORMALIZED

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

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)
- \( = T(n-2) + 6n - 6 - 5 + 6n - 5 \) (compare new terms)
- \( = T(n-3) + 6(n-2) - 5 + 6n - 6 \) (new terms)
- \( = T(n-3) + 6n - 6 + 2(6n - 5) - 6 \) (substitute)
- \( = T(n-3) + 3(6n - 5) - 6(1 + 2) \) (compare new terms)
- ... identify patterns and guess what happens in the limit
- \( = T(0) + m(6n - 5) - 6(1 + 2 + 3 + \ldots + (n-1)) = guess(n) \)

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:
    \[ an^2 + bn + c \]
  - So, just carry the unknown constants into the proof!
  - You can then determine what the constants must be for the proof to work out

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 \( guess(n+1) = T(n+1) \).
  - \( T(n+1) = T(n) + 6(n+1) - 5 \) (by definition)
  - \( = guess(n) + 6(n+1) - 5 \) (by inductive hypothesis)
  - \( = 3n^2 + 4n + 5 \) (substitute & simplify)
  - \( guess(n+1) = 3(n+1)^2 - 2(n+1) + 4 \) (by definition)
  - \( = 3n^2 + 4n + 5 = T(n+1) \) (simplify)
\[ T(0) = 4; T(n) = T(n-1) + 6n - 5; \text{guess } n = an^2 + bn + c \]

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 \( c = 4 \) (\( a, b \) are not constrained)
- Inductive case: suppose \( \text{guess } n = T(n) \) for \( n \geq 0 \),
  
  - show \( \text{guess } (n+1) = T(n+1) \)

\[ T(n + 1) = T(n) + 6(n + 1) - 5 \]  
  
  - (by definition)

\[ = an^2 + bn + 4 + 6(n + 1) - 5 \]  
  
  - (by inductive hypothesis)

\[ = an^2 + (b + 6)n + 5 \]  
  
  - (simplify)

\section*{M master theorem for recurrences}

Provides a formula for solving many recurrence relations

- We start with a simplified version

Consider recurrence: \( T(1) = 4; T(n) = aT(n/2) + \Theta(n^2) \)
  
  - \( a \geq 1, b \geq 2 \) and \( n \) is a power of \( b \) (i.e., \( n = b^i \) for integer \( i \))

\[ \begin{align*}
\text{if BaseCase[1], return Result(1)} \\
\text{subcases = {}} \\
\text{for } j \text{ in } 1..n \\
\text{let } x = \text{subproblem of size } n/b \\
\text{subcases}[j] = \text{BaseCase}(x) \\
\text{end} \\
\text{solution = combine in } n'' \text{ time}
\end{align*} \]

\section*{Rearranging}

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

- \( x = \log_b a \)
- \( x \) relates # of subproblems to their size
- Rearranging we have \( b^x = a \)

\[ \begin{align*}
S(n) &= da^i + \sum_{i=0}^{i-1} ca^i \left( \frac{b}{b^i} \right)^j \\
&= da^i + \sum_{i=0}^{i-1} ca^i \left( \frac{b}{b^i} \right)^j \\
&= da^i + \sum_{i=0}^{i-1} ca^i \left( \frac{b}{b^i} \right)^j \\
&= da^i + \sum_{i=0}^{i-1} ca^i \left( \frac{b}{b^i} \right)^j \\
&= da^i + \sum_{i=0}^{i-1} ca^i \left( \frac{b}{b^i} \right)^j
\end{align*} \]

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

\[ \begin{align*}
&= da^i + \sum_{i=0}^{i-1} ca^i \left( \frac{b}{b^i} \right)^j \\
&= da^i + \sum_{i=0}^{i-1} ca^i \left( \frac{b}{b^i} \right)^j \\
&= da^i + \sum_{i=0}^{i-1} ca^i \left( \frac{b}{b^i} \right)^j \\
&= da^i + \sum_{i=0}^{i-1} ca^i \left( \frac{b}{b^i} \right)^j \\
&= da^i + \sum_{i=0}^{i-1} ca^i \left( \frac{b}{b^i} \right)^j
\end{align*} \]

- Also \( da^i = (b^i)^x = d(b^i)^j \)
- Since \( n = b^i \) this is just \( da^i \)

\[ \begin{align*}
So T(n) &= da^i + \sum_{i=0}^{i-1} ca^i \left( \frac{b}{b^i} \right)^j \\
&= da^i + \sum_{i=0}^{i-1} ca^i \left( \frac{b}{b^i} \right)^j \\
&= da^i + \sum_{i=0}^{i-1} ca^i \left( \frac{b}{b^i} \right)^j
\end{align*} \]

\section*{Solving the geometric seq}

\[ \begin{align*}
T(n) &= dn^r + cn^t \sum_{i=0}^{i-1} r^i \\
&= dn^r + cn^t \sum_{i=0}^{i-1} r^i
\end{align*} \]

\[ \begin{align*}
&= dn^r + cn^t \sum_{i=0}^{i-1} r^i \\
&= dn^r + cn^t \sum_{i=0}^{i-1} r^i \\
&= dn^r + cn^t \sum_{i=0}^{i-1} r^i
\end{align*} \]

\section*{Recall formula}

\[ \sum_{i=0}^{i-1} r^i = \frac{r^{i+1} - 1}{r-1} \quad \text{if } r > 1 \\
= \frac{r^{i+1} - 1}{r-1} \quad \text{if } r = 1 \\
= \frac{r^{i+1} - 1}{r-1} \quad \text{if } 0 < r < 1
\]

So different solutions depending on \( r \)

\begin{itemize}
\item Case 1: \( r = b^{x+y} > 1 \) \( x - y > 0 \) \( x > y \)
\item Case 2: \( r = b^{x+y} = 1 \) \( x - y = 0 \) \( x = y \)
\item Case 3: \( 0 < r = b^{x+y} < 1 \) \( x - y < 0 \) \( x < y \)
\end{itemize}
\[ \sum_{i=0}^{n-1} a r^i = \begin{cases} \frac{a r^n - 1}{r-1} & \text{if } r > 1 \\ n a & \text{if } r = 1 \\ \frac{a}{1-r} & \text{if } 0 < r < 1 \end{cases} \]

\[ T(n) = d n^r + c n^j \sum_{i=0}^{n-1} r^i = d n^r + c n^j \Theta(1) \]

\[ T(n) \in \Theta(n^r) \]

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

\[ T(n) = \Theta(n^r + n^{y(x)}) = \Theta(n^r + n^{y(b^r)}) = \Theta(n^r + n^{y(b^r)}) \]

Recall \( b^r = n \), so \( T(n) = \Theta(n^r + n^{y(b^r)}) = \Theta(n^r + n^{y(b^r)}) \)

So \( T(n) = \Theta(n^r) \)

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

\[ T(n) = d n^r + c n^j \sum_{i=0}^{n-1} r^i = d n^r + c n^j \Theta(1) \]

\[ T(n) \in \Theta(n^r + n^{j(r)}) \]

Recall \( b^r = n \), so \( T(n) = \Theta(n^r + n^{j(r)}) \)

So \( T(n) = \Theta(n^r + n^{j(r)}) = \Theta(n^r \log n) \)

**Case 3:** \( 0 < r = b^r < 1 \)  \( \Rightarrow \)  \( x - y < 0 \)  \( \Rightarrow \)  \( x < y \)

\[ T(n) = d n^r + c n^j \sum_{i=0}^{n-1} r^i = d n^r + c n^j \Theta(1) \]

Since \( x < y \), we simply have \( T(n) = \Theta(n^r) \)

\[ y = \frac{1}{2} - \frac{1}{2} \]

\[ x = \frac{1}{2} \]

\[ \sigma = 0 \]

\[ b = 2 \]

\[ y = 1 \]

\[ x = 0 \]

**SOME BONUS INTUITION FOR R CASES**

Recall: simplified master theorem

\[ T(n) = d n^r + c n^j \sum_{i=0}^{n-1} r^i = d n^r + c n^j \Theta(1) \]

\[ x = \log_d r \]

\[ a = 0 \]

\[ b = 1 \]

\[ y = 1 \]

\[ x = 0 \]

**SOLVING THE GEOMETRIC SEQ**

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

**Master Theorem**

**Simplified version**

Consider recurrence:

\[ T(n) = a T\left(\frac{n}{b}\right) + \Theta(n^r) \]

And let \( s = \log_b a \).

\[ T(n) \in \begin{cases} \Theta(n^r) & \text{if } r < s \\ \Theta(n^s \log n) & \text{if } s < r \\ \Theta(n^r) & \text{if } r > s \end{cases} \]

**Questions:**

\[ a = 0, b = 2, y = 1, x = 0 \]

\[ x = 0 \]

which \( \Theta \) function?

\[ T(n) = 3 T\left(\frac{n}{2}\right) + n \]

\[ a = 2, b = 2, y = 0, x = 1 \]

\[ \Theta(n \log n) = \Theta(\log n) \]

\[ T(n) = 7 T\left(\frac{n}{2}\right) + n \]

\[ a = 2, b = 2, y = 1, x = 0 \]

\[ \Theta(n \log n) = \Theta(n) \]

\[ T(n) = 2 T\left(\frac{n}{2}\right) + n \]

\[ a = 2, b = 2, y = 0, x = 1 \]

\[ \Theta(n) = \Theta(n) \]

\[ T(n) = 2 T\left(\frac{n}{2}\right) + \log n \]

\[ a = 2, b = 2, y = 0, x = 1 \]

\[ \Theta(n \log n) = \Theta(n \log n) \]

\[ T(n) = 2 T\left(\frac{n}{2}\right) + n \]

\[ a = 2, b = 2, y = 1, x = 0 \]

\[ \Theta(n \log n) = \Theta(n \log n) \]
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 \( x = \log_b n \). Then

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

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>2 ( j )</td>
<td>( 2^j )</td>
<td>( 2^j )</td>
</tr>
<tr>
<td>( j - 1 )</td>
<td>2 ( j - 1 )</td>
<td>( 2^{j-1} )</td>
<td>( 2^{j-1} )</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

- **Case 1** (\( y < x \)): \( \Theta((bn)^y) \)
- **Case 2** (\( y = x \)): \( \Theta((bn)^y \log bn) \)
- **Case 3** (\( y > x \)): \( \Theta((bn)^y) \)

Can tackle \( \Omega \) similarly to get \( \Theta \)

**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

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

Since \( n = 2^j \), we have \( j = \log_2 n \) and \( T(n) = \Theta(n(\log n)^2) \).

**MASTER THEOREM WHEN \( b^{j-1} < n < b^j \)**

- **Case 1** (\( y < x \)): \( \Theta((bn)^y) \)
- **Case 2** (\( y = x \)): \( \Theta((bn)^y \log bn) \)
- **Case 3** (\( y > x \)): \( \Theta((bn)^y) \)

**MASTER THEOREM WHEN \( b^{j-1} < n < b^j \)**

- **Case 1** (\( y < x \)): \( \Theta(n^{y}) \)
- **Case 2** (\( y = x \)): \( \Theta(n^{y} \log n) \)
- **Case 3** (\( y > x \)): \( \Theta(n^{y}) \)

**MASTER THEOREM WHEN \( b^{j-1} < n < b^j \)**

- **Case 1** (\( y < x \)): \( \Theta((bn)^y) \)
- **Case 2** (\( y = x \)): \( \Theta((bn)^y \log bn) \)
- **Case 3** (\( y > x \)): \( \Theta((bn)^y) \)

**MASTER THEOREM WHEN \( b^{j-1} < n < b^j \)**

- **Case 1** (\( y < x \)): \( \Theta(n^{y}) \)
- **Case 2** (\( y = x \)): \( \Theta(n^{y} \log n) \)
- **Case 3** (\( y > x \)): \( \Theta(n^{y}) \)

**MASTER THEOREM WHEN \( b^{j-1} < n < b^j \)**

- **Case 1** (\( y < x \)): \( \Theta(n^{y}) \)
- **Case 2** (\( y = x \)): \( \Theta(n^{y} \log n) \)
- **Case 3** (\( y > x \)): \( \Theta(n^{y}) \)