CS 341: ALGORITHMS
Lecture 4: divide and conquer

Slides by Trevor Brown (some material from Doug Stinson’s slides)
trevor.brown@uwaterloo.ca
(http://tbrown.pro)
THIS TIME

• Divide and conquer paradigm
• Merge sort
• Recurrence relations (and solving them)
  • Recursion tree method
  • Guess and check method
• Time permitting
  • Master theorem
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., \( S = \text{Combine}(S_1, \ldots, S_a) \).
When you’re finished with background material and finally get to merge sort.
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 $\lfloor n/2 \rfloor$ 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 $\textit{Merge}$ to merge $A_L$ and $A_R$ into a single sorted array. Recall that this can be done in time $Θ(n)$ 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

105 7 13 8 14 1 19 11

4 10 98 16 31 5 21 12
MERGE: CONQUER AND COMBINE
MERGE SIMULATION 1

L

5 31

R

12 21

O

5 12 21 31
ANOTHER EXAMPLE MERGE STEP
MERGE SIMULATION 2

L
4 10 96 98

R
5 12 21 31

O
4 5 10 12 21 31 96 98
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    sR = Mergesort(aR)
8    return Merge(sL, sR)
PSEUDOCODE FOR MERGE

```plaintext
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
```

There are still elements left in both arrays.
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)

RECURRENT 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 $T(...)$ 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

\[ T(n) \]
\[ T(n/2) \]
\[ T(n/2) \]

... 

\[ T(n/4) \]
\[ T(n/4) \]

... 

\[ T(n/8) \]

...
**Recursion Tree Method**

![Recursion Tree](image)

<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 = \(cn \times \#\) 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
  
  • Recursively substitute the formula into itself
  
  • Try to identify patterns to **guess** the final closed form

• **Check** that the guess was correct
WORKED EXAMPLE

Recurrence: \( T(0) = 4 \); \( T(n) = T(n-1) + 6n - 5 \)

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

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

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

• \( = 3n^2 - 2n + 4 \)

• Are we done?

• It depends… The form of \( \text{guess}(n) \) was an \textbf{educated guess}.
  • What we just saw is typically enough for this course.

• To be completely formal, we \textbf{prove} it correct using \textbf{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 \( guess(n + 1) = T(n + 1) \).

\[
\begin{align*}
T(n + 1) &= T(n) + 6(n + 1) - 5 \\
&= guess(n) + 6(n + 1) - 5 \\
&= 3n^2 + 4n + 5 \\
guess(n + 1) &= 3(n + 1)^2 - 2(n + 1) + 4 \\
&= 3n^2 + 4n + 5 = T(n + 1)
\end{align*}
\]
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 \textit{iff} \( c = 4 \) \( (a, b \text{ can be anything}) \)

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

\[
T(n + 1) = T(n) + 6(n + 1) - 5 \quad \text{(by definition)}
\]
\[
= \text{guess}(n) + 6(n + 1) - 5 \quad \text{(by inductive hypothesis)}
\]
\[
= an^2 + bn + 4 + 6(n + 1) - 5 \quad \text{(substitute)}
\]
\[
= an^2 + (b + 6)n + 5 \quad \text{(simplify)}
\]
• Recall: \( \text{guess}(n) = an^2 + bn + c \) where \( c = 4 \)

• Inductive case: \textbf{suppose} \( \text{guess}(n) = T(n) \) for \( n \geq 0 \), \textbf{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)

• \( = an^2 + (2a + b)n + (a + b + 4) \) (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 \textbf{correct} for \( a = 3, b = -2, c = 4 \)
TIME PERMITTING

Might not get here...
MASTER THEOREM FOR RECURRENCES

• Provides a formula for solving many recurrence relations
• We start with a simplified version

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

\[
T(n) \in \begin{cases} 
\Theta(n^x) & \text{if } y < x \\
\Theta(n^x \log n) & \text{if } y = x \\
\Theta(n^y) & \text{if } y > x.
\end{cases}
\]
Suppose that $a \geq 1$ and $b \geq 2$ are integers and

$$T(n) = aT\left(\frac{n}{b}\right) + cn^y, \quad T(1) = d.$$ 

Let $n = b^j$.

<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>$cn^y$</td>
<td>$cn^y$</td>
</tr>
<tr>
<td>$j - 1$</td>
<td>$a$</td>
<td>$c(n/b)^y$</td>
<td>$ca(n/b)^y$</td>
</tr>
<tr>
<td>$j - 2$</td>
<td>$a^2$</td>
<td>$c(n/b^2)^y$</td>
<td>$ca^2(n/b^2)^y$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>$a^{j-1}$</td>
<td>$c(n/b^{j-1})^y$</td>
<td>$ca^{j-1}(n/b^{j-1})^y$</td>
</tr>
<tr>
<td>0</td>
<td>$a^j$</td>
<td>$d$</td>
<td>$da^j$</td>
</tr>
</tbody>
</table>

Must sum the values over all levels!
Summing the values at all levels of the recursion tree, we have that

$$T(n) = d \alpha^j + cn^y \sum_{i=0}^{j-1} \left( \frac{a}{b^y} \right)^i.$$ 

Recall that $b^x = a$ and $n = b^j$. Hence $\alpha^j = (b^x)^j = (b^j)^x = n^x$.

The formula for $T(n)$ is a geometric sequence with ratio $r = a/b^y = b^{x-y}$:

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

There are three cases, depending on whether $r > 1$, $r = 1$ or $r < 1$. 
THE THREE CASES FOR $r$

<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.
Let 

\[ S = \sum_{i=0}^{j-1} r^i. \]

In case 1, we have \( x > y \) so \( r > 1 \). \( S \in \Theta(r^j) \), so \( T(n) \in \Theta(n^x + n^yr^j) \). However, 

\[ r^j = (b^{x-y})^j = (b^j)^{x-y} = n^{x-y}. \]

Therefore 

\[ T(n) \in \Theta(n^x + n^yn^{x-y}) = \Theta(n^x). \]

In case 2, we have \( x = y \) so \( r = 1 \). \( S \in \Theta(j) = \Theta(\log n) \), so 

\[ T(n) \in \Theta(n^x + n^y \log n) = \Theta(n^x + n^x \log n) = \Theta(n^x \log n). \]

In case 3, we have \( x < y \) so \( r < 1 \). \( S \in \Theta(1) \), so 

\[ T(n) \in \Theta(n^x + n^y) = \Theta(n^y). \]

The complexity does not depend on the initial value \( d \).
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?
**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

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

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>:</td>
<td>:</td>
<td>:</td>
<td>:</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>

Must **sum** the values over all levels!
REVISITING THE RECURRENCE 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 \( T(n) \in \Theta(n(\log n)^2) \).
NEXT TIME

• More divide and conquer problems
• Non-dominated points
• Multiplication of large numbers