CS 341: ALGORITHMS

Trevor Brown

trevor.brown@uwaterloo.ca
THIS TIME

• Divide and conquer paradigm (intro to…)
• Merge sort
• Recurrence relations
  • Tree recursion method: merge sort as an example
  • Guess and check method
• [time permitting] Master theorem
DIVIDE AND CONQUER

A useful design strategy / paradigm
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., $S = \text{Combine}(S_1, \ldots, S_a)$. 
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 $\lfloor \frac{n}{2} \rfloor$ elements in $A$ and $A_R$ consists of the last $\lceil \frac{n}{2} \rceil$ 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.
DIVIDE
MERGE: CONQUER AND COMBINE
MERGE SIMULATION 1

L

\[
\begin{array}{cc}
5 & 31 \\
\uparrow & \uparrow \\
\end{array}
\]

R

\[
\begin{array}{cc}
12 & 21 \\
\uparrow & \uparrow \\
\end{array}
\]

O

\[
\begin{array}{cccc}
5 & 12 & 21 & 31 \\
\end{array}
\]
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
Algorithm: Mergesort($A : array; n : integer$)

if $n = 1$
then $S \leftarrow A$
\begin{align*}
   n_L & \leftarrow \left\lfloor \frac{n}{2} \right\rfloor \\
   n_R & \leftarrow \left\lfloor \frac{n}{2} \right\rfloor \\
   A_L & \leftarrow [A[1], \ldots, A[n_L]]
\end{align*}

else
\begin{align*}
   A_R & \leftarrow [A[n_L + 1], \ldots, A[n]] \\
   S_L & \leftarrow \text{Mergesort}(A_L, n_L) \\
   S_R & \leftarrow \text{Mergesort}(A_R, n_R) \\
   S & \leftarrow \text{Merge}(S_L, n_L, S_R, n_R)
\end{align*}

return $(S, n)$
While both arrays still contain elements to be merged

Handle case where the right array empties first (and left has some elements)

Handle case where the left array empties first

PSEUDOCODE FOR MERGE

```plaintext
Merge(AL, nL, AR, nR)
AOut = empty array of size nL+nR
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
```
## Analysis of Mergesort

**Algorithm:** `Mergesort(A : array; n : integer)`

- **if** $n = 1$  
  - then $S \leftarrow A$

- else
  
  \[
  \begin{align*}
  n_L &\leftarrow \left\lceil \frac{n}{2} \right\rceil &\text{O(1)} \\
  n_R &\leftarrow \left\lfloor \frac{n}{2} \right\rfloor &\text{O(1)} \\
  A_L &\leftarrow [A[1], \ldots, A[n_L]] &\text{O(n)} \\
  A_R &\leftarrow [A[n_L + 1], \ldots, A[n]] &\text{O(n)} \\
  S_L &\leftarrow \text{Mergesort}(A_L, n_L) &??
  \\
  S_R &\leftarrow \text{Mergesort}(A_R, n_R) &??
  \\
  S &\leftarrow \text{Merge}(S_L, n_L, S_R, n_R) &\text{O(n)}
  \end{align*}
  \]

- **return** $(S, n)$
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:

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

We use the recursion tree method

This is informal! We ignore floor/ceil by assuming $n=2^k$, and skip lots of math

MergeSort takes $O(n \log n)$ time on an input of size $n$. 

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

Total = $cn \log_2(n)$

$cn + cn + 2(cn/2) + 4(cn/4) + \ldots + n(c) = cn + cn + cn + \ldots = cn \log_2(n)$
RECURRANCE RELATIONS
Analysis tool for recursive algorithms
RECURRENCE 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.
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: \( T(0) = 4; \quad T(n) = T(n - 1) + 6n - 5 \) (substitute)

\[
\begin{align*}
T(n) &= (T(n - 2) + 6(n - 1) - 5) + 6n - 5 \\
&= T(n - 2) + 6n - 6 - 5 + 6n - 5 \\
&= T(n - 2) + 2(6n - 5) - 6
\end{align*}
\]

\[
\begin{align*}
&= (T(n - 3) + 6(n - 2) - 5) + 2(6n - 5) - 6 \\
&= T(n - 3) + 6n - 2(6) - 5 + 2(6n - 5) - 6 \\
&= T(n - 3) + 3(6n - 5) - 6(1 + 2)
\end{align*}
\]

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

\[
T(n - n) + n(6n - 5) - 6(1 + 2 + 3 + \cdots + (n - 1)) = guess(n)
\]
Recurrence: \( T(0) = 4 ; \quad T(n) = T(n - 1) + 6n - 5 \)

- \( \text{guess}(n) = T(n - n) + n(6n - 5) - 6(1 + 2 + 3 + \cdots + (n - 1)) \)
- \( = T(0) + 6n^2 - 5n - 6 \sum_{i=1}^{n-1} i \) (simplify)
- \( = 4 + 6n^2 - 5n - 6n(n - 1)/2 \)
- \( = 4 + 6n^2 - 5n - 3n^2 + 3n \)
- \( = 3n^2 - 2n + 4 \)

- Are we done?
- No! This is a **guess**.
- Must **prove** it is correct using induction
PROOF

• Recurrence: \( T(0) = 4; \quad T(n) = T(n-1) + 6n - 5 \)
• Our guess: \( guess(n) = 3n^2 - 2n + 4 \)
• Want to prove: \( guess(n) = T(n) \) for all \( n \)
• Base case: \( guess(0) = 3(0)^2 - 2(0) + 4 = T(0) \)
• Inductive case: assume \( guess(n) = T(n) \), and show \( guess(n+1) = T(n+1) \).
  • \( T(n+1) = T(n) + 6(n+1) - 5 \) \hspace{1cm} (by definition)
  • \( = guess(n) + 6(n+1) - 5 \) \hspace{1cm} (by inductive hypothesis)
  • \( = (3n^2 - 2n + 4) + 6(n+1) - 5 = 3n^2 + 4n + 5 \)
  • \( guess(n+1) = 3(n+1)^2 - 2(n+1) + 4 \) \hspace{1cm} (by definition)
  • \( = 3(n^2 + 2n + 1) - 2n - 2 + 4 = 3n^2 + 4n + 5 = T(n+1) \). QED.
ANOTHER APPROACH

• Instead of substituting and carefully identifying the pattern, and computing exact constants in our quadratic function, you might simply guess that the result is a quadratic function
  • $an^2 + bn + c$ for some unknown constants $a, b, c$
  • And then carry the unknown constants into the proof!
    • In the inductive step, you can determine what the constants must be, for the proof to work out
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 $T(n)$. 
**Master Theorem for Recurrences**

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

Suppose that $a > 1$ 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

$T(n) \in \begin{cases} 
\Theta(n^x) & \text{if } y < x \\
\Theta(n^x \log n) & \text{if } y = x \\
\Theta(n^y) & \text{if } y > x.
\end{cases}$
PROOF OF SIMPLIFIED MASTER THEOREM

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>\text{...}</td>
<td>\text{...}</td>
<td>\text{...}</td>
<td>\text{...}</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 OVER ALL LEVELS**

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

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

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

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

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

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.
MATH DETAILS FOR EACH CASE

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 \( T(n) \in \Theta(n^x + n^y r^j) \).

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

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

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: $a=?$  $b=?$  $y=?$  $x=\text{?}$

which $\Theta$ function?
GENERAL MASTER THEOREM

Suppose that $a > 1$ 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

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

Example recurrence:

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

Arbitrary function of $n$
(not just $cn^\gamma$)

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

| level \( j \) | \# nodes \( \begin{cases} 1 \\
 j - 1 \\
 j - 2 \\
 \vdots \\
 1 \\
 0 \end{cases} \) | value at each node \( \begin{cases} j2^j \\
 (j - 1)2^{j-1} \\
 (j - 2)2^{j-2} \\
 \vdots \\
 2^1 \\
 1 \end{cases} \) | value of the level \( \begin{cases} j2^j \\
 (j - 1)2^j \\
 (j - 2)2^j \\
 \vdots \\
 2^j \\
 2^j \end{cases} \) |

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

• More divide and conquer problems
  • Non-dominated points
  • Multi-precision multiplication (very likely)
  • Selection (somewhat likely)
  • Closest pair (unlikely)