Recurrence Equation

- A recurrence equation arises when an algorithm contains recursive calls to itself.
- Many divide and conquer algorithms give rise to recurrence equations of the form:
  \[ T(n) = aT\left(\frac{n}{b}\right) + f(n) \quad a, b \text{ constants} \]
  \[ T(1) = O(1) \text{ (typically)} \]
- These are generally solved in one of three ways:
  - Iterative method
  - Recursion-tree method
  - Master method
Merge Sort: Iterative Method 1

- From last time: For (2 –Way) Merge Sort, we know:
  \[ T(1) = 1 \]
  and
  \[ T(n) = 2 \ T(n/2) + n \]
  = \[ 2^k \ T(n/2^k) + k \ n \]
  = \[ 2^{\log_2 n} \ T(1) + (\log_2 n) n \]
  = \[ T(1) + (\log_2 n) n \]
  = \[ n (\log_2 n) \]

Merge Sort: Iterative Method 2

From last time: For (2 –Way) Merge Sort, we know:
\[ T(1) = 1 \] and \[ T(n) = 2 \ T(n/2) + n \]

Instead of proceeding as before, divide both sides by \( n \):

\[
\frac{T(n)}{n} = \frac{T(n/2)}{n/2} + 1
\]
\[
\frac{T(n/2)}{n/2} = \frac{T(n/4)}{n/4} + 1
\]
\[
\frac{T(n/4)}{n/4} = \frac{T(n/8)}{n/8} + 1
\]
... \[
\frac{T(n/2^{k-1})}{n/2^{k-1}} = \frac{T(n/2^k)}{n/2^k} + 1
\]
... \[
\frac{T(2)}{2} = \frac{T(1)}{1} + 1
\]
Merge Sort: Recursion Tree

From last time: For (2-Way) Merge Sort, we know:
\[ T(1) = 1 \quad \text{and} \quad T(n) = 2T(n/2) + n \]

Each node represents the cost incurred at that level of recursion:

<table>
<thead>
<tr>
<th>Level</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>n</td>
</tr>
<tr>
<td>1</td>
<td>2(n/2)=n</td>
</tr>
<tr>
<td>2</td>
<td>4(n/4)=n</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Total Cost:

Binary Search: Iterative Method

- \( T(n) = \quad \), \( T(1) = \)
Integer Multiplication: Recursive

- x, y are n digit numbers
  - Standard multiplication is $O(n^2)$
- Instead write
  
  $$x = 10^{n/2}a + b, \quad y = 10^{n/2}c + d$$
  where a, b, c, and d are $n/2$ digit numbers
- Then
  
  $$xy = 10^n ac + 10^{n/2}(a+b)(c+d) + bd$$

is a recursive equation for computing the product $xy$.

- Question: What is corresponding recursion equation for this algorithm? Is this a complexity class faster than using standard multiplication?

Note:

$$ad + bc = (a+b)(c+d) - ac - bd$$

Integer Multiplication: Solve recursion

- $T(n) = 4T(n/2) + n$
Gaussian Multiplication

- Gauss determined a method for multiplying two numbers (see note on integer multiplication slide) whose corresponding recursion equation is:
  
  \[ T(n) = 3T(n/2) + O(n) \]
  
  Is this faster than the standard algorithm?

Master Method: Setup

Suppose given a recurrence equation of the form

\[ T(n) = aT(n/b) + O(n^d) \]

with \(T(n) = O(1)\) for small \(n\) (generally we have \(T(1) = O(1)\)).

From previous work, we know:
- \(a\) = number of recursive calls per node (\(\geq 1\))
- \(b\) = shrinkage of input size per recursive call (\(\geq 1\))
- \(d\) = exponent in combining step (\(\geq 0\))
Master Method: Statement

- Given $T(n) = aT(n/b) + O(n^d)$ then

  \[
  T(n) = \begin{cases}
  O(n^d \log n) & \text{if } a = b^d \quad \text{(base doesn’t matter)} \\
  O(n^d) & \text{if } a < b^d \\
  O(n^{\log_b a}) & \text{if } a > b^d \quad \text{(base matters)}
  \end{cases}
  \]

Master Method: Examples

- Merge Sort:

- Integer Multiplication Mod:

- Gaussian Integer Multiplication:

- Strassen Matrix Multiplication (later): $T(n) = 7T(n/2) + n^2$
### Recurrence Relations to Remember

<table>
<thead>
<tr>
<th>( T(n) )</th>
<th>Algorithm</th>
<th>( \Theta(n) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(n) = T(n/2) + O(1) )</td>
<td>Binary Search</td>
<td>( O(\lg n) )</td>
</tr>
<tr>
<td>( T(n) = T(n-1) + O(1) )</td>
<td>Sequential Search</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>( T(n) = 2 \cdot T(n/2) + O(1) )</td>
<td>Tree Traversal</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>( T(n) = T(n-1) + O(n) )</td>
<td>Selection Sort</td>
<td>( O(n^2) )</td>
</tr>
<tr>
<td>( T(n) = 2 \cdot T(n/2) + O(n) )</td>
<td>Merge Sort</td>
<td>( O(n \lg n) )</td>
</tr>
</tbody>
</table>

### For Next Time

- So far we’ve covered chapters 1, 2, and 3 and parts of 4.
- Quicksort