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., $S = Combine(S_1,...,S_a)$.

D&C PROTO-ALGORITHM

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

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 **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 \( O(n) \) 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.

MERGE: CONQUER AND COMBINE

<table>
<thead>
<tr>
<th>105</th>
<th>7</th>
<th>8</th>
<th>13</th>
<th>14</th>
<th>19</th>
<th>105</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>31</td>
<td>4</td>
<td>5</td>
<td>10</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>98</td>
<td>5</td>
<td>12</td>
<td>21</td>
<td>31</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>58</td>
<td>12</td>
<td>21</td>
<td>31</td>
<td>19</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>19</td>
<td>11</td>
<td>19</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>21</td>
<td>31</td>
<td>19</td>
<td>11</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>14</td>
<td>19</td>
<td>11</td>
<td>4</td>
<td>10</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>19</td>
<td>11</td>
<td>4</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>21</td>
<td>31</td>
<td>19</td>
<td>4</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>16</td>
<td>31</td>
<td>4</td>
<td>10</td>
<td>12</td>
<td>16</td>
<td>31</td>
</tr>
<tr>
<td>98</td>
<td>5</td>
<td>12</td>
<td>21</td>
<td>31</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>58</td>
<td>12</td>
<td>21</td>
<td>31</td>
<td>19</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>19</td>
<td>11</td>
<td>19</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>12</td>
<td>21</td>
<td>31</td>
<td>19</td>
<td>11</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>14</td>
<td>19</td>
<td>11</td>
<td>4</td>
<td>10</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>19</td>
<td>11</td>
<td>4</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>21</td>
<td>31</td>
<td>19</td>
<td>4</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>16</td>
<td>31</td>
<td>4</td>
<td>10</td>
<td>12</td>
<td>16</td>
<td>31</td>
</tr>
</tbody>
</table>

MERGE SIMULATION

<table>
<thead>
<tr>
<th>( L )</th>
<th>( R )</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 10 16 98 5</td>
<td>105 13 14 19 11</td>
</tr>
<tr>
<td>4 5 10 12 21 31 96 98</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
<tr>
<td>105 8 13 14 19 11 4 10 16 98 8 12 21 31</td>
<td></td>
</tr>
</tbody>
</table>

PSEUDOCODE FOR MERGESORT

1. **Mergesort**(*A[1..n]*)
2. if \( n = 1 \) then return \( A \)
3. \( nL = \text{ceil}(n/2) \)
4. \( aL = A[1..nL] \)
5. \( aR = A[nL+1..n] \)
6. \( aL = \text{Mergesort}(aL) \)
7. \( aR = \text{Mergesort}(aR) \)
8. return **merge**(*aL*, *aR*)

PSEUDOCODE FOR MERGE

```python
while \( iL < nL \) and \( iR < nR \):
    if \( aL[iL] < aR[iR] \):
        \( aOut[iOut] = aL[iL] \)
        \( iL++ \)
    else:
        \( aOut[iOut] = aR[iR] \)
        \( iR++ \)
    \( iOut++ \)
while \( iL < nL \):
    \( aOut[iOut] = aL[iL] \)
    \( iL++ \)
while \( iR < nR \):
    \( aOut[iOut] = aR[iR] \)
    \( iR++ \)
return \( aOut \)
```

There are still elements left in both arrays.
Right array is out of elements.
Left array is out of elements.
ANALYSIS OF MERGESORT

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

RECURRENCE RELATIONS

A crucial 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.

RECURSION TREE METHOD

Evaluating recurrences with $T(n/2)$ terms

MATHEMATICALLY EXPRESSING THE COMPLEXITY OF MERGESORT

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

divide takes time $T(1)$

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

combine takes time $O(n)$

Recurrence relation:

$T(n) = \begin{cases} 
T(1) + T(1) + O(n) & \text{if } n > 1 \\
\Theta(1) & \text{if } n = 1 
\end{cases}$

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

$T(n) = \Theta(n \log n)$

Can also compute using a table...
Suppose we have the following recurrence:

\[ T(n) = \begin{cases} \frac{1}{2}, & \text{if } n > 0 \\ \frac{1}{2} + \frac{n}{2}, & \text{if } n = 0 \end{cases} \]

where \( n \) 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 reaches 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 \( T(n) \).

**Recursion Tree Method: Worked Example**

**Recurrence:**

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

**Step 1:**

- \( T(n-1) = T(n-1-1) + 6(n-1) - 5 \)
- \( T(n) = (T(n-2) + 6(n-1) - 5) + 6n - 5 \) (substitute)
- \( = T(n-2) + 2(6n-5) - 6 \) (try to preserve structure)
- \( = T(n-3) + 3(n-1) = 6n = 6 \) (substitute)

**Step 2:**

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

**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: 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

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

- Use \( 1 + 2 + \cdots + (n-1) = \frac{n(n-1)}{2} \)
- \( \text{guess}(n) = 4 + 6n^2 - 5n + 6n(n-1)/2 \) (simplify)
- \( = 3n^2 - 2n + 4 \)
- Are we done?
- The form of \( \text{guess}(n) \) was an educated guess.
- To be sure, we must prove it correct using induction

**Substitution Method: Worked Example**

**Recurrence:**

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

**Step 1:**

- \( T(n-1) = T(n-1-1) + 6(n-1) - 5 \)
- \( T(n) = (T(n-2) + 6(n-1) - 5) + 6n - 5 \) (substitute)
- \( = T(n-2) + 2(6n-5) - 6 \) (try to preserve structure)
- \( = T(n-3) + 3(n-1) = 6n = 6 \) (substitute)

**Step 2:**

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

**Recall:**

- \( T(0) = 4; \quad T(n) = T(n-1) + 6n - 5 \)
- \( guess(n) = 3n^2 - 2n + 4 \)
- 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 \( n \geq 0 \),
- Prove \( guess(n+1) = T(n+1) \)

- \( T(n+1) = T(n) + 6(n+1) - 5 \) (by definition)
- \( = guess(n) + 6(n+1) - 5 \)
- \( = 3n^2 - 2n + 4 + 6n + 1 - 5 \) (by inductive hypothesis)
- \( = 3n^2 + 4n + 5 \) (substitute)
- \( = 3(n+1)^2 - 2(n+1) + 4 \) (simplify)
- \( = 3n^2 + 4n + 5 = T(n+1) \) (by definition)
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:
  - \( T(n) = an^2 + bn + c \) 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

Recall: \( \text{guess}(n) = an^2 + bn + c \) where \( c = 4 \)

- Inductive case: \( \text{suppose } \text{guess}(n) = T(n) \) for \( n \geq 0 \)
- \( T(n+1) = an^2 + (b+6)n + 5 \) (continue previous slide)
- \( \text{guess}(n+1) = a(n+1)^2 + b(n+1) + 4 \) (by definition and \( c = 4 \))
- \( = an^2 + 2an + b + a + b + 4 \) (simplify, and...)
- \( = an^2 + (2a+b)n + (a+b+4) \) (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 into second to get \( b = 5 - 4 = 1 \)

Simplified Master Theorem

- Provides a formula for solving many recurrence relations
- We start with a simplified version
- Consider recurrence: \( T(1) = d \); \( T(n) = aT(n/2) + \Theta(n^e) \)
where \( a \geq 1, b \geq 2 \) and \( e \) is a power of \( \log \) (i.e., \( e = \log_b a \)

DERIVING THE SIMPLIFIED MASTER THEOREM

- \( T(n) = d + \sum_{i=0}^{\log_b n-1} \Theta(n^e) \) where \( a \geq 1, b \geq 2 \) and \( n = b^i \)

REARRANGING

- \( T(n) = da^i + \sum_{i=0}^{\log_b n-1} \Theta(n^e) \)
- Let \( l = \log_b a \)
- \( x \) relates # of subproblems to their size
- Rearranging we have \( b^x = a \)
- \( \text{So } T(n) = da^i + \sum_{i=0}^{\log_b n-1} \Theta(n^{ei}) \)
- \( = da^i + \sum_{i=0}^{\log_b n-1} \Theta(n^{(ei)/l}) \)
- Also \( da^i = d(a^i)^{1/l} = d(b^x)^x \)
- Hence \( n = b^i \) this is not \( x \)
- \( = da^i + \sum_{i=0}^{\log_b n-1} \Theta((b^x)^x) \)
- and we can simplify: let \( r = a^x \)
SOLVING THE GEOMETRIC SEQ

- Formula: \( T(n) = dn^x + cn^y \sum_{i=0}^{k} r^i \) where \( r = b^{-y} \)
- Geo. Seq. formula: \( \sum_{i=0}^{k} r^i = \begin{cases} \frac{r^{k+1} - 1}{r - 1} & \text{if } r > 1 \\ \frac{ja - a}{r^j - 1} & \text{if } r = 1 \\ \frac{ja - a}{j - 1} & \text{if } 0 < r < 1 \end{cases} \)

- So different solutions depending on \( r \)
  - Case 1: \( r = b^{-y} > 1 \) \( x - y > 0 \) \( x > y \)
  - Case 2: \( r = b^{-y} = 1 \) \( x - y = 0 \) \( x = y \)
  - Case 3: \( 0 < r = b^{-y} < 1 \) \( x - y < 0 \) \( x < y \)

Recall: \( T(0) = d \) so \( T(n) = T(n) + c \sum_{i=0}^{k} r^i \) where \( r = b^{-y} \)

\( T(n) = \Theta(n^y + n^x \sum_{i=0}^{k} r^i) \) since \( x = y \)

\( \sum_{i=0}^{k} r^i = \Theta(1) \) so \( T(n) = \Theta(n^x) \)

\( \Theta(n^x + n^y) = \Theta(n^x + n^y) \) since \( x = y \)

\( \text{So } T(n) = \Theta(n^x) \)

SOLVING THE GEOMETRIC SEQ

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

- Case 2: \( r = b^{-y} = 1 \) \( x - y = 0 \) \( x = y \)
- Case 3: \( 0 < r = b^{-y} < 1 \) \( x - y < 0 \) \( x < y \)

Recall \( b^i = n \), so \( \log_b b^i = \log_b n \). This means \( j \in \Theta(\log n) \).

\( \text{So } T(n) = \Theta(n^x + n^y \log n) = \Theta(n^x \log n) \)

SOLVING THE GEOMETRIC SEQ

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

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

Recall \( b^i = n \), so \( \log_b b^i = \log_b n \).

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

MASTER THEOREM FOR RECURRENCES

- Simplified version

Consider recurrence:
\( T(n) = af \left( \frac{n}{b} \right) + \Theta(n^x) \) where \( a \geq 1, b \geq 2 \) and \( n = b^y \)

And let \( x = \log_b n \)

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

SOME BONUS INTUITION FOR R CASES

Recall: \( T(n) = dn^x + cn^y \sum_{i=0}^{k} r^i \) where \( r = b^{-y} \)

- case 1: heavy top \( r > 1 \) \( y < x \) \( T(n) \in \Theta(n^x) \)
- case 2: balanced \( r = 1 \) \( y = x \) \( T(n) \in \Theta(n^x \log n) \)
- case 3: heavy bottom \( r < 1 \) \( y > x \) \( T(n) \in \Theta(n^y) \)

Heavy top means that the value of the recursion tree is dominated by the value of the leaf nodes.

Balanced means that the values of the levels of the recursion tree are constant (except for the last level).

Heavy bottom means that the value of the recursion tree is dominated by the value of the root node.
WORKED EXAMPLES

Recall: simplified master theorem

Suppose that $a \geq 1$ and $b > 1$. Consider the recurrence

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

where $n$ is a power of $b$. Denote $x = \log_b n$. Then

$$T(n) \in \begin{cases} a(x + 1) & \text{if } y < x \\ \Theta((b^y)^x) & \text{if } y = x \\ \Theta((b^y)^{2x}) & \text{if } y > x \end{cases}$$

Questions: $a=?$ $b=?$ $y=?$ $x=?$

which $\Theta$ function?

**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 $b^j$

$$\begin{align*} T(n) & \leq T\left(b^j\right) \\
& \in \begin{cases} \Theta((b^j)^x) & \text{if } y < x \\ \Theta((b^j)^{log_b n}) & \text{if } y = x \\ \Theta((b^j)^{\frac{n}{b^{j-1}}}) & \text{if } y > x \end{cases} \end{align*}$$

**REVISITING THE RECURSION TREE METHOD**

- Some recurrences with complex $f(n)$ functions (such as $f(n) = \log n$) can still be solved “by hand”

**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 $x = \log_b n$. Then

$$T(n) \in \begin{cases} a(x + 1) & \text{if } f(n) = O(n^{e-1}) \text{ for some } e > 0 \\ \Theta(n^x) & \text{if } f(n) = \Theta(n^x) \\ \Theta(n^{xlog_n a}) & \text{if } f(n) = \Theta(n^{xlog_n a}) \text{ is an increasing function of } n \\ \Theta(f(n)) & \text{if } f(n) = \Theta(f(n)) \text{ for some } e > 0 \end{cases}$$

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

<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>$j-1$</td>
<td>$2^j$</td>
<td>$2^j - 1$</td>
<td>$2^j - 1$</td>
</tr>
<tr>
<td>$j-2$</td>
<td>$2^j$</td>
<td>$2^j - 2$</td>
<td>$2^j - 2$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$0$</td>
<td>$2^j$</td>
<td>$2^j - 3$</td>
<td>$2^j - 3$</td>
</tr>
</tbody>
</table>

Note: $\log_2 n = j$
REVISITING THE RECURSION TREE METHOD

• Recall: \( n = 2^j; \ T(1) = 1, \ T(2) = n \log n \)

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

\[
T(n) = 2^j \left( 1 + \sum_{i=0}^{j-1} i \right) = 2^j \left( 1 + \frac{j(j+1)}{2} \right).
\]

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