First, let us draw the recursion tree for the merge procedure we saw last week. The tree is leaf dominated and the total number of leaves is \( n \). This allows us to guess the solution of this recurrence as \( \Theta(n) \). But we must verify this solution using induction.

This method of solving recurrences involves guessing the answer and then proving it true. This is often called the “substitution method.” Before we get into this complicated recurrence, let’s try to solve some simpler recurrences. First, let’s think through what it means when we write \( \theta(f(n)) \) in an expression (e.g., when we write \( 2W(n/2) + \theta(n) \) in the recurrence above). In these expression, we write \( \theta(f(n)) \) in place of some function \( g(n) \in O(f(n)) \). From the definition of \( O(\cdot) \), this means that there exist positive constants \( N_0 \) and \( c \) such that for all \( n \geq N_0 \), we have \( g(n) \leq c \cdot f(n) \). It follows that there exist constants \( k_1 \) such that for all \( n \geq 1 \), and \( k_2 \) such that

\[
g(n) \leq k_1 \cdot f(n) + k_2,
\]

where, for example, we can take \( k_1 = c \) and \( k_2 = \sum_{i=1}^{N_0} g(i) \), assuming \( f \) and \( g \) are non-negative functions.

By this argument, we can establish that

\[
W(n) \leq 2W(n/2) + k_1 \cdot n + k_2,
\]

where \( k_1 \) and \( k_2 \) are constants. Since this technique relies on guessing an answer, you can sometimes fool yourself by giving a false proof. The following are some tips to avoid common mistakes:

1. Spell out the constants. Do not use big-\( O \)—we need to be precise about constants, so big-\( O \) makes it super easy to fool ourselves.

2. Be careful that the induction goes in the right direction.

3. Add additional lower-order terms, if necessary, to make the induction go through.

We will start with a simple recurrence: Specifically, let’s try \( W(n) = 2W(n/2) + O(n) \). We know from master method, that the solution is \( O(n \lg n) \). We will prove the following theorem:

**Theorem 1** If \( W(n) \leq 2W(n/2) + k \cdot n \) for \( n > 1 \) and \( W(1) \leq k \) for \( n \leq 1 \), then for some constants \( \kappa_1 \) and \( \kappa_2 \),

\[
W(n) \leq \kappa_1 \cdot n \log n + \kappa_2.
\]
Proof. Let \( \kappa_1 = 2k \) and \( \kappa_2 = k \). For the base case \((n = 1)\), we check that \( W(1) = k \leq \kappa_2 \). For the inductive step \((n > 1)\), we assume that

\[
W(m) \leq \kappa_1 \cdot m \lg(m) + \kappa_2,
\]

for all \( m < n \). And we’ll show that \( W(n) \leq \kappa_1 \cdot n \log n + \kappa_2 \). To show this, we substitute an upper bound for \( W(n/2) \) from our assumption into the recurrence, yielding

\[
W(n) \leq 2W(n/2) + k \cdot n
\]

\[
\leq 2(\kappa_1 \cdot \frac{n}{2} \lg(n/2) + \kappa_2) + k \cdot n
\]

\[
= \kappa_1 n (\lg n - 1) + 2\kappa_2 + k \cdot n
\]

\[
= \kappa_1 n \lg n + \kappa_2 + (k \cdot n + \kappa_2 - \kappa_1 \cdot n)
\]

\[
\leq \kappa_1 n \lg n + \kappa_2,
\]

where the final step follows because \( k \cdot n + \kappa_2 - \kappa_1 \cdot n \leq 0 \) as long as \( n > 1 \).

Let’s do another one: \( W(n) \leq 2W(n/2) + O(\lg n) \). Again, from master method, we know the solution is \( O(n) \).

**Theorem 2** If \( W(n) \leq 2W(n/2) + k \cdot \lg n \) for \( n > 1 \) and \( W(1) \leq k \) for \( n \leq 1 \), then for some constants \( \kappa_1 \) and \( \kappa_2 \),

\[
W(n) \leq \kappa_1 \cdot n - \kappa_2 \cdot \lg n - \kappa_3.
\]

**Proof.** Let \( \kappa_1 = 2k \), \( \kappa_2 = \kappa_3 = k \). We begin with the base case. Clearly, \( W(1) = k \leq \kappa_1 - \kappa_3 \).

For the inductive step, we assume that \( W(m) \leq \kappa_1 \cdot m - \kappa_2 \cdot \lg m - \kappa_3 \) for all \( m < n \). We substitute the inductive hypothesis into the recurrence and obtain

\[
W(n) \leq 2W(n/2) + k \cdot \lg n
\]

\[
\leq 2(\kappa_1 \frac{n}{2} - \kappa_2 \lg(n/2) - \kappa_3) + k \cdot \lg n
\]

\[
= \kappa_1 n \lg n - 2\kappa_2 (\lg n - 1) - 2\kappa_3 + k \cdot \lg n
\]

\[
= (\kappa_1 n \lg n - \kappa_2 \lg n - \kappa_3) + (k \lg n - \kappa_2 \lg n + 2(\kappa_2 - \kappa_3))
\]

\[
\leq (\kappa_1 n \lg n - \kappa_2 \lg n - \kappa_3),
\]

where the final step uses the fact that \( (k \lg n - \kappa_2 \lg n + 2(\kappa_2 - \kappa_3)) \leq 0 \) by our choice of \( \kappa \)'s.

Now we can do the one we want to:

**Theorem 3** Show using induction that the recurrence \( W(n) = W(\alpha n) + W((1 - \alpha)n) + \Theta(\lg n) \) solves to \( \Theta(n) \).

**Proof.** We will try to prove that \( W(n) \leq c_1 n - c_2 \lg n \) **Base Case:** We need \( W(1) \leq c_1 n \).
We will guess, as an inductive hypothesis, that \(W(m) \leq c_1 m - c_2 \lg m\) for all \(m < n\) and for some constant \(c_1\) and \(c_2\). **Inductive Step:** Let us try to prove it for \(m = n\).

\[
\begin{align*}
W(n) & \leq W(\alpha n) + W((1 - \alpha)n) + c' \lg n \\
& \leq c_1 \alpha n - c_2 \lg(\alpha n) + c_1((1 - \alpha)n) - c_2 \lg((1 - \alpha)n) + c' \lg n \\
& = c_1 n - c_2 \lg n - (c_2 \lg n + c_2 \lg(\alpha(1 - \alpha)) - c' \lg n) \\
& \leq c_1 n - c_2 \lg n \text{ (if the residual (} c_2 \lg n + c_2 \lg(\alpha(1 - \alpha)) - c' \lg n \text{) } \geq 0) }
\end{align*}
\]

We can pick \(c_2\) large enough that the residual is always larger than 0.

**Work and Span of Parallel Merge Sort using Parallel Merge**

We can use this parallel merge procedure as a subroutine of merge sort and our work remains \(\Theta(n \lg n)\). If we substitute the span of merge back into the Merge Sort equation, we get

\[
S_{\text{MergeSort}}(n) = S_{\text{MergeSort}}(n/2) + S_{\text{Merge}}(n) \\
= S_{\text{MergeSort}}(n/2) + \Theta(\lg^2 n) \\
= \Theta(\lg^3 n)
\]

Therefore, the parallelism of this new merge sort procedure is \(\Theta(n \lg n /\lg^3 n) = \Theta(n /\lg^2 n)\). Therefore, we now have polynomial amount of parallelism.