With sequential algorithms, we often considered $T(n)$ (the runtime of the algorithm). Now, we’ll consider a more general notion:

Let $T_P(n)$ be the runtime of an algorithm using $P$ processors.

There are two important runtime quantities for a parallel algorithm:
- How long it would take if it were fully sequential (work)
- How long it would take if it were as parallel as possible (span)
For each “type” of tree, figure out work(−) and span(−) of findMin in terms of the number of nodes, $n$.

A (Parallel) Algorithm

```c
int findMin(Node current) {
    if (current is a leaf) {
        return current.data;
    }
    return min(current.data, findMin(left), findMin(right));
}
```

Degenerate Tree

```
1
  2
   3
    4
```

Perfect Tree

```
20
  30
    40
  60
    70
     80
```

```
A (Parallel) Algorithm

```java
int findMin(Node current) {
    if (current is a leaf) {
        return current.data;
    }

    return min(current.data, findMin(left), findMin(right));
}
```

To calculate work, we just do our standard analysis. First, we make a recurrence:

\[
\text{work}(n) = \begin{cases} 
0 & \text{if } n = 0 \\
O(1) & \text{if } n = 1 \\
\text{work}(0) + \text{work}(n-1) + 1 & \text{otherwise}
\end{cases}
\]
A (Parallel) Algorithm

```java
int findMin(Node current) {
    if (current is a leaf) {
        return current.data;
    }

    return min(current.data, findMin(left), findMin(right));
}
```

Degenerate Tree

To calculate span, we assume all calls are in parallel. We look for the longest dependence chain. We make a recurrence:

\[
\begin{align*}
\text{Span}(1) &= 0 \\
\text{Span}(n) &= \max(\text{Span}(0), \text{Span}(n-1)) + 1
\end{align*}
\]
A (Parallel) Algorithm

```java
int findMin(Node current) {
    if (current is a leaf) {
        return current.data;
    }

    return min(current.data, findMin(left), findMin(right));
}
```

To calculate work, we just do our standard analysis. First, we make a recurrence:

\[
\text{work}(1) = 1 \\
\text{work}(n) = 2 \cdot \text{work}(n/2) + 1
\]

\[\Theta(n)\]
A (Parallel) Algorithm

```c
int findMin(Node current) {
    if (current is a leaf) {
        return current.data;
    }

    return min(current.data, findMin(left), findMin(right));
}
```

To calculate span, we take the \( \text{max} \) of the recursive calls. First, we make a recurrence:

\[
\text{span}(1) = 1 \\
\text{span}(n) = \text{max} (\text{span}(n/2), \text{span}(n/2)) + 1
\]

\( \Theta (\log n) \)
A (Parallel) Algorithm

```c
int findMin(Node current) {
    if (current is a leaf) {
        return current.data;
    }

    return min(current.data, findMin(left), findMin(right));
}
```

To calculate span, we take the \textbf{max} of the recursive calls. First, we make a recurrence:

\[
\text{span}(n) = \begin{cases} 
    \mathcal{O}(1) & \text{if } n = 1 \\
    \max(\text{span}(n/2), \text{span}(n/2)) + \mathcal{O}(1) & \text{otherwise}
\end{cases}
\]

Master Theorem says this recurrence is \( \Theta(\lg n) \).

Again, this proves our intuition that parallelizing tree algorithms helps.

**But what does it mean for work to be \( \Theta(n) \) and span to be \( \Theta(\lg n) \)?**
Okay, but we don’t have $\infty$ processors... 

Consider $T_P$. We know the following:

- $T_P \geq \frac{T_1}{p}$,
Okay, but we don’t have $\infty$ processors...

Consider $T_P$. We know the following:

- $T_P \geq \frac{T_1}{P}$, the case where all the processors are always busy.
- $T_P \geq T_\infty$, $T_\infty$ is the length of the critical path which the algorithm must go through.

So, in an optimal execution, asymptotically, we know:

$$T_P \in \Theta\left(\frac{T_1}{P} + T_\infty\right)$$
Applying Our Asymptotic Bound

Minimum in a Perfect Tree

When calculating the minimum element in a tree, we had:

- $\text{work}(n) \in \Theta(n)$
- $\text{span}(n) \in \Theta(\lg n)$

So, we expect the algorithm to take $\mathcal{O}\left(\frac{n}{P} + \lg n\right)$

Another Example

Suppose we have the following work and span:

- $\text{work}(n) \in \Theta(n^2)$
- $\text{span}(n) \in \Theta(n)$

So, we expect the algorithm to take $\mathcal{O}\left(\frac{n^2}{P} + n\right)$