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)

# Analyzing a Parallel Algorithm



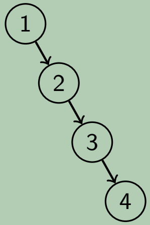
For each "type" of tree, figure out work(−) and span(−) of `findMin` in terms of the number of nodes, $n$.

## A (Parallel) Algorithm

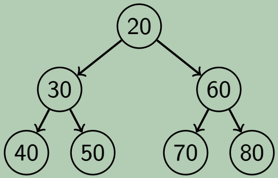
```
int findMin(Node current) {
   if (current is a leaf) {
      return current.data;
   }

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

## Degenerate Tree




## Perfect Tree

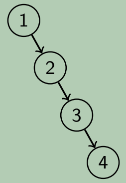


## A (Parallel) Algorithm

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

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

## Degenerate Tree




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

$$\begin{cases} \text{work}(1) = O(1) \\ \text{work}(n) = \text{work}(0) + \text{work}(n-1) + 1 \end{cases}$$

$\Theta(n)$

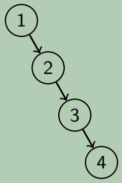
# Analyzing a Parallel Algorithm: Span of Degenerate Tree 

**A (Parallel) Algorithm**

```
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:

Span(1)

Span(n) = max(~~Span(0)~~, span(n-1)) + 1

$\Theta(n)$

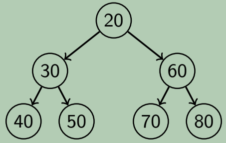
# Analyzing a Parallel Algorithm: Work of Perfect Tree 

**A (Parallel) Algorithm**

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

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

**Perfect Tree**




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

$$\text{work}(1) = 1$$

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

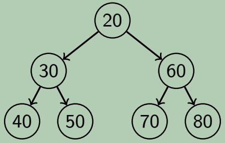
$$\Theta(n)$$

## A (Parallel) Algorithm

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

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

## Perfect Tree




To calculate span, we take the **max** of the recursive calls. First, we make a recurrence:

$$\text{Span}(1) = 1$$

$$\text{Span}(n) = \max(\text{Span}(n/2), \text{Span}(n/2)) + 1$$

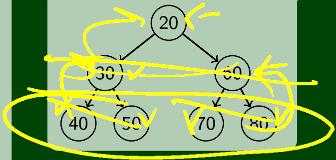
$$\Theta(\lg n)$$

## A (Parallel) Algorithm

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

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

## Perfect Tree




To calculate span, we take the **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:

- $T_P \geq \dfrac{T_1}{P}$,

**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:**

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

$\frac{n}{\lg n}$ $\quad$ $n$

### Minimum in a Perfect Tree

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

- work $(n) \in \Theta(n)$
- span $(n) \in \Theta(\lg n)$

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

work(n) ≥ span(n)

### Another Example

Suppose we have the following work and span:

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

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