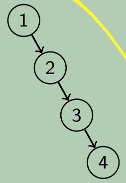
Analyzing a Parallel Algorithm: Work of Degenerate Tree

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

$$T(1) = 1$$

$$T(n) = T(0) + T(n-1) + 1$$

$$\Theta(n) \checkmark$$

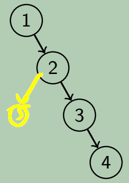
Analyzing a Parallel Algorithm: Span of Degenerate Tree

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:

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

$\text{Span}(n) = \max(\text{Span}(0), \text{Span}(n-1)) + 1$

$\Theta(n)$ ✓

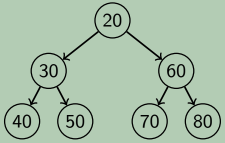
Analyzing a Parallel Algorithm: Work of Perfect Tree

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:

$$T(n) = 2T(n/2) + 1$$

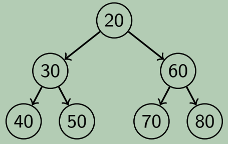
Analyzing a Parallel Algorithm: Span of Perfect Tree

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

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