14th of October, 2025

1. Consider the function $f : \mathbb{N} \to \mathbb{N}$ defined recursively below.

$$f(n) := \begin{cases} 2 & \text{if } n = 0 \\ 3 & \text{if } n = 1 \\ 3f(n-1) - 2f(n-2) & \text{if } n \geqslant 2 \end{cases}$$

Prove that $f(n) = 2^n + 1$ for all $n \in \mathbb{N}$.

Proof.

Basis Step:
We can clearly see that $f(0) = 2 = 1 + 1 = 2^0 + 1$.
Similarly, we have that $f(1) = 3 = 2 + 1 = 2^1 + 1$.

Inductive Step:
Let $k \in \mathbb{N}$ such that $k \geqslant 2$ and suppose, for all $j \in \mathbb{N}$, that if $j < k$, then $f(j) = 2^j + 1$.
Now, recall that $f(k) = 3f(k-1) - 2f(k-2)$ by the definition of f.

$$\begin{aligned} f(k) &= 3f(k-1) - 2f(k-2) && \text{by the definition of } f \\ &= 3(2^{k-1} + 1) - 2(2^{k-2} + 1) && \text{by the \textit{inductive hypothesis}} \\ &= 3 \cdot 2^{k-1} - 2 \cdot 2^{k-2} + 3 - 2 && \text{by distributing multiplication over addition} \\ &= 3 \cdot 2^{k-1} - 2^{k-1} + 1 && \text{because } 2 \cdot 2^{k-2} = 2^{k-1} \\ &= 2 \cdot 2^{k-1} + 1 && \text{because } 3x - 2x = x \text{ for all } x \in \mathbb{R} \\ &= 2^k + 1 && \text{because } 2 \cdot 2^{k-1} = 2^k \end{aligned}$$

Therefore, we know $f(n) = 2^n + 1$ for all $n \in \mathbb{N}$ as desired. Q.E.D.

2. Consider the function $g : \mathbb{N}_+ \to \mathbb{N}_+$ defined recursively below.

$$\begin{aligned} g(1) &:= 1 \\ g(z) &:= 2g(z-1) + 3 \quad \text{for all } z \in \mathbb{N} - \{0, 1\} \end{aligned}$$

Let's find a closed form for g by *unrolling*.

$$\begin{aligned} g(n) &= 2g(n-1) + 3 && = 2^1 g(n-1) + 2^0 \cdot 3 \\ &= 2\big(2g(n-2)+3\big) + 3 && = 2^2 g(n-2) + 2^0 \cdot 3 + 2^1 \cdot 3 \\ &= 2^2\big(2g(n-3)+3\big) + 2 \cdot 3 && = 2^3 g(n-3) + 2^0 \cdot 3 + 2^1 \cdot 3 + 2^2 \cdot 3 \\ & && \vdots \\ & && = 2^k g(n-k) + 3\sum_{i=0}^{k-1} 2^i && \text{... by unrolling the definition } k \text{ times} \\ & && \vdots \\ & && = 2^{n-1} g\big(n-(n-1)\big) + 3\sum_{i=0}^{n-1-1} 2^i && \text{We reach our base case when } n-k=1, \text{ which is equivalent to } k=n-1. \\ & && = 2^{n-1} g(1) + 3\sum_{i=0}^{n-2} 2^i \\ & && = 2^{n-1} + 3\left(2^{n-1} - 1\right) && \text{Recall } \sum_{i=0}^{m} 2^i = 2^{m+1} - 1 \text{ for all } m \in \mathbb{N}. \\ & && = 2^{n+1} - 3 \end{aligned}$$

Let's prove $g(n) = 2^{n+1} - 3$ for all $n \in \mathbb{N}_+$ by induction.

Proof.

Basis Step:
Observe $g(1) = 1 = 4 - 3 = 2^{1+1} - 3$.

Inductive Step:
Let $k \in \mathbb{N}_+$ and assume $(\forall \ell \in \mathbb{N}_+)(\ell < k \Rightarrow g(\ell) = 2^{\ell+1} - 3)$. Now, observe the following derivation.

$$\begin{aligned} g(k) &= 2g(k-1) + 3 && \text{by the definition of } g \\ &= 2(2^k - 3) + 3 && \text{by the } \textit{inductive hypothesis} \\ &= 2^{k+1} - 2 \cdot 3 + 3 \\ &= 2^{k+1} - 3 \end{aligned}$$

Therefore, $g(n) = 2^{n+1} - 3$ for all $n \in \mathbb{N}_+$ as desired. Q.E.D.

3. When we analyse the `binary_search` algorithm, we might describe the amount of computational work it takes to search through a list with the recursive function $T : \mathbb{N}_+ \to \mathbb{N}_+$ given below.

$$T(n) := \begin{cases} 4 & \text{if } n = 1 \\ T(n/2) + 4 & \text{if } n \geqslant 2 \end{cases}$$

Let's find a closed form for T by *unrolling*.

$$\begin{aligned} T(n) &= T(n/2) + 4 &&= T(n/2^1) + 1 \cdot 4 \\ &= T(n/4) + 4 + 4 &&= T(n/2^2) + 2 \cdot 4 \\ &= T(n/8) + 4 + 4 + 4 &&= T(n/2^3) + 3 \cdot 4 \\ & &&\vdots \\ & &&= T(n/2^k) + 4k \\ & &&\vdots \\ & &&= T\left(\frac{n}{2^{\log_2(n)}}\right) + 4\log_2(n) \\ & &&= T(1) + 4\log_2(n) \\ & &&= 4 + 4\log_2(n) \end{aligned}$$

. . . by unrolling the definition k times

We reach our base case when $n/2^k = 1$, which is equivalent to $k = \log_2(n)$.

Let's prove $T(n) = 4\log_2(n) + 4$ for all $n \in \mathbb{N}_+$ by induction.

Proof.

Basis Step:
Observe that $T(1) = 4 = 4 + 0 = 4 + \log_2(1)$.

Inductive Step:
Let $k \in \mathbb{N}_+$ and suppose, for all $j \in \mathbb{N}_+$, that $T(j) = 4\log_2(j) + 4$ whenever $j < k$. We can now plainly see the following.

$$\begin{aligned} T(k) &= T(k/2) + 4 && \text{by the definition of } T \\ &= \left(4\log_2(k/2) + 4\right) + 4 && \text{by the } \textit{inductive hypothesis} \\ &= 4\left(\log_2(k) - \log_2(2)\right) + 4 + 4 \\ &= 4\log_2(k) - 4 + 4 + 4 \\ &= 4\log_2(k) + 4 \end{aligned}$$

Therefore, $T(n) = 4\log_2(n) + 4$ for all $n \in \mathbb{N}_+$ as desired. Q.E.D.

4. When we analyse `merge_sort`, we might describe the amount of computational work it takes to sort a list with the recursive function $T : \mathbb{N}_+ \to \mathbb{N}_+$ given below.

$$T(n) := \begin{cases} 0 & \text{if } n = 1 \\ 2T(n/2) + n & \text{if } n \geqslant 2 \end{cases}$$

Let's find a closed form for T by *unrolling*.

$$\begin{aligned}
T(n) &= 2T(n/2) + n &&= 2^1 T(n/2^1) + 1n \\
&= 2\big(2T(n/4) + n/2\big) + n &&= 2^2 T(n/2^2) + 2n \\
&= 2\Big(2\big(2T(n/8) + n/4\big) + n/2\Big) + n &&= 2^3 T(n/2^3) + 3n \\
& &&\vdots \\
& &&= 2^k T(n/2^k) + kn && \text{...by unrolling the definition } k \text{ times} \\
& &&\vdots \\
& &&= 2^{\log_2(n)} T\left(\frac{n}{2^{\log_2(n)}}\right) + \log_2(n) n && \text{We reach our base case when } n/2^k = 1, \text{ which is equivalent to } k = \log_2(n). \\
& &&= nT(1) + n\log_2(n) \\
& &&= n\log_2(n)
\end{aligned}$$

Let's prove that $T(n) = n\log_2(n)$ for all $n \in \mathbb{N}_+$ by induction.

Proof.

Basis Step:
Observe that $T(1) = 0 = 1 \cdot \log_2(1)$.

Inductive Step:
Let $k \in \mathbb{N}_+$ and assume, for every positive natural number $m < k$, that $T(m) = m\log_2(m)$. We can now make the following observation.

$$\begin{aligned}
T(k) &= 2T\left(\frac{n}{2^k}\right) + k && \text{by the definition of } T \\
&= 2\left(\frac{k}{2}\log_2\left(\frac{k}{2}\right)\right) + k && \text{by the } \textit{inductive hypothesis} \\
&= \frac{2k}{2}\Big(\log_2(k) - \log_2(2)\Big) + k \\
&= k\Big(\log_2(k) - 1\Big) + k \\
&= k\log_2(k) - k + k \\
&= k\log_2(k)
\end{aligned}$$

Therefore, we conclude $T(n) = n\log_2(n)$ for all $n \in \mathbb{N}_+$ as desired.

Q.E.D.