S.KIMMEL

Recurrence & Time Complexity

When an algorithm has loops, we have tools for analyzing time complexity. What about recursive algorimths?

Factorial(n)
1. if(n==1): return 1
2. else: return n·Factorial(n-1)

T(n) is time complexity of Factorial(n)

- $T(n) = C$ if $n=1$ ← Initial conditions
- $T(n) = \underbrace{D}_{\substack{\text{constant \# of operations}\\ \text{in this call}}} + \underbrace{T(n-1)}_{\substack{+\text{ all operations done in}\\ \text{recursive call}}}$ if $n>1$ ← Recurrence Relation

} Recurrence Relation

some constant

e.g. $T(3) = D + T(2)$

$T(3) = D + D + T(1) = 2D + C$

* Sometimes can have multiple initial conditions

$T(n) = T(n-1) + T(n-2)$ ⇐ if recursive relation depends on more than just n-1

$T(1) = A$ $\quad$ $T(2) = B$

$T(3) = A + B$

S.KIMMEL

Solving a Recurrence Relation (iterative method)

$$T(n) = D + \underbrace{T(n-1)}$$
$$= D + D + T(n-2)$$
$$= D + D + D + T(n-3)$$

Looks like pattern is

$$T(n) = kD + T(n-k)$$

We know $T(1) = 1$

$n-k=1 \Rightarrow n-1=k$ Plug in k:

$$T(n) = (n-1)D + T(1) = nD + C - D = O(n)$$

Not really rigorous... Instead:

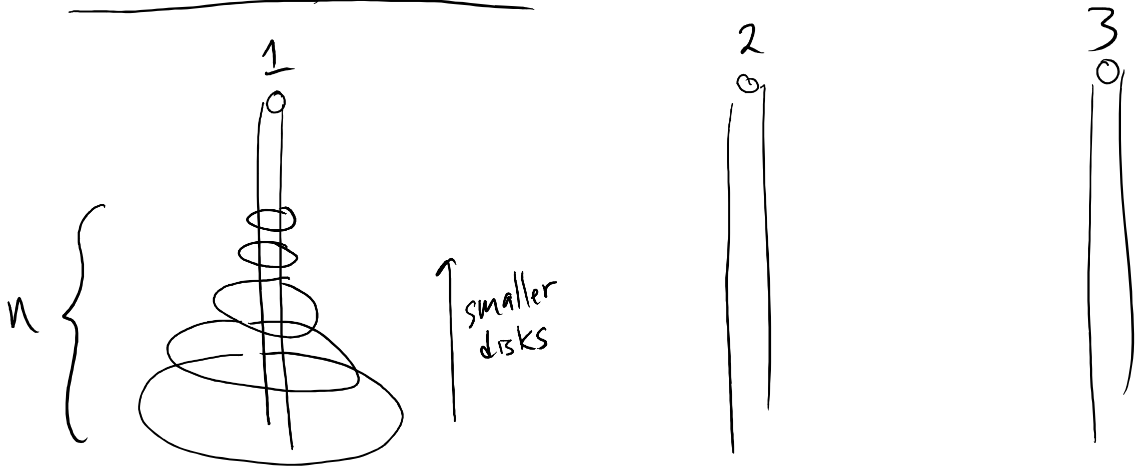
1. Use this process to guess solution
2. Use (strong) induction to prove.

We'll hopefully get back to this ~~next~~ later

S. KIMMEL

Another example...

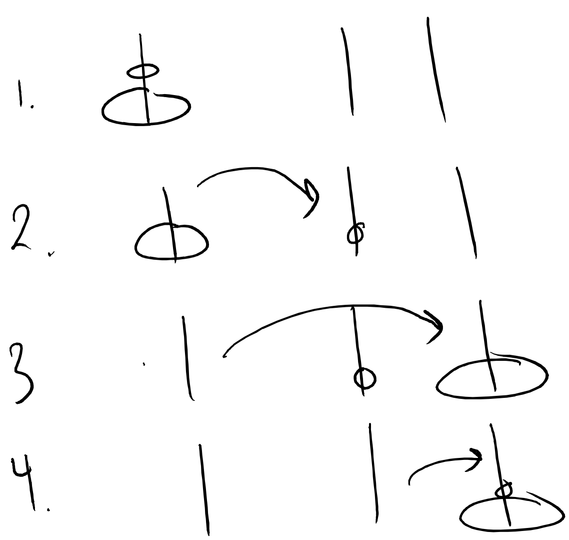
Towers of Hanoi



Task: Move Tower from 1 to 3 without putting larger disk on top of smaller.

Q: How many moves do you need?

ex



$n=2 \rightarrow 3$ moves

S. KIMMEL

Q: Which of the following recurrence relations is correct if $T(n) = \#$ of moves w/ n disks

A) $T(1) = 1$; $T(n) = 2 + T(n-1)$

B) $T(2) = 1$; $T(n) = 2T(n-1)$

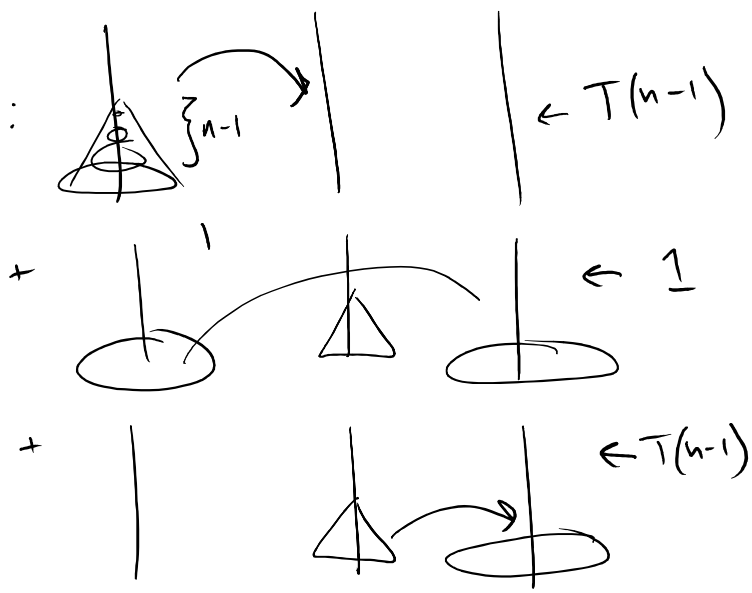
C) $T(1) = 1$; $T(n) = 2T(n-1)$

D) $T(1) = 1$; $T(n) = 2T(n-1) + 1$

Answer D:

$T(1) = 1$

$T(n)$:



S.KIMMEL

Solve for $T(n)$ using iterative method

$$
\begin{aligned}
T(n) &= 2T(n-1)+1 \\
&= 2\left(2\cdot T(n-2)+1\right)+1 = 4\cdot T(n-2)+2+1 \\
&= 4\cdot\left(2\cdot T(n-3)+1\right)+2 = 8\cdot T(n-3)+4+2+1 \\
&= 8\left(2\cdot T(n-4)+1\right)+4+2 = 16\cdot T(n-4)+8+4+2+1 \\
&\vdots \\
&= 2^k T(n-k) + \sum_{j=1}^{k-1} 2^j
\end{aligned}
$$

Base case: $n-k=1 \Rightarrow k=n-1$

$$
\begin{aligned}
&= 2^{n-1}T(1) + \sum_{j=0}^{n-2} 2^j \\
&= \underbrace{\sum_{j=0}^{n-1} 2^j = 1+2+4+8+16+32+\cdots+2^{n-2} = \frac{2^n-1}{2-1}}_{\text{Will prove in Hw}} \\
&= 2^n-1 = O(2^n)
\end{aligned}
$$

Worksheet!

S.KIMMEL