Solving Recurrences

Thomas Schwarz, SJ

Recurrences

- The equivalent of differential equations in the discrete
  - Calculate an amount based on differences or quotients
  - And one or more initial values
- Some categories are simple to solve
  - E.g. linear recurrences
    - f_n is a linear combination of previous values
    - $f_n = f_{n-2} + f_{n-3}; \quad f_1 = f_2 = f_3 = 1$ (Padovan numbers)
    - $f_n = 2f_{n-1} + f_n; \quad f_1 = 0, f_2 = 1$ (Pell numbers)

Recurrences

- Statements about sequences defined by recurrences are usually proven via induction
- Example: Pell numbers
  $p_n = 2p_{n-1} + p_{n-2};\ p_0 = 0,\ p_1 = 1$
- Matrix formula:
  - $\begin{pmatrix} p_{n+1} & p_n \\ p_n & p_{n-1} \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}^n$

Proving $\begin{pmatrix} p_{n+1} & p_n \\ p_n & p_{n-1} \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}^n$ for Pell numbers

- Induction Proof
  - Induction Base
  - Induction Step:
    - Induction Hypothesis
    - To show:

Proving $\begin{pmatrix} p_{n+1} & p_n \\ p_n & p_{n-1} \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}^n$ for Pell numbers

- Induction Proof
  - Induction Base
  - Induction Step:
    - Induction Hypothesis
    - To show:

Proving $\begin{pmatrix} p_{n+1} & p_n \\ p_n & p_{n-1} \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}^n$ for Pell numbers

- Induction Proof
  - Induction Base
    - For $n = 1$, the left side is

$$\begin{pmatrix} p_2 & p_1 \\ p_1 & p_0 \end{pmatrix} = \begin{pmatrix} 2 \cdot 1 + 0 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}^1$$

which is equal to the right side.

  - Induction Step:
    - Induction Hypothesis
    - To show:

Proving $\begin{pmatrix} p_{n+1} & p_n \\ p_n & p_{n-1} \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}^n$ for Pell numbers

- Induction Step
  - Induction Hypothesis:
    - $\begin{pmatrix} p_{n+1} & p_n \\ p_n & p_{n-1} \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}^n$
  - To show:
    - $\begin{pmatrix} p_{n+2} & p_{n+1} \\ p_{n+1} & p_n \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}^{n+1}$

Proving $\begin{pmatrix} p_{n+1} & p_n \\ p_n & p_{n-1} \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}^n$ for Pell numbers

- To show: $\begin{pmatrix} p_{n+2} & p_{n+1} \\ p_{n+1} & p_n \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}^{n+1}$

$$\text{LHS} = \begin{pmatrix} 2p_{n+1} + p_n & 2p_n + p_{n-1} \\ p_{n+1} & p_n \end{pmatrix}$$

$$= \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} p_{n+1} & p_n \\ p_n & p_{n-1} \end{pmatrix}$$

$$= \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}^n$$

$$= \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}^{n+1}$$

Pell Equation

- Use the power of Linear Algebra II
  - Calculate eigenvalues and eigenvectors and obtain

$$\begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix} = U \cdot \begin{pmatrix} 1-\sqrt{2} & 0 \\ 0 & 1+\sqrt{2} \end{pmatrix} \cdot U^{-1}$$

    - with

$$U = \begin{pmatrix} 1-\sqrt{2} & 1+\sqrt{2} \\ 1 & 1 \end{pmatrix} \qquad U^{-1} = \begin{pmatrix} -\frac{1}{2\sqrt{2}} & -\frac{-1-\sqrt{2}}{2\sqrt{2}} \\ \frac{1}{2\sqrt{2}} & -\frac{1-\sqrt{2}}{2\sqrt{2}} \end{pmatrix}$$

Pell Equation

- Why is this cool:
  - This Jordan decomposition works very well with matrix powers

    The core is $D = \begin{pmatrix} 1-\sqrt{2} & 0 \\ 0 & 1+\sqrt{2} \end{pmatrix}$

  - Then:
    $P^n = (UDU^{-1}) \cdot (UDU^{-1}) \cdot \ldots \cdot (UDU^{-1}) = UD^nU^{-1}$

    Where $D^n = \begin{pmatrix} (1-\sqrt{2})^n & 0 \\ 0 & (1+\sqrt{2})^n \end{pmatrix}$

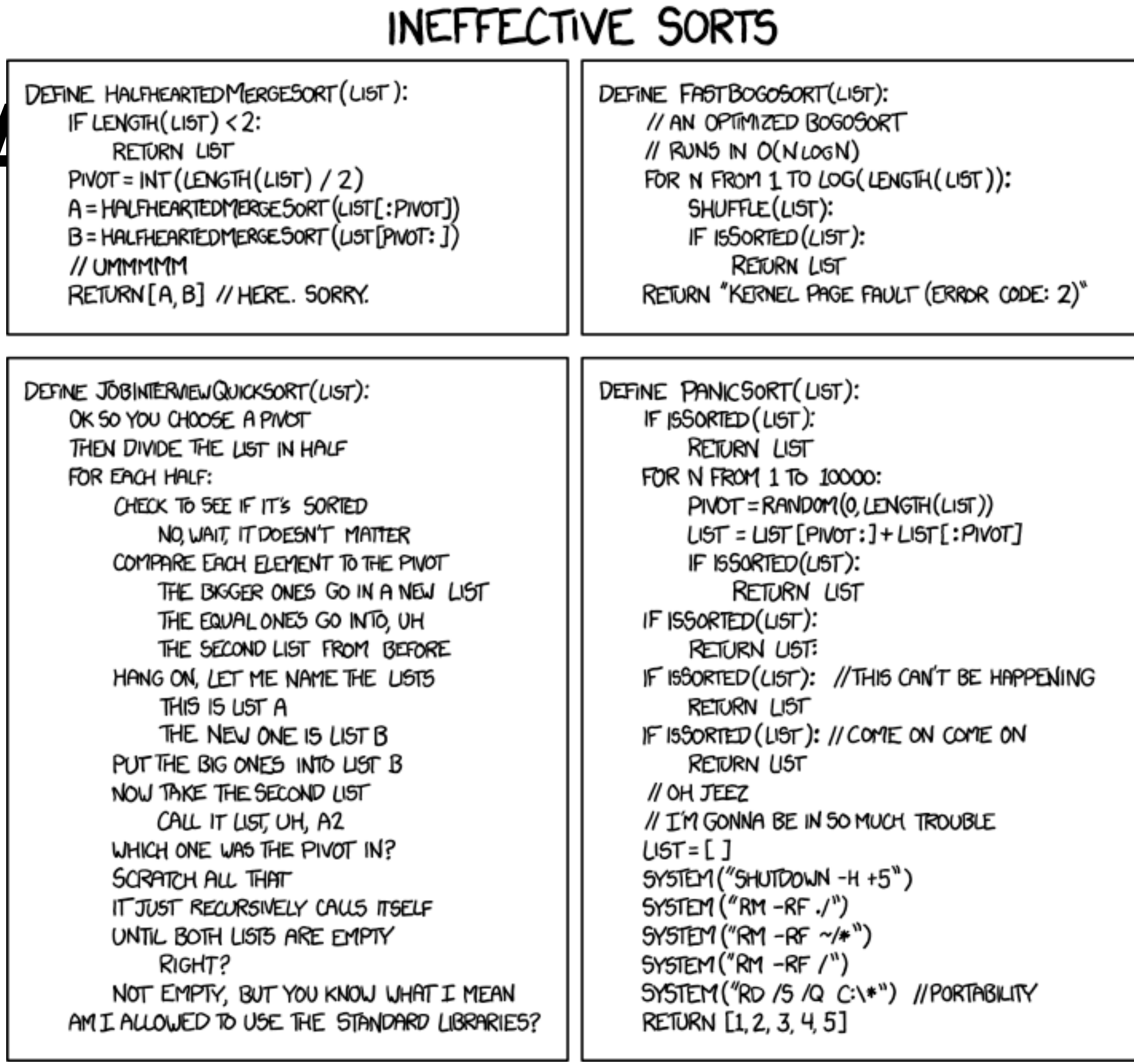
Pell Equation

- This gives us a nice formula, which we can also prove by induction:

$$p_n = \frac{(1+\sqrt{2})^n - (1-\sqrt{2})^n}{2\sqrt{2}}$$

- Since $1 + \sqrt{2} >> 1 - \sqrt{2}$ and the latter is negative, for large *n*

$$p_n \approx \frac{(1+\sqrt{2})^n}{2\sqrt{2}}$$


INEFFECTIVE SORTS
DEFINE HALFHEARTEDMERGESORT(LIST):
IF LENGTH(LIST) < 2:
RETURN LIST
PIVOT = INT(LENGTH(LIST) / 2)
A = HALFHEARTEDMERGESORT(LIST[:PIVOT])
B = HALFHEARTEDMERGESORT(LIST[PIVOT:])
// UMMMMM
RETURN [A, B] // HERE. SORRY.
DEFINE FASTBOGOSORT(LIST):
// AN OPTIMIZED BOGOSORT
// RUNS IN O(N LOGN)
FOR N FROM 1 TO LOG(LENGTH(LIST)):
SHUFFLE(LIST):
IF ISSORTED(LIST):
RETURN LIST
RETURN "KERNEL PAGE FAULT (ERROR CODE: 2)"
DEFINE JOBINTERVIEWQUICKSORT(LIST):
OK SO YOU CHOOSE A PIVOT
THEN DIVIDE THE LIST IN HALF
FOR EACH HALF:
CHECK TO SEE IF IT'S SORTED
NO, WAIT, IT DOESN'T MATTER
COMPARE EACH ELEMENT TO THE PIVOT
THE BIGGER ONES GO IN A NEW LIST
THE EQUAL ONES GO INTO, UH
THE SECOND LIST FROM BEFORE
HANG ON, LET ME NAME THE LISTS
THIS IS LIST A
THE NEW ONE IS LIST B
PUT THE BIG ONES INTO LIST B
NOW TAKE THE SECOND LIST
CALL IT LIST, UH, A2
WHICH ONE WAS THE PIVOT IN?
SCRATCH ALL THAT
IT JUST RECURSIVELY CALLS ITSELF
UNTIL BOTH LISTS ARE EMPTY
RIGHT?
NOT EMPTY, BUT YOU KNOW WHAT I MEAN
AM I ALLOWED TO USE THE STANDARD LIBRARIES?
DEFINE PANICSORT(LIST):
IF ISSORTED(LIST):
RETURN LIST
FOR N FROM 1 TO 10000:
PIVOT = RANDOM(0, LENGTH(LIST))
LIST = LIST[PIVOT:] + LIST[:PIVOT]
IF ISSORTED(LIST):
RETURN LIST
IF ISSORTED(LIST):
RETURN LIST:
IF ISSORTED(LIST): //THIS CAN'T BE HAPPENING
RETURN LIST
IF ISSORTED(LIST): // COME ON COME ON
RETURN LIST
// OH JEEZ
// I'M GONNA BE IN SO MUCH TROUBLE
LIST = [ ]
SYSTEM("SHUTDOWN -H +5")
SYSTEM("RM -RF ./")
SYSTEM("RM -RF ~/*")
SYSTEM("RM -RF /")
SYSTEM("RD /S /Q C:*") //PORTABILITY
RETURN [1, 2, 3, 4, 5]

Analysis of Quicksort

- We want to sort an array
- Idea of quicksort:
  - Pick a random pivot
  - Divide the array in elements smaller and larger than the pivot
  - Recursively order the two subarrays
  - Combine the two subarrays into one

Analysis of Quicksort

- Example of a divide and conquer algorithm:
  - We divide the array into two parts i.e. we divide the problem into sub-problems
  - We recursively sort the sub-arrays, i.e we solve the sub-problems
  - We combine the sub-arrays, i.e. we conquer the problem by combining the sub-problems

Analysis of Quicksort

- Ideally: Pivot is <u>always</u> in the middle
  - Then, time T to sort n elements is
    - $T(n) = T(n/2) + T(n/2) + cn$
      - Here, c is a constant representing the work of choosing the pivot, dividing the array and merging the arrays
    - An exact formula:
      - Would round up and down and be more clear on the linear work:
    - $T(n) = T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + O(n)$

Analysis of Quicksort

- How to solve $T(n) = T(n/2) + T(n/2) + cn$?
  - Substitution method:
    - Substitute formula into itself

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

$$= 2(2T(\frac{n}{4}) + c\frac{n}{2}) + cn \;\; = 4T(\frac{n}{4}) + cn + cn$$

$$= 8T(\frac{n}{8}) + cn + cn + cn$$

$$= \ldots = 2^m T(\frac{n}{2^m}) + m \cdot cn$$

Analysis of Quicksort

$$T(n) = 2^m T(\frac{n}{2^m}) + m \cdot cn$$

$$m = \lceil \log_2(n) \rceil :$$

$$T(n) = 2^m T(1) + cmn \leq (n+1)cmn = O(\log(n)n)$$

Analysis of Quicksort

- Worst behavior:
  - The pivot is the maximum or the minimum
    - One of the list is empty
    - The other list contains everything but the pivot
    - Recurrence is now
      - $T(n) = cn + T(n-1)$

Analysis of Quicksort

- Solving $T(n) = cn + T(n-1)$
- Substitution method:
  - $T(n) = cn + T(n-1)$
  - $= cn + c(n-1) + T(n-2)$
  - $= cn + c(n-1) + c(n-2) + T(n-3)$
  - $\dots$
  - $= c(n + (n-1) + (n-2) + \dots 2) + T(1)$
  - $= c\dfrac{(n+2)(n-1)}{2} + T(1)$

Analysis of Quicksort

- In the worst case, quicksort is quadratic

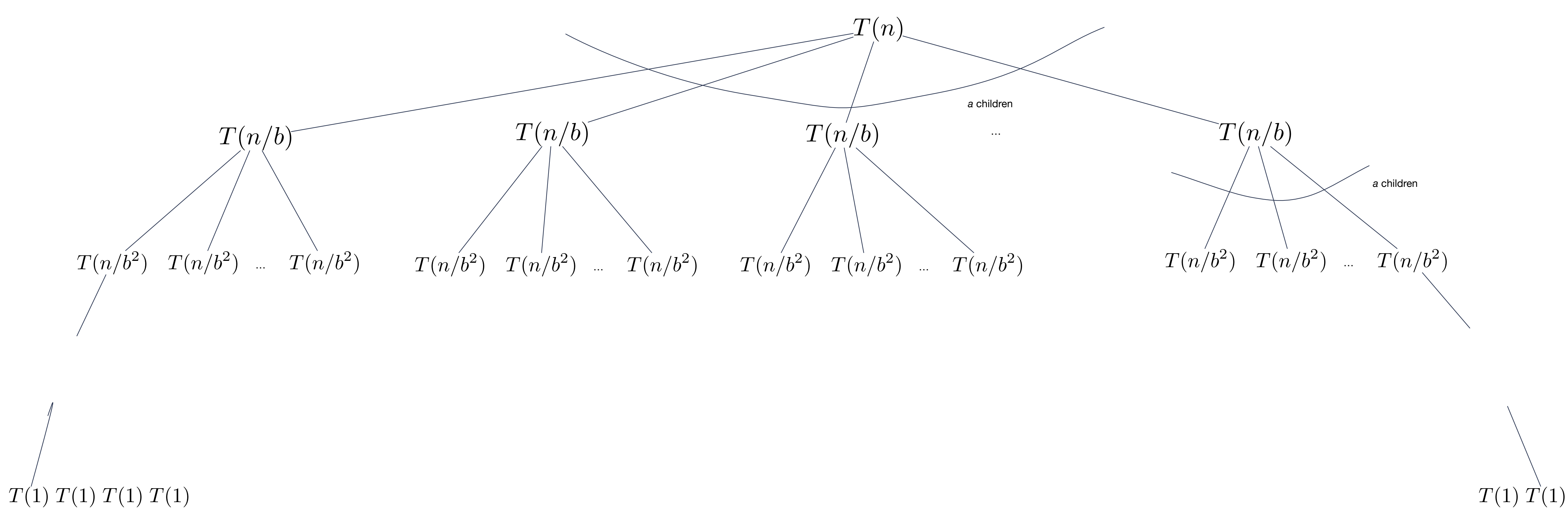
A family of Recursion Equations

- Divide and conquer frequently lead to recursions of the form

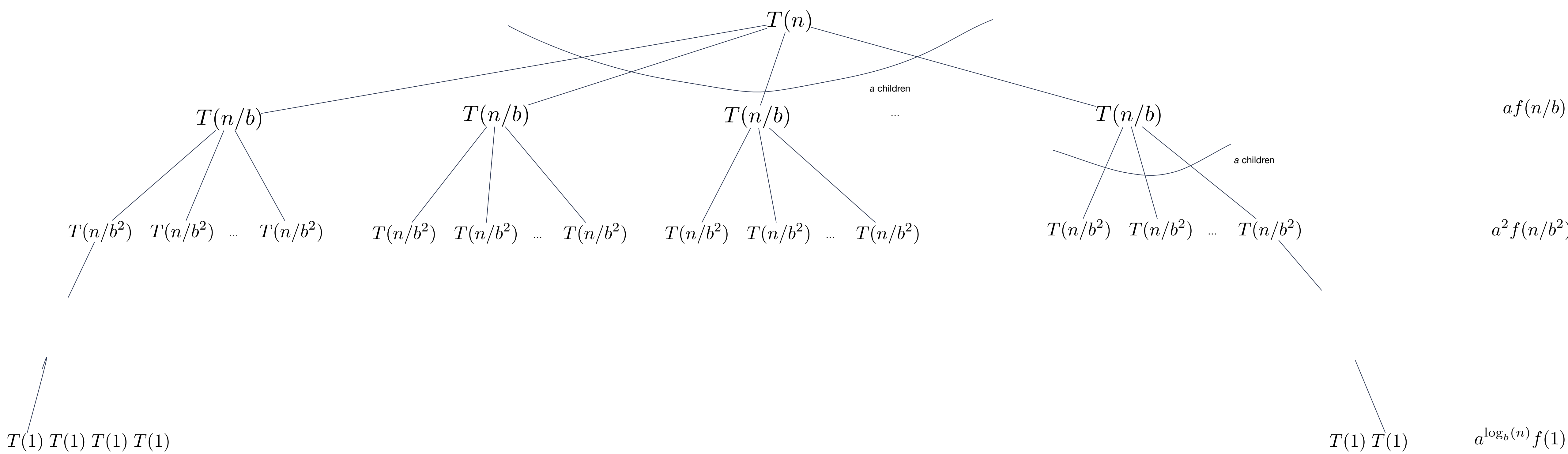
$$T(n) = aT(n/b) + f(n)$$

A family of Recursion Equations

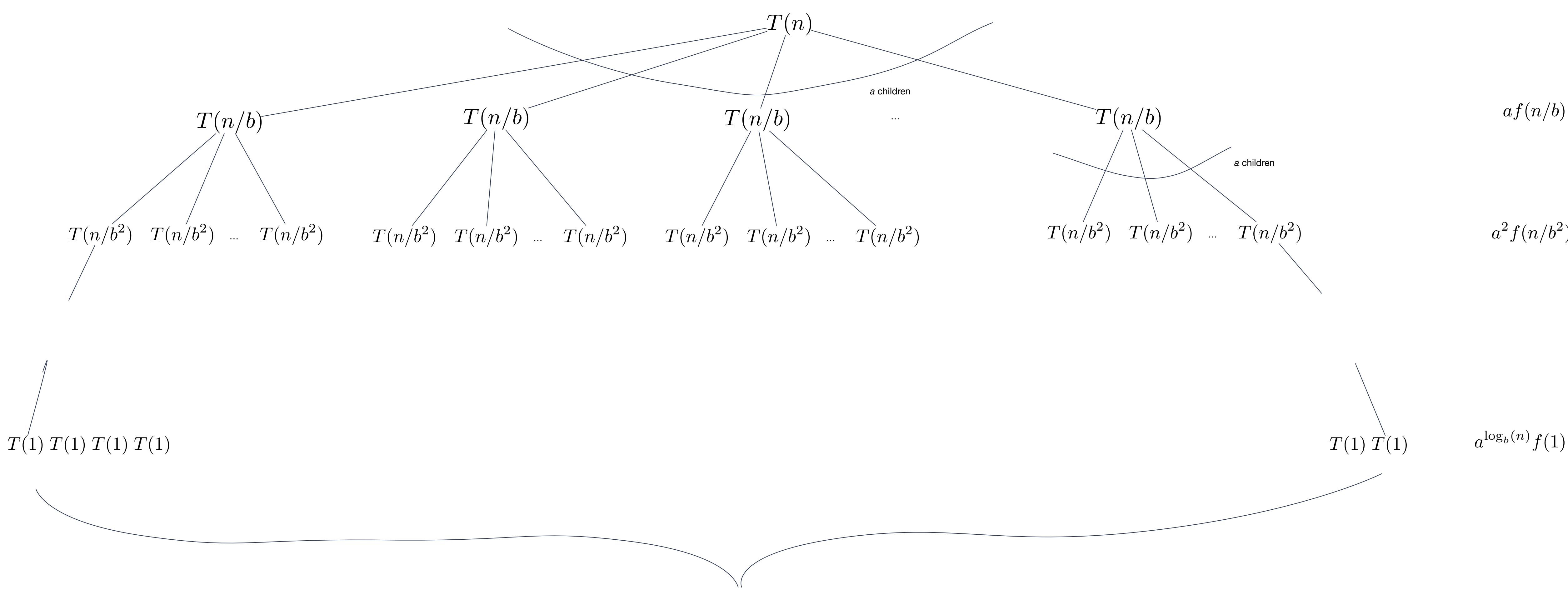
- Solve Recurrence using a tree:



A family of Recursion Equations



A family of Recursion Equations



$$a^{\log_b n} = \exp\log(a^{\log_b n}) = \exp(\log_b(n)\log a)) =$$

$$\exp(\log(n)\log(b)^{-1}\log(a)) = \exp(log_b(a)\log(n)) = \log(n^{\log_b(a)})$$

A family of Recursion Equations

- Total is

$$\sum_{j=0}^{\log_b n-1} a^j f(n/b^j) + cn^{\log_b a}$$

- Need to compare *f* with power of *n* in order to see what dominates

A family of Recursion Equations

$$f(n) = O(n^{\log_b a - \epsilon}) \implies T(n) = \Theta(n^{\log_b a})$$

$$f(n) = \Theta(n^{\log_b a}) \implies T(n) = \Theta(n^{\log_b a} \log n)$$

$f(n) = \Omega(n^{\log_b a + \epsilon})$ **and** $af(n/b) \leq cf(n)$ **eventually** $\implies T(n) = \Theta(f(n))$

A family of Recursion Equations

- There are gaps between the three cases, where the master theorem does not apply

Examples

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

$$n^{\log_2 2} = n = f(n)$$

Case 2

$$T(n) = \Theta(n \log n)$$

Examples

$$T(n) = 3T(n/2) + n$$

$$\log_2 3 = 1.58496$$

$$n = O(n^{\log_2 3 - 0.1})$$

$$\implies T(n) = \Theta(n^{\log_2 3})$$

Examples

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

$$a = 1,\ b = 2$$

$$\log_2 1 = 0$$

$$n = \Omega(n^{0+1/2})$$

$$\implies T(n) = \Theta(n)$$

Examples

$$T(n) = 3T(n/3) + n\log n$$

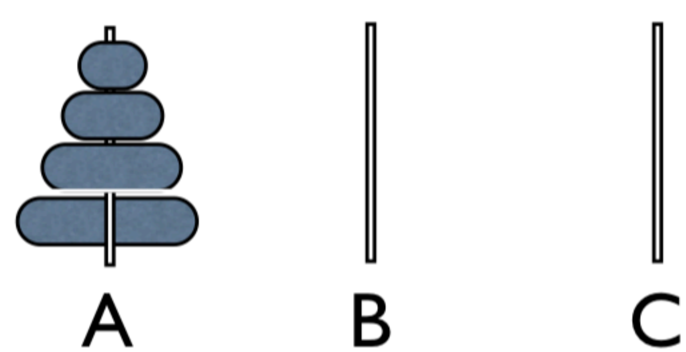
$a = 3$ $b = 3$ **so compare with** n

$n\log n \notin \Theta(n)$ $\quad n\log n \notin \Omega(n^{1+\epsilon})$

Falls into the gap between Case 2 and Case 3

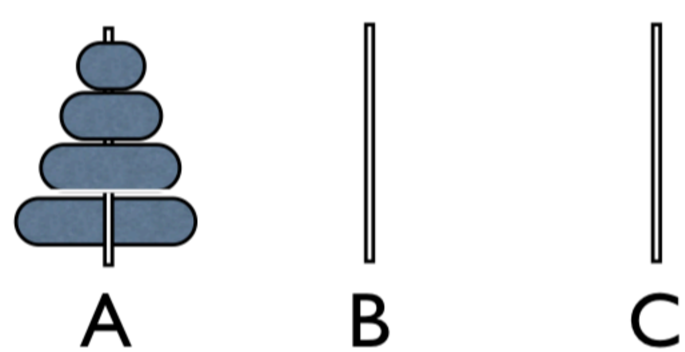
Tower of Hanoi

- *n* disks of *n* different parameters are on Peg A.
- Need to move them to Peg C subject to
  - Can only one disk at a time
  - Can only place smaller disk on bigger ones



Tower of Hanoi: Algorithm

- Recursive Solution
  - One disk: Just move the disk (1 move)
  - General case: Move top *n*-1 disks from A to C. Move remaining disk to B. Move *n*-1 disks from C to A



Tower of Hanoi: Evaluation

- If $T(n)$ is the number of moves for *n* disks, then
- $$T(1) = 1 \qquad T(n+1) = 2T(n) + 1$$

Solving the recurrence

$$
\begin{aligned}
T(n) &= 2T(n-1) + 1 \\
&= 2(2T(n-2) + 1) + 1 = 4T(n-2) + 2 + 1 \\
&= 2^3 T(n-3) + 4 + 2 + 1 \\
&= 2^4 T(n-4) + 2^3 + 2^2 + 1 \\
&= \vdots \\
&= 2^{n-1} + 2^{n-2} + \ldots 2^2 + 2^1 + 2^0 \\
&= 2^n - 1
\end{aligned}
$$

Tower of Hanoi: Proof

- Given the recurrence relation $T(1) = 1; \quad T(n+1) = 2T(n) + 1$
- Show that $T(n) = 2^n - 1$
- Proof by induction:
  - Base case: For $n = 1$, we have $T(1) = 1 = 2^1 - 1$
- Induction step:
  - Hypothesis: $T(n) = 2^n - 1$
  - To show: $T(n+1) = 2^{n+1} - 1.$
  - Proof:

$$T(n+1) = 2T(n) + 1 = 2(2^n - 1) + 1 = 2^{n+1} - 2 + 1 = 2^{n+1} - 1$$

The Upper Bound Trap

- What is wrong here.
  - Show that $T(1) = 1; \quad T(n+1) = 2T(n) + 1$ implies $T(n) \leq 2^n$
  - Induction base: same as before
  - Induction step:
    - Hypothesis: $T(n) = 2^n$
    - To show: $T(n+1) \leq 2^{n+1}$
    - Proof Attempt:

$T(n+1) = 2T_n + 1$ (recurrence)

$\leq 2 \cdot 2^n + 1$ (induction hypothesis)

$= 2^{n+1} + 1$

- And we are stuck

The Upper Bound Trap

- However, we can prove a **stronger** proposition and the proof goes through:
  - Show that $T(1) = 1; \quad T(n+1) = 2T(n) + 1$ implies $T(n) \leq 2^n - 1$
  - Induction base: same as before
  - Induction step:
    - Hypothesis: $T(n) \leq 2^n - 1$
    - To show: $T(n+1) \leq 2^{n+1}$
    - Proof:

$T(n+1) = 2T_n + 1$ (recurrence)

$\leq 2 \cdot (2^n - 1) + 1$ (induction hypothesis)

$= 2^{n+1} - 1$

- And we are done

Linear Recurrence Examples

- Pell numbers
  - $P_0 = 0,\ P_1 = 1,\ P_n = 2P_{n-1} + P_{n-2}$
- Example of linear recurrence
  - Assume solution is of the form a^n
  - This results in
    - $a^n = 2a^{n-1} + a^{n-2}$
  - We can divide by a^{n-2} to get
    - $a^2 = 2a + 1$
      - $\Rightarrow a^2 - 2a + 1 - 2 = 0 \Rightarrow (a-1)^2 = 2$
  - This means $a = 1 - \sqrt{2}$ or $a = 1 + \sqrt{2}$

Linear Recurrence Example

- Reversely, for these a : $a^n = 2a^{n-1} + a^{n-2}$
- Solutions are given by linear combinations
  - with $a_1 = 1 + \sqrt{2}$, $a_2 = 1 - \sqrt{2}$
  - $P_n = ca_1^n + da_2^n$
- Now we need to fit the two initial conditions
  - $ca_1^0 + da_2^0 = 0, ca_1^1 + da_2^1 = 1$
  - The first equation gives $c = -d$, the second gives $c(1+\sqrt{2}) - c(1-\sqrt{2}) = 1$, which is equivalent to $c = \dfrac{1}{2\sqrt{2}}$
  - Thus, the closed form is $P_n = \dfrac{(1+\sqrt{2})^n + (1-\sqrt{2})^n}{2\sqrt{2}}$