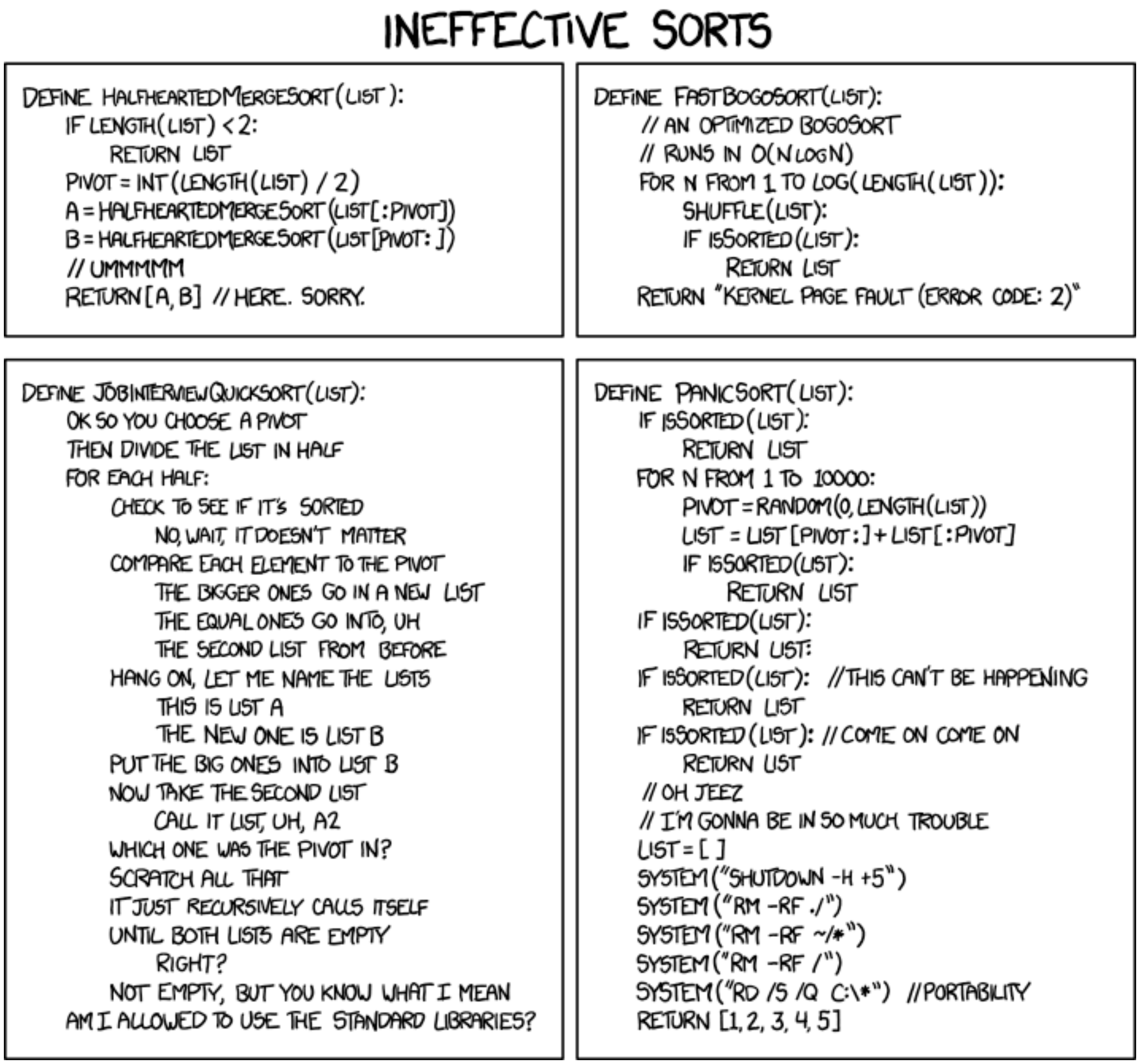
Solving Recurrence Relationships

Thomas Schwarz, SJ

Analysis of Quicksort

- You should have seen this before!
- 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



```
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 LOG N)
    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]
```

xkcd.com/1185

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 always 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 time to choose a pivot, divide the array, and to combine the arrays.
                - Dividing the array means looking at all elements
        - An exact formula would use rounding down and also take cognizance of the intricacies of dividing and combining
            - $T(n) = 2T(\lfloor n/2 \rfloor) + O(n)$

Analysis of Quicksort

- How to solve a recurrence $T(n) = T(n/2) + T(n/2) + cn$
  - Notice, that there is no base case.
    - This is typically, $T(1)$ is always some constant

Analysis of Quicksort

- How do we solve a recurrence like this?
  - Use Mathematica or a similarly sophisticated math tool
  - Guess a solution and use a proof by induction
  - Use substitution until you see a pattern and then prove the pattern by induction
  - Use a recurrence tree
  - Use the Master Theorem (from the book)

Analysis of Quicksort

- Substitution Method:

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

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

$$= 4\left(2T(\frac{n}{8}) + c\frac{n}{4}\right) + cn + cn = 8T(\frac{n}{8}) + cn + cn + cn$$

$$= \vdots$$

$$= C + cn + \ldots + cn + cn + cn$$

Analysis of Quicksort

- $T(n) = C + cn + \ldots + cn + cn + cn$
- How many addend cn?
  - We get an addend each time we divide by 2
  - Can divide n $\log_2(n)$ before getting 1
  - Therefore:
  - $T(n) = \log_2(n)cn + C = O(\log(n)n)$

Analysis of Quicksort

- Now we need to prove it.
  - We start with the induction step
    - Hypothesis: $T(n) \leq C \log_2(n)n$
    - To show: $T(n+1) \leq C \log_2(n+1)(n+1)$
  - That is awkward, so we do not do this
  - Use STRONG INDUCTION instead
    - Hypothesis: $T(m) \leq C \log_2(m)m$ for all $m < n$
    - To show: $T(n) \leq C \log_2(n)n$
  - This one can use the recursion
  - Notice, we did not specify $C > 0$

Analysis of Quicksort

- We calculate:

$$
\begin{aligned}
T(n) &= 2T(\frac{n}{2}) + cn \quad \text{(Recurrence formula with a different c)} \\
&\leq C\log_2(\frac{n}{2})\frac{n}{2} + cn \quad \text{(Using the strong hypothesis)} \\
&= C(\log_2(n) - 1)\frac{n}{2} + cn \\
&\leq C(\log_2(\frac{n}{2})\frac{n}{2} + cn \\
&= C\log_2(n)n + (cn - C\log_2(n)\frac{n}{2}) \\
&\leq C\log_2(n)n \quad \text{IF the right parenthesis is negative}
\end{aligned}
$$

- by adding and subtracting the desired expression

Analysis of Quicksort

- The correction should be negative:

$$cn - C\log_2(n)\frac{n}{2} \;\leq\; 0$$

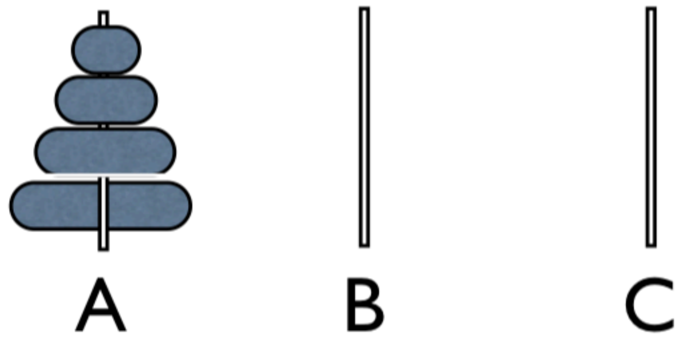
$$\iff cn \leq C\log_2(n)\frac{n}{2}$$

$$\iff 2c \leq C\log_2(n)$$

- which is true if $n \geq 4$ and $C \geq c$.
- We also need to make C large enough so that $T(2) \leq C$.
- Exact analysis is mathematically more involved!

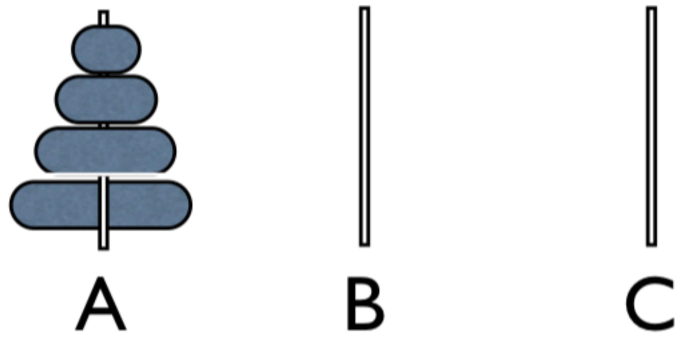
Tower of Hanoi

- *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

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

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:

$$
\begin{aligned}
T(n+1) &= 2T_n + 1 \quad \text{(recurrence)} \\
&\leq 2 \cdot (2^n - 1) + 1 \quad \text{(induction hypothesis)} \\
&= 2^{n+1} - 1
\end{aligned}
$$

- 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 $P_n = \dfrac{(1+\sqrt{2})^n + (1-\sqrt{2})^n}{2\sqrt{2}}$

An ugly recurrence

- Let's look at $T(n) = \sqrt{n}T(\sqrt{n}) + n$.
- First try: $T(n) = O(n \log n)$
  - Assume $T(n) \leq Cn \log n$
  - Induction step:
    - $T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n$ (recurrence)
    - $= \sqrt{n}C\sqrt{n} \log(\sqrt{n}) + n$ (ind. hyp.)
    - $= Cn\frac{1}{2} \log n + n$ (algebra)
    - $\leq Cn \log(n)$ (if $n \leq n\frac{C}{2} \log n$)

An ugly recurrence

- Condition $n \leq n\frac{C}{2}\log n$ is true if and only if
  - $1 \leq \frac{C}{2}\log(n)$
- which is always true if n is large enough
- Thus indeed: $T(n) = O(n\log n)$
- An easy proof usually means that we were not aggressive enough
-

An ugly recurrence

- Can we prove that $T(n) = \Omega(n \log n)$?
    - If we assume $T(n) \geq Dn \log(n)$, what happens
        - $T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n$ (Recurrence)
        - $\geq \sqrt{n} \cdot D\sqrt{n} \log(\sqrt{n}) + n$ (I. H.)
        - $= \frac{D}{2} n \log(n) + n$
        - $\geq Dn \log(n)$ only if $1 > \frac{D}{2} \log(n)$
- But this is never true for large n: $T(n) \notin \Omega(n \log n)$

An ugly recurrence

- Given $T(n) = \sqrt{n}T(\sqrt{n}) + n$
- Let's try whether $T(n) = \Theta(n)$.
  - Your turn: Show that $T(n) \geq n$.

An ugly recurrence

- Solution:
  - $T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n \geq n$

An ugly recurrence

- But can we show $T(n) = O(n)$?
- Your turn

An ugly recurrence

- Solution:
  - $T(n) = \sqrt{n}T(\sqrt{n}) + n$ recurrence
  - $\leq \sqrt{n}C\sqrt{n} + n$ I.H.
  - $= Cn + n$ algebra
  - $= C(n+1)$
  - $\nleq Cn$ **So, we are stuck**

An ugly recurrence

- Need something between n and $n\log(n)$
  - Let's try $T(n) = \Theta(n\log(\log(n))$
  - $T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n$ (recurrence)
  - $\leq \sqrt{n} \cdot C\sqrt{n}\log(\log(\sqrt{n})) + n$
  - $= Cn\log(\dfrac{\log(n)}{2}) + n$
  - $= Cn\log(\log(n)) - Cn\log(2) + n$
  - $= Cn\log(\log n)) - Cn + n$ (log base 2)
  - which works with $C > 1$, $T(n) = O(n\log(\log(n))$
- (For the induction base we can pick C large enough)

An ugly recurrence

- Your turn:
  - Show $T(n) = \Omega(n \log(\log(n)))$

An ugly recurrence

- Solution:
  - $T(n) = \sqrt{n} \cdot T(\sqrt{n}) + n$ (recurrence)
  - $\geq \sqrt{n} \cdot D\sqrt{n} \log(\log(\sqrt{n})) + n$
  - $= Dn \log(\frac{\log(n)}{2}) + n$
  - $= Dn \log(\log(n)) - Dn \log(2) + n$
  - $= Dn \log(\log n)) - Dn + n$ (log base 2)
- which works if $D \leq 1$.