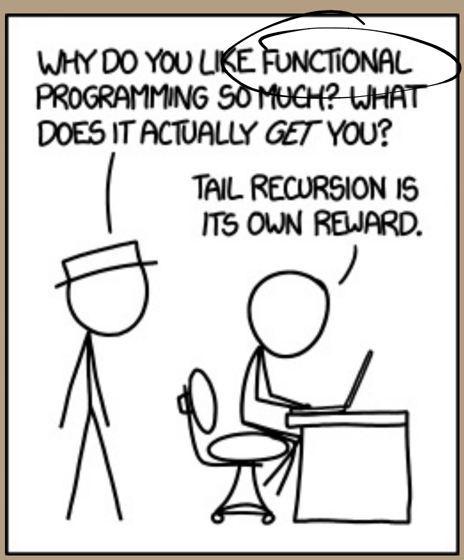
Algorithms



Recursion

Recap/notes

- Perusall: • open from Canvas
  • 5 comments each from here out, please
- HW0: Wednesday.
  Note: You submit in groups! Please add to one (even if solo). Before adding to an existing one, ask!
- Office hours:
  Tomorrow 10am
  (on discord, switch to Zoom if needed)

Recursion

- If you can solve directly (usually because input is small), do it!
- Otherwise, reduce to simple (usually smaller) instances of the same problem.

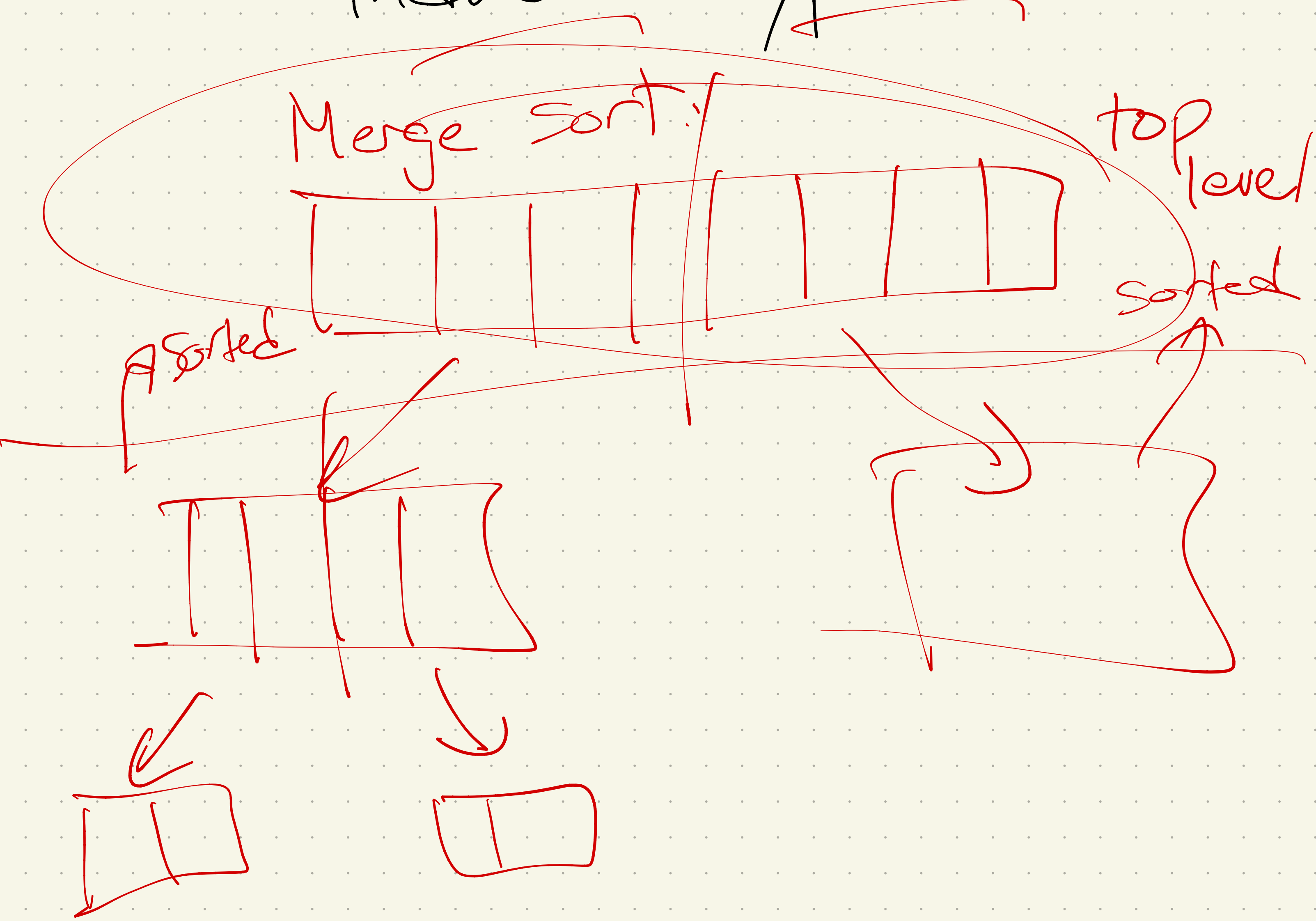
Result:

Code that calls smaller instance of itself.

Recursion Fairy

- Helps to solidify that "black box" mentality, so you don't keep unpacking the next level.

(She's also called the "induction hypothesis".)



Towers of Hanoi: runtime

Let $H(k)$ = runtime of size k instance

```
HANOI(n, src, dst, tmp):
  if n > 0
    HANOI(n − 1, src, tmp, dst)    ⟨⟨Recurse!⟩⟩
    move disk n from src to dst
    HANOI(n − 1, tmp, dst, src)    ⟨⟨Recurse!⟩⟩
```

Figure 1.4. A recursive algorithm to solve the Tower of Hanoi

1 op − $O(1)$

$H(n-1)\cdot 1$

+1 move

$+\ H(n-1)\cdot 1$

How?? (no loop, + calls itself!)

Goal: calculate $H(n)$

$$H(n) = 2H(n-1) + k\,O(1)$$

if, move disk 1

$$a_n = 2a_{n-1} + 1$$

char eqn → homogenous

inhomogeneous (in Rosen)

$$H(n) = 2(H(n-1)) + 1$$

"unrolling"

$$= 2[2H(n-2) + 1] + 1$$

$$= 2(2(2H(n-3)+1)+1] + 1 = \dots$$

$$= \underbrace{2(2(2(2\cdots}_{n}(H(0)+1)+1)+1)+1 \in O(2^n)$$

$H(0) = 1$

4, 2, 1

$$G(n) = \underbrace{G(n-1)} + 1$$ "unroll"

$$= (\underbrace{G(n-2)} + 1) + 1$$

$$= ((G(n-3) + 1) + 1) + 1$$

$$\vdots$$

$$= \underbrace{((((\cdots(G(1)+1)+1)+1)+1)\cdots)}_{\text{repeat } n-1 \text{ times}} + 1$$

= add 1, n times

$$= O(n)$$

if $n > 1$
recurse on $n-1$
do $O(1)$ work

For loop: $i = n \dots 1$

Q: Thm, lemma, corollary, etc.
The difference?

there isn't - one is
just "bigger"

lemma = small thm

In merge, lemma is
actually harder!
Not length, but
"importance."
In this book, main
runtime/correctness is
thm, subroutines
get lemmas.

Lemma: Merge subroutine correct merges $A[1..m]$ & $A[m+1..n]$ (assuming they are sorted.)

```
MERGE(A[1..n], m):
  i ← 1; j ← m+1
  for k ← 1 to n
    if j > n
        B[k] ← A[i]; i ← i+1
    else if i > m
        B[k] ← A[j]; j ← j+1
    else if A[i] < A[j]
        B[k] ← A[i]; i ← i+1
    else
        B[k] ← A[j]; j ← j+1
  for k ← 1 to n
    A[k] ← B[k]
```

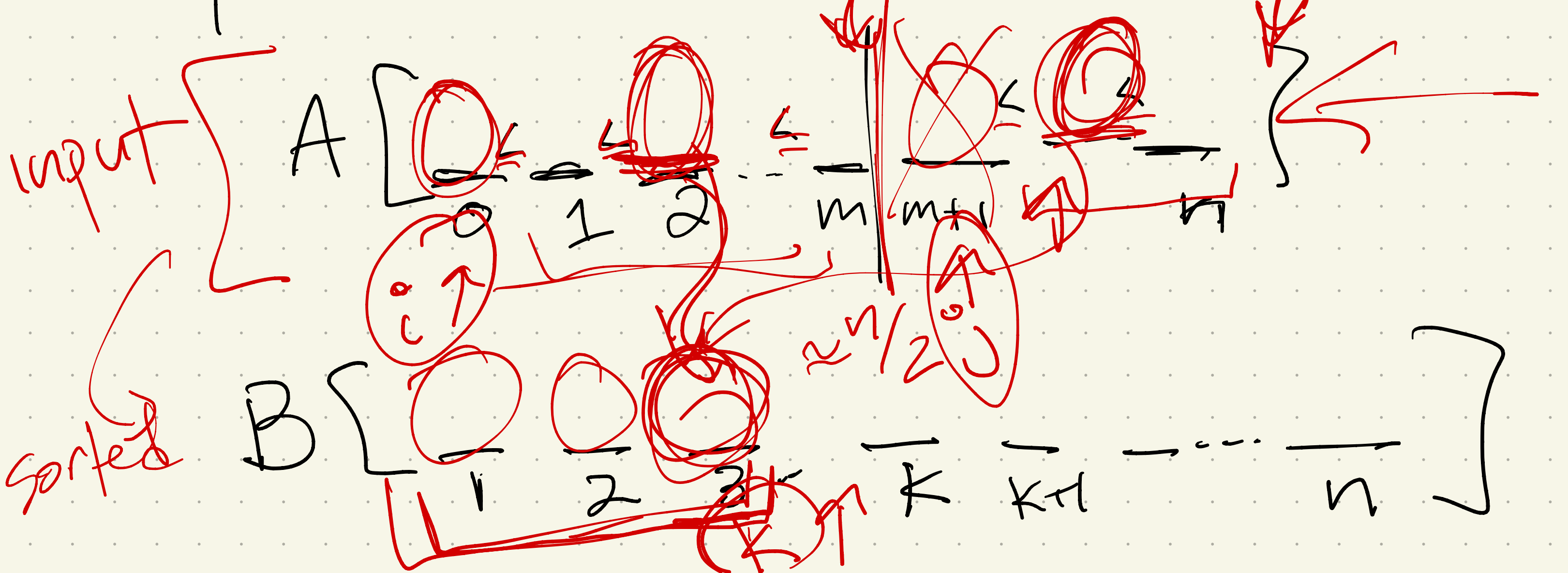
Pf: Assume $A[1..n]$ as

induction: on the loop! k

his: $n-k$

Show: $\forall k \in [0..n]$ if first k are sorted into B, then last part works too.

show iteration $k+1$ work



So: Base case: ~~$k>n$, so on "last" loop iteration: So $B[1..n]$ is sorted!~~

$k=0$ (either way)

Ind Hyp: assume worked up to $(k-1)^{st}$ loop iteration

$\rightarrow A[\underset{\uparrow i}{1} \; \dots \; m \mid m+1 \; \dots \; \underset{\uparrow j}{} \; n]$ $\quad i \le m+1 \quad j \le n+1$

$\rightarrow B[\underbrace{1 \; \dots \; k-1}_{\text{sorted}} \mid k \; \dots \; n]$

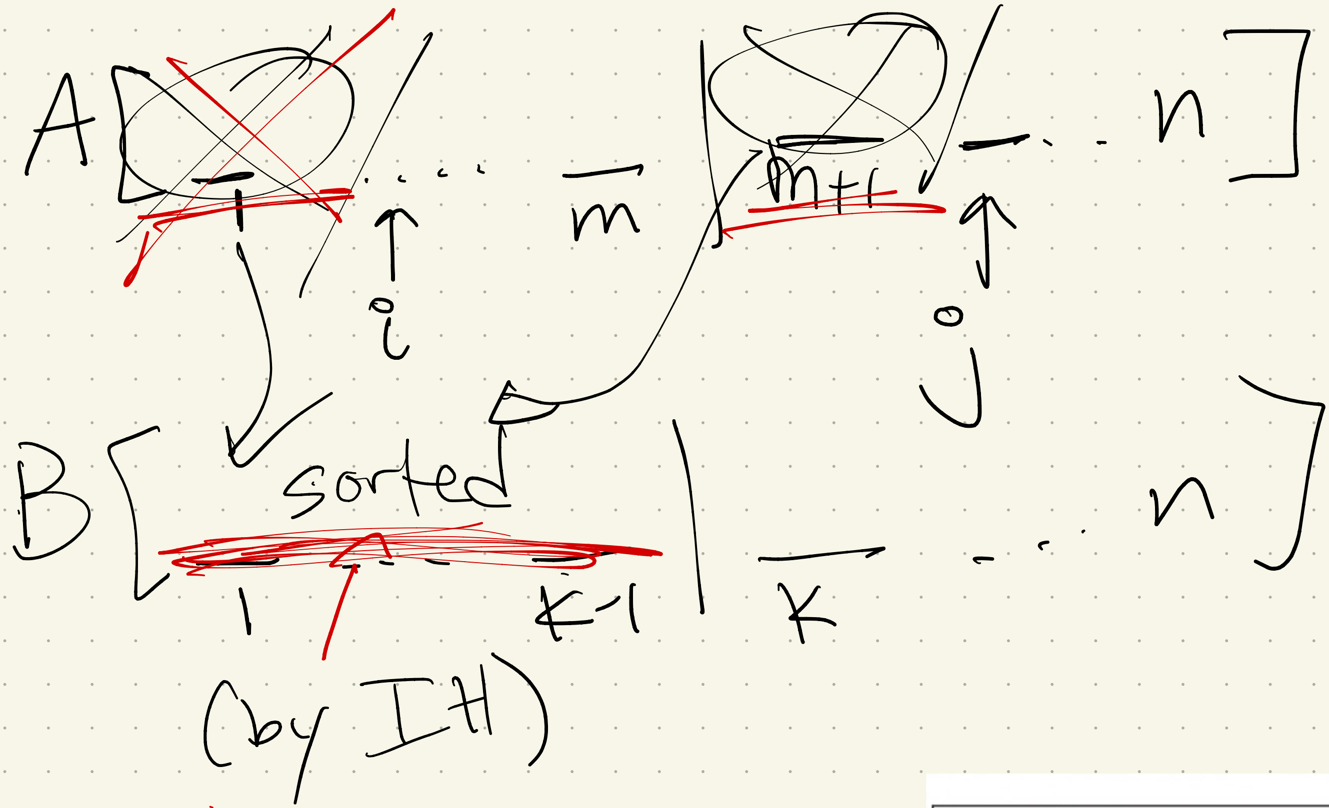
Ind Step: now — k^{th} loop iteration

Well: Cases! Why?

$i > m$: done w/ 1^{st} half

$j > n$: done w/ 2^{nd} half

$i + j$ in middle



Then, our cases:

$i > m$: copy next thing from 2^{nd} half

$j > n$: copy from 1st half

owise: find smaller of $A[i]$ & $A[j]$, & copy it to k^{th} spot

```
MERGE(A[1..n], m):
  i ← 1; j ← m + 1
  for k ← 1 to n
      if j > n
          B[k] ← A[i]; i ← i + 1
      else if i > m
          B[k] ← A[j]; j ← j + 1
      else if A[i] < A[j]
          B[k] ← A[i]; i ← i + 1
      else
          B[k] ← A[j]; j ← j + 1
  for k ← 1 to n
      A[k] ← B[k]
```

Nice part:

Once we know MERGE works, the induction for MergeSort is pretty easy!

Note: example of "strong" induction
(in MergeSort)

Base case: $n=0$ or 1

IH works for $k<n$
(believe in rec. fairy)

MS: $n \longrightarrow$ divide in half

works for $\frac{n}{2}$, + merge correctly merges the 2

$\blacksquare$

Domain transformation:

$$T(n) = T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + O(n)$$

Quicksort:

$$T(n) = \max_{1 \le r \le n} \left(T(r-1) + T(n-r) + O(n) \right)$$

Solving:

Note: "Median of three"
- Somewhat better can still be good!

Remember, while $O(n^2)$ worst case, this is the best sorting algorithm in practice.

Issues to consider:

Recursion Trees:

Let's start with an example.

$$T(n) = 3T\left(\frac{n}{2}\right) + n^2$$

How can I "visualize" the time spent?

Note on reading:

If you don't follow the bit on ignoring floors & ceilings — don't stress!

I need you to know you can do this, but won't ask you to prove it.