Algorithms, Spring '25

Recursion

Recap

- HW0 is due today
- Readings - any class day by 8am
- HW1 - posted later today
  - gradescope: click from Canvas to link things
- TA office hours
  - posted in syllabus

Bug on induction last time:

Where I went wrong: full binary trees → every node has 0 or 2 children

Let's derive the right formula:

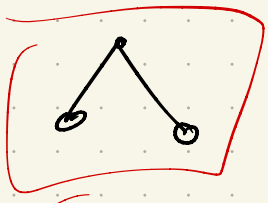
Base case(s):

#nodes

height 0 •

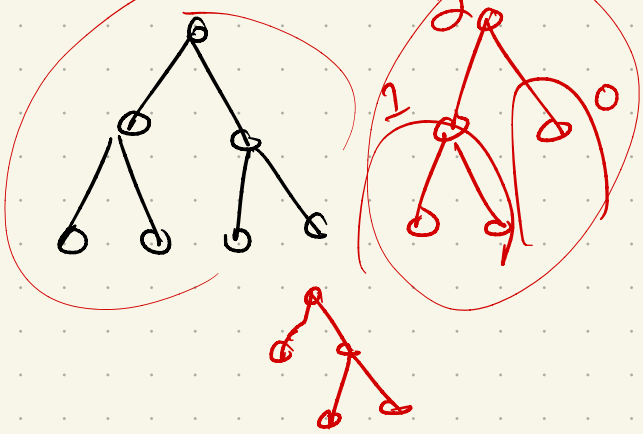
$= 1 = 2^1 - 1$

height 1



$= 3 = 2^2 - 1$

height 2



$\leq 7 = 2^3 - 1$ nodes

Powers of 2 pattern:

$\leq 2^{h+1} - 1$ nodes

Correct proof:

Show that for a full binary tree of height h,

$$\#\text{nodes} \leq 2^{h+1} - 1$$

Induction on height h:

Base case: $h = 0$

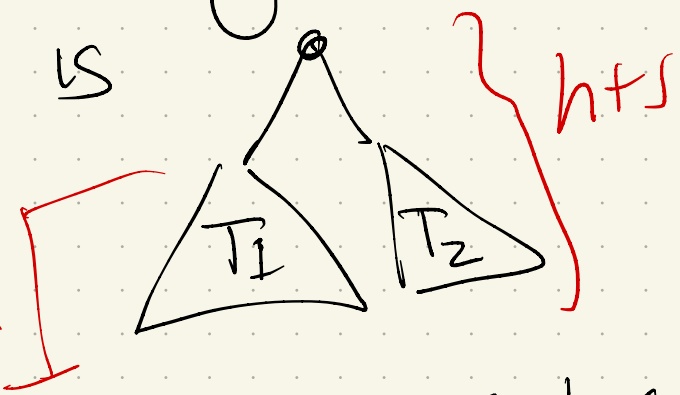
height 0 tree: • 1 node

$2^{h+1} - 1 = 2^{0+1} - 1 = 1$ ✓

Inductive step: A full binary tree with height $\leq h$ has $2^{h+1} - 1$ nodes

(use this to show height $h+1$ tree has $\leq 2^{h+2} - 1$)

Ind step: A binary tree of height $h+1$ is built from a root plus two children of height $\leq h$ (since 1 level taller after the root)



By IH, each of T_1 & T_2 have $\leq 2^{h+1} - 1$ nodes

Total nodes:

root + nodes of T_1 + nodes of T_2

$$\leq 1 + (2^{h+1} - 1) + (2^{h+1} - 1)$$

$$= 2 \cdot 2^{h+1} - 1 = 2^{h+2} - 1$$

Example (& tie to runtimes):

Multiplication:

Input: 2 numbers ↳ in decimal

$X[0..m-1]$, $Y[0..n-1]$ ← m digit, ← n digit

& $X = \sum_{i=0}^{m-1} X[i] \cdot 10^i$, $Y = \sum_{j=0}^{n-1} Y[j] \cdot 10^j$

Example: $X = 25968$

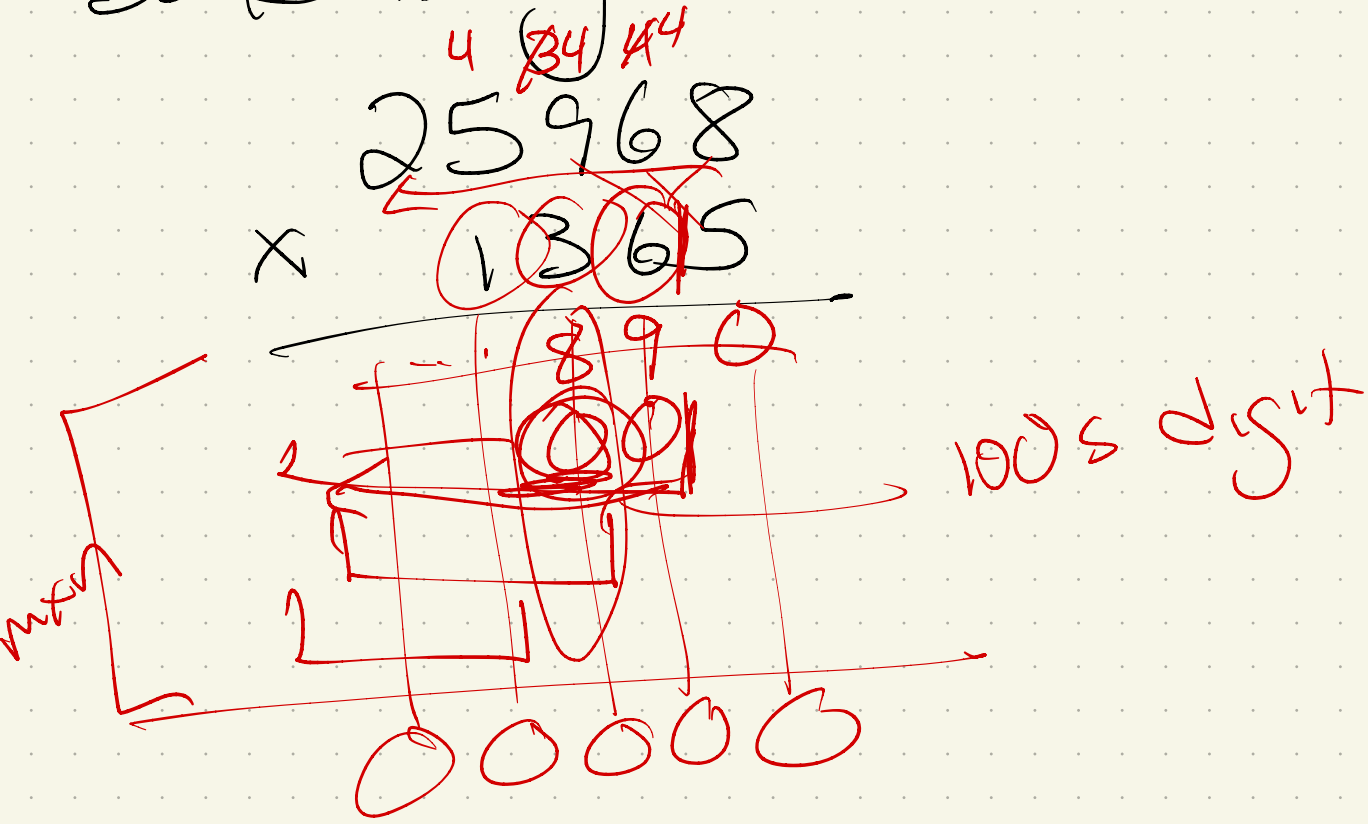
$Y = 1365$

↳ $X = 8 \cdot 10^0 + 6 \cdot 10^1 + 9 \cdot 10^2 + 5 \cdot 10^3 + 2 \cdot 10^4$

$X[0..4] = [8, 6, 9, 5, 2]$

0 1 2 3 4

Back to grade school:



Abstract:

Find all digits:

+ multiply every X[i] w/ Y[j]

+ how many powers of 10 they get:

$$X[i] \cdot Y[j] \cdot 10^i \cdot 10^j$$

$$X \cdot Y = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} X[i] \cdot Y[j] \cdot 10^{i+j}$$

Another view: Instead of just adding all, search for all that land in one spot k:

```
FibonacciMultiply(X[0..m−1], Y[0..n−1]):
  hold ← 0
  for k ← 0 to n + m − 1
      for all i and j such that i + j = k
          hold ← hold + X[i] · Y[j]
    Z[k] ← hold mod 10
    hold ← ⌊hold/10⌋
  return Z[0..m + n − 1]
```

10^k

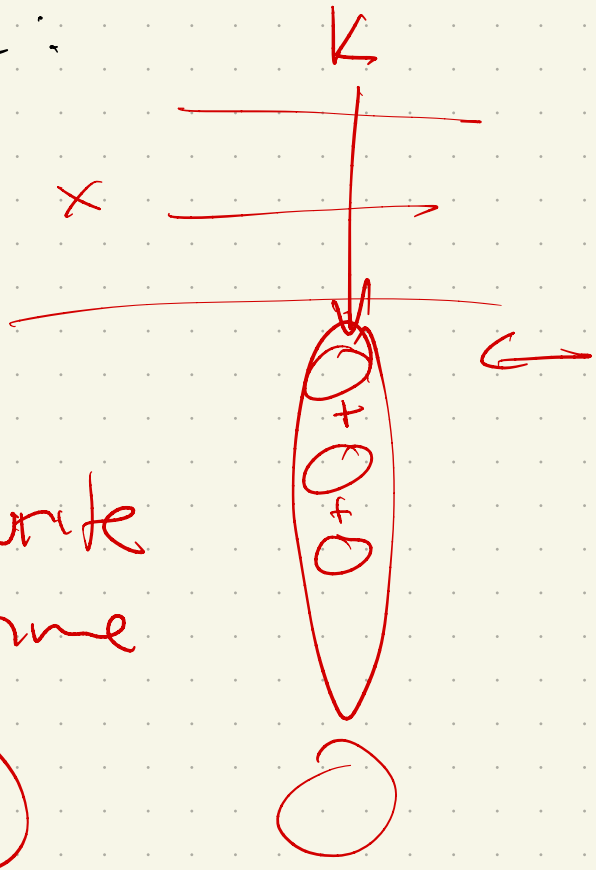
carry

Space & runtime:

$O(m+n)$ items

Space:

one "column" to sum, but overwrite in hold each time



runtime: $O(m \cdot n)$

Recursive Algorithms : Chapter 1

1st part

Setup, plus (hopefully) familiar examples:

- Towers of Hanoi ★
- Merge sort

Later:

- Recap of recurrences & "Master theorem"
- Linear time selection
- Multiplication (again)
- Exponentiation

A high level note on recursion:

Recursion really can be simpler + useful!

often depends upon the language and setup.

Counter-intuitive, but that's mostly because you haven't had a lot of practice.

Functional languages:

↳ no variables

all computation is done via functions + recursions

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:

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".)

Multiplication: How?

$$x \cdot y = \begin{cases} 0 & \text{if } x = 0 \\ \lfloor x/2 \rfloor \cdot (y+y) & \text{if } x \text{ is even} \\ \lfloor x/2 \rfloor \cdot (y+y) + y & \text{if } x \text{ is odd} \end{cases}$$

smaller!

← base case

$\lfloor \frac{x}{2} \rfloor = x/2$

Why? Proof by cases:

if x is even:

then $\exists k \in \mathbb{Z}$ s.t. $x = 2k$

$x \cdot y = (2k) \cdot y = k(2y)$

$= \lfloor \frac{x}{2} \rfloor (y+y)$

if x is odd:

then $\exists \ell \in \mathbb{Z}$ s.t. $x = 2\ell + 1$

$x = 7$

$x = 2 \cdot 3 + 1$

$\ell = \lfloor \frac{x}{2} \rfloor = 3$

$(2\ell+1)y = 2\ell y + y$

$= \ell(2y) + y$

$= \ell(y+y) + y$

Note: historical name! Not a commentary..

```
PEASANTMULTIPLY(x, y):
  if x = 0
    return 0
  else
    x' ← ⌊x/2⌋
    y' ← y + y
    prod ← PEASANTMULTIPLY(x', y')   ⟨⟨Recurse!⟩⟩
    if x is odd
      prod ← prod + y
    return prod
```

bit-shifting

1 addition

1 addition

P(n) runtime for input of size n

$$x \cdot y = \left\lfloor \frac{x}{2} \right\rfloor (y+y)$$

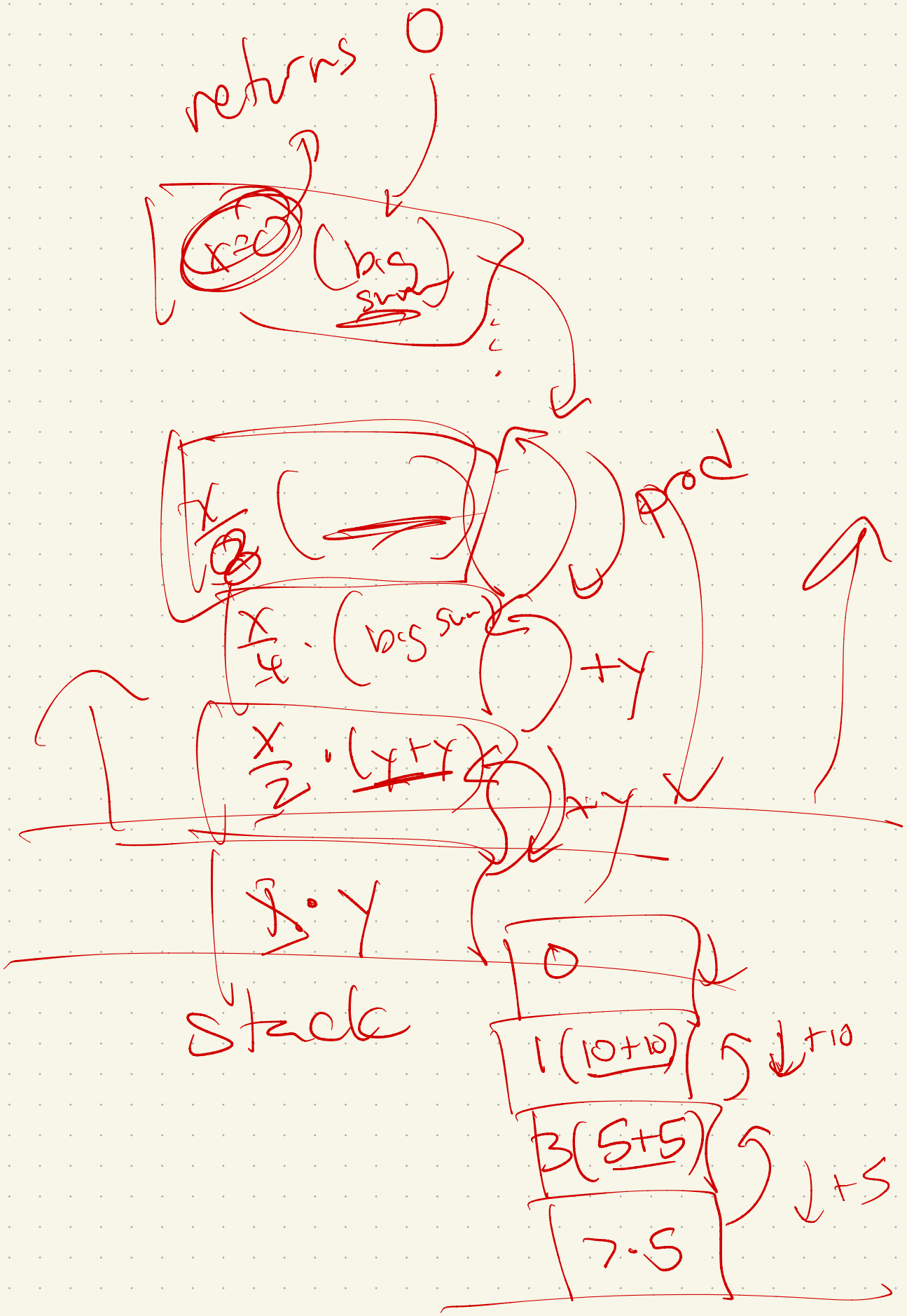
$\hookrightarrow \frac{x}{y}$ (+ + ~)

Runtime:

$$P(n) = \underbrace{3}_{O(1)} + P\left(\frac{n}{2}\right)$$

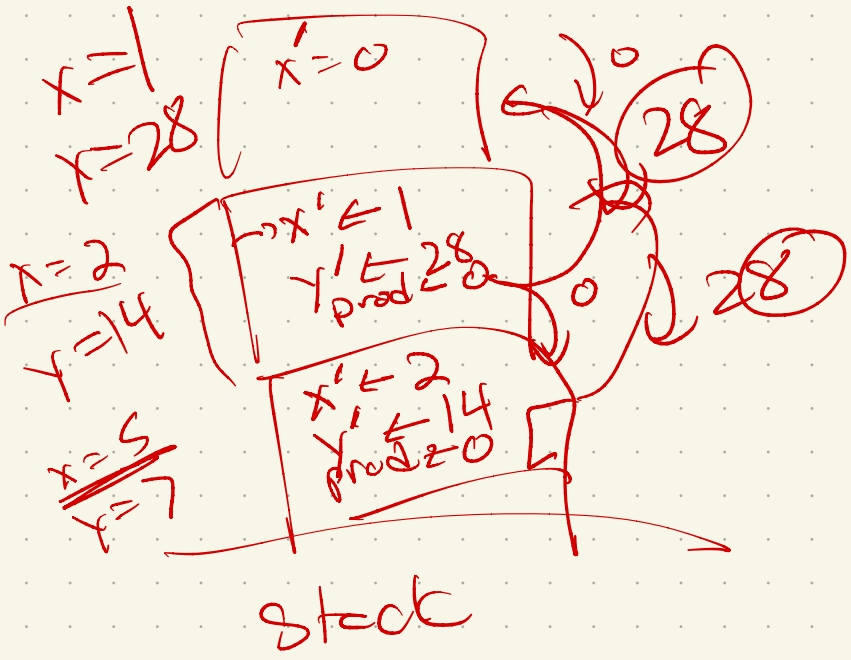
write the recursion!

Base case: $P(0) = O(1)$


returns 0
x=0
(b·s sum
x/2 (
x/2 · (b·s sum
+y
x/2 · (y+y
+y
y · y
stack
0
1 (10+10)
+10
3 (5+5)
+5
7 · 5

```
PeasantMultiply(x, y):
  if x = 0
      return 0
  else
      x' ← ⌊x/2⌋
      y' ← y + y
      prod ← PeasantMultiply(x', y')     ⟨⟨Recurse!⟩⟩
      if x is odd
          prod ← prod + y
      return prod
```

$x = 5$ $y = 7$



Correctness

Towers of Hanoi: runtime

```
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

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

Proof of correctness:

Runtime (for Hanoi):

```
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

Merge Sort:

Divide + conquer recurrences + proof of correctness

```
MERGESORT(A[1..n]):
  if n > 1
      m ← ⌊n/2⌋
      MERGESORT(A[1..m])          ⟨⟨Recurse!⟩⟩
      MERGESORT(A[m+1..n])        ⟨⟨Recurse!⟩⟩
      MERGE(A[1..n], m)
```

```
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]
```

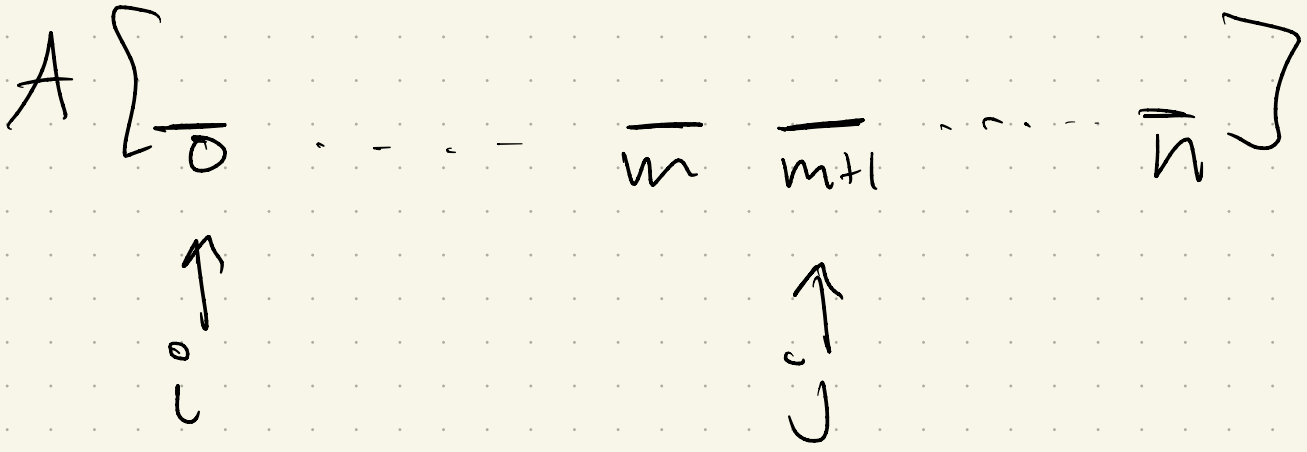
Figure 1.6. Mergesort

First: correctness, in 2 parts.

Part 1: Merge works

Setup: Given $A[1..n]$ and an index m with $1 \le m \le n$ where $A[1..m]$ + $A[m+1..n]$ are sorted, MERGE correctly sorts $A[1..n]$ by end.

How?



and $k \leftarrow 0$ to n

So: at iteration k, show we correctly copy k^{th} sorted element.

Backwards induction:
Consider what is left to sort, ie $n-k$.

Spps $k=n$:

IH:
Now, let $k < n$, & suppose works for any value greater than k:

IS: 4 cases:

```
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]
```

Mergesort: runtime