Algorithms - Spring '25

Induction
Pseudocode
Runtime

Recap

- Reading due by 8am, every day we have class
  - no excuses, but I'll drop lowest 3 to allow for illness / forgetting / travel
- HW0: due next Wed
- No class or office hour Monday
- Extra office hour on Tuesday from 3-5pm

Last time:

- Big-O
- Identities + Summations
- Induction

(4) Induction:

There is a template!

Base case: Prove statement for small value

Ind hypothesis: Assume true for values $\leq k$.

Ind. step: Prove true for next value $k+1$

Think of this as "automating" a proof.

- $P(1)$
- $\forall k \geq 1$, if $P(k)$ then $P(k+1)$

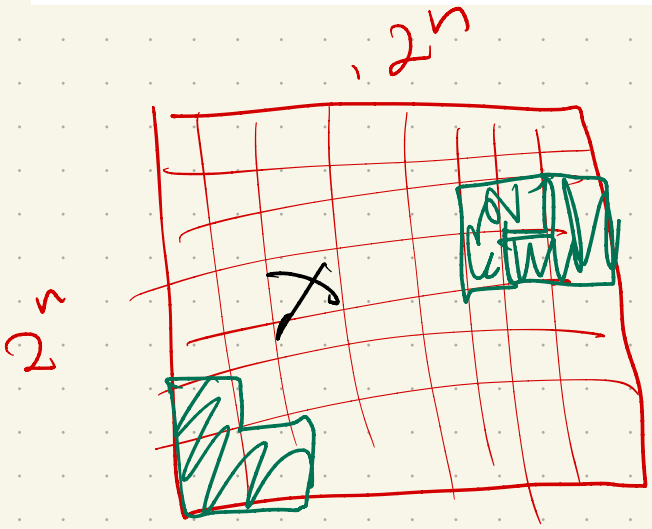
IH

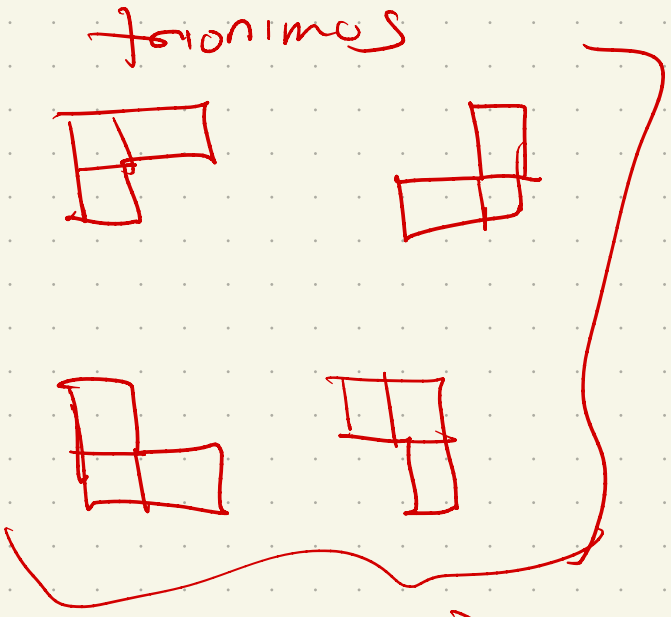
IS

Learn it, use it, love it!

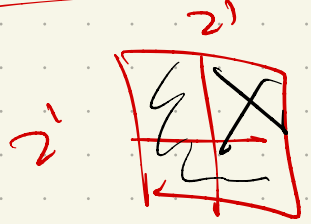
"Structural" induction:

Let n be a positive integer. Show that every $2^n \times 2^n$ checkerboard with one square removed can be tiled using right triominoes, where these pieces cover three squares at a time, as shown in Figure 4.



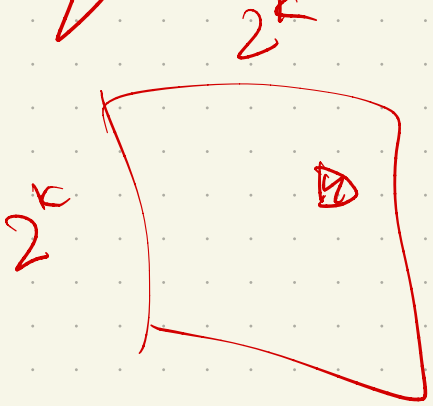


Base case: $2' \times 2'$ board



No matter which square is removed, I can cover the other 3

IH: Assume I can tile any $2^k \times 2^k$ board with any 1 square removed



IS: Show I can do a $2^{k+1} \times 2^{k+1}$ board:

Induction on graphs / tree

Let $h(T)$ = height of a full binary tree

full → every node has 0 or 2 children

$$h(T) = \begin{cases} 0 & \text{if } T \text{ is a single node} \\ 1 + \max(h(T\text{'s children})) \end{cases}$$

Claim: The number of nodes in a full binary tree is $\leq 2^{h(T)+1}$

Claim: The number of nodes in a full binary tree is $\leq 2^{h(T)} + 1$.

Proof:

5 Pseudo code & runtime:
Discrete math examples
(from Rosen textbook)

ALGORITHM 1 Finding the Maximum Element in a Finite Sequence.

procedure $max(a_1, a_2, \ldots, a_n$: integers)
$max := a_1$
for $i := 2$ **to** n
    if $max < a_i$ **then** $max := a_i$
return max {max is the largest element}

ALGORITHM 2 The Linear Search Algorithm.

procedure *linear search*(x: integer, $a_1, a_2, \ldots, a_n$: distinct integers)
$i := 1$
while ($i \leq n$ and $x \neq a_i$)
    $i := i + 1$
if $i \leq n$ **then** $location := i$
else $location := 0$
return $location$ {$location$ is the subscript of the term that equals x, or is 0 if x is not found}

Pseudocode conventions here:

Variable assignment:

Boolean comparison:

```
if (x = 5)
```

Arrays: A[0..n-1]

- each A[i]

Loops:

```
for i ← 1 to 100
    A[i] ← i
```

Again, takes practice! Read intro chapter + then give HW0/HW1 a try.

Pseudocode format:

In a pinch, pretend you're in Python / Ruby.

High level & readable.

I realize this is not a "definition" - that is the point!

It's about effective communication.

Initially:

- lots of examples
- lots of practice
- reach out if you have questions
- in a pinch - peer evaluation!

Example (& tie to runtimes):

Multiplication:

Input: 2 numbers
↳ in decimal

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

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

Example: $X = 25968$

$Y = 1365$

Back to grade school:

$$\begin{array}{r} 25968 \\ \times\ 1365 \\ \hline \end{array}$$

Abstract:

Find all digits:

+ how many powers of 10 they get:

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

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]
```

Space & runtime:

More complex: recursion!

Algorithm 1 Quicksort

```
1: procedure QUICKSORT(A, p, r)
2:     if p < r then
3:         q = PARTITION(A, p, r)
4:         QUICKSORT(A, p, q − 1)
5:         QUICKSORT(A, q + 1, r)
6:     end if
7: end procedure
8: procedure PARTITION(A, p, r)
9:     x = A[r]
10:    i = p − 1
11:    for j = p to r − 1 do
12:        if A[j] < x then
13:            i = i + 1
14:            exchange A[i] with A[j]
15:        end if
16:        exchange A[i] with A[r]
17:    end for
18: end procedure
```

QuickSort Pseudocode Example

Any function which calls itself (on a smaller size)

Multiplication: How?

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

Why? Proof by cases:

if x is even:

if x is odd:

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
```

Runtime:

Correctness

Recursive Algorithms : Chapter 1

1st half:

Setup, plus (hopefully) familiar examples:

- Towers of Hanoi
- Merge sort

2nd half:

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