

# Algorithms - Spring '25

Greedy:

Intervals

Huffman Codes

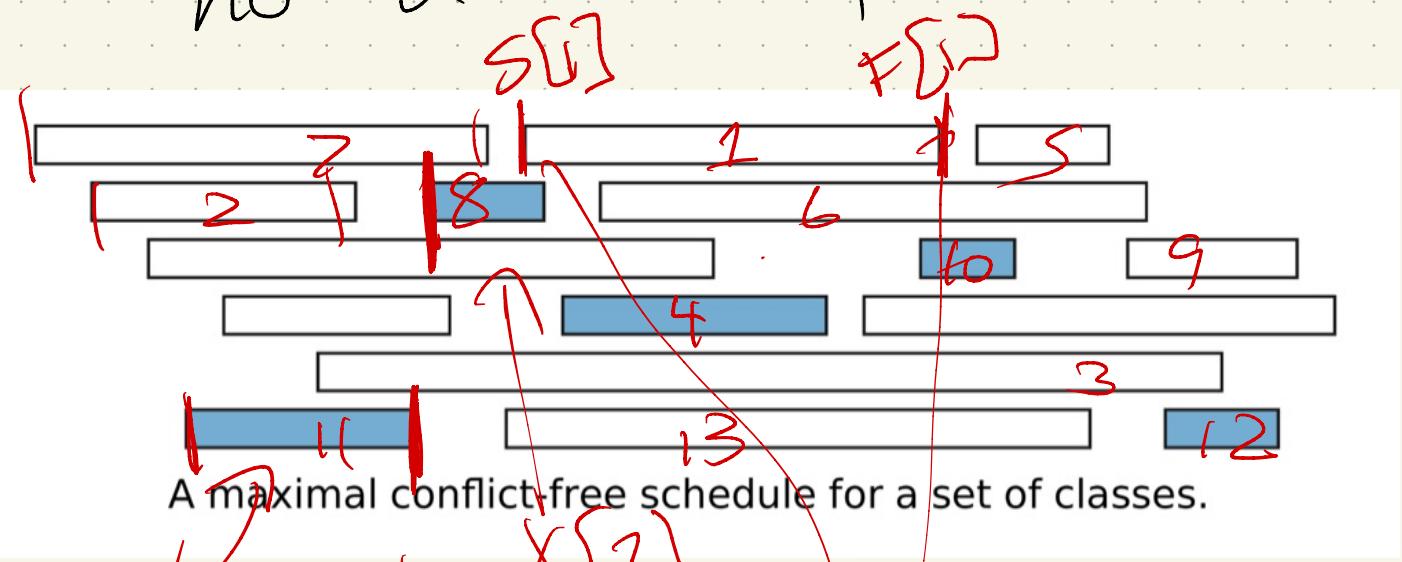


# Recap

- Office hours: today at 2pm  
(me) & 3:15 pm (TAs)
- HW3 due
- HW4— oral grading  
next Thurs/Friday
- Midterm: March 4, 8am

# Problem: Interval Scheduling

Given a set of events (ie intervals, with a start and end time), select as many as possible so that no 2 overlap.



$X[1] = 1$      $X[2]$   
More formally:

Two arrays

$S[1..n]$ :

$F[1..n]$ :

$X[1..n]$  instead  
 $\cup S[X[i]]$   
 $F[X[i]]$

Goal: A subset  $X \subseteq \{1..n\}$  as big as possible s.t.  $\forall i$   
 $F[i] \leq S[i+1]$

How would we formalize a dynamic programming approach?

Recursive structure:

Consider job 1;

take it  
↳ add to X

recurse on  $2-n$

Don't

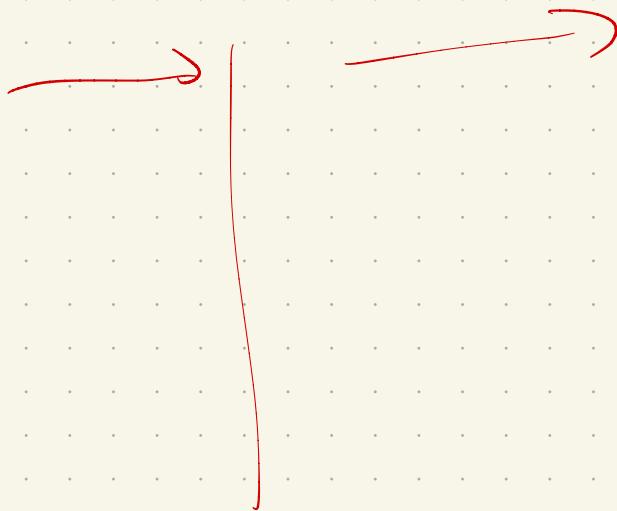
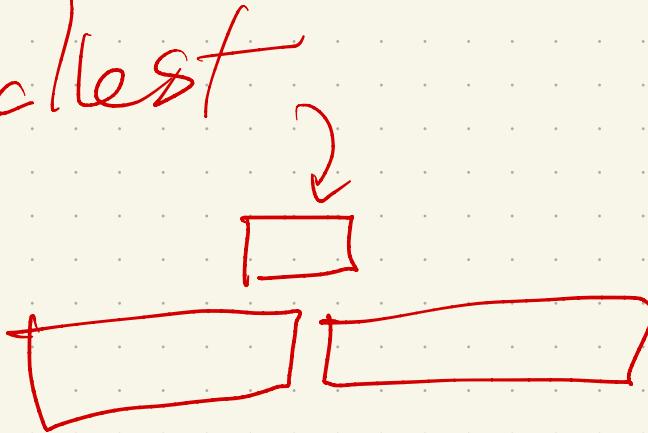
recurse on  $2-n$

# Intuition for greedy:

Consider what might be a good first one to choose.

## Ideas?

Smallest



Key intuition:

If it finishes as early as possible, we can fit more things in!

So - strategy:

The code:

GREEDYSCHEDULE( $S[1..n], F[1..n]$ ):

sort  $F$  and permute  $S$  to match

$count \leftarrow 1$

$X[count] \leftarrow 1$

for  $i \leftarrow 2$  to  $n$

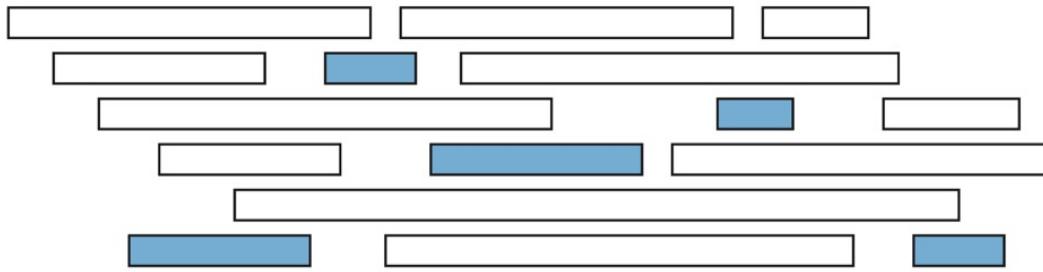
    if  $S[i] > F[X[count]]$

$count \leftarrow count + 1$

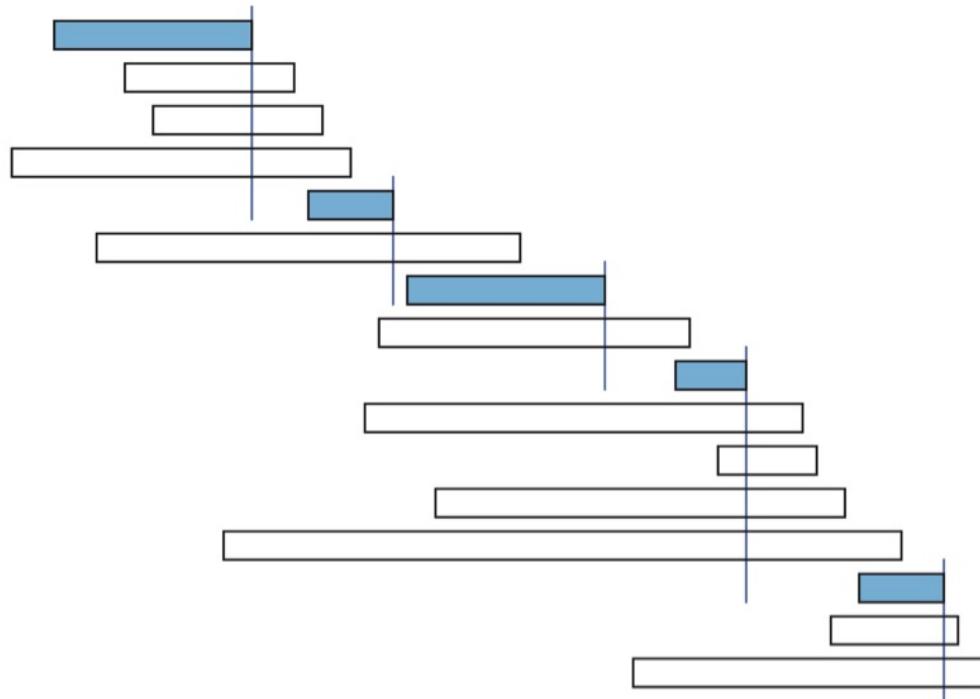
$X[count] \leftarrow i$

return  $X[1..count]$

Picture:



A maximal conflict-free schedule for a set of classes.



The same classes sorted by finish times and the greedy schedule.

## Correctness:

Why does this work?

Note: No longer trying all possibilities or relying on optimal substructure!

So we need to be very careful on our proofs.

(Clearly, intuition can be wrong!)

Lemma: We may assume the optimal schedule includes the class that finishes first.

Pf:

Thm: The greedy schedule is optimal.

Pf: Suppose not.

Then  $\exists$  an optimal schedule that has more intervals than the greedy one.

Consider first time they differ:

Greedy:  $g_1 \ g_2 \dots g_i \dots g_k$

OPT:  $o_1 \ o_2 \dots o_i \dots o_l$

Example: Huffman trees

Many of you saw this in data structures.  
Why?

- cool use of trees
- non-trivial use of other data structure

Really - it's greedy!

Idea: Want to compress  
data, to use fewest possible  
bits.

Goal: Minimize Cost

↳ here, minimize total length of encoded message :

Input: Frequency counts  
 $f[1..n]$

Compute: binary tree

Leaves:

$$\text{cost}(T) = \sum_{i=1}^n f[i] \cdot \text{depth}(i)$$

# Let's be greedy!

To do this, we'll need to use the array  $f$ :

This sentence contains three a's, three c's, two d's, twenty-six e's, five f's, three g's, eight h's, thirteen i's, two l's, sixteen n's, nine o's, six r's, twenty-seven s's, twenty-two t's, two u's, five v's, eight w's, four x's, five y's, and only one z.

If we ignore punctuation & spaces (just to keep it simple), we get:

A	C	D	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	Z
3	3	2	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	1

Which letters should be deeper (or shallower)?

(ie: How to be greedy?)

## Huffman's alg.:

Take the two least frequent characters.

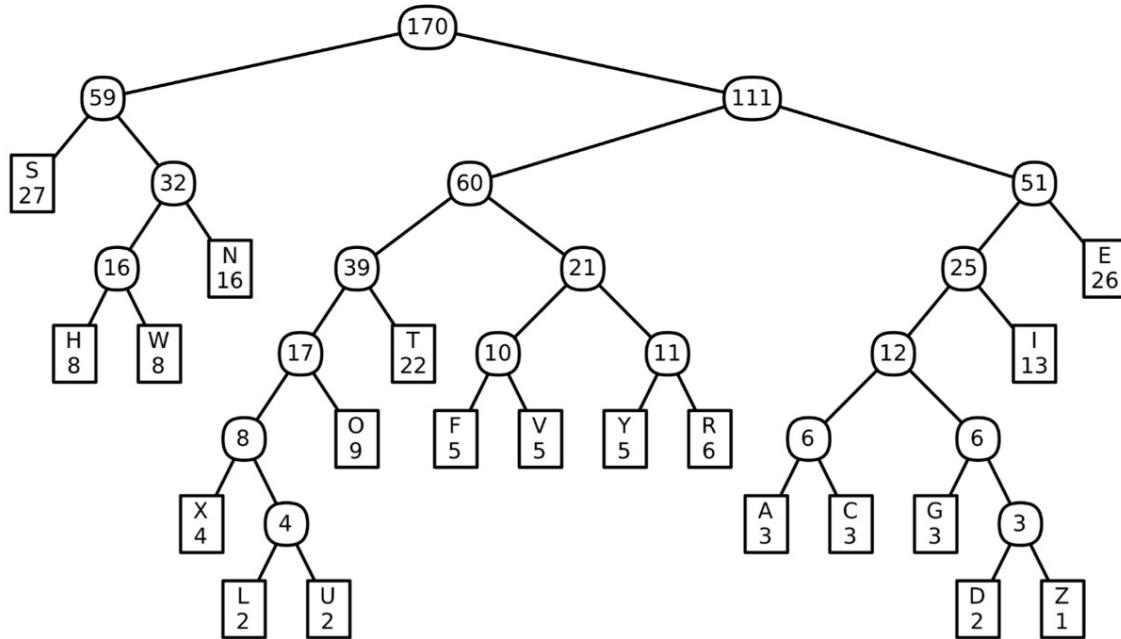
Merge them into one letter, which becomes a new "leaf".

A	C	D	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	Z
3	3	2	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	1



A	C	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	¶
3	3	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	3

In the end, get a tree with letters at the leaves:



A Huffman code for Lee Sallows' self-descriptive sentence; the numbers are frequencies for merged characters

A	C	D	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	Z
3	3	2	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	1

If we use this code, the encoded message starts like this:

1001 0100 1101 00 00 111 011 1001 111 011 110001 111 110001 10001 011 1001 110000 ...  
 T H I S S E N T E N C E C E C O N T A ...

# How many bits?

char.	A	C	D	E	F	G	H	I	L	N	O	R	S	T	U	V	W	X	Y	Z
freq.	3	3	2	26	5	3	8	13	2	16	9	6	27	22	2	5	8	4	5	1
depth	6	6	7	3	5	6	4	4	7	3	4	4	2	4	7	5	4	6	5	7
total	18	18	14	78	25	18	32	52	14	48	36	24	54	88	14	25	32	24	25	7

Total is  $\sum f[i] \cdot \text{depth}(i)$   
= 646 bits here

How would ASCII do on these  
170 letters

Implementation: use priority queue

BUILDHUFFMAN( $f[1..n]$ ):

for  $i \leftarrow 1$  to  $n$

$L[i] \leftarrow 0; R[i] \leftarrow 0$

INSERT( $i, f[i]$ )

for  $i \leftarrow n$  to  $2n - 1$

$x \leftarrow \text{EXTRACTMIN}()$

$y \leftarrow \text{EXTRACTMIN}()$

$f[i] \leftarrow f[x] + f[y]$

$L[i] \leftarrow x; R[i] \leftarrow y$

$P[x] \leftarrow i; P[y] \leftarrow i$

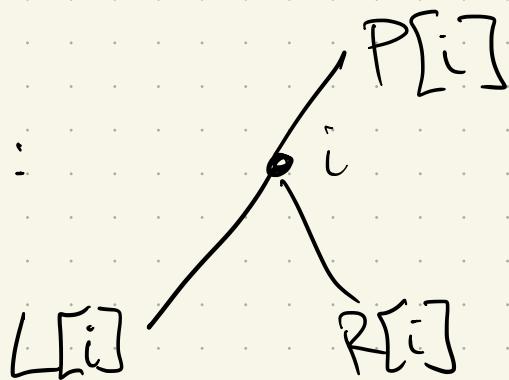
INSERT( $i, f[i]$ )

$P[2n - 1] \leftarrow 0$

3 arrays:  $L, R, P$

to encode the tree

node  $i$ :



So:

BANANA

index: 1 2 3 4  $\leftarrow n$

letters: B A N EoM

Freq: f,

BUILDHUFFMAN( $f[1..n]$ ):

for  $i \leftarrow 1$  to  $n$

$L[i] \leftarrow 0; R[i] \leftarrow 0$

INSERT( $i, f[i]$ )

for  $i \leftarrow n$  to  $2n - 1$

$x \leftarrow \text{EXTRACTMIN}()$

$y \leftarrow \text{EXTRACTMIN}()$

$f[i] \leftarrow f[x] + f[y]$

$L[i] \leftarrow x; R[i] \leftarrow y$

$P[x] \leftarrow i; P[y] \leftarrow i$

INSERT( $i, f[i]$ )

$P[2n - 1] \leftarrow 0$

L: 1 2 3 4 5 6 7

R:

P:

Runtime?

BUILDHUFFMAN( $f[1..n]$ ):

for  $i \leftarrow 1$  to  $n$

$L[i] \leftarrow 0; R[i] \leftarrow 0$

INSERT( $i, f[i]$ )

for  $i \leftarrow n$  to  $2n - 1$

$x \leftarrow \text{EXTRACTMIN}()$

$y \leftarrow \text{EXTRACTMIN}()$

$f[i] \leftarrow f[x] + f[y]$

$L[i] \leftarrow x; R[i] \leftarrow y$

$P[x] \leftarrow i; P[y] \leftarrow i$

INSERT( $i, f[i]$ )

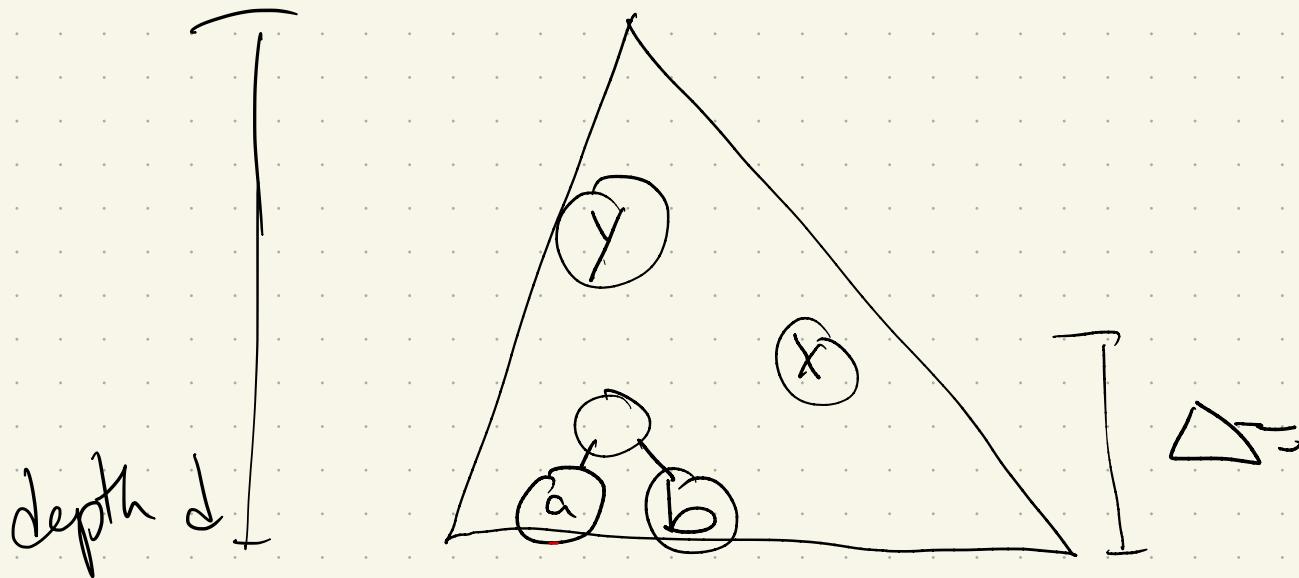
$P[2n - 1] \leftarrow 0$

# Correctness :

1<sup>st</sup> Lemma: There is an optimal prefix tree where the two least common letters are siblings at the largest depth.

Pf: Suppose not. Then

optimal tree  $T$  has some depth  $d$ , but at least 2 common letters  $x + y$  are not at that depth.



Note some other letters  $a + b$  are deepest

Pf cont:

Know  $f[x] \leq f[a]$ ,

but  $\text{depth}(a) = \text{depth}(x) + \Delta$

+ recall that:

$$\text{cost}(T) = \sum_{i=1}^n f[i] \cdot \text{depth}(i)$$

Build  $T'$ :

Thm: Huffman trees are optimal.

Pf:

Use induction (+ swap).

BC: For  $n=1, 2$ , or  $3$ , Huffman works

Why?

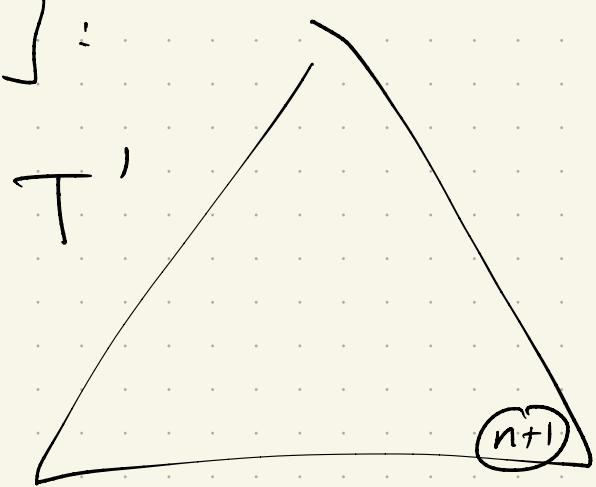
IH: Assume Huffman works on  $\leq n-1$  characters

IS: Input  $F[1..n]$ , + spp's

$F[1] + F[2]$  are min freq.

↳ create a smaller array

IS : optimal tree  $T'$  of  
 $F[3..n+1]$ :



Note:  $n+1$   
is in tree

Build a tree  $T$  for  $F[1..n]$ :

Claim:  $T$  is optimal.

Why?

Why is  $T$  optimal??  
(we know  $T'$  is  $\rightarrow IH!$ )

Cost( $T$ ) =

$$\sum_{i=1}^n F[i] \cdot \text{depth}[i]$$

$$= \text{cost}(T') + \underbrace{\text{changes we made}}$$

↗

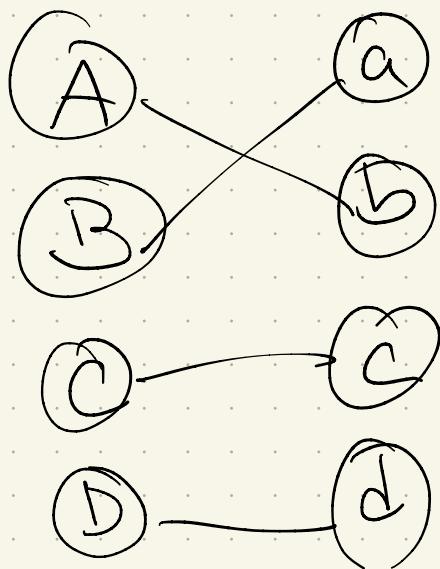
# Stable matching

Really useful! Many variants:

- ties
- incomplete preference lists
- one side picks many from the other
- "egalitarian" matchings
- minimizing "regret"

Really a lot of choices to be made.

First: "unstable":



- $(A, a)$  is unstable
  - If A prefers a to current match
  - and a prefers A to current match

In a sense: if put together + realizing they both prefer each other, would  $(A, a)$  leave current matches?

↳ unstable!

History: used to be "stable marriage"

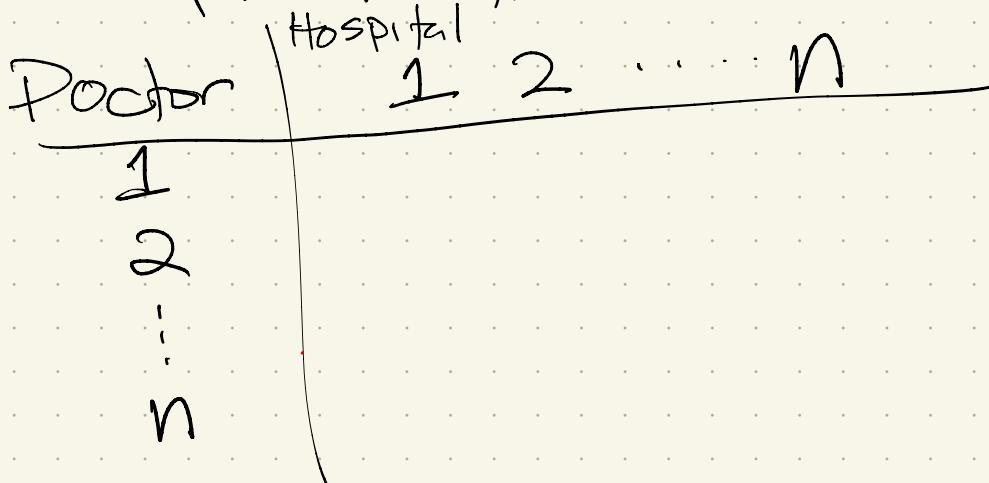
(long history of strange papers + variants.)

# Algorithm: (wikipedia)

## Algorithm [edit]

```
algorithm stable_matching is
    Initialize all  $m \in M$  and  $w \in W$  to free
    while  $\exists$  free man  $m$  who still has a woman  $w$  to propose to do
         $w :=$  first woman on  $m$ 's list to whom  $m$  has not yet proposed
        if  $w$  is free then
             $(m, w)$  become engaged
        else some pair  $(m', w)$  already exists
            if  $w$  prefers  $m$  to  $m'$  then
                 $m'$  becomes free
                 $(m, w)$  become engaged
            else
                 $(m', w)$  remain engaged
            end if
        end if
    repeat
```

In book, data structures matter  
for runtime:



Not obvious why it works.  
(or even how to be greedy!)

Good example of why the  
proof matters.

Nice example of fairness:

This algorithm sucks for  
one side.

(Not all solutions are equal!)

How to even define "fair"?