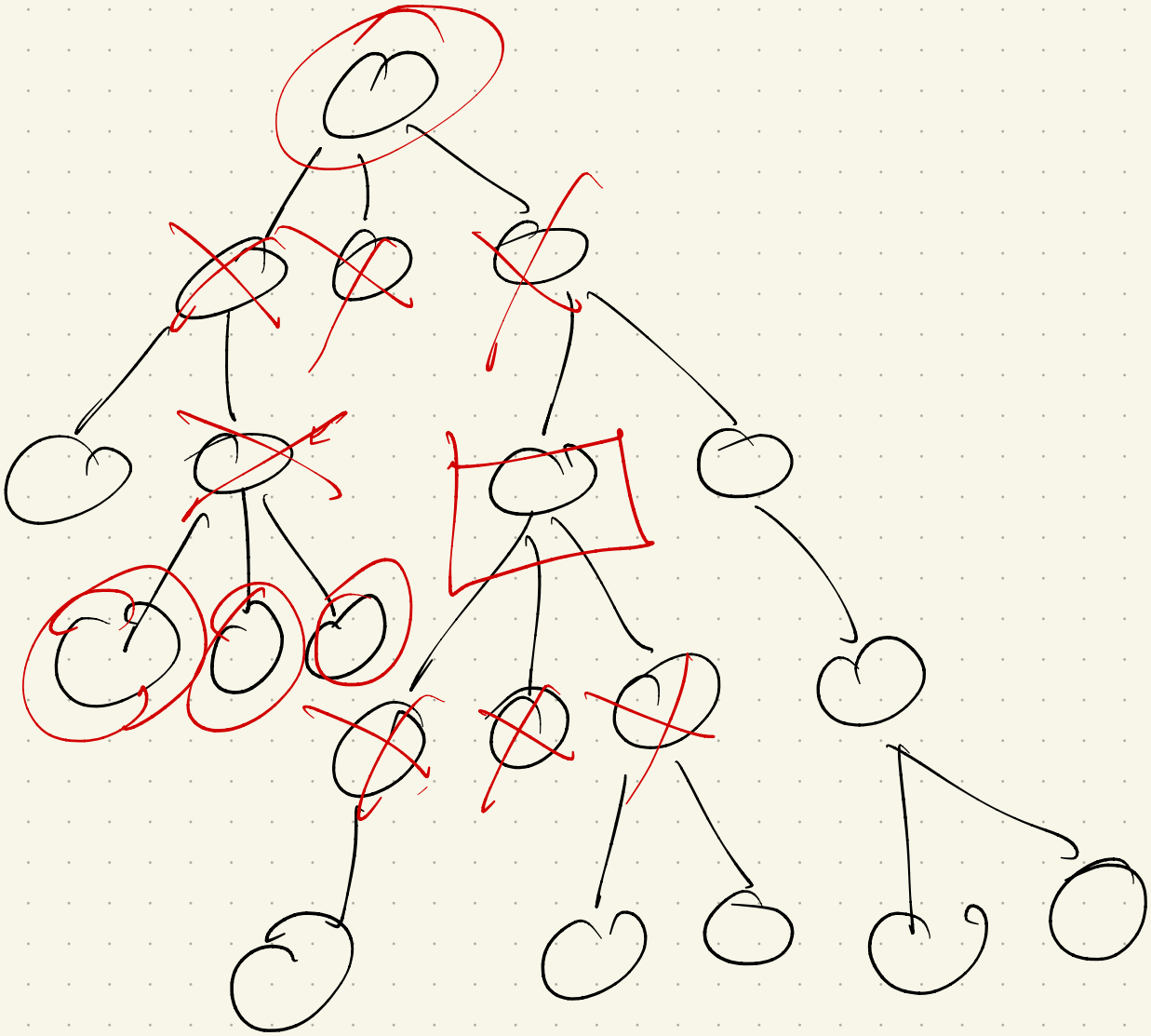
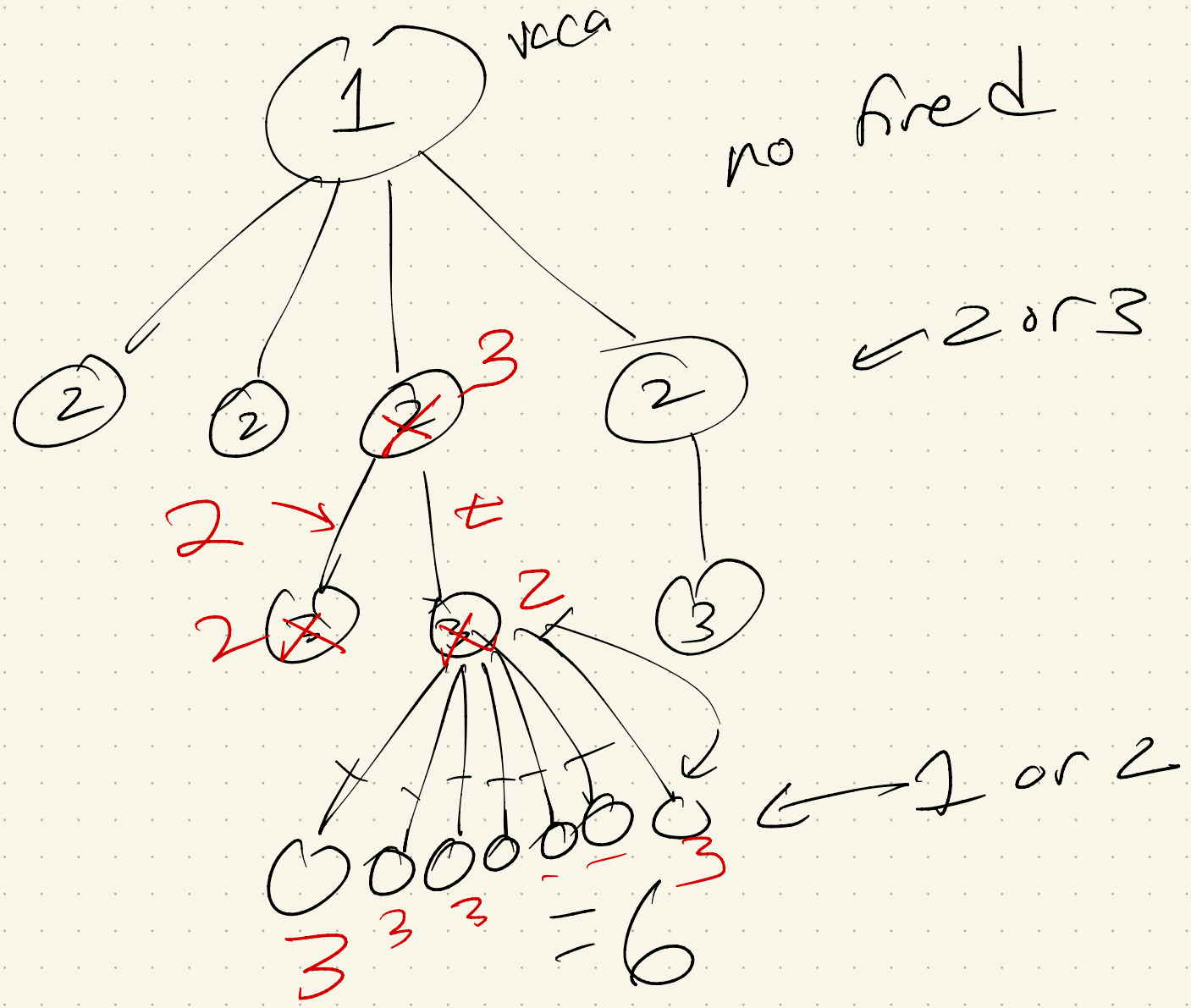
Transition: DS+ Algorithms

Greedy Algorithms

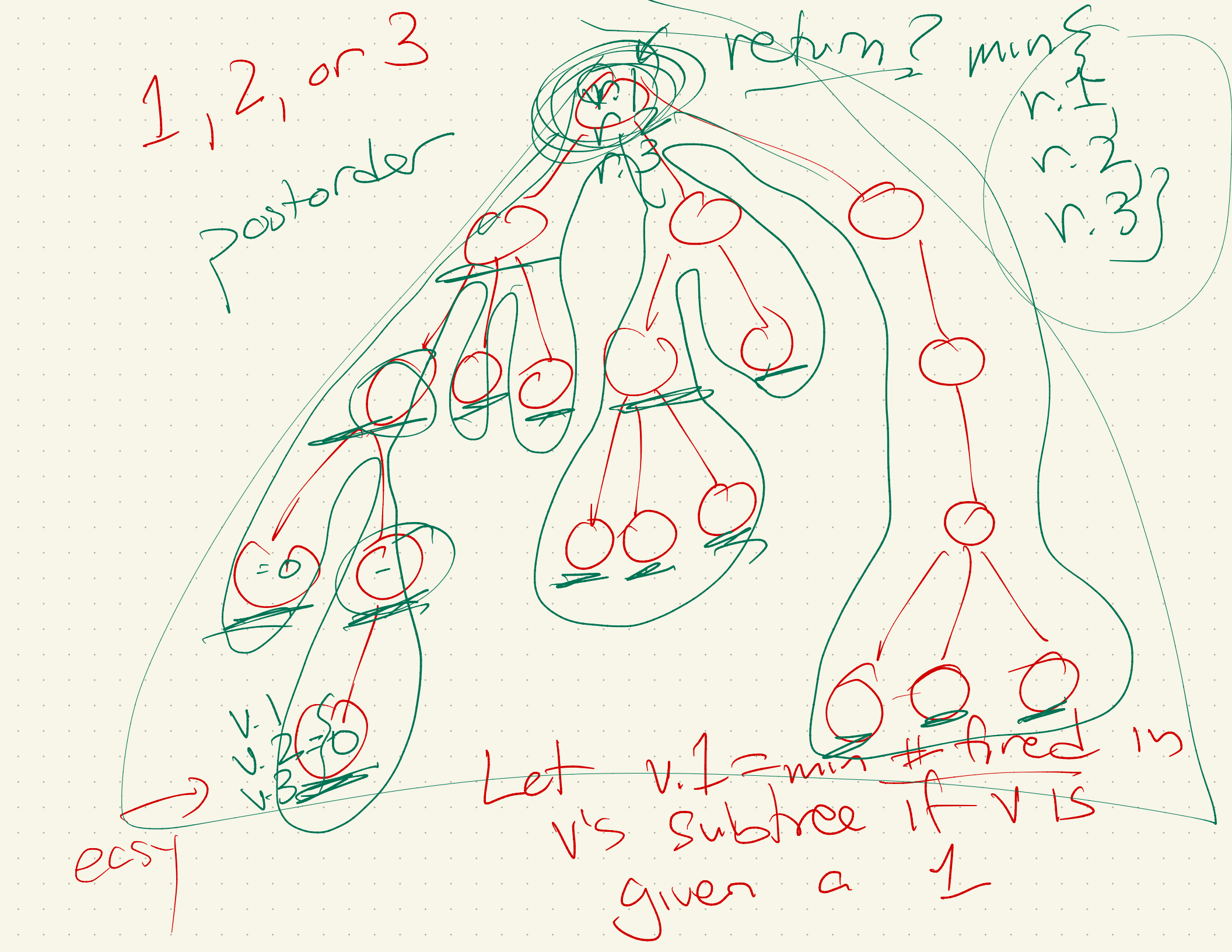
Worksheet

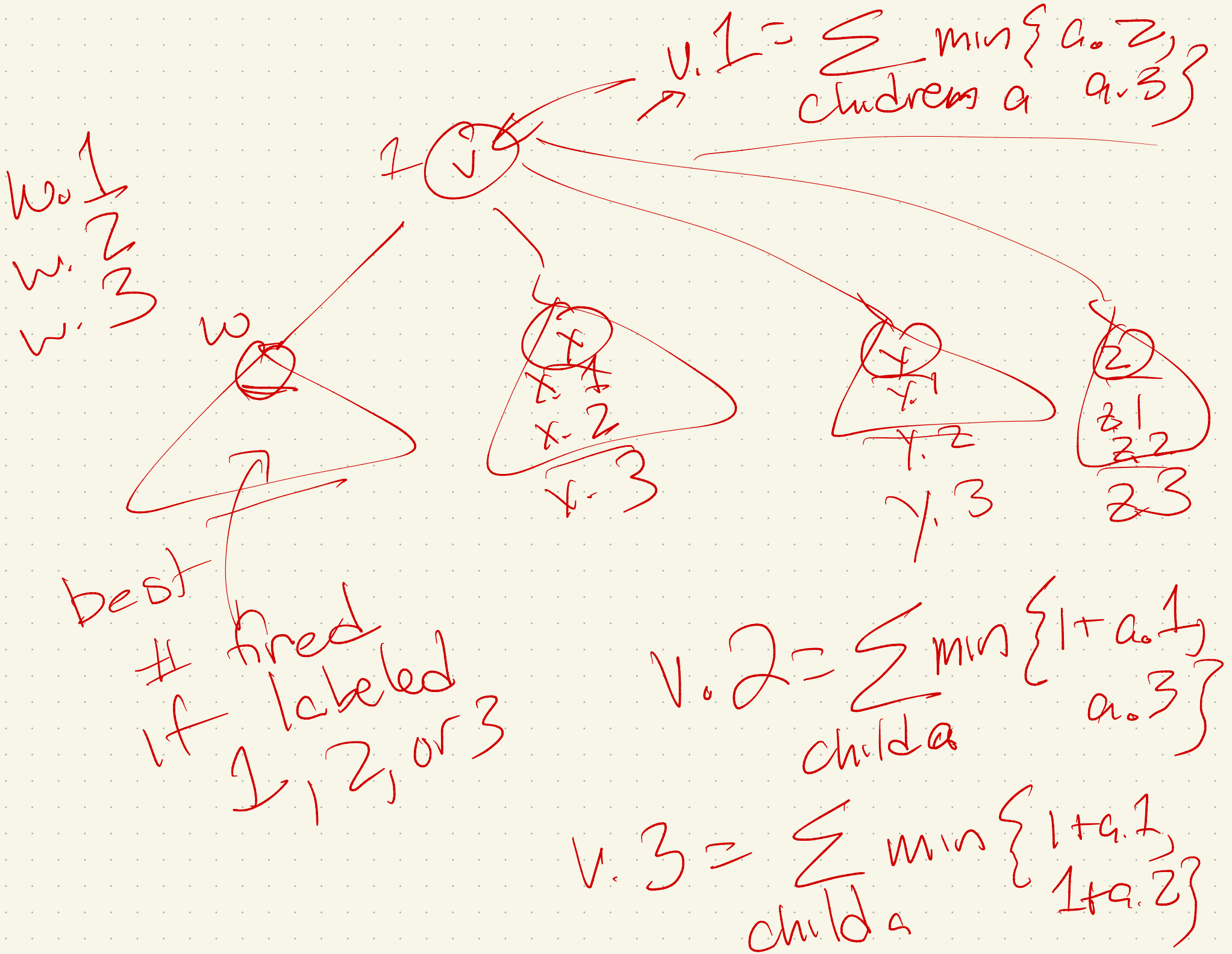


Invite some people



vaca
1
no fired
2
2
2
3
2
2 or 3
2
2
2
3
1 or 2
3
3
3
3
= 6


1, 2, or 3
postorder
return? min{
r.1
r.2
r.3}
=0
-
v.1
v.2=0
v.3
easy
Let v.1 = min # freed in
v's subtree if v is
given a 1

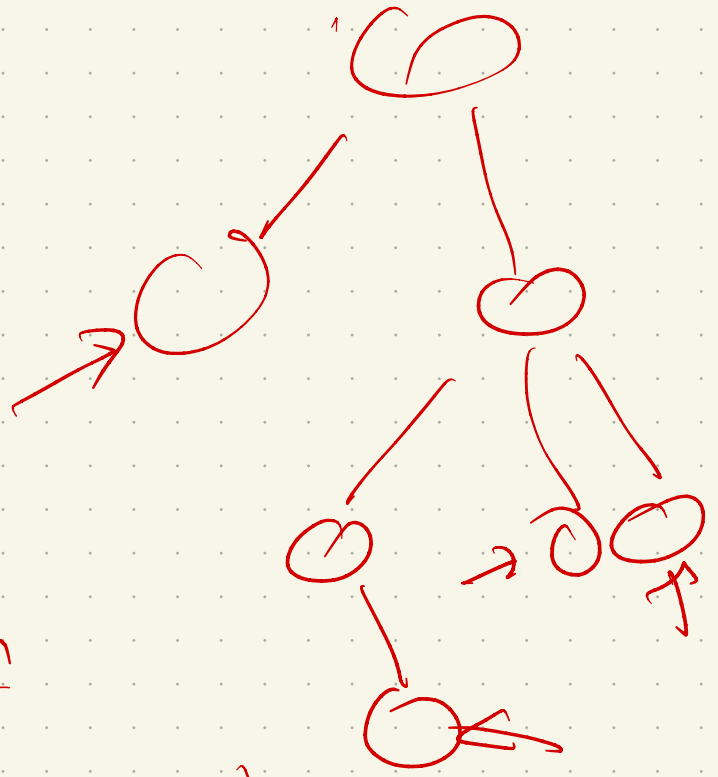

v.1 = Σ min{a.2, a.3}
children a
v
w
x
y
z
w.1
w.2
w.3
x.1
x.2
x.3
y.1
y.2
y.3
z.1
z.2
z3
best
fired
if labeled
1, 2, or 3
v.2 = Σ min{1+a.1, a.3}
child a
v.3 = Σ min{1+a.1, 1+a.2}
child a

if v is a leaf:

v.1 = 0

v.2 = 0

v.3 = 0



if v has children:

v.1 = { see last

v.2 = { see post

v.3 = { see post

plug into postorder code

for v in postorder trav.

if v is leaf:

v.1, v.2, v.3 = 0

else

fill in v.1, v.2, v.3

v.1 = Σ

v.2 = Σ

v.3 = Σ

function(v):

postorder rec

if v if a leaf save $v.1, v.2, v.3 = c$

else

do children for each child a of v:

function(a)

compute for v

$v.1 = \sum_{a\ \text{child}} \min\{a.1, a.2\}$

$v.2 = \sum_{a\ \text{child}} \min\{1 + a.1, a.2\}$

$v.3 = \min\{1 + a.1, 1 + a.2\}$



Dynamic Programing vs Greedy

Dyn. pro: try all possibilites
↳ but intelligently!

In greedy algorithms, we avoid building all possibilites.

How?

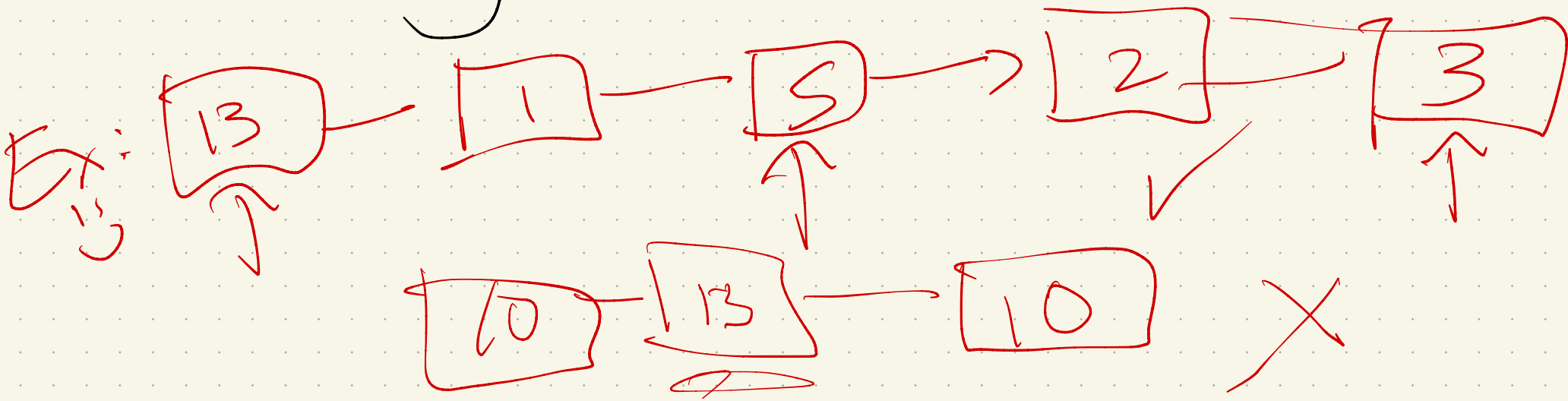
- Some part of the problem's structure lets us pick a local "best" and have it lead to a global best.

But - be careful!

Students often design a greedy strategy, but don't check that it yields the best global one.

Example (from worksheet):

Robbing houses, no 2 in a row:



Overall greedy strategy:

- Assume optimal is different than greedy
- Find the "first" place they differ.
- Argue that we can exchange the two without making optimal worse.

$\Longrightarrow$ there is no "first place" where they must differ, so greedy in fact is an optimal solution

(Note: proof by contradiction here.)

First example in the book:

Storing files on tape.

Input: n files, each with a length
& # times it will be accessed:

$L[1..n]$ & $F[1..n]$

length

frequency

Goal: Minimize access time

$$\text{Cost} = \sum_k (\#\text{times access job } k) \times (\text{time to access job } k)$$

Example: 3 files

A: length 1, frequency 1

B: length 2, frequency 2

C: length 10, frequency 12

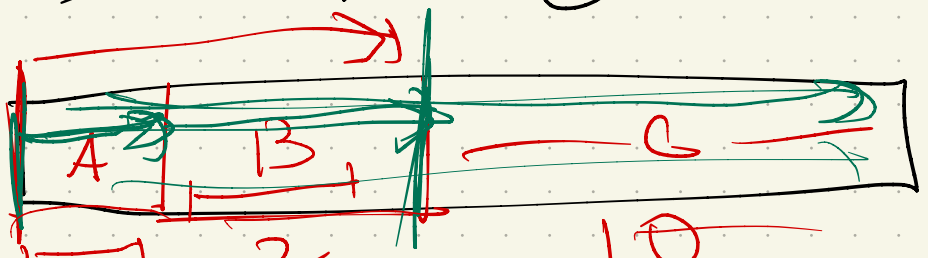
1/1

2/15

10/12

Bad ex

~~Shortest first:~~

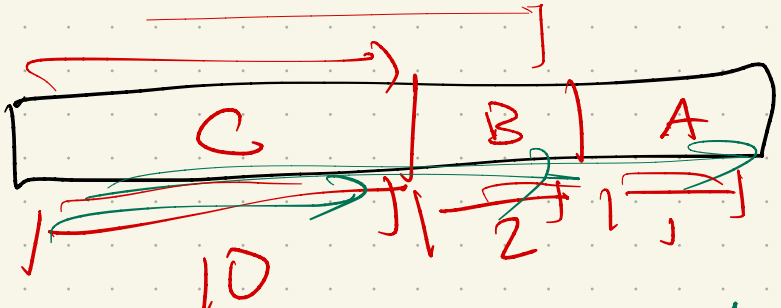


Seek cost

Cost: $1 \cdot 1 + 2 \cdot 3 + 12 \cdot 13 = 163$

Sort by freq:

Most frequent first:

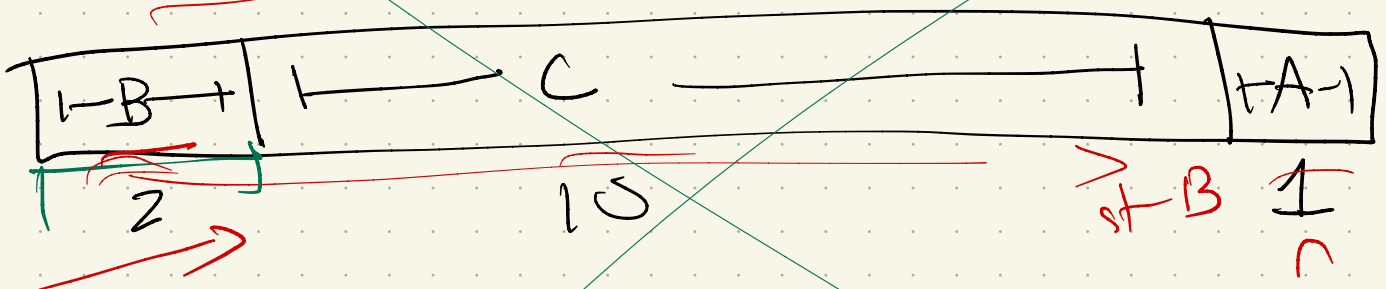


Cost: $12 \cdot 10 + 2 \cdot 12 + 1 \cdot 13$

$= 157$

Optimal: !!

B, then C, then A:



Why?

cost = $2 \cdot 2 + 12 \cdot 12 + 1 \cdot 13$

$= 161$

$\pi(0) = 2$

$\pi(2) = 3$

$\pi(3) = 1$

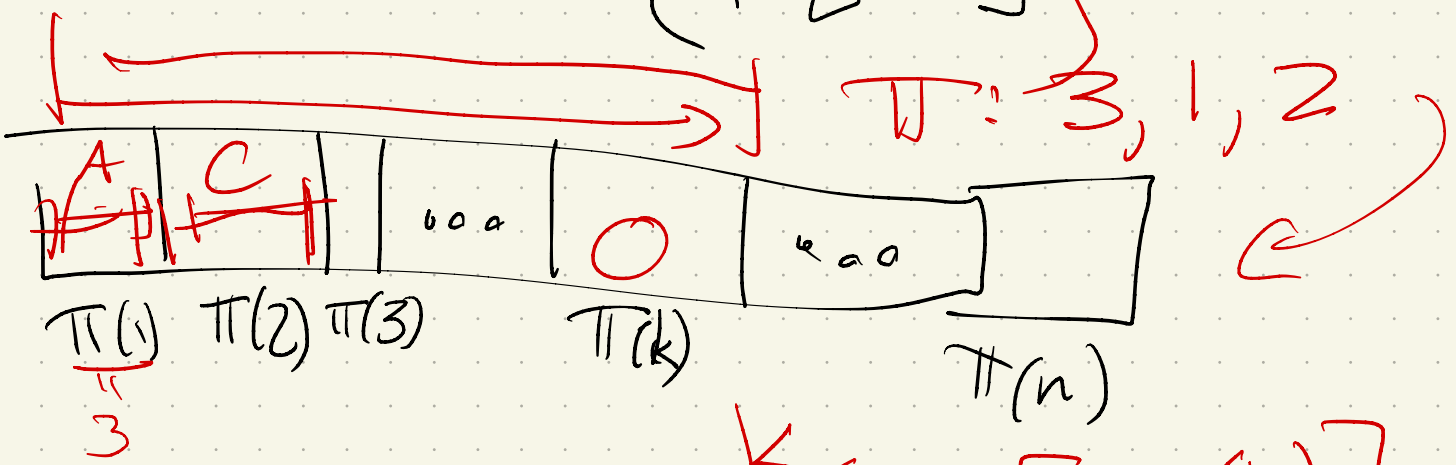
π: 2, 3, 1

His maths:

We can re-order: permutation π

input order was $\begin{cases} L[1..n] \\ F[1..n] \end{cases}$ 1..n

Reorder:



cost to access k^{th} one. $\sum_{i=1}^{k} L[\pi(i)]$

And: how many times do access it? $F[\pi(k)]$

Total: $(L[\pi(1)] + L(\pi(2)) + \cdots L(\pi(k)]$

$$\Sigma cost(\pi) = \sum_{k=1}^{n} \left(F[\pi(k)] \cdot \sum_{i=1}^{k} L[\pi(i)] \right) = \sum_{k=1}^{n} \sum_{i=1}^{k} \left(F[\pi(k)] \cdot L[\pi(i)] \right).$$

perm: 1, 2, 3

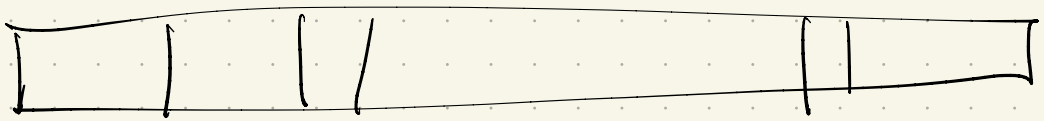
1, 2, 3: $\pi(1) = 1, \pi(2) = 2, \pi(3) = 3$

3, 2, 1: $\pi(1) = 3, \pi(2) = 2, \pi(3) = 1$

2, 3, 1: $\pi(1) = 2, \pi(2) = 3, \pi(1) = 1$

Lemma: Sort by $\frac{L[i]}{F[i]}$ & will get optimal schedule.

pf: Suppose we sort.



& $\forall i$, $\frac{L[i]}{F[i]} \leq \frac{L[i+1]}{F[i+1]}$

Suppose this is not optimal.
What does that mean?

Well, OPT must be different,
so $\exists$ out of order pair:

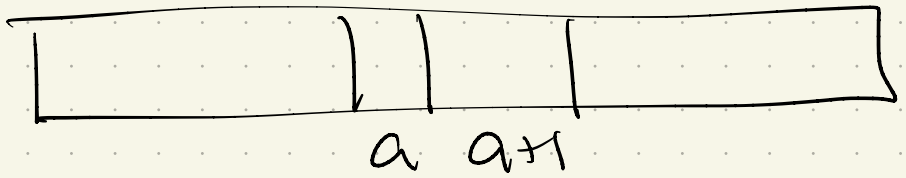
OPT : i

with $\frac{L[a]}{F[a]} > \frac{L[a+1]}{F[a+1]}$

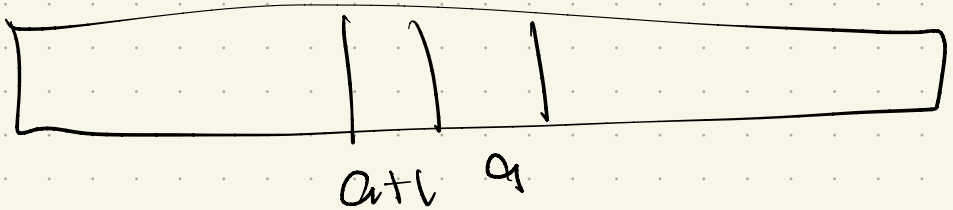
If OPT, must beat our "sorted" solution.

What if we swap a & $a+1$?

Before:



After:



What is the difference?

So: algorithm

- Calculate $\frac{L[a]}{F[a]}$ for all a.
- Sort, + permute order of jobs to match.

Runtime: