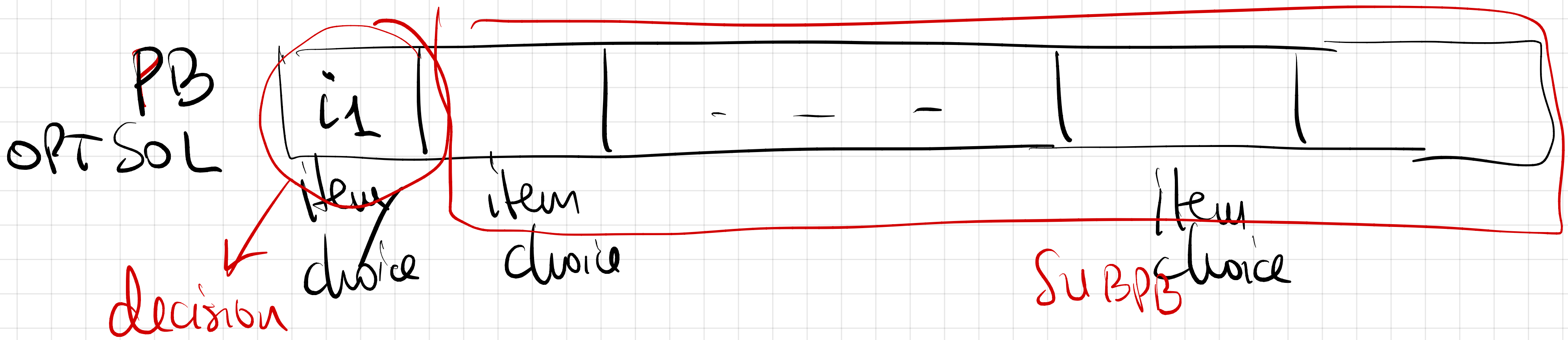
Lecture 5 : Greedy Algorithms (Divide & Conquer)



Procedure

1) understand OPTSOL (SOL)
2) understand PB → SUBPB — dependencies
3) DIVIDE & CONQUER
   - SPLIT into subpb (make first decision)
   - SOLVE subproblems
   - COMBINE SUBPB-SOL GLUE

SUBPB glued ⇓ OPTSOL

Divide & Conquer PB

Greedy

- Divide/decide/split without solving SUBPB
  - Greedy criteria (math proof OPTSOL)
- SUBPb, solve it/then
- Combine/Glue

DP

- Solve many subpb (some that we don't need)
- make decision/split/divide
  - based on subpb solution ("dynamic")
- Combine/Glue

decision/split PB ⟶ SUBPB

OPTSOL = SOL(PB) ⟵ (glue) SOL(subpb)

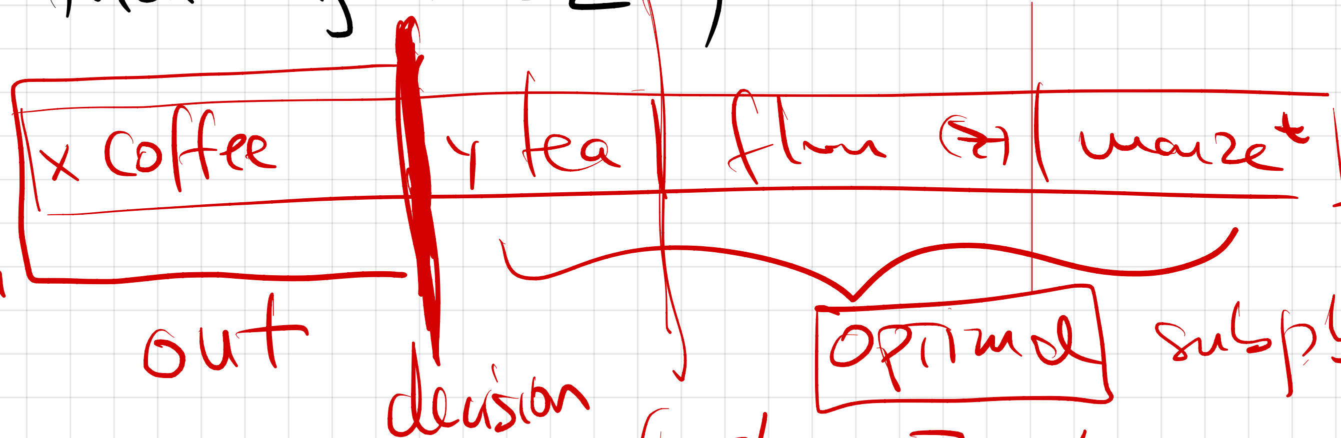
Greedy 1: Fractional Knapsack

total weight = $Z = 50$

items	coffee	tea	flour	maize
total weight (LB)	25	50	20	70
total val	30	40	15	10
q_i:	30/25	40/50	15/20	10/70

Task: put items in knapsack (fractional quant ok) and total weight $\leq Z$, max value.

OPTSOL characterization



$Z' = Z - x$

items	c	t	f	m
	$25-x$	50	20	70
val	$30 - \frac{x}{25}\cdot 30$?	40	15	10

Greedy criteria: (pick) item highest total value

(1)

tea val = 40

either { • use entire quantity available
OR
• knapsack full }

exercise
counterexample

Greedy 2: (pick) item smallest total weight

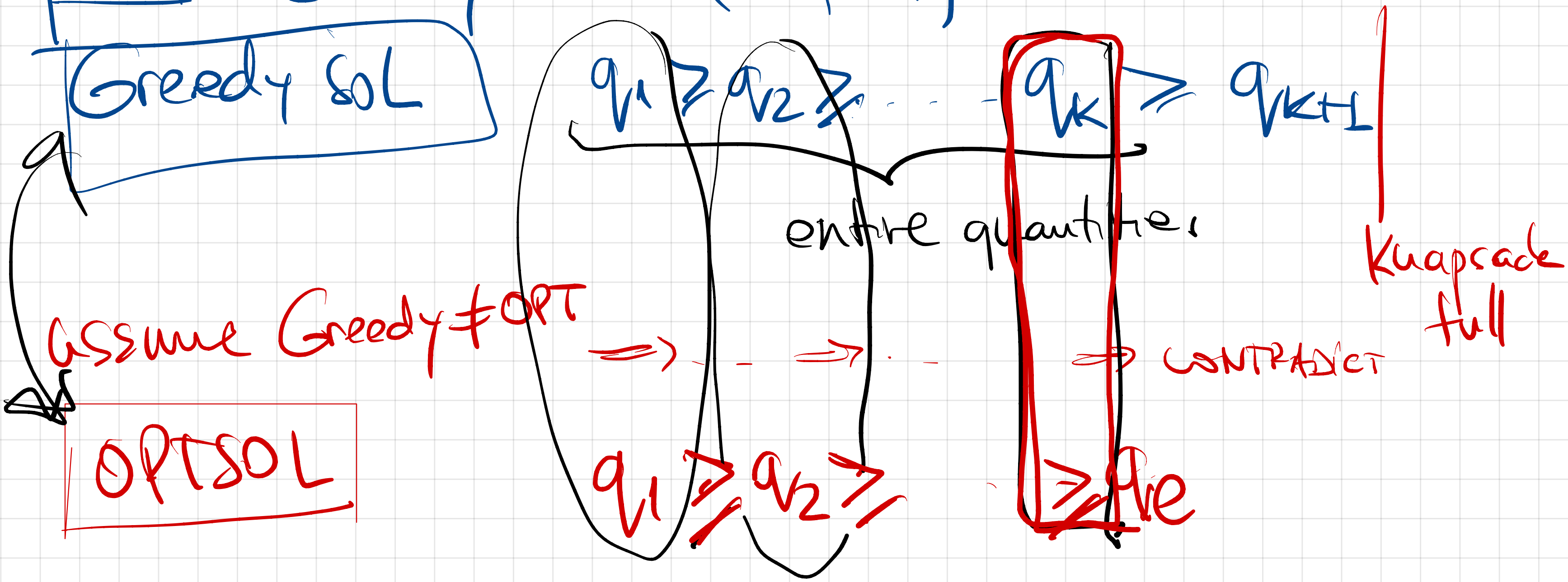
Greedy (correct): Sort items by quality = $\frac{\text{value}}{\text{weight}}$

$q_1 \geq q_2 \geq \dots \geq q_n$

(pick) highest quality

coffee first.

Proof: Greedy choice (sig of) ⟹ OPT SO



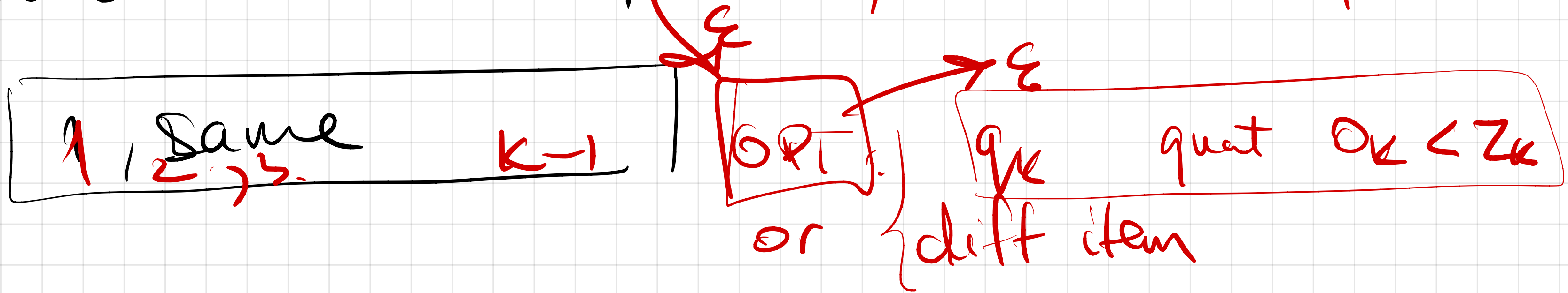
Step-by-step analysis GREEDY vs OPT

First item: is the same? is it same amount?

if yes+yes second item, is the same? same quantity?

yes+yes move to 3rd: is the same? same quantity

GRE ≠ OPT at some k ! NO! Greedy q_k AMAP quant z_k



then at this point I am going to improve OPTSOL

exchange:

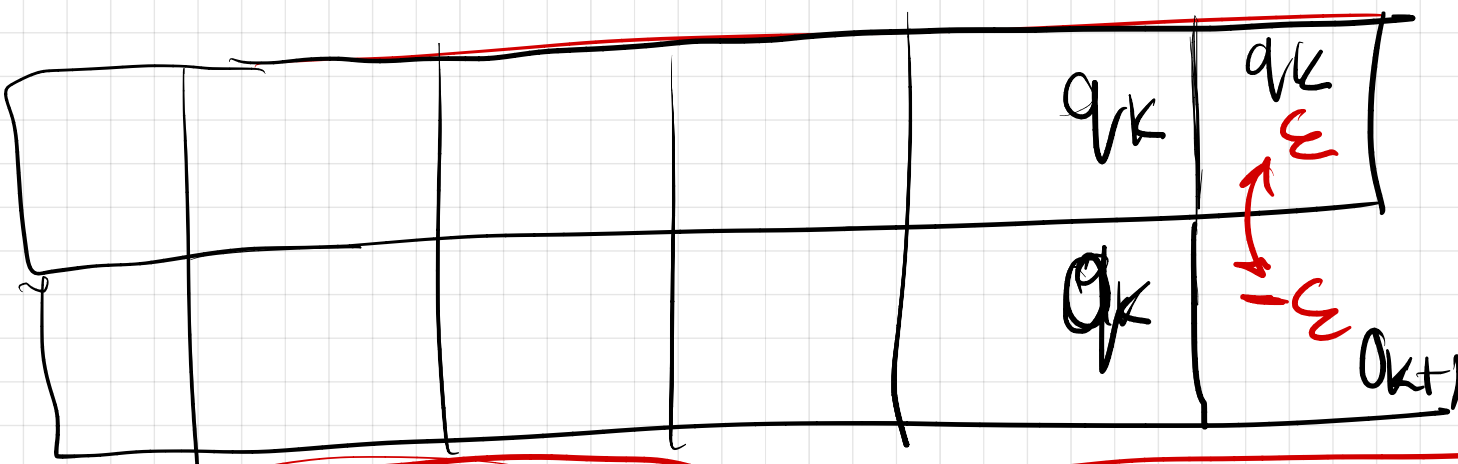
ε-weight take some item$_k$ in OPTSOL → out

replace it with ε-weight of q_k (Greedy sol) > better quality on going OPTSOL

weight same $-\varepsilon+\varepsilon$, quality

value $\Delta = \varepsilon(q_k - o_k) > 0$

(q_k → Gre, > 0, o_k → OPTSO) CONTRAD!



if Greedy ≠ OPT ⟹ OPT is not actually not best value CONTRAD

Activity Selection PB

n activities (start, finish time)

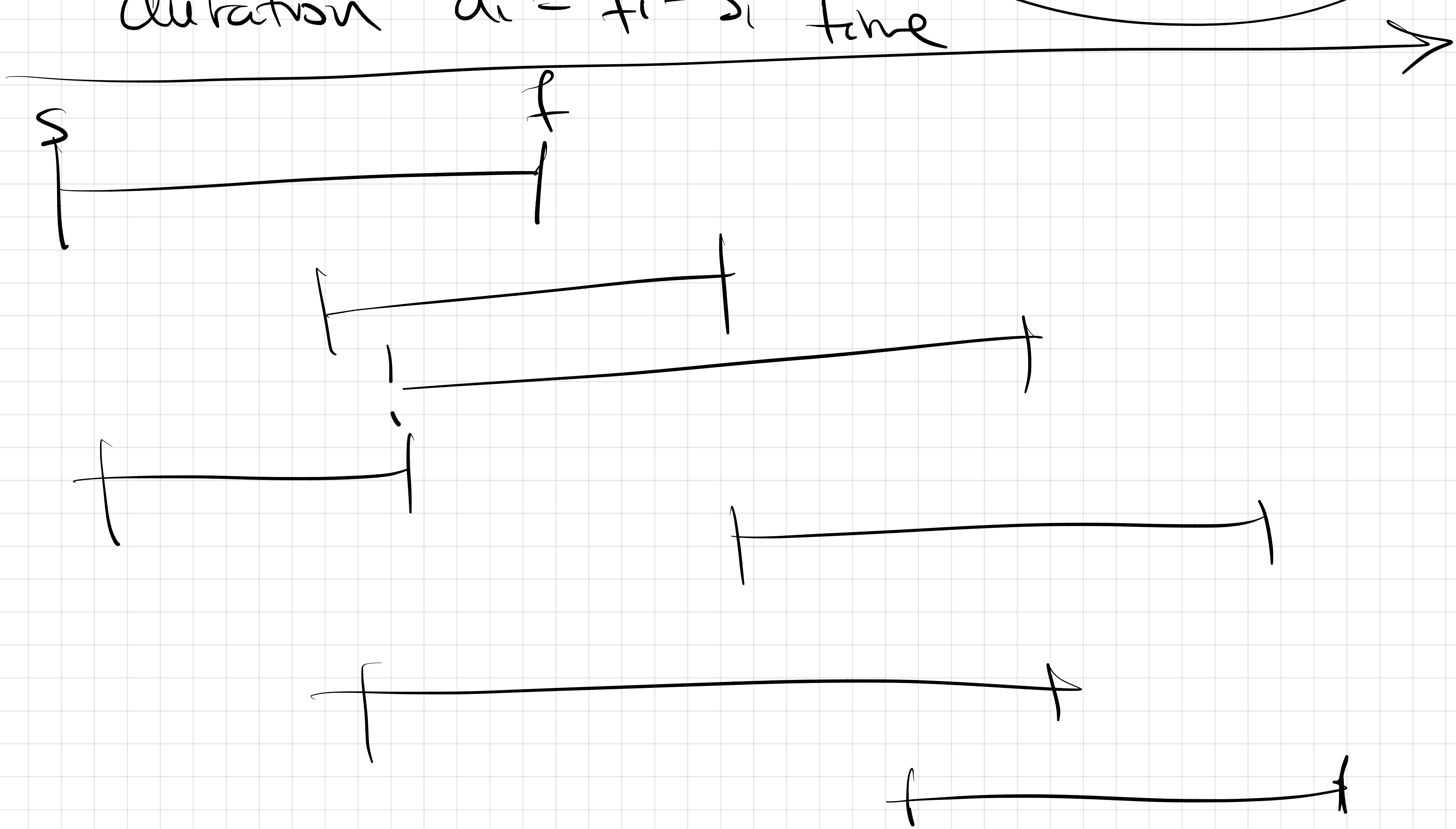
given

$s_1 \ s_2 \ s_3 \ \ldots \ s_n$

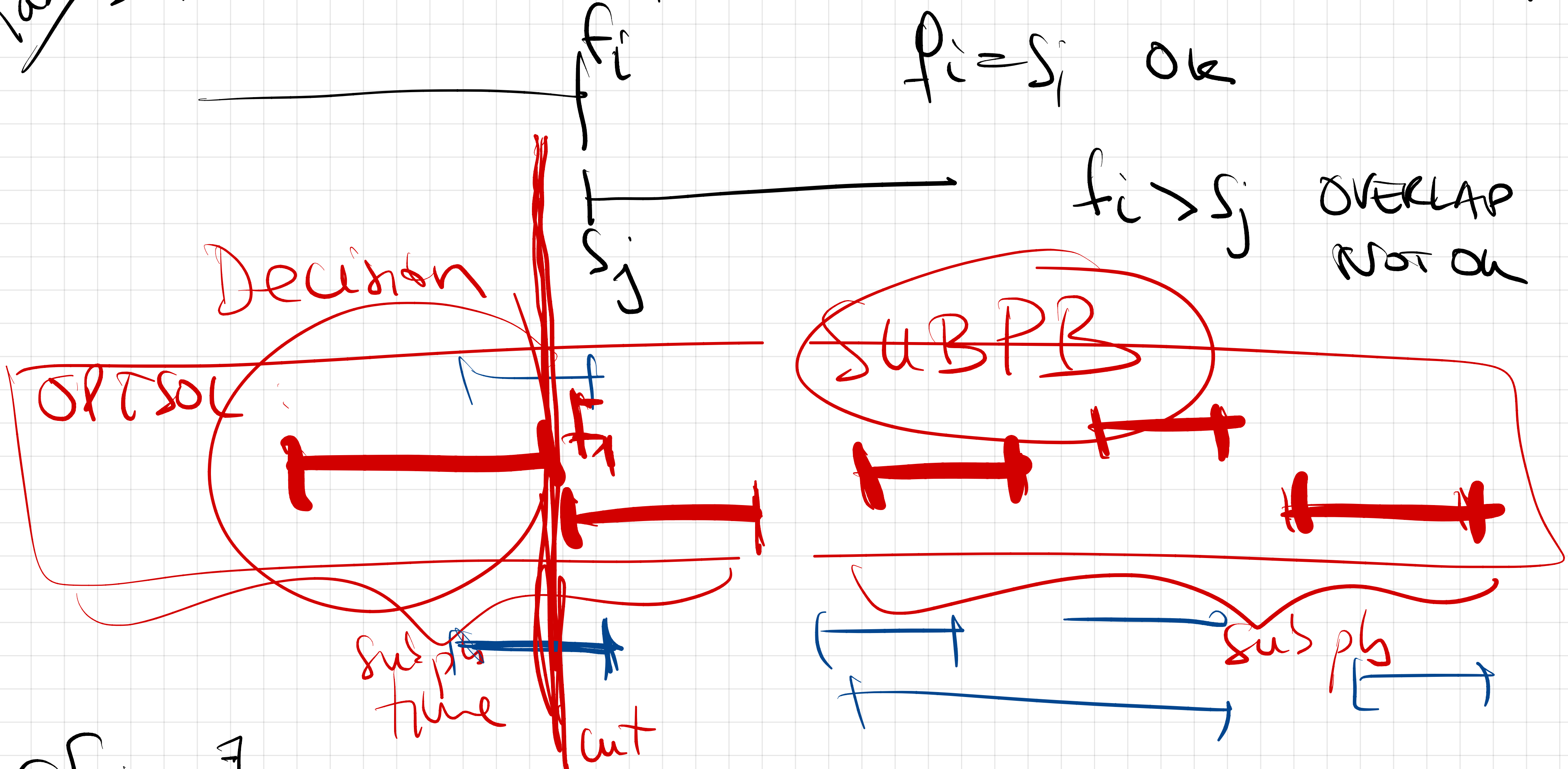
$f_1 \ f_2 \ f_3 \ \ldots \ f_n$

$s_i < f_i$

duration $d_i = f_i - s_i$



Task / Select max # of activities that do not overlap

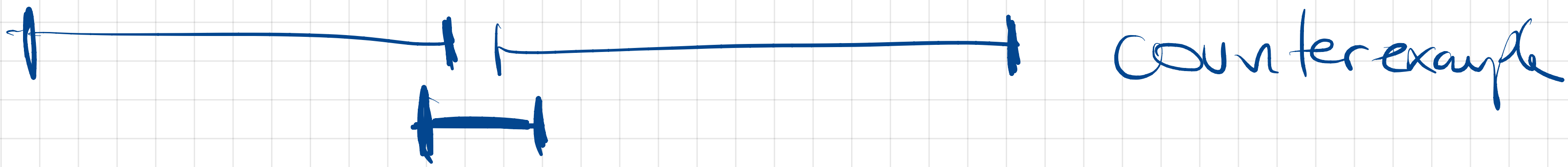


C[input] = OBJ value

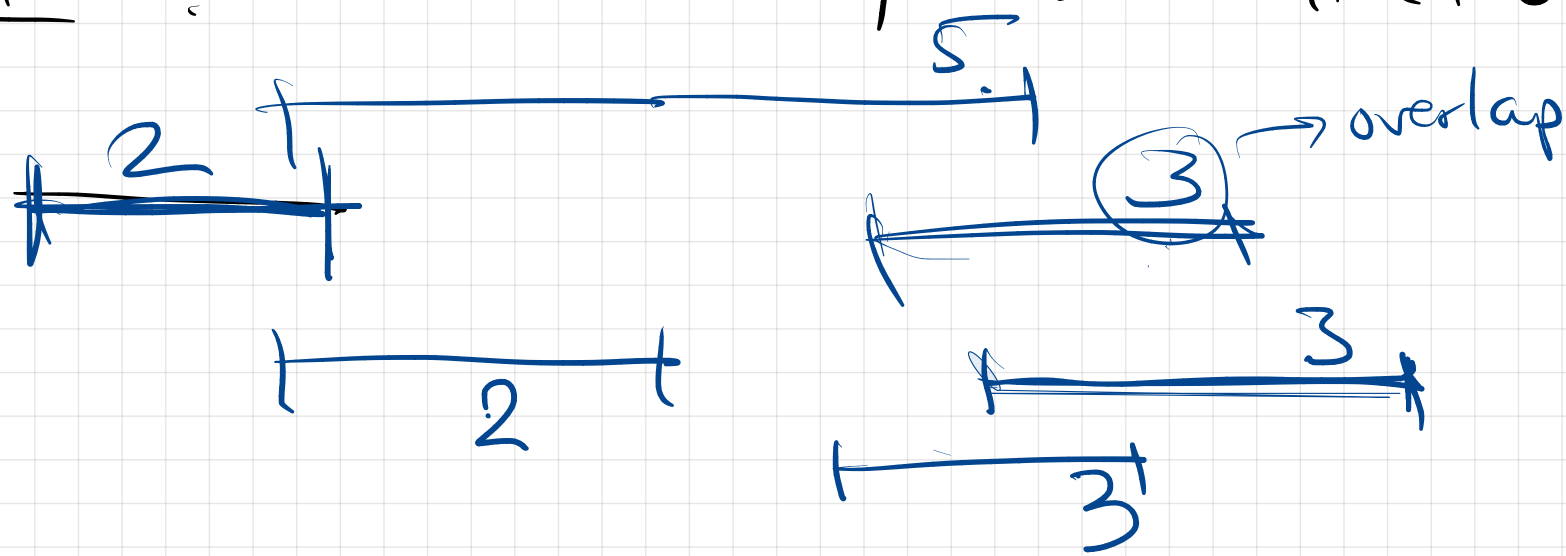
$C[(s_1,f_1)(s_2,f_2) \dots (s_n,f_n)] = \#$ max select

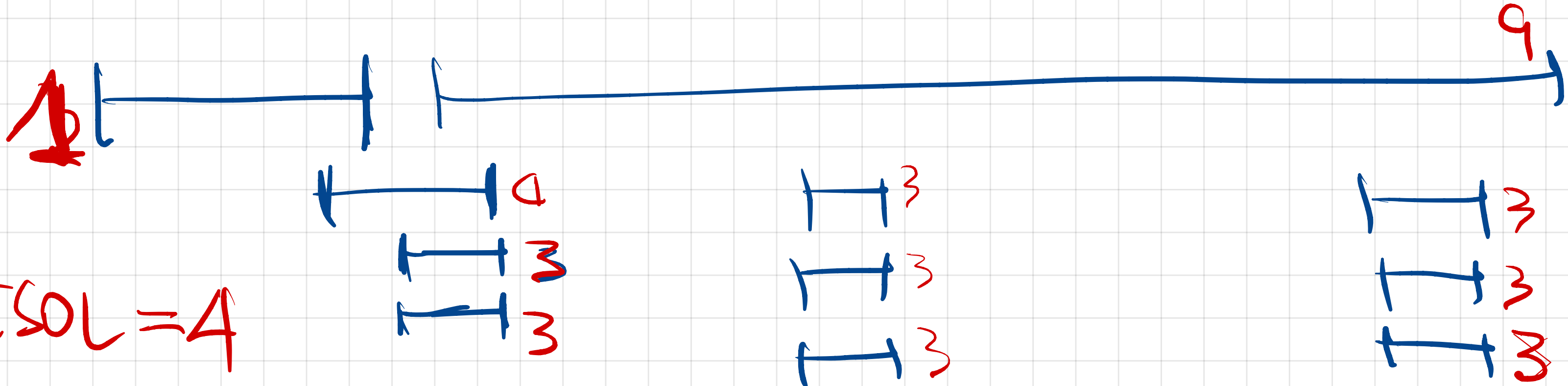
$= 1 + C\left[\begin{array}{c}\text{all activities } j \\ s_j > F_1\end{array}\right]$ time ~~$= F_1$~~

Greedy criteria: choose activity smalles duration $f_i - s_i$



Greedy 2: choose activity smalles #act overlap

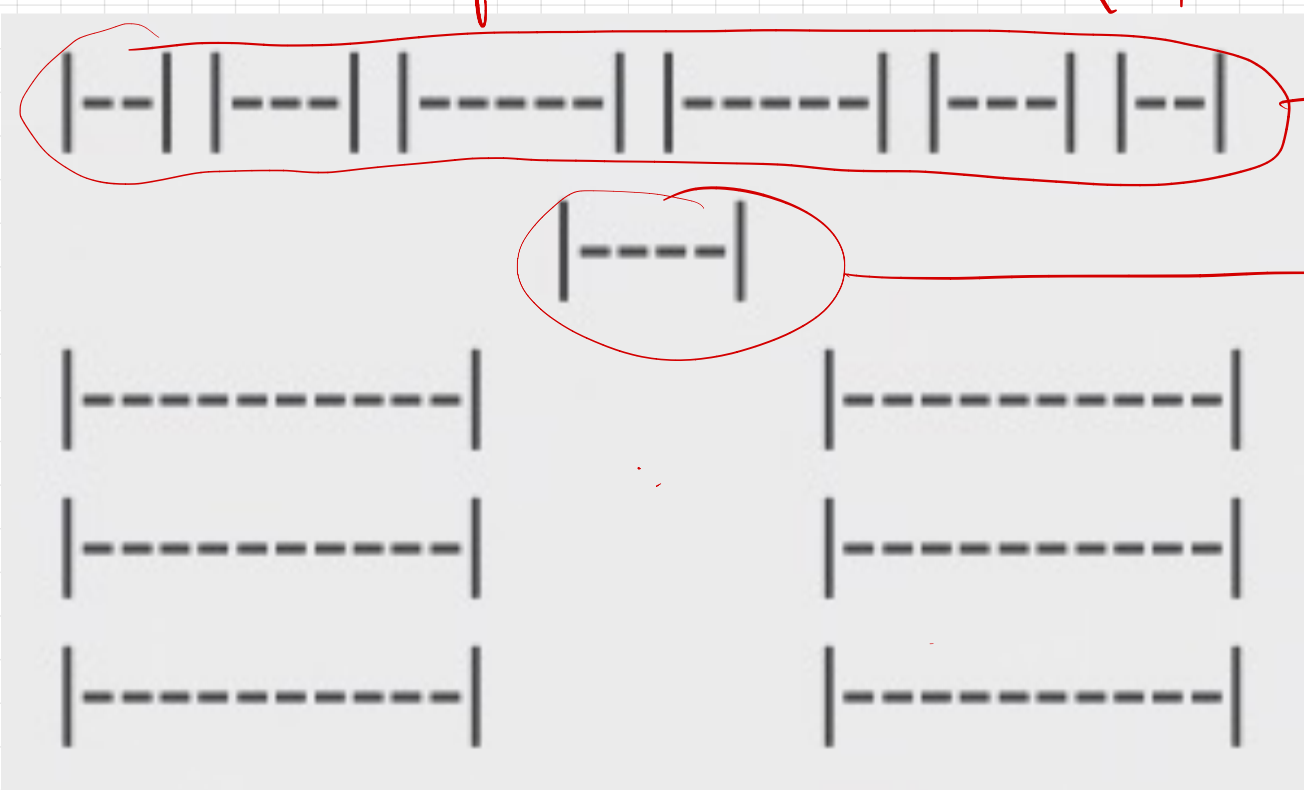




OPTSOL = 4

Greedy by "min overlap": counterexample? INCORRECT!

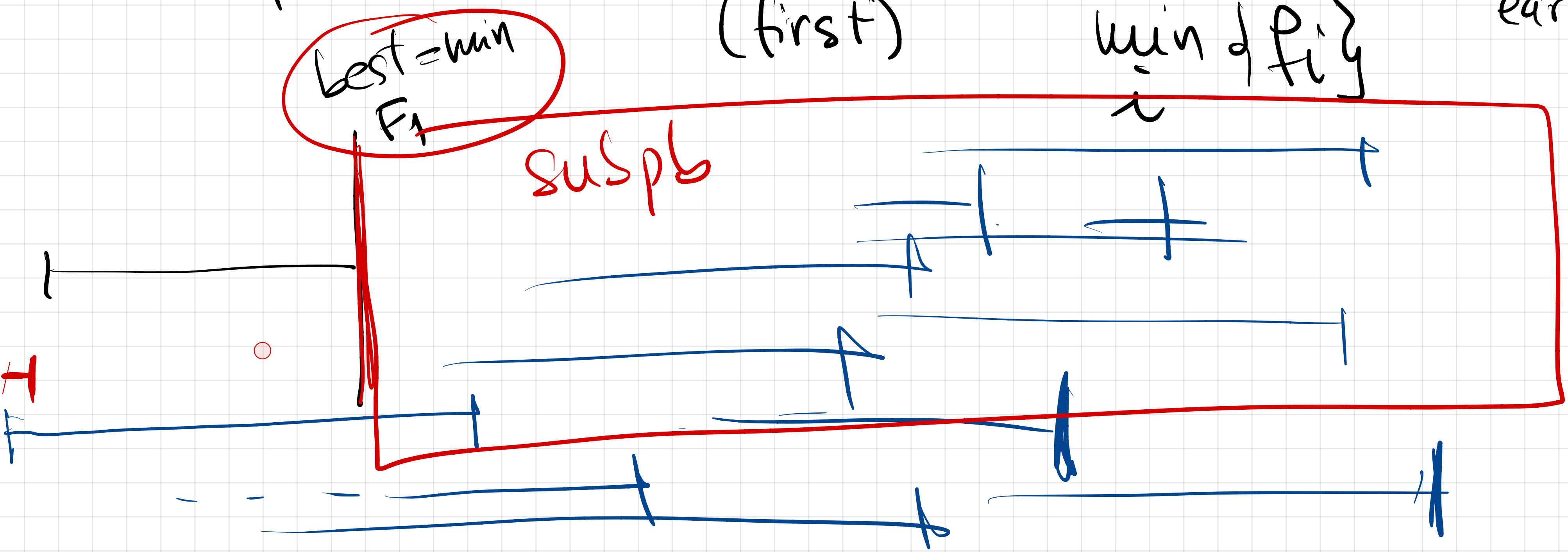
Counterexample – take two (by students)



→ OPTSOL

→ greedy choice prevents OPTSOL

Greedy Correct: choose / pick (first) activiti finishes first earliest $\min_i \{f_i\}$



ALG: Sort act by f_i

Loop
- pick first (available) $(s_i - f_i)\ a_i$
- delete/invalidate all act overlapping a_i

  $\forall j\ s_j < f_i$

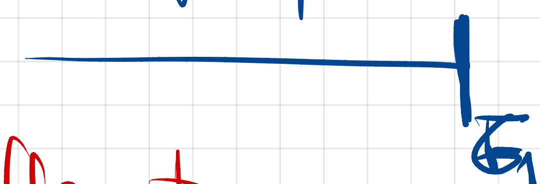
while you can

Greedy sol as good as any other

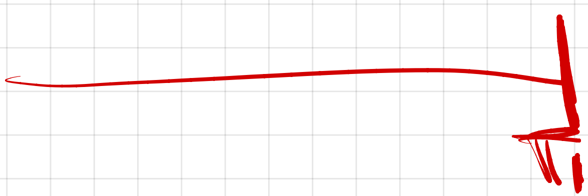
I can follow any subseq choices

proof: Stay ahead argument (∨ exchange)

after greedy choice ⇒ SUBG (act available, time available) $s_j > G_1$, time $> G_1$

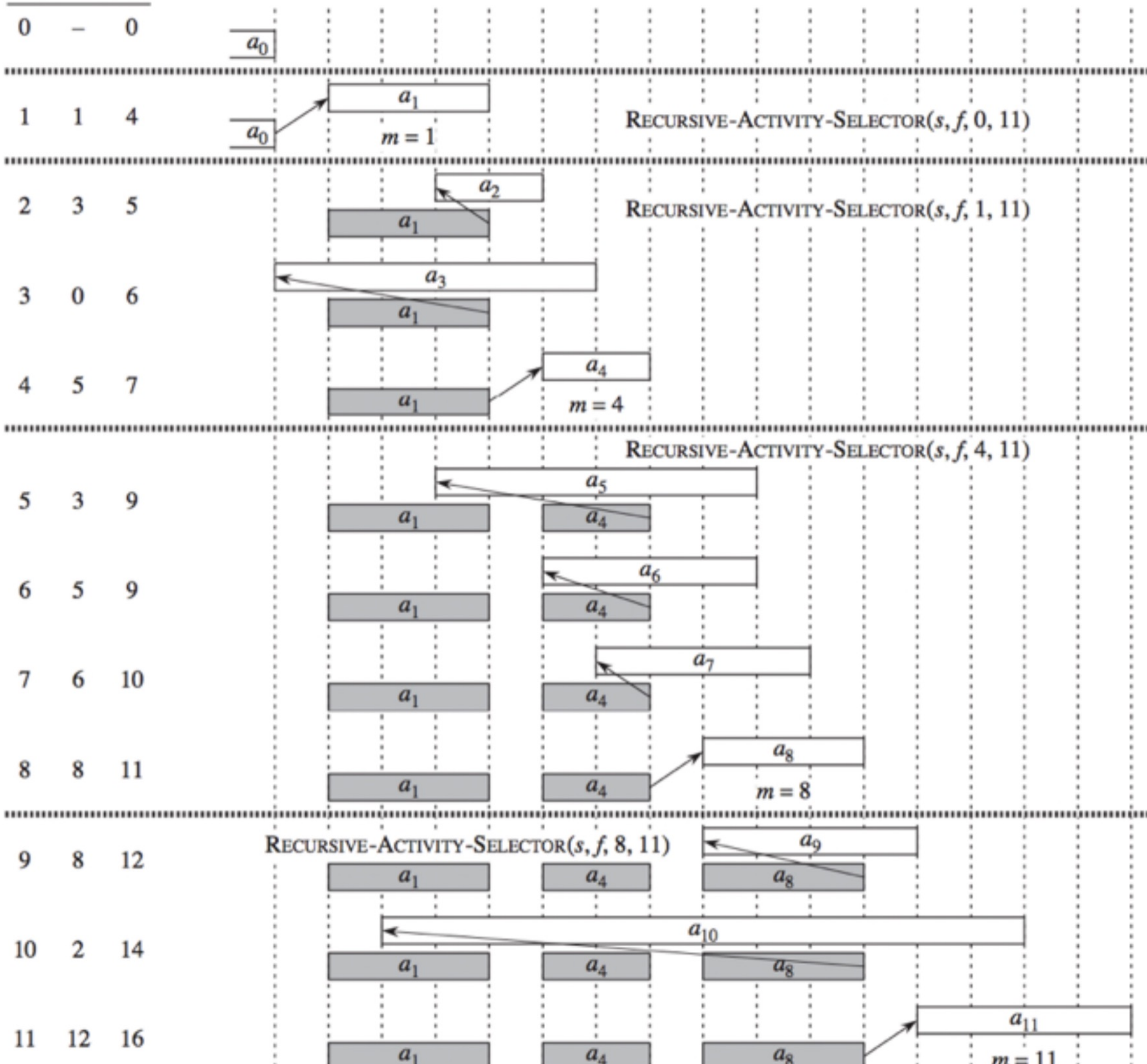


different first choice T → SUBT (act avail $s_j > T_1$, time $> T_1$)



OBJ

$C[SUBG] \geq C[SUBT]$



0 – 0
a_0
1 1 4
a_1
$m = 1$
RECURSIVE-ACTIVITY-SELECTOR(s, f, 0, 11)
2 3 5
a_2
RECURSIVE-ACTIVITY-SELECTOR(s, f, 1, 11)
3 0 6
a_3
4 5 7
a_4
$m = 4$
RECURSIVE-ACTIVITY-SELECTOR(s, f, 4, 11)
5 3 9
a_5
6 5 9
a_6
7 6 10
a_7
8 8 11
a_8
$m = 8$
RECURSIVE-ACTIVITY-SELECTOR(s, f, 8, 11)
9 8 12
a_9
10 2 14
a_{10}
11 12 16
a_{11}
$m = 11$

type of arguments:
- stay ahead
- exchange
- contradict
- induction

ind step greedy proof

if first n Greedy choices are part of some OPTSOL $\Longrightarrow$ the first n+1 Greedy choices are part of some OPTSOL

Coin Change Denominations example U.S.

$\{d_1, d_2, d_3 \ldots d_k\} = \{1, 5, 10, 25, 50, 100\}$

n cents give change $n = d_{i_1} + d_{i_2} + d_{i_3} \ldots$

min # of coins

Greedy: largest coin that fits. yes for US.

30 = 25 + 5

29 = 25 + 1 + 1 + 1 + 1

22 = 10 + 10 + 1 + 1

exercise: proof greedy works for U.S. denom

Greedy does not work

denom = {1, 2, 7, 10}

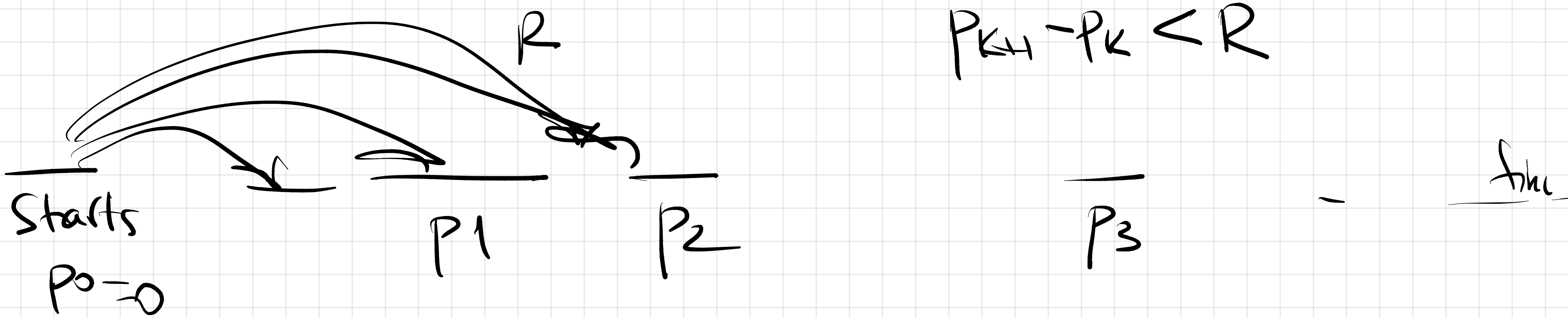
n = 14 = 10 + 2 + 2 not optimal

** What denom set makes greedy work?

Frog can jump R feet

River with stones in pos $p_1, p_2, \ldots, p_n \in \mathbb{R}^+$

$p_{k+1} - p_k < R$



min #jumps

Greedy: jump as far as possible

Stay ahead arg (exercise)

Egyptian fractions $x \in \mathbb{Q}^* \{ \frac{a}{b} \}$

$$x = \sum_{k=1}^{I} \frac{1}{n_k}$$

- is this always possible — yes

if yes, alg = greedy

ex

$$\frac{5}{6} = \frac{1}{2} + \frac{1}{3}$$

$$\frac{7}{15} = \frac{1}{3} + \frac{2}{15} = \frac{1}{3} + \frac{1}{8} + \frac{1}{120}$$

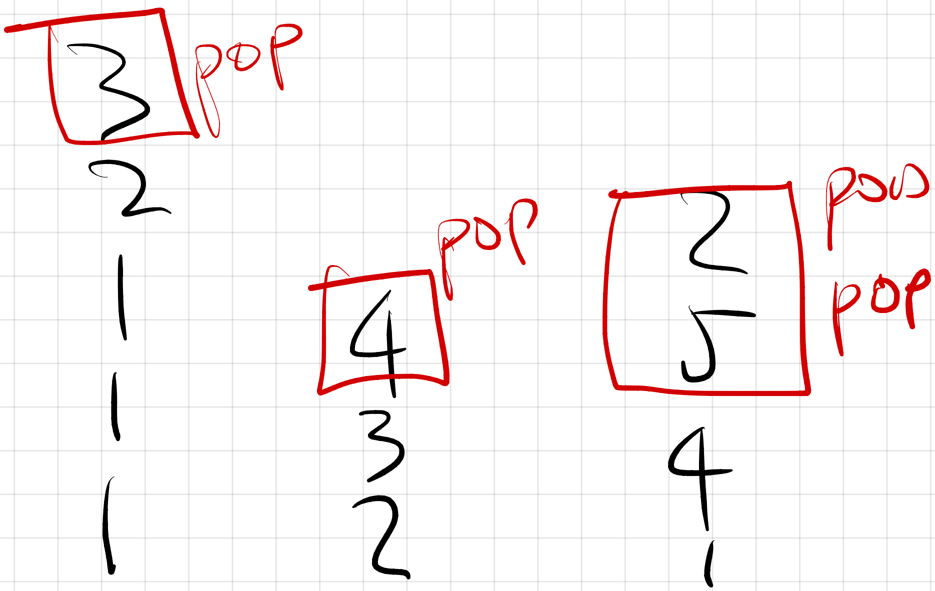
$$\frac{5}{121} = \frac{1}{33} + \frac{1}{121} + \frac{1}{363}$$ not greedy

Greedy: take fraction with smallest denom. possible

$$\frac{5}{121} = \frac{1}{25} + \frac{1}{757} + \frac{1}{763309} + \frac{1}{\text{big}} + \frac{1}{\text{big}}$$

(Th) process is not infinite because the numerator of sums keep decreasing (exercise)

3 Stacks get them to have same sum (by pops) (if possible)



Greedy: pop from the stack with highest sum

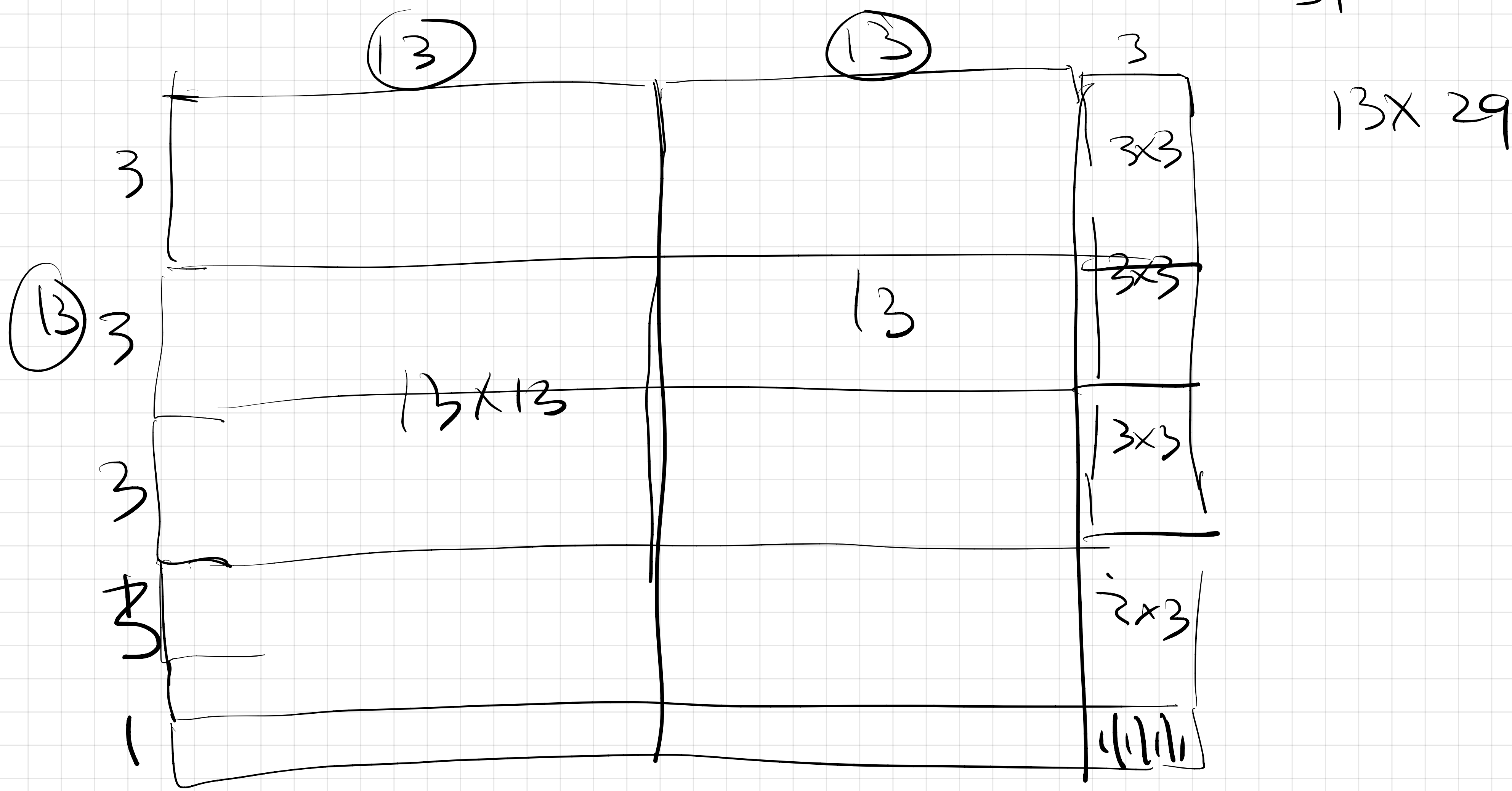
$$\Theta(\sum sizes)$$

<u>proof</u>: if there is a solution, and currently the highest-sum stack "S" is <u>not</u> equal with the others, then "S" top element has to be popped.
So doing it is not a waste of time!

cut paper m×n into squares

∈ ℤ×ℤ

want: min # squares



Greedy: cut □ = side ? exercise

Huffman Codes

symbols set	a	b	c	d	e	f	g
probab of occ	.05	.05	.1	.2	.3	.2	.1

want encode symbol ⟶ string of bits
ex b → 0101?

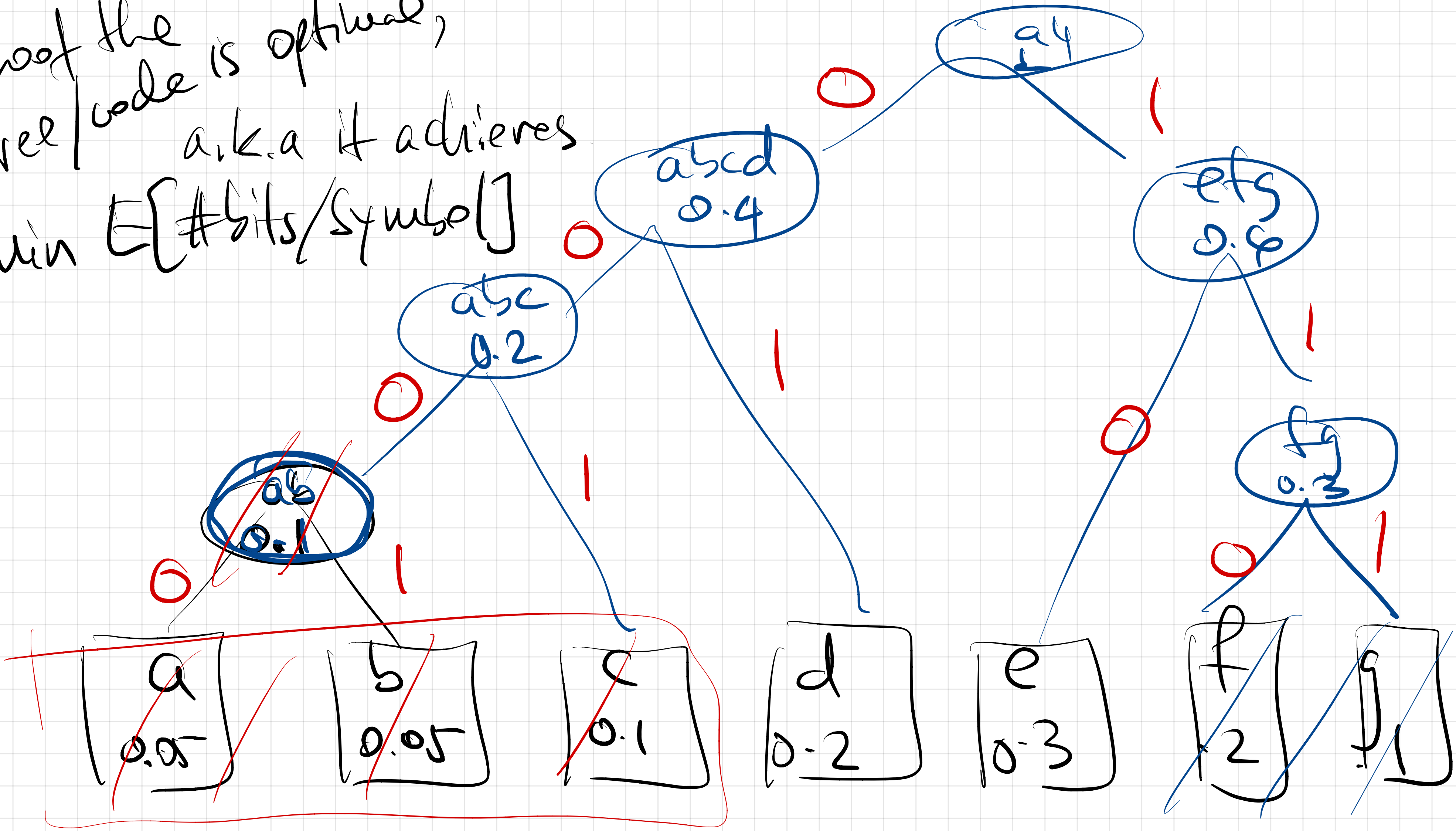
1) cond: be able to decode "prefix free property"

010 code ⟹ 010 any not a code

2) obj minimize $E[\#\text{bits}/\text{symbol}]$

exercise

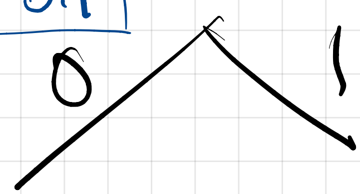
Proof the tree/code is optimal,
a.k.a it achieves
min E[#bits/symbol]



Greedy: join the two smallest ones → tree

ab → ab 0.1

Huffman code:



b = 0001
e = 10
d = 01

clearly: prefix-free ✓