Quiz + go over quiz

Done at 9:~~45~~ 50

we can take a little more time if needed

1. (5 points) The pseudocode for Dijkstra's algorithm is given below. The input to the algorithm is a directed graph G with edge weights denoted by ℓ_e for edge e.

Dijkstra's algorithm. Dijkstra's alg runs in $\Omega(\log n)$ time in worst case. T/F

```
Let S be the set of explored nodes
For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V:
  Select a node v ∉ S with at least one edge from S for which d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e
  is as small as possible
  Add v to S and define d(v) = d'(v)
```

Notice that the while loop runs exactly $n-1$ times. Select all that are true:

- The while loop runs $\Omega(n)$ times.
- The while loop runs $O(n)$ times.

True

$\Theta(n)$ times

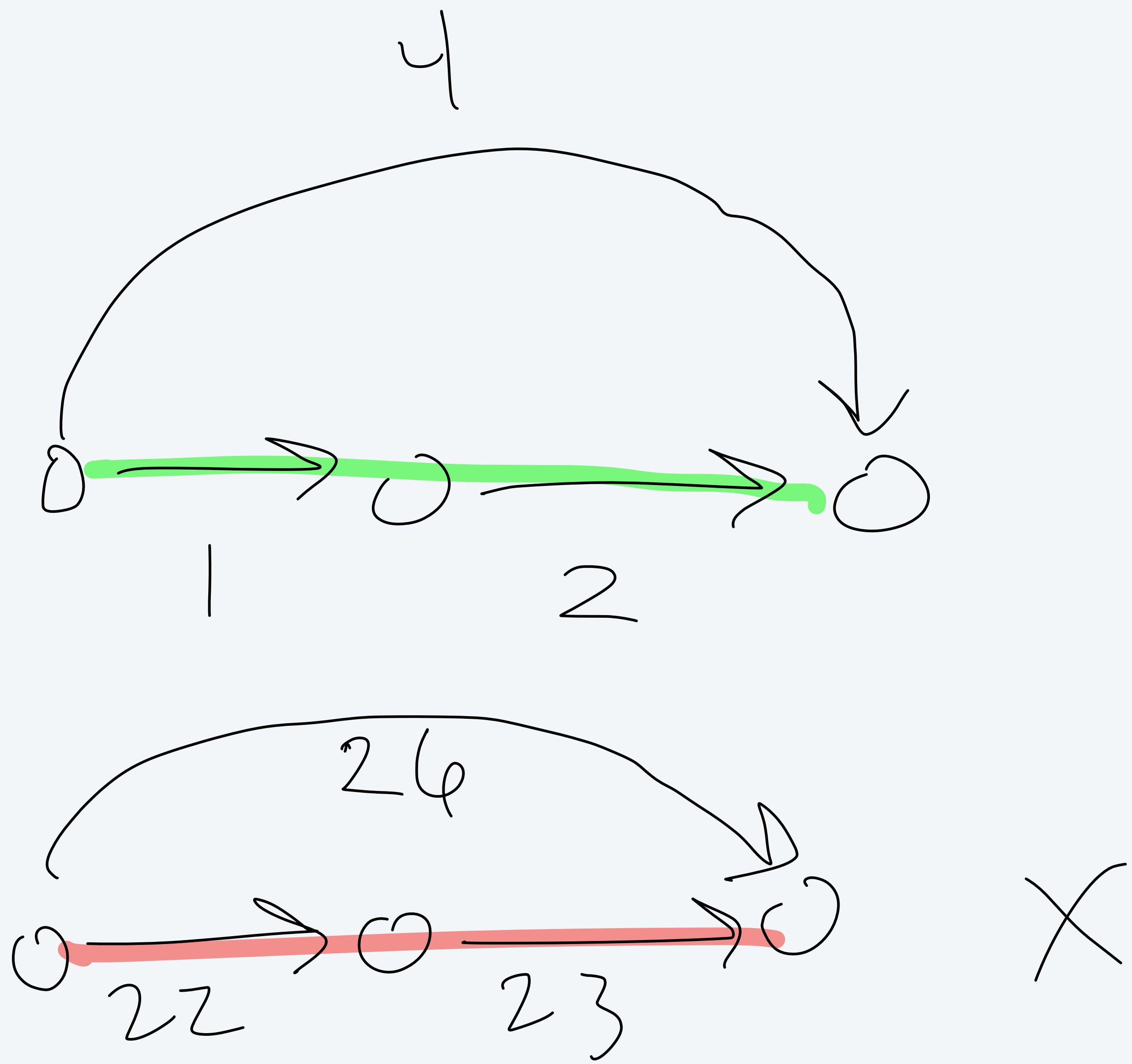
- In the worst case, Dijkstra's algorithm runs in $\Omega(n)$ time.
- In the best case, Dijkstra's algorithm runs in $\Omega(n)$ time.
- In the average case, Dijkstra's algorithm runs in $\Omega(n)$ time.

True

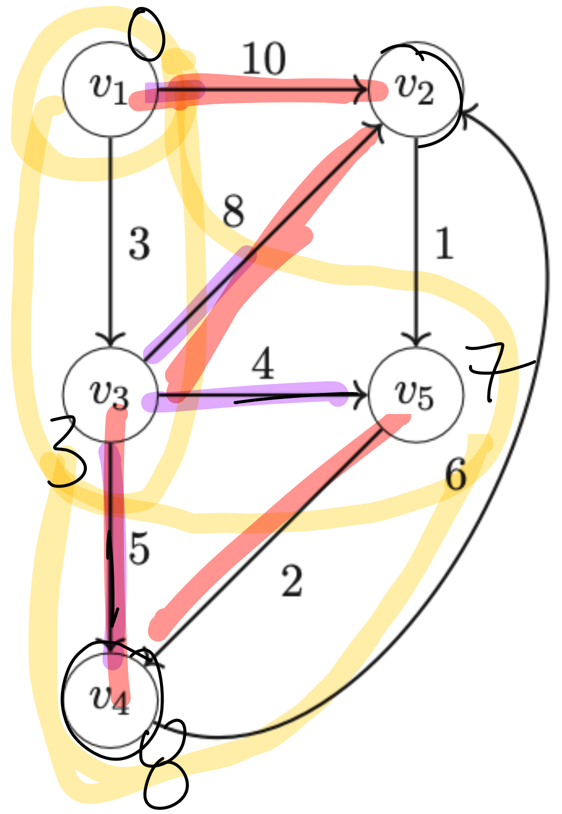
$f(n)$ is Ω of $g(n)$ if there exist $n_0, c \geq 0$ s.t. for all $n \geq n_0$, $f(n) \geq c \cdot g(n)$

2. (5 points) Suppose we are given an instance of the Shortest s-t Path Problem on a directed graph G. We assume that all edge costs are positive and distinct. Let P be a minimum-cost s-t path for this instance. Now suppose we replace each edge cost c_e by $c_e + 21$, thereby creating a new instance of the problem with the same graph but different costs.

Give a counterexample demonstrating that P may not still be a minimum-cost s-t path for this new instance.



3. (5 points) Trace the execution of Dijkstra's algorithm on the following graph using the table below. Notice that some of it is already filled in for you.



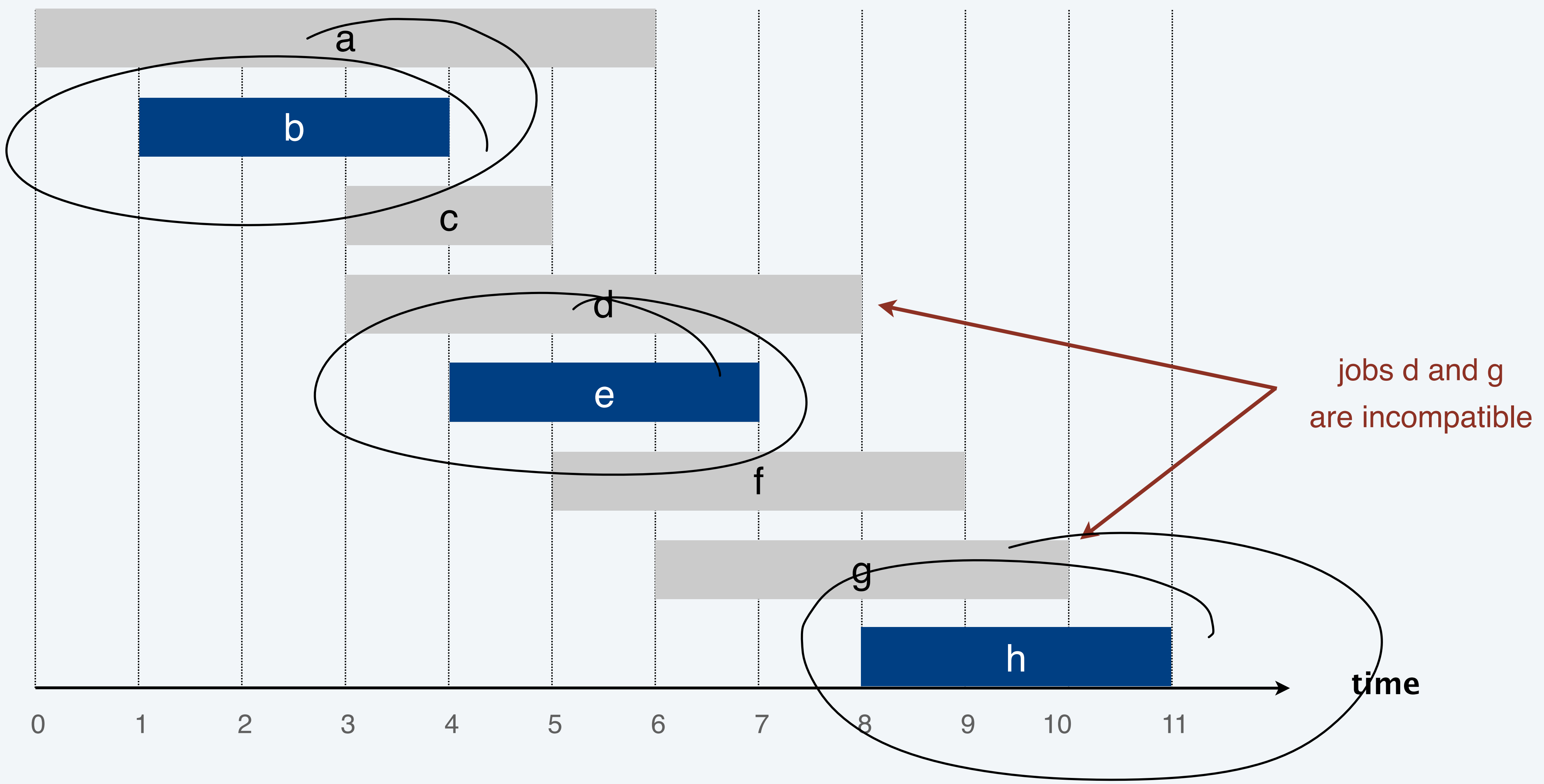
$d'(v_2) = \min(10, 11)$
$d'(v_4) = \min(8)$
$d'(v_5) = \min(7)$

$d'(v_2) = \min(10, 11)$
$d'(v_4) = \min(8, 9)$

	current S	current $d(u)$ values for $u \in S$	all $v \notin S$ with at least one edge from S	values of $d'(v)$	v to add to S
set up	n/a	n/a	n/a	n/a	v_1
while loop run 1	$\{v_1\}$	$d(v_1) = 0$	v_2, v_3	$d'(v_2) = 10, d'(v_3) = 3$	v_3
while loop run 2	$\{v_1, v_3\}$	$d(v_3) = 3$	v_4, v_5, v_2	$d'(v_2) = 10$ $d'(v_4) = 8$ $d'(v_5) = 7$	v_5
while loop run 3	$\{v_1, v_3, v_5\}$	$d(v_5) = 7$	v_4, v_2	$d'(v_2) = 10$ $d'(v_4) = 8$	v_4
while loop run 4					

Interval scheduling

- Job j starts at s_j and finishes at f_j.
- Two jobs are compatible if they don't overlap.
- Goal: find maximum subset of mutually compatible jobs.



Some local criteria that won't work…

Give a counterexample to why a greedy algorithm using each of the following local criteria would yield a global optimal solution for every input.

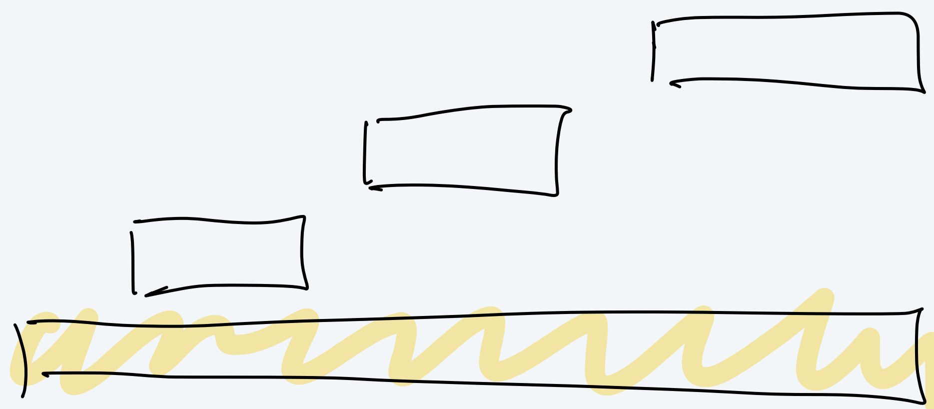
A. [Earliest start time] Consider jobs in ascending order of s_j.

B. [Shortest interval] Consider jobs in ascending order of $f_j - s_j$.

C. [Fewest conflicts] Consider jobs in ascending order of conflicts.

Consider jobs in order of s_j ✗

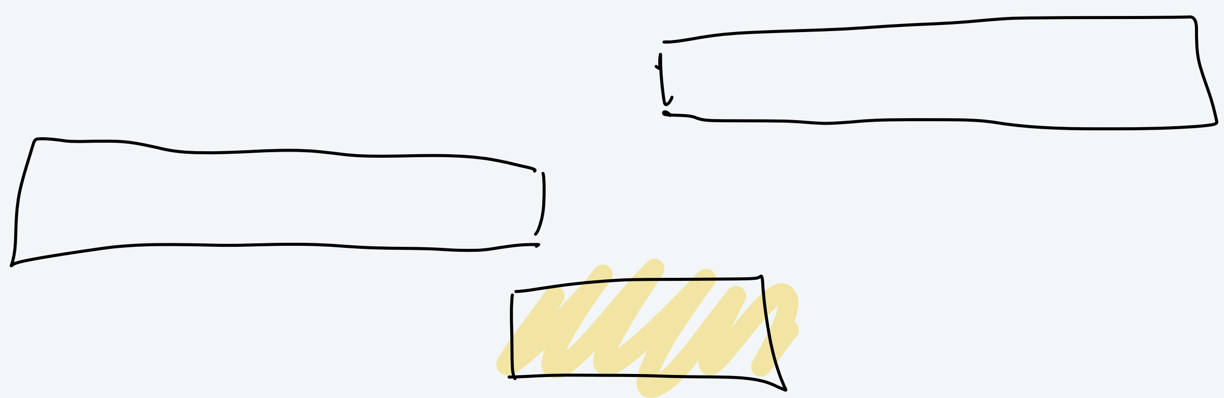
Counter example input:



optimal solution selects 3 jobs

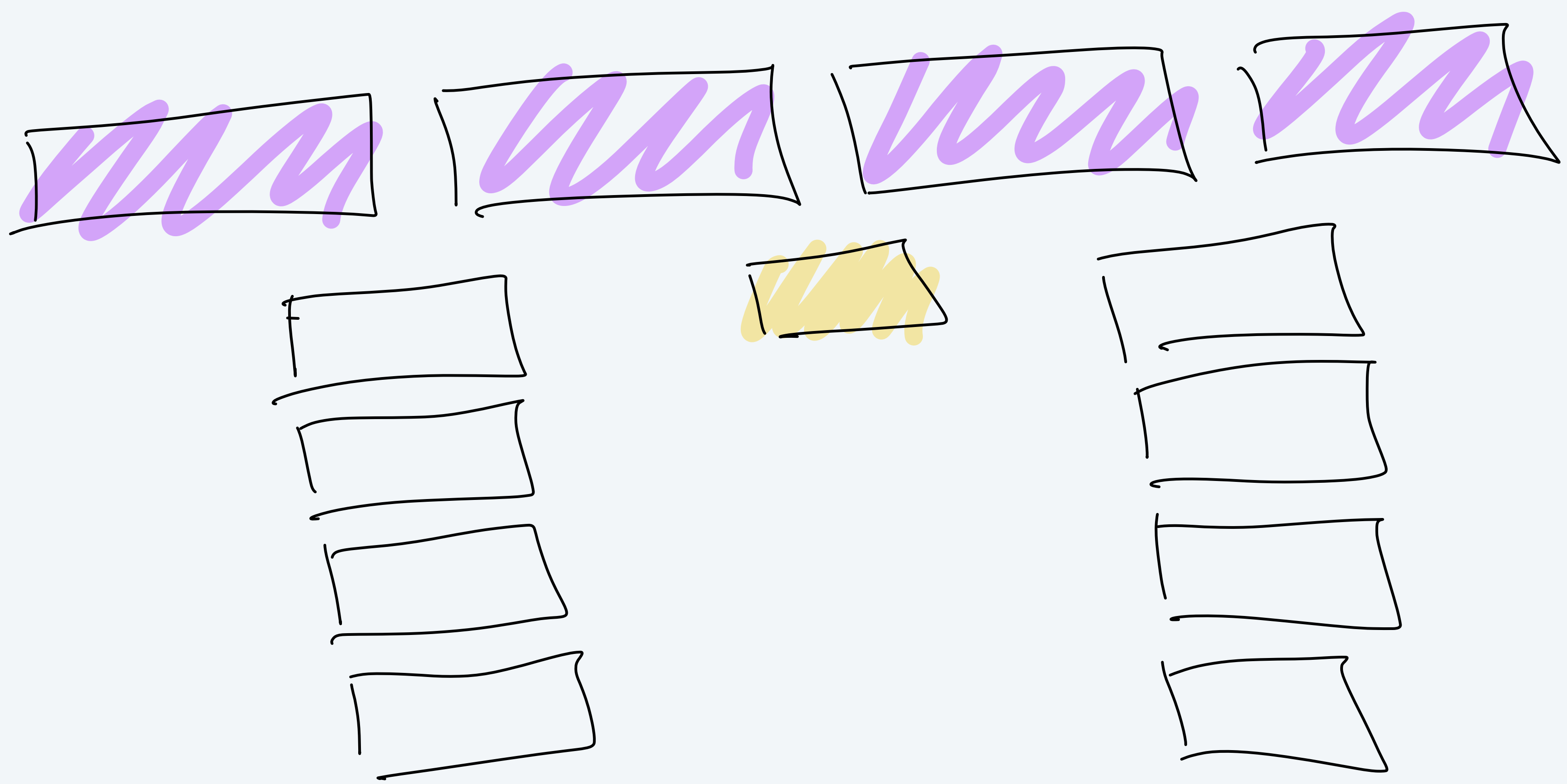
Consider jobs in order of $f_j - s_j$ ✗

Counter example:



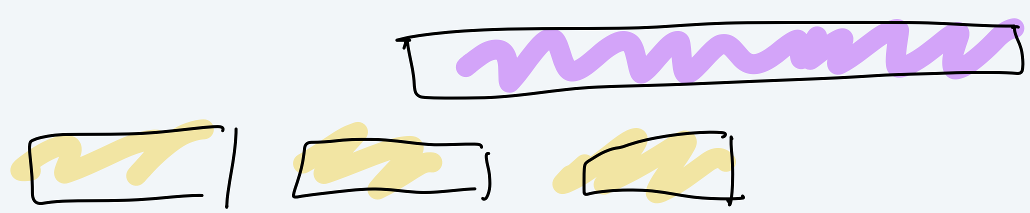
Consider jobs in order of fewest conflicts

Counter example

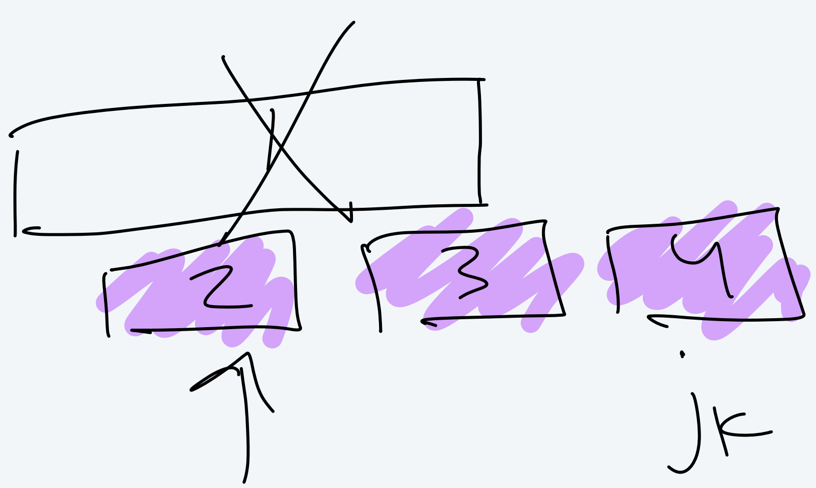


A criteria that will work

Any ideas? descending finish time. (last finishing time first)



earliest finishing time first? yes!



A criteria that will work

Any ideas?

sort jobs by finishing time and renumber so that $f_1 \le f_2 \le \cdots \le f_n$.

```
S = {}
for j = 1 to n:
    if (job j is compatible with jobs in S):
        add job j to S
return S
```