Lecture 21: Interval Scheduling

COSC 311 *Algorithms*, Fall 2022

Greedy Algorithms

So far: focused on (greedy) graph algorithms

1. Finding Eulerian circuits
   - greedily collect new edges
2. BFS (unweighted SSSP)
   - greedily explore nearest edges
3. Dijkstra (weighted SSSP)
   - greedily find closest vertex
4. Prim (MST)
   - greedily add lightest outgoing edge
5. Kruskal (MST)
   - greedily add lightest edge that doesn't create a cycle

Today

Interval Scheduling

1. Interval scheduling problem & motivation
2. Greedy algorithm for interval scheduling
3. Analysis technique: algorithm stays ahead

Interval Scheduling

Motivation.

- Timed access to finite resource, e.g., CPU time, classroom
- Receive requests consisting of
  - start time s
  - end time $t > s$
- Only one request can be serviced at a time
  - cannot service two "overlapping" requests

$r: [1, 3]$ $r'': [4, 6]$

$r': [2, 4]$

Interval Scheduling

Motivation.

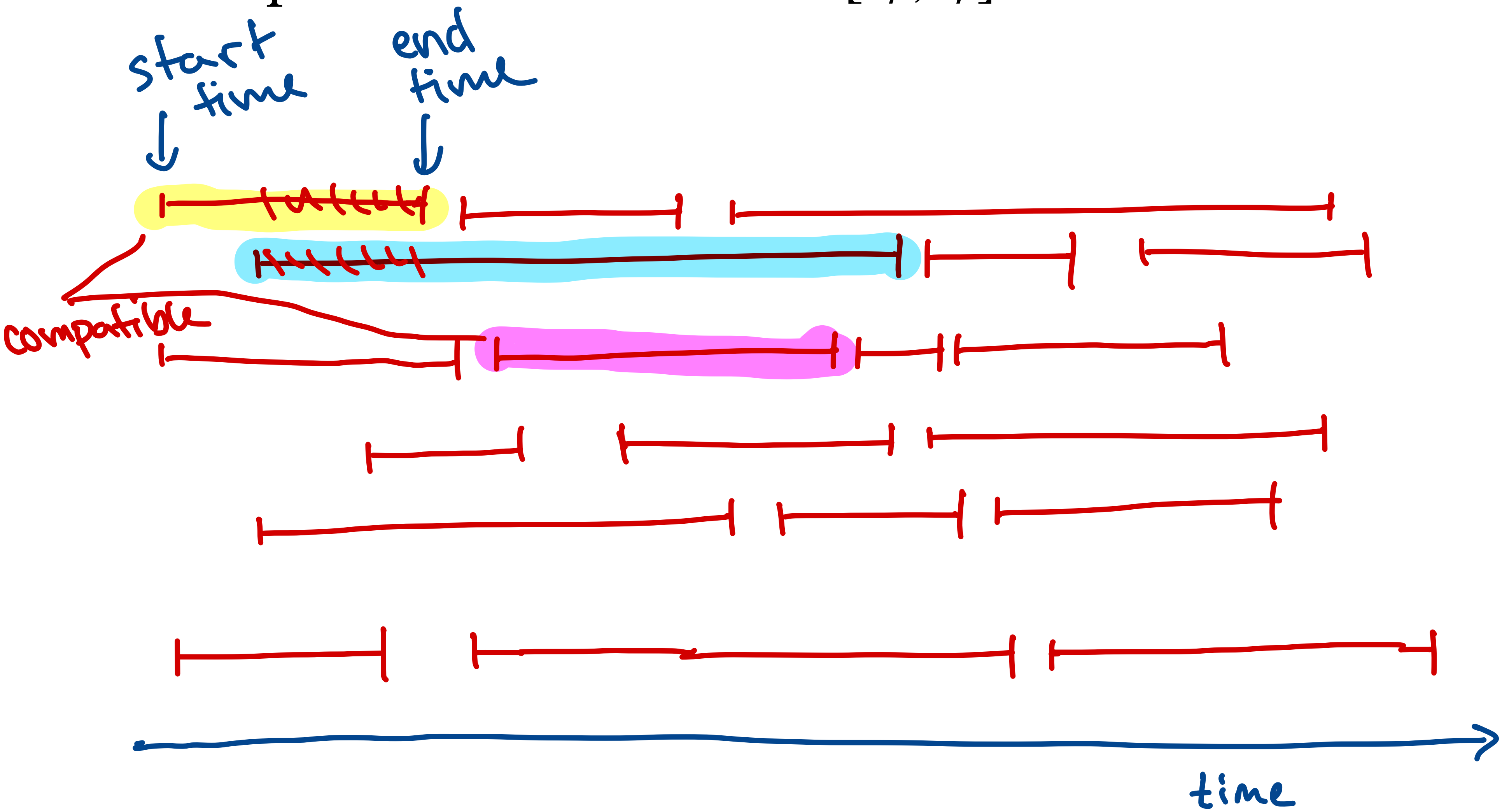
- Timed access to finite resource, e.g., CPU time
- Receive requests consisting of
  - start time s
  - end time $t > s$
- Only one request can be serviced at a time
  - cannot service two "overlapping" requests

Goal. Find a set of requests to service that are:

1. *feasible:* no two requests overlap
2. *optimal:* as many requests as possible are serviced, subject to feasibility

Example and Geometric View

View requests as intervals: $r = [s_r, t_r]$



A Meta-Strategy

To find a *feasible* set of requests to service:

1. Pick a request r to service (according to some criteria)
2. Remove all requests r' that overlap with r
3. Repeat 1 and 2 until all requests have been chosen or removed

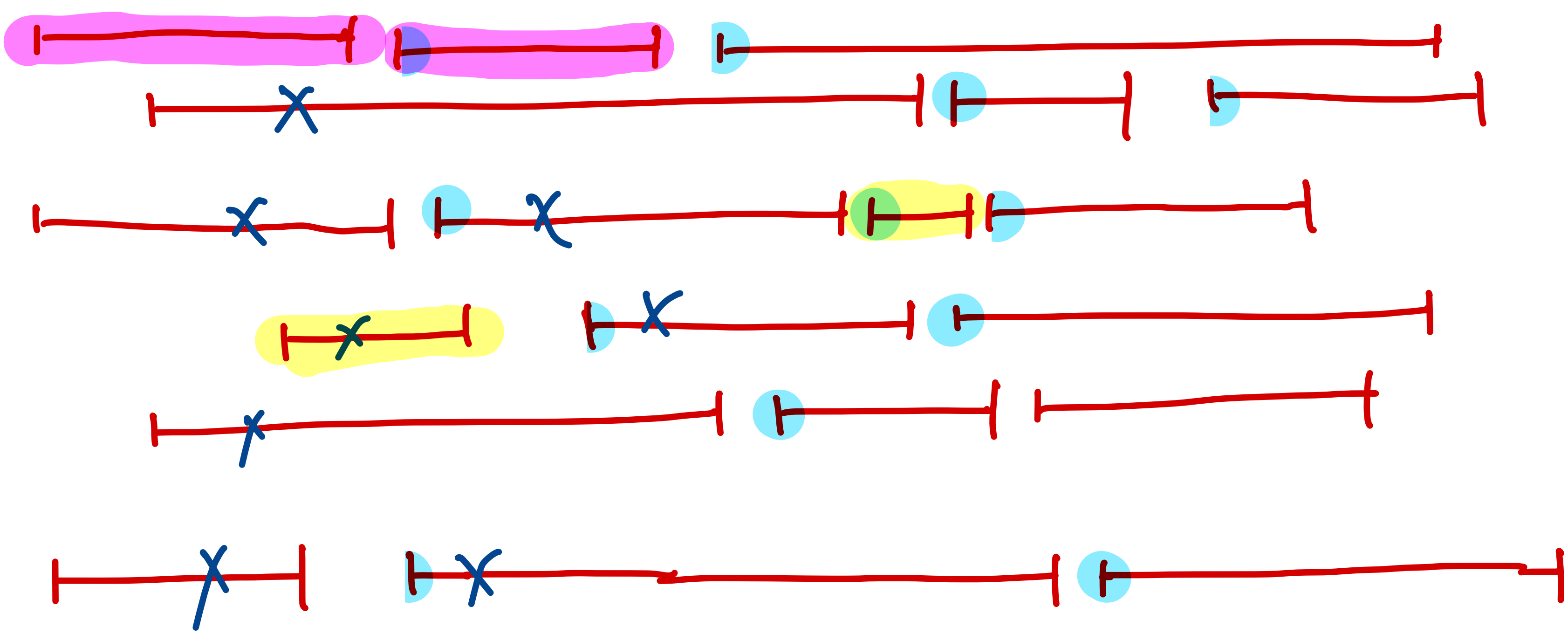
Observe. This will always give a feasible set of requests.

Question. How to select a request at each step?

Natural Greedy Selection Strategies

Select request...

1. ...with earliest start time
2. ...with shortest duration
3. ...that overlaps the fewest other requests



Natural Greedy Selection Strategies

Select request...

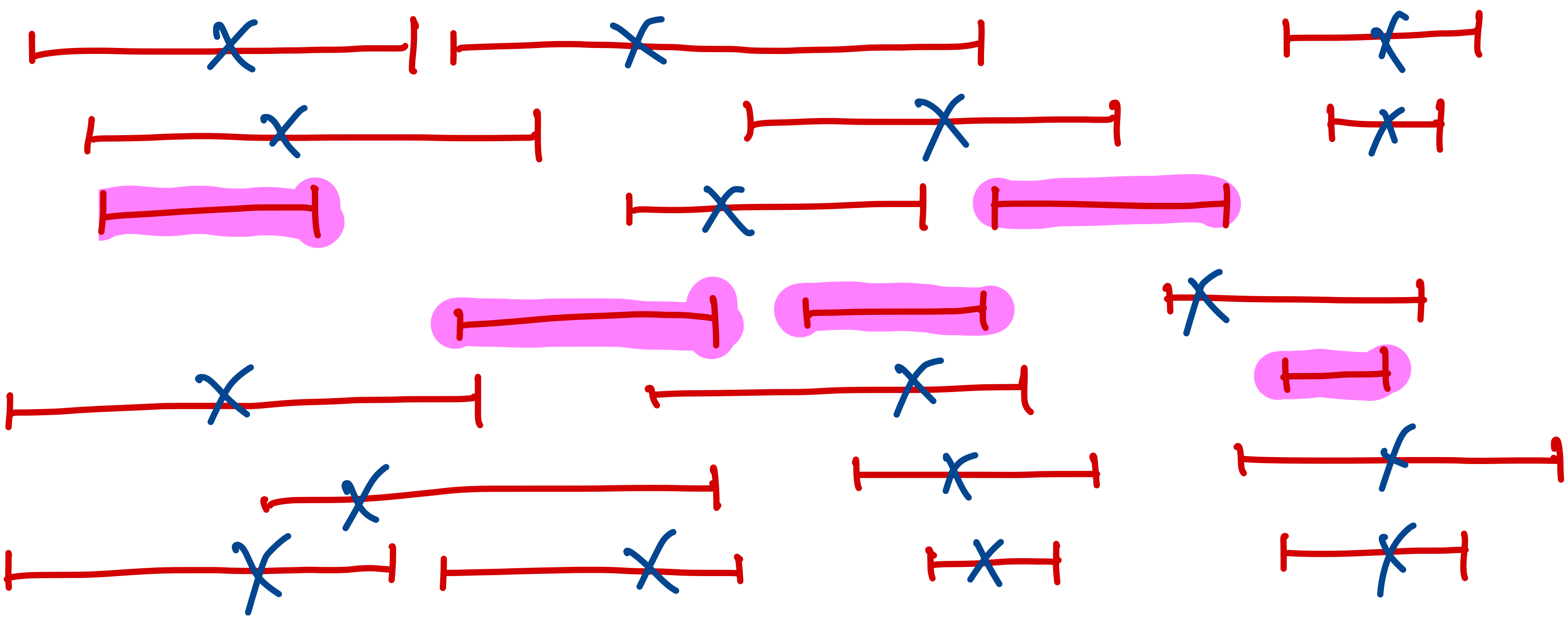
1. ...with earliest start time
2. ...with shortest duration
3. ...that overlaps the fewest other requests

Exercise. Show that none of these strategies are guaranteed to find a maximum feasible collection of requests.

Another Strategy

Earliest deadline first:

- select request with earliest deadline

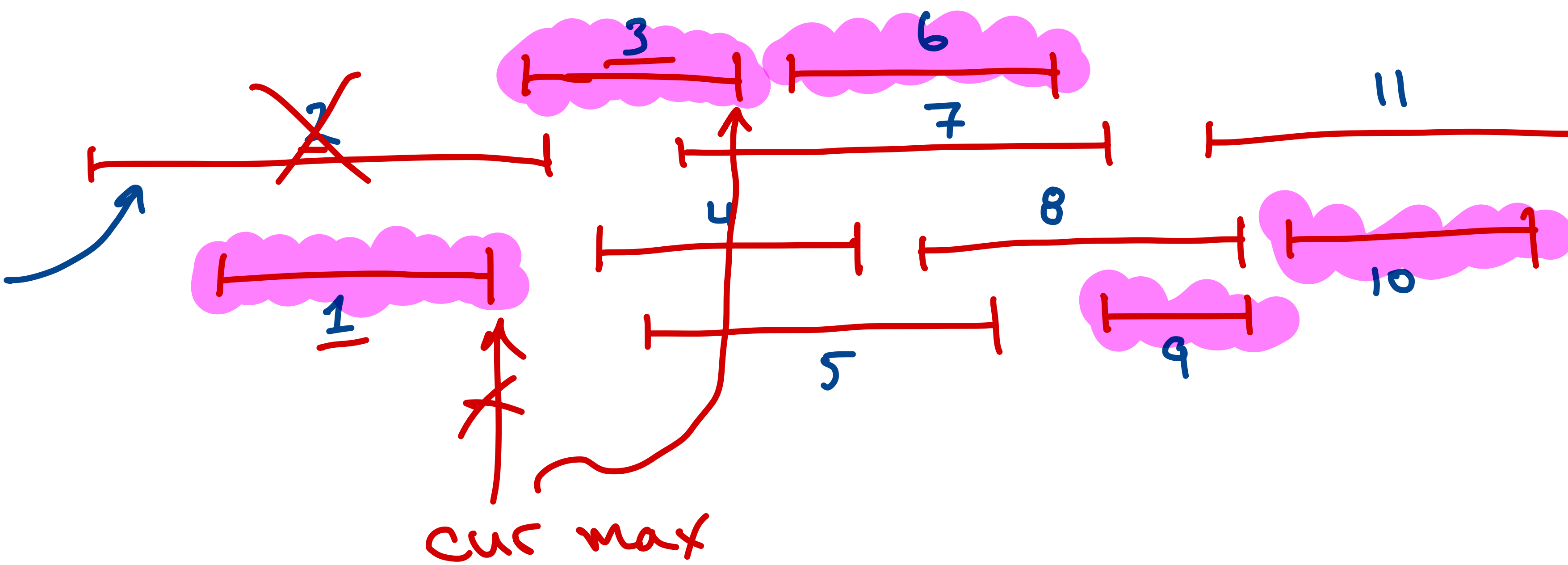


Find 5 compatible requests

EDF in Pseudocode

```
# R a collection of requests, r = (s, t)
EDF(R):
  sort R in ascending order of end time t
  curMax <- -infinity
  S <- empty collection
  for each request r = (s, t) in R do
    | if s > curMax then
    | | add r to S
    | | curMax <- t
  return S
```

right endpt of latest request added so far



Correctness I

Clear: EDF returns a feasible collection of requests

Claim. EDF returns a maximum feasible collection of requests.

Proof strategy. "EDF stays ahead"

- S is set of requests selected by EDF
- S_{opt} is optimal (maximum) feasible collection

To show:

- each request chosen in S is at least as good as corresponding request in S_{opt}
-

Correctness II

collection of req. found by EDF

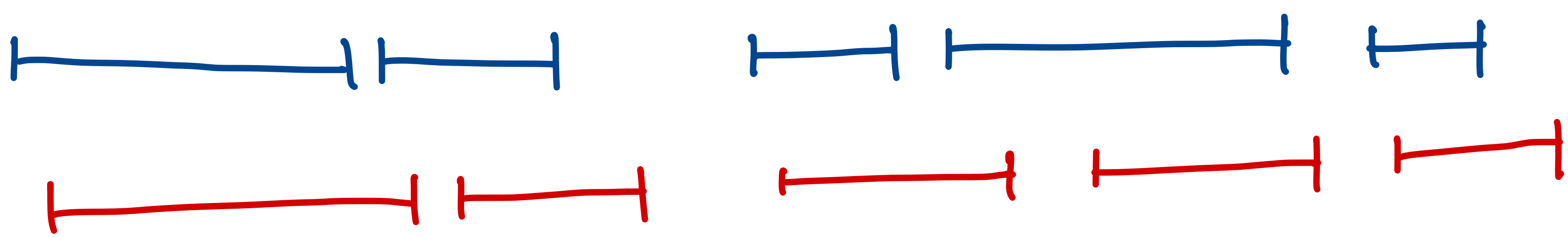
Notation:

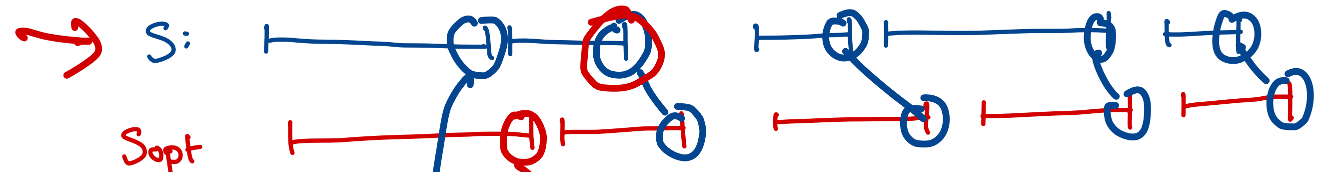
- $S = \{(s_1, t_1), (s_2, t_2), \dots, (s_k, t_k)\}$
  - $t_1 < s_2, t_2 < s_3, \dots$
  - $\boxed{k}$ is number of requests selected by EDF
- $S_{\text{opt}} = \{(s'_1, t'_1), (s'_2, t'_2), \dots, (s'_\ell, t'_\ell)\}$
  - $t'_1 < s'_2, t'_2 < s'_3, \dots$
  - $\boxed{\ell}$ is number of requests selected by S_{opt}

Want to show: $\boxed{k \geq \ell}$

* S:

* Sopt





Correctness III

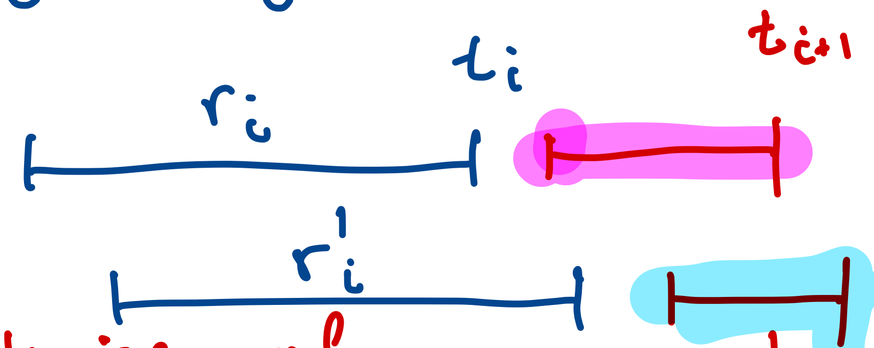
Sub-claim. For all $i = 1, 2, \ldots, k$, $t_i \le t'_i$.

Proof. Induction on i.

Base case $i = 1$. EDF chooses (s_1, t_1) w/ t_1 min among all intervals in input. ✓

$\Rightarrow t_1 \le t'_1$

<u>Inductive step</u>. Assume true for i i.e. $t_i \le t'_i$



min among all intervals after t_i (w/ $s > t_i$)

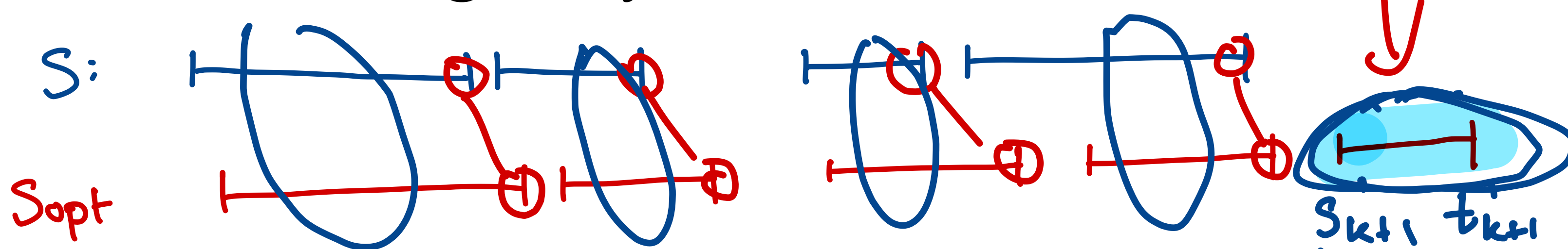
comes after $t'_i \ge t_i$ $\Rightarrow$ comes after t_i

By choice of t_{i+1}, have $t_{i+1} \le t'_{i+1}$

contained in opt, but not added by EDF

Correctness IV

Sub-claim $\implies$ Claim. Argue by contradiction.



Suppose not, i.e. S_{opt} contains strictly more intervals.

- k intervals added by EDF
- by sub-claim $t_k \le t'_k$
- by feasibility and ordering of intervals
  $s'_{k+1} > t'_k \ge t_k$
- this gives contradiction because r'_{k+1} was not added by EDF

n = total # of req.

Running Time?

```
# R a collection of requests, r = (s, t)
EDF(R):
    sort R in ascending order of end time t
    curMax <- -infinity
    S <- empty collection
    for each request r = (s, t) in R do
    | if s > curMax then
    | | add r to S
    | | curMax <- t
    return S
```

$O(n \log n)$ using, eg. merge sort

n iter

$O(1)$

$O(n \log n)$.

Conclusion

1. Earliest deadline first strategy finds a maximum feasible collection of requests.
2. "Algorithm stays ahead" analysis establishes correctness
3. Running time of EDF is $O(n \log n)$

Next Time

Start dynamic programming!