Advanced Algorithmic Techniques (COMP523)

Greedy Algorithms

Recap and plan

- **Last lecture:**
  - Directed Acyclic Graphs (DAGs)
  - Topological Ordering
  - Finding strongly connected components
- **This lecture:**
  - The Greedy approach
  - Interval Scheduling

The Greedy approach

- The goal is to come up with a global solution.
- The solution will be built up in small consecutive steps.
- For each step, the solution will be the best possible myopically, according to some criterion.

Interval Scheduling

Interval Scheduling

- A set of requests {*1*, *2*, … , *n*}.

Interval Scheduling

- A set of requests {*1*, *2*, ... , *n*}.
    - Each request has a starting time s(*i*) and a finishing time f(*i*).

Interval Scheduling

- A set of requests $\{1, 2, \ldots, n\}$.
  - Each request has a starting time $s(i)$ and a finishing time $f(i)$.
  - Alternative view: Every request is an interval $[s(i), f(i)]$.

Interval Scheduling

- A set of requests $\{1, 2, \ldots, n\}$.
  - Each request has a starting time $s(i)$ and a finishing time $f(i)$.
  - Alternative view: Every request is an interval $[s(i), f(i)]$.
- Two requests i and j are compatible if their respective intervals do not overlap.

Interval Scheduling

- A set of requests {*1*, *2*, … , *n*}.
  - Each request has a starting time s(*i*) and a finishing time f(*i*).
  - Alternative view: Every request is an interval [s(*i*), f(*i*)].
- Two requests *i* and *j* are compatible if their respective intervals do not overlap.
- Goal: Output a schedule which maximises the number of compatible intervals.

The Greedy Approach

The Greedy Approach

- We start by selecting an interval [s(*i*), f(*i*)] for some request *i*.

The Greedy Approach

- We start by selecting an interval [s(i), f(i)] for some request i.
- We include this interval in the schedule.

The Greedy Approach

- We start by selecting an interval [s(*i*), f(*i*)] for some request *i*.
- We include this interval in the schedule.
- This necessarily means that we can not include any other interval that is not compatible with [s(*i*), f(*i*)].

The Greedy Approach

- We start by selecting an interval [s(*i*), f(*i*)] for some request *i*.
- We include this interval in the schedule.
- This necessarily means that we can not include any other interval that is not compatible with [s(*i*), f(*i*)].
- We will continue with some compatible interval [s(*j*), f(*j*)] and repeat the same process.

The Greedy Approach

- We start by selecting an interval [s(i), f(i)] for some request i.
- We include this interval in the schedule.
- This necessarily means that we can not include any other interval that is not compatible with [s(i), f(i)].
- We will continue with some compatible interval [s(j), f(j)] and repeat the same process.
- We terminate when there are no more compatible intervals to consider.

Example

Example

Example

Example

Example

Example

Example

The Greedy Approach

The Greedy Approach

- We start by selecting an interval [s(i), f(i)] for some request i.

The Greedy Approach

- We start by selecting an interval [s(i), f(i)] for some request i.

- Let's try to make this more concrete.

The Greedy Approach

- We start by selecting an interval [s(*i*), f(*i*)] for some request *i*.
- Let's try to make this more concrete.
- Option 1: Choose the available interval that starts earliest.

Example

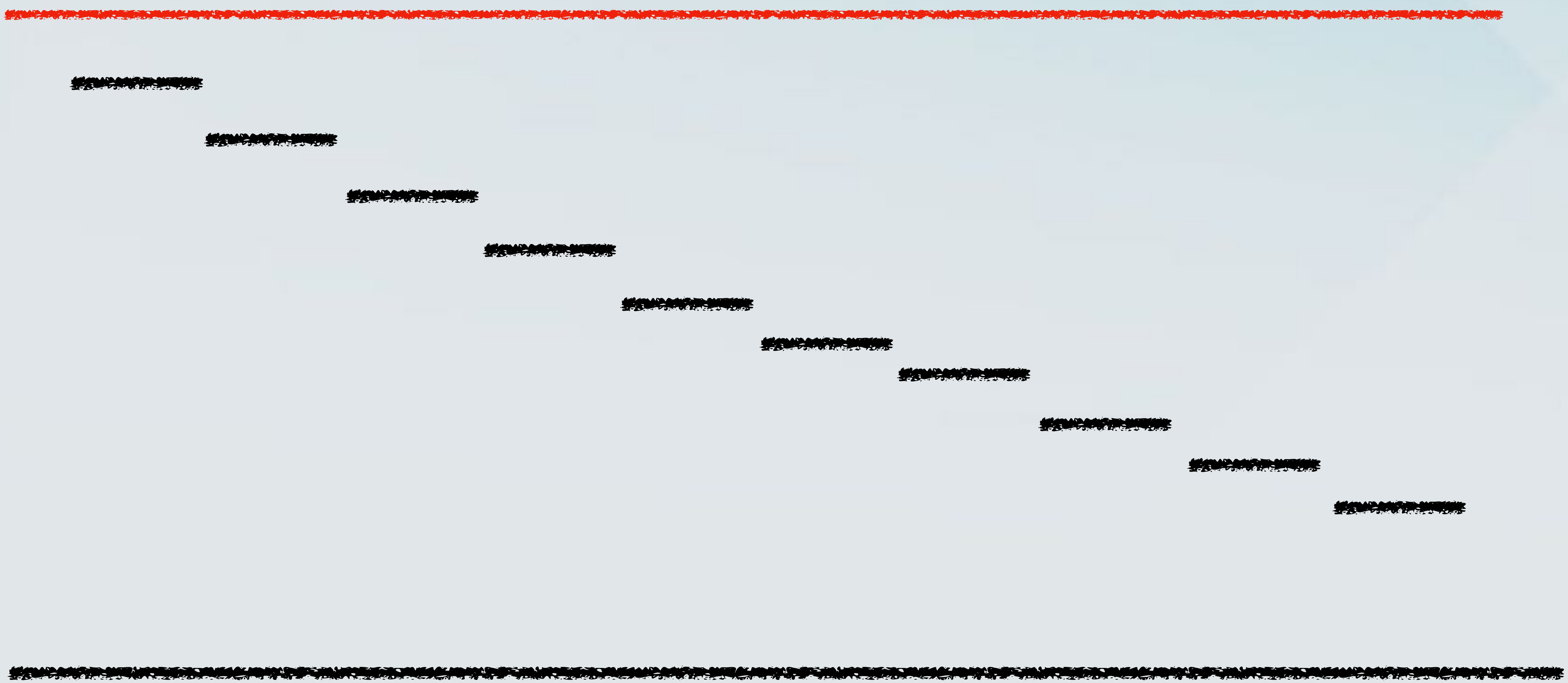
Example

Example

Is this the best we can do?

Is this always optimal?

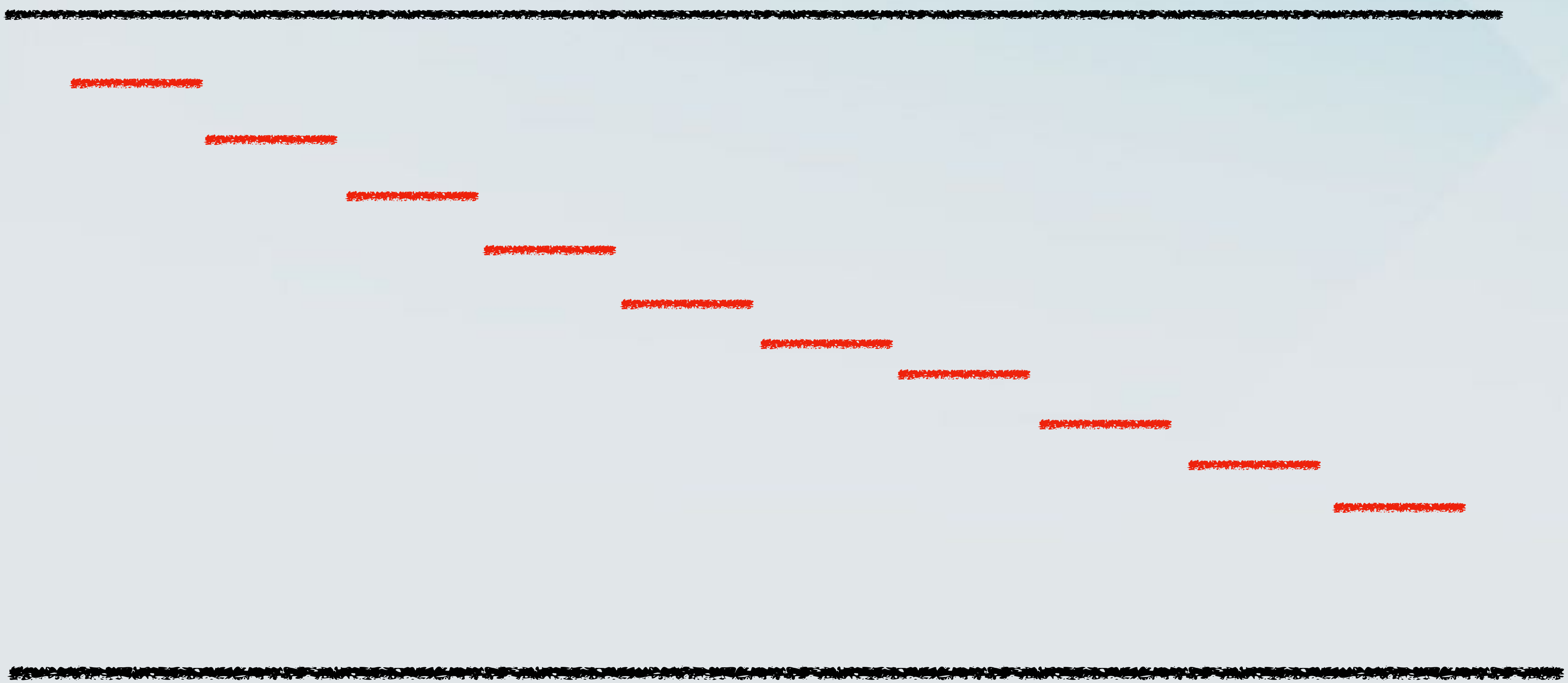
Is this always optimal?



The Greedy Approach

- We start by selecting an interval [s(*i*), f(*i*)] for some request *i*.
- Let's try to make this more concrete.
- Option 1: Choose the available interval that starts earliest.
- Option 2: Choose the smallest available interval.

Choosing the smallest interval



Is this always optimal?

Is this always optimal?

The Greedy Approach

- We start by selecting an interval [s(*i*), f(*i*)] for some request *i*.
- Let's try to make this more concrete.
- Option 1: Choose the available interval that starts earliest.
- Option 2: Choose the smallest available interval.
- Option 3: Something more clever.
    - Find the interval that minimises the number of "conflicts".

Minimum number of conflicts

Is this always optimal?

Is this always optimal?

Is this always optimal?

Is this always optimal?

Something even more clever

Something even more clever

- Select the interval [$s(i)$, $f(i)$] that finishes first (smallest $f(i)$).
- Intuition: The resource becomes free as soon as possible, but we still satisfy one request.

Greedy Algorithm for interval scheduling

IntervalScheduling($[s(i), f(i)]_{i=1 \text{ to } n}$)

Let R be the set of requests, let A be *empty*
While R is *not empty*
    Choose a request *i* with the smallest f(*i*).
    Add *i* to A
    Delete all requests from R that are not compatible with request *i*.

Return the set A of accepted requests

Correctness

Correctness

- Does the Greedy algorithm produce an optimal schedule?

Correctness

- Does the Greedy algorithm produce an optimal schedule?
- Does the Greedy algorithm produce a feasible (or acceptable) schedule?

Correctness

- Does the Greedy algorithm produce an optimal schedule?
- Does the Greedy algorithm produce a feasible (or acceptable) schedule?
    - Yes, since it removes in each step the intervals which are not compatible with what has been chosen.

Arguing for optimality

Arguing for optimality

- Some notation:

Arguing for optimality

- Some notation:
    - O is the optimal schedule. Recall, that A is the schedule of the Greedy algorithm.

Arguing for optimality

- Some notation:
  - O is the optimal schedule. Recall, that A is the schedule of the Greedy algorithm.
  - Let i_1, i_2, … , i_k be the order in which the intervals were added to A by the algorithm.

Arguing for optimality

- Some notation:
  - O is the optimal schedule. Recall, that A is the schedule of the Greedy algorithm.
  - Let $i_1, i_2, \ldots, i_k$ be the order in which the intervals were added to A by the algorithm.
    - Note that $|A| = k$.

Arguing for optimality

- Some notation:
  - O is the optimal schedule. Recall, that A is the schedule of the Greedy algorithm.
  - Let i_1, i_2, … , i_k be the order in which the intervals were added to A by the algorithm.
    - Note that $|A| = k$.
  - Let j_1, j_2, … , j_m be the set of requests in O.

Arguing for optimality

- Some notation:
  - O is the optimal schedule. Recall, that A is the schedule of the Greedy algorithm.
  - Let $i_1, i_2, \ldots, i_k$ be the order in which the intervals were added to A by the algorithm.
    - Note that $|A| = k$.
  - Let $j_1, j_2, \ldots, j_m$ be the set of requests in O.
    - Note that $|O| = m$.

Arguing for optimality

- Some notation:
  - O is the optimal schedule. Recall, that A is the schedule of the Greedy algorithm.
  - Let i_1, i_2, … , i_k be the order in which the intervals were added to A by the algorithm.
    - Note that $|A| = k$.
  - Let j_1, j_2, … , j_m be the set of requests in O.
    - Note that $|O| = m$.
  - We will prove that $m=k$. (*Why is that enough?*)

Arguing for optimality

Arguing for optimality

- Let j_1, j_2, … , j_m be the set of requests in O.

Arguing for optimality

- Let j_1, j_2, … , j_m be the set of requests in O.
  - Assume wlog that this is in order of increasing $s(j_h)$.

Arguing for optimality

- Let j_1, j_2, … , j_m be the set of requests in O.
  - Assume wlog that this is in order of increasing $s(j_h)$.
  - Since O is feasible, this is also in order of increasing $f(j_h)$.

Arguing for optimality

- Let $j_1, j_2, \ldots, j_m$ be the set of requests in O.
  - Assume wlog that this is in order of increasing $s(j_h)$.
  - Since O is feasible, this is also in order of increasing $f(j_h)$.
- Claim: $f(i_1) \leq f(j_1)$

Arguing for optimality

- Let j_1, j_2, … , j_m be the set of requests in O.
  - Assume wlog that this is in order of increasing $s(j_h)$.
  - Since O is feasible, this is also in order of increasing $f(j_h)$.
- Claim: $f(i_1) \leq f(j_1)$
  - Because i_1 is chosen to be the interval with the smallest $f(i_h)$.

Arguing for optimality

- Claim: $f(i_1) \leq f(j_1)$
  - Because i_1 is chosen to be the interval with the smallest $f(i_h)$.

Arguing for optimality

- Claim: $f(i_1) \leq f(j_1)$
  - Because i_1 is chosen to be the interval with the smallest $f(i_h)$.
- Lemma: For all indices $r \leq k$, it holds that $f(i_r) \leq f(j_r)$

Arguing for optimality

- Claim: $f(i_1) \leq f(j_1)$
  - Because i_1 is chosen to be the interval with the smallest $f(i_h)$.
- Lemma: For all indices $r \leq k$, it holds that $f(i_r) \leq f(j_r)$
  - Proof by induction:

Arguing for optimality

- Claim: $f(i_1) \leq f(j_1)$
    - Because i_1 is chosen to be the interval with the smallest $f(i_h)$.
- Lemma: For all indices $r \leq k$, it holds that $f(i_r) \leq f(j_r)$
    - Proof by induction:
        - Base Case ($r=1$), by Claim.

Arguing for optimality

- Claim: $f(i_1) \leq f(j_1)$
  - Because i_1 is chosen to be the interval with the smallest $f(i_h)$.
- Lemma: For all indices $r \leq k$, it holds that $f(i_r) \leq f(j_r)$
  - Proof by induction:
    - Base Case ($r=1$), by Claim.
    - Induction Step. Assume it is true for $r-1$ (IH), we will prove it for r.

Induction step proof

Induction step proof

- We know that $f(j_{r-1}) \leq s(j_r)$ (why?)

Induction step proof

- We know that $f(j_{r-1}) \leq s(j_r)$ (why?)
  - Because the intervals of O are compatible.

Induction step proof

- We know that $f(j_{r-1}) \leq s(j_r)$ (why?)
  - Because the intervals of O are compatible.
- We know that $f(i_{r-1}) \leq f(j_{r-1})$ (why?)

Induction step proof

- We know that $f(j_{r-1}) \leq s(j_r)$ (why?)
  - Because the intervals of O are compatible.
- We know that $f(i_{r-1}) \leq f(j_{r-1})$ (why?)
  - By the Induction Hypothesis.

Induction step proof

- We know that $f(j_{r-1}) \leq s(j_r)$ (why?)
  - Because the intervals of O are compatible.
- We know that $f(i_{r-1}) \leq f(j_{r-1})$ (why?)
  - By the Induction Hypothesis.
- What does that mean for the interval $j_r = (s(j_r), f(j_r))$?

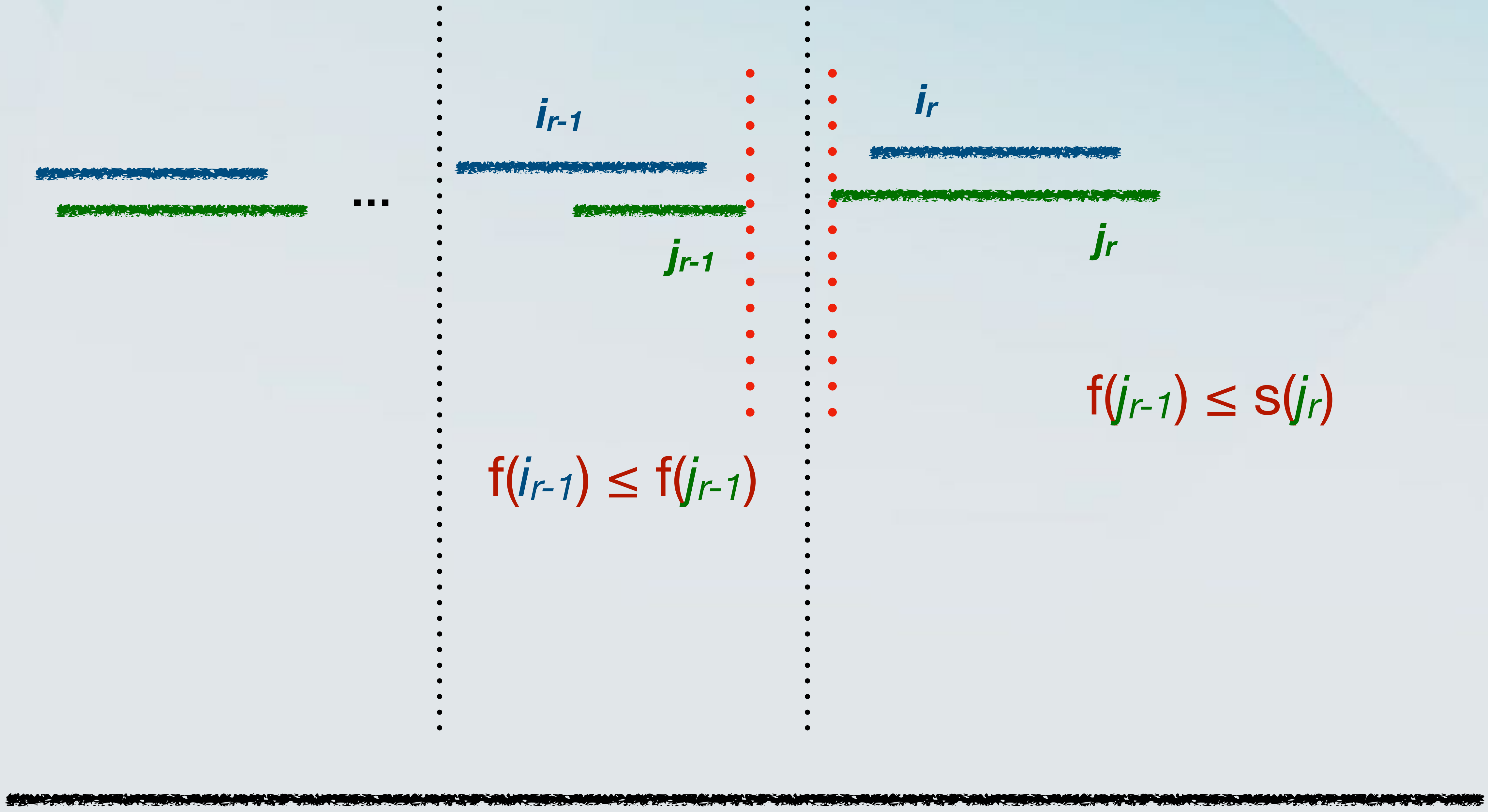
Induction step proof

- We know that $f(j_{r-1}) \leq s(j_r)$ (why?)
  - Because the intervals of O are compatible.
- We know that $f(i_{r-1}) \leq f(j_{r-1})$ (why?)
  - By the Induction Hypothesis.
- What does that mean for the interval $j_r = (s(j_r), f(j_r))$?
  - When the Greedy algorithm selected i_r , j_r was in the set R of available intervals.

Induction step proof

- We know that $f(j_{r-1}) \leq s(j_r)$ (why?)
  - Because the intervals of O are compatible.
- We know that $f(i_{r-1}) \leq f(j_{r-1})$ (why?)
  - By the Induction Hypothesis.
- What does that mean for the interval $j_r = (s(j_r), f(j_r))$?
  - When the Greedy algorithm selected i_r , j_r was in the set R of available intervals.
- This means that $f(i_r) \leq f(j_r)$, as otherwise the algorithm would have selected j_r instead.

With a picture



Completing the proof

Completing the proof

- By contradiction: To the contrary, assume that $m > k$
- For $r=k$, the Lemma gives us that $f(i_k) \leq f(j_k)$.
- Since $m > k$, there is an extra request j_{k+1} in O.
- $s(j_{k+1}) > f(j_k) \geq f(i_k)$.
- The greedy algorithm would have continued with j_{k+1}.

Running Time

Running Time

- Sort intervals in terms of increasing $f(i)$.

Running Time

- Sort intervals in terms of increasing $f(i)$.
- We select the first interval in the ordering.

Running Time

- Sort intervals in terms of increasing f(i).
- We select the first interval in the ordering.
- For any consecutive interval j in the ordering, we check if f(i) ≤ s(j).

Running Time

- Sort intervals in terms of increasing f(i).
- We select the first interval in the ordering.
- For any consecutive interval j in the ordering, we check if f(i) ≤ s(j).
    - If yes, we select it and continue with the same checks for this new interval.

Running Time

- Sort intervals in terms of increasing f(*i*).
- We select the first interval in the ordering.
- For any consecutive interval *j* in the ordering, we check if f(*i*) ≤ s(*j*).
    - If yes, we select it and continue with the same checks for this new interval.
    - If not, we move on to the next interval.

Running Time

- Sort intervals in terms of increasing f(*i*).
- We select the first interval in the ordering.
- For any consecutive interval *j* in the ordering, we check if f(*i*) ≤ s(*j*).
  - If yes, we select it and continue with the same checks for this new interval.
  - If not, we move on to the next interval.
- The running time is O(*n log n*).