Greedy Algorithms

T. M. Murali

February 10, 15, 2021
Algorithm Design

- Start discussion of different ways of designing algorithms.
- Greedy algorithms, divide and conquer, dynamic programming.
- Discuss principles that can solve a variety of problem types.
- Design an algorithm, prove its correctness, analyse its complexity.
Algorithm Design

- Start discussion of different ways of designing algorithms.
- Greedy algorithms, divide and conquer, dynamic programming.
- Discuss principles that can solve a variety of problem types.
- Design an algorithm, prove its correctness, analyse its complexity.
- Greedy algorithms: make the current best choice.
Interval Scheduling

Interval Partitioning

Minimising Lateness

Input: Start and end time of each ride.
Constraint: Cannot be in two places at one time.
Goal: Compute the largest number of rides you can be on in one day.

T. M. Murali February 10, 15, 2021 CS 4104: Greed is Good
Input: Start and end time of each ride.
Constraint: Cannot be in two places at one time.
Goal: Compute the largest number of rides you can be on in one day.
Interval Scheduling

**INSTANCE:** Set \( \{(s(i), f(i)), 1 \leq i \leq n\} \) of start and finish times of \( n \) jobs.

**SOLUTION:** The largest subset of mutually compatible jobs.

- Two jobs are *compatible* if they do not overlap.
- This problem models the situation where you have a resource, a set of fixed jobs, and you want to schedule as many jobs as possible.
- For any input set of jobs, algorithm must provably compute the largest set of compatible jobs.
Interval Scheduling Example

(a) (b)
(c) (d)

Solutions (c) and (d) are optimal.

▶ Each contains four jobs.
▶ No set of mutually compatible jobs can contain more than four jobs.
Interval Scheduling Example

(a) Not compatible

(b) Mutually compatible

(c) Mutually compatible

(d) Mutually compatible

Solutions (c) and (d) are optimal.

- Each contains four jobs.
- No set of mutually compatible jobs can contain more than four jobs.
Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what order should we process the jobs?
Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what order should we process the jobs?
  
  **Earliest start time** Increasing order of start time $s(i)$.
  **Earliest finish time** Increasing order of finish time $f(i)$.
  **Shortest interval** Increasing order of length $f(i) - s(i)$.
  **Fewest conflicts** Increasing order of the number of conflicting jobs.
Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what order should we process the jobs? ▶ Poll
  - Earliest start time Increasing order of start time \( s(i) \).
  - Earliest finish time Increasing order of finish time \( f(i) \).
  - Shortest interval Increasing order of length \( f(i) - s(i) \).
  - Fewest conflicts Increasing order of the number of conflicting jobs.

(a) (b)
(c) (d)

Time

Time

Time

Time
Template for Greedy Algorithm

- Process jobs in some order. Add next job to the result if it is compatible with the jobs already in the result.
- Key question: in what order should we process the jobs?
  - **Earliest start time** Increasing order of start time $s(i)$.
  - **Earliest finish time** Increasing order of finish time $f(i)$.
  - **Shortest interval** Increasing order of length $f(i) - s(i)$.
  - **Fewest conflicts** Increasing order of the number of conflicting jobs.

(a) Shortest interval
(b) Earliest start time
(c) Minimum conflicts
(d) Shortest Interval, Min. conflicts
Interval Scheduling Algorithm: Earliest Finish Time

- Schedule jobs in order of earliest finish time (EFT).

Initially let $R$ be the set of all requests, and let $A$ be empty
While $R$ is not yet empty
  - Choose a request $i \in R$ that has the smallest finishing time
  - Add request $i$ to $A$
  - Delete all requests from $R$ that are not compatible with request $i$
EndWhile
Return the set $A$ as the set of accepted requests
Interval Scheduling Algorithm: Earliest Finish Time

- Schedule jobs in order of earliest finish time (EFT).

Initially let $R$ be the set of all requests, and let $A$ be empty

While $R$ is not yet empty

- Choose a request $i \in R$ that has the smallest finishing time
- Add request $i$ to $A$
- Delete all requests from $R$ that are not compatible with request $i$

EndWhile

Return the set $A$ as the set of accepted requests

- Claim: $A$ is a compatible set of jobs.
Interval Scheduling Algorithm: Earliest Finish Time

- Schedule jobs in order of earliest finish time (EFT).

Initially let $R$ be the set of all requests, and let $A$ be empty

While $R$ is not yet empty

- Choose a request $i \in R$ that has the smallest finishing time
- Add request $i$ to $A$
- Delete all requests from $R$ that are not compatible with request $i$

EndWhile

Return the set $A$ as the set of accepted requests

Claim: $A$ is a compatible set of jobs. Proof follows by construction, i.e., the algorithm computes a compatible set of jobs.
Ideas for Analysing the EFT Algorithm

- We need to prove that $|A|$ (the number of jobs in $A$) is the largest possible in any set of mutually compatible jobs.
Ideas for Analysing the EFT Algorithm

- We need to prove that $|A|$ (the number of jobs in $A$) is the largest possible in any set of mutually compatible jobs.
- Proof idea 1: algorithm makes the best choice at each step, so it must choose the largest number of mutually compatible jobs.
Ideas for Analysing the EFT Algorithm

- We need to prove that $|A|$ (the number of jobs in $A$) is the largest possible in any set of mutually compatible jobs.

- Proof idea 1: algorithm makes the best choice at each step, so it must choose the largest number of mutually compatible jobs.
  - What does “best” mean?
  - This idea is too generic. It can be applied even to algorithms that we know do not work correctly.
Ideas for Analysing the EFT Algorithm

- We need to prove that $|A|$ (the number of jobs in $A$) is the largest possible in \textit{any} set of mutually compatible jobs.
- Proof idea 1: algorithm makes the best choice at each step, so it must choose the largest number of mutually compatible jobs.
  - What does “best” mean?
  - This idea is too generic. It can be applied even to algorithms that we know do not work correctly.
- Proof idea 2: at each step, can we show algorithm has the “better” solution than any other answer?
  - What does “better” mean?
  - How do we measure progress of the algorithm?
Ideas for Analysing the EFT Algorithm

- We need to prove that \(|A|\) (the number of jobs in \(A\)) is the largest possible in *any* set of mutually compatible jobs.

- Proof idea 1: algorithm makes the best choice at each step, so it must choose the largest number of mutually compatible jobs.
  - What does “best” mean?
  - This idea is too generic. It can be applied even to algorithms that we know do not work correctly.

- Proof idea 2: at each step, can we show algorithm has the “better” solution than any other answer?
  - What does “better” mean?
  - How do we measure progress of the algorithm?

- Basic idea of proof:
  - We can sort jobs in any solution in increasing order of their finishing time.
  - Finishing time of job number \(r\) selected by \(A\) ≤ finishing time of job number \(r\) selected by any other algorithm.
Let $O$ be an optimal set of jobs. We will show that $|A| = |O|$.

Let $i_1, i_2, \ldots, i_k$ be the set of jobs in $A$ in order.

Let $j_1, j_2, \ldots, j_m$ be the set of jobs in $O$ in order, $m \geq k$.

Claim: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$. 

Prove by induction on $r$. 

Claim: $m = k$. 

Claim: The greedy algorithm returns an optimal set $A$. 

T. M. Murali February 10, 15, 2021 CS 4104: Greed is Good
Analysing the EFT Algorithm

- Let $O$ be an optimal set of jobs. We will show that $|A| = |O|$.
- Let $i_1, i_2, \ldots, i_k$ be the set of jobs in $A$ in order.
- Let $j_1, j_2, \ldots, j_m$ be the set of jobs in $O$ in order, $m \geq k$.
- Claim: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$. Prove by induction on $r$. 
Analysing the EFT Algorithm

- Let $O$ be an optimal set of jobs. We will show that $|A| = |O|$.
- Let $i_1, i_2, \ldots, i_k$ be the set of jobs in $A$ in order.
- Let $j_1, j_2, \ldots, j_m$ be the set of jobs in $O$ in order, $m \geq k$.
- Claim: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$. Prove by induction on $r$.

![Diagram](image.png)

*Figure 4.3* The inductive step in the proof that the greedy algorithm stays ahead.
Analysing the EFT Algorithm

- Let $O$ be an optimal set of jobs. We will show that $|A| = |O|$.
- Let $i_1, i_2, \ldots, i_k$ be the set of jobs in $A$ in order.
- Let $j_1, j_2, \ldots, j_m$ be the set of jobs in $O$ in order, $m \geq k$.
- Claim: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$. Prove by induction on $r$.

Can the greedy algorithm’s $r^{th}$ interval really finish later?

\[ i_{r-1} \quad j_{r-1} \quad i_r \quad ? \quad j_r \]

**Figure 4.3** The inductive step in the proof that the greedy algorithm stays ahead.

- Claim: $m = k$. 
Analysing the EFT Algorithm

- Let $O$ be an optimal set of jobs. We will show that $|A| = |O|$.
- Let $i_1, i_2, \ldots, i_k$ be the set of jobs in $A$ in order.
- Let $j_1, j_2, \ldots, j_m$ be the set of jobs in $O$ in order, $m \geq k$.
- Claim: For all indices $r \leq k$, $f(i_r) \leq f(j_r)$. Prove by induction on $r$.

Can the greedy algorithm’s $r^{th}$ interval really finish later?

![Diagram](image)

Figure 4.3 The inductive step in the proof that the greedy algorithm stays ahead.

- Claim: $m = k$.
- Claim: The greedy algorithm returns an optimal set $A$. 

---

T. M. Murali February 10, 15, 2021 CS 4104: Greed is Good
Implementing the EFT Algorithm

1. Reorder jobs so that they are in increasing order of finish time.
2. Store starting time of jobs in an array $S$.
3. $k = 1$.
4. While $k \leq |S|$, 
   1. Output job $k$.
   2. Let finish time of job $k$ be $f$.
   3. Iterate over $S$ from index $k$ onwards to find the first index $i$ such that $S[i] \geq f$.
   4. $k = i$
Implementing the EFT Algorithm

1. Reorder jobs so that they are in increasing order of finish time.
2. Store starting time of jobs in an array $S$.
3. $k = 1$.
4. While $k \leq |S|$, 
   1. Output job $k$.
   2. Let finish time of job $k$ be $f$.
   3. Iterate over $S$ from index $k$ onwards to find the first index $i$ such that $S[i] \geq f$.
   4. $k = i$
   • Must be careful to iterate over $S$ such that we never scan same index more than once.
   • Running time is $O(n \log n)$, dominated by sorting.
<table>
<thead>
<tr>
<th>CRN</th>
<th>Course Code</th>
<th>Course Title</th>
<th>Credits</th>
<th>Room</th>
<th>Instructor</th>
<th>Days</th>
<th>Time</th>
<th>Location</th>
<th>Room Code</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>12616</td>
<td>CS-4104</td>
<td>Data and Algorithm Analysis</td>
<td>3</td>
<td></td>
<td>CA Shaffer</td>
<td>T R</td>
<td>2:00PM</td>
<td>SURGE 107</td>
<td>14T</td>
<td></td>
</tr>
<tr>
<td>18154</td>
<td>CS-4104</td>
<td>Data and Algorithm Analysis</td>
<td>3</td>
<td></td>
<td>TM Murali</td>
<td>M W</td>
<td>2:30PM</td>
<td>SURGE 104C</td>
<td>14M</td>
<td></td>
</tr>
<tr>
<td>12617</td>
<td>CS-4114</td>
<td>Formal Languages</td>
<td>3</td>
<td></td>
<td>L Zhang</td>
<td>T R</td>
<td>9:30AM</td>
<td>MCB 129</td>
<td>09T</td>
<td></td>
</tr>
<tr>
<td>18155</td>
<td>CS-4204</td>
<td>Computer Graphics</td>
<td>3</td>
<td></td>
<td>D Gracanin</td>
<td>T R</td>
<td>11:00AM</td>
<td>MCB 224</td>
<td>11T</td>
<td></td>
</tr>
<tr>
<td>19593</td>
<td>CS-4264</td>
<td>Principles Computer Security</td>
<td>3</td>
<td></td>
<td>KE Giles</td>
<td>M W</td>
<td>2:30PM</td>
<td>GOODW 135</td>
<td>14M</td>
<td></td>
</tr>
<tr>
<td>12618</td>
<td>CS-4284</td>
<td>Systems &amp; Networking Capstone</td>
<td>3</td>
<td></td>
<td>GV Back</td>
<td>M W</td>
<td>2:30PM</td>
<td>MCB 238</td>
<td>14M</td>
<td></td>
</tr>
<tr>
<td>18156</td>
<td>CS-4304</td>
<td>Compiler Design</td>
<td>3</td>
<td></td>
<td>C Jung</td>
<td>T R</td>
<td>8:00AM</td>
<td>GOODW 125</td>
<td>08T</td>
<td></td>
</tr>
<tr>
<td>12620</td>
<td>CS-4604</td>
<td>Int Data Base Mgt Sys</td>
<td>3</td>
<td></td>
<td>RJ Quintin</td>
<td>M W</td>
<td>4:00PM</td>
<td>SURGE 109</td>
<td>16M</td>
<td></td>
</tr>
<tr>
<td>12621</td>
<td>CS-4624</td>
<td>Multimedia/Hypertext</td>
<td>3</td>
<td></td>
<td>EA Fox</td>
<td>T R</td>
<td>3:30PM</td>
<td>SURGE 109</td>
<td>15T</td>
<td></td>
</tr>
<tr>
<td>12622</td>
<td>CS-4644</td>
<td>Creative Computing Studio</td>
<td>3</td>
<td></td>
<td>SR Harrison</td>
<td>W</td>
<td>2:30PM</td>
<td>MAC 253A</td>
<td>14W</td>
<td></td>
</tr>
</tbody>
</table>

**Comments for CRN 12622:** Prerequisite: C or better in CS 3724 OR CS 3744

<table>
<thead>
<tr>
<th>CRN</th>
<th>Course Code</th>
<th>Course Title</th>
<th>Credits</th>
<th>Room</th>
<th>Instructor</th>
<th>Days</th>
<th>Time</th>
<th>Location</th>
<th>Room Code</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>12623</td>
<td>CS-4654</td>
<td>Intermed Data Analytics &amp; ML</td>
<td>3</td>
<td></td>
<td>RB Gramacy</td>
<td>M W</td>
<td>4:00PM</td>
<td>SEITZ 313</td>
<td>16M</td>
<td></td>
</tr>
<tr>
<td>12624</td>
<td>CS-4704</td>
<td>Software Engineering Capstone</td>
<td>3</td>
<td></td>
<td>KR Edmison</td>
<td>M W</td>
<td>4:00PM</td>
<td>NCB 170</td>
<td>16M</td>
<td></td>
</tr>
</tbody>
</table>

**Comments for CRN 12624:** Prerequisite: C or better in CS 3704 OR CS 3714

<table>
<thead>
<tr>
<th>CRN</th>
<th>Course Code</th>
<th>Course Title</th>
<th>Credits</th>
<th>Room</th>
<th>Instructor</th>
<th>Days</th>
<th>Time</th>
<th>Location</th>
<th>Room Code</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>12625</td>
<td>CS-4784</td>
<td>Human-Computer Interact Capstone</td>
<td>3</td>
<td></td>
<td>AL Kavanaugh</td>
<td>F</td>
<td>1:00PM</td>
<td>MAC 253A</td>
<td>13F</td>
<td></td>
</tr>
</tbody>
</table>

**Comments for CRN 12625:** Prerequisite: CS 3724 required; CS 3714 or 3744 recommended

<table>
<thead>
<tr>
<th>CRN</th>
<th>Course Code</th>
<th>Course Title</th>
<th>Credits</th>
<th>Room</th>
<th>Instructor</th>
<th>Days</th>
<th>Time</th>
<th>Location</th>
<th>Room Code</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>19924</td>
<td>CS-4784</td>
<td>Human-Computer Interact Capstn</td>
<td>3</td>
<td></td>
<td>DS McCrickard</td>
<td>F</td>
<td>12:30PM</td>
<td>MCB 655</td>
<td>12F</td>
<td></td>
</tr>
</tbody>
</table>
Input: Start and end time of each class.
Constraint: Cannot schedule two overlapping classes to the same room.
Output: Assign each class to a room and use smallest number of rooms possible.
Interval Partitioning

**Interval Partitioning**

**INSTANCE:** Set \( \{(s(i), f(i)), 1 \leq i \leq n\} \) of start and finish times of \( n \) jobs.

**SOLUTION:** A partition of the jobs into \( k \) sets, where each set of jobs is mutually compatible, and \( k \) is minimised.

- This problem models the situation where you a set of fixed jobs, and you want to schedule all jobs using as few resources as possible.
The depth of a set of intervals is the maximum number of intervals that contain any time point.

Figure 4.4 (a) An instance of the Interval Partitioning Problem with ten intervals (a through j). (b) A solution in which all intervals are scheduled using three resources: each row represents a set of intervals that can all be scheduled on a single resource.
The depth of a set of intervals is the maximum number of intervals that contain any time point.

Claim: In any instance of INTERVAL PARTITIONING, $k \geq \text{depth}$.
The depth of a set of intervals is the maximum number of intervals that contain any time point.

Claim: In any instance of INTERVAL PARTITIONING, $k \geq \text{depth}$.

Is it possible to compute the depth efficiently? Is $k = \text{depth}$?
Computing the Depth of the Intervals

How efficiently can we compute the depth of a set of intervals?

Algorithm:
1. Sort the start times and finish times of the jobs into a single list $L$.
2. $d \leftarrow 0$.
3. For $i$ ranging from 1 to $n$:
   - If $L_i$ is a start time, increment $d$ by 1.
   - If $L_i$ is a finish time, decrement $d$ by 1.
4. Return the largest value of $d$ computed in the loop.

The algorithm runs in $O(n \log n)$ time.
Computing the Depth of the Intervals

How efficiently can we compute the depth of a set of intervals?

1. Sort the start times and finish times of the jobs into a single list $L$.
2. $d \leftarrow 0$.
3. For $i$ ranging from 1 to $2n$
   1. If $L_i$ is a start time, increment $d$ by 1.
   2. If $L_i$ is a finish time, decrement $d$ by 1.
4. Return the largest value of $d$ computed in the loop.
Computing the Depth of the Intervals

• How efficiently can we compute the depth of a set of intervals?

1. Sort the start times and finish times of the jobs into a single list $L$.
2. $d \leftarrow 0$.
3. For $i$ ranging from 1 to $2n$
   1. If $L_i$ is a start time, increment $d$ by 1.
   2. If $L_i$ is a finish time, decrement $d$ by 1.
4. Return the largest value of $d$ computed in the loop.

• Algorithm runs in $O(n \log n)$ time.
Interval Partitioning Algorithm

- First, compute the depth $d$ of the intervals.
Interval Partitioning Algorithm

- First, compute the depth $d$ of the intervals.

Sort the intervals by their start times, breaking ties arbitrarily
Let $I_1, I_2, \ldots, I_n$ denote the intervals in this order
For $j = 1, 2, 3, \ldots, n$
  - For each interval $I_i$ that precedes $I_j$ in sorted order and overlaps it
    - Exclude the label of $I_i$ from consideration for $I_j$
  Endfor
If there is any label from $\{1, 2, \ldots, d\}$ that has not been excluded then
  - Assign a nonexcluded label to $I_j$
Else
  - Leave $I_j$ unlabeled
Endif
Endfor

- Claim: Every interval gets a label and no pair of overlapping intervals get the same label.
Interval Partitioning Algorithm

- First, compute the depth $d$ of the intervals.

Sort the intervals by their start times, breaking ties arbitrarily
Let $I_1, I_2, \ldots, I_n$ denote the intervals in this order
For $j = 1, 2, 3, \ldots, n$
  - For each interval $I_i$ that precedes $I_j$ in sorted order and overlaps it
    - Exclude the label of $I_i$ from consideration for $I_j$
  Endfor
If there is any label from $\{1, 2, \ldots, d\}$ that has not been excluded then
  - Assign a nonexcluded label to $I_j$
Else
  - Leave $I_j$ unlabeled
Endif
Endfor

- Claim: Every interval gets a label and no pair of overlapping intervals get the same label.
- Claim: The greedy algorithm is optimal.
Interval Partitioning Algorithm

- First, compute the depth $d$ of the intervals.

Sort the intervals by their start times, breaking ties arbitrarily.
Let $I_1, I_2, ..., I_n$ denote the intervals in this order.
For $j = 1, 2, 3, ..., n$
  For each interval $I_i$ that precedes $I_j$ in sorted order and overlaps it
    Exclude the label of $I_i$ from consideration for $I_j$
  Endfor
If there is any label from $\{1, 2, ..., d\}$ that has not been excluded then
  Assign a nonexcluded label to $I_j$
Else
  Leave $I_j$ unlabeled
Endif
Endfor

- Claim: Every interval gets a label and no pair of overlapping intervals get the same label.
- Claim: The greedy algorithm is optimal.
- The running time of the algorithm is $O(n \log n)$. 
Interval Partitioning Algorithm

- First, compute the depth $d$ of the intervals.

Sort the intervals by their start times, breaking ties arbitrarily
Let $I_1, I_2, \ldots, I_n$ denote the intervals in this order
For $j = 1, 2, 3, \ldots, n$
  - For each interval $I_i$ that precedes $I_j$ in sorted order and overlaps it
    - Exclude the label of $I_i$ from consideration for $I_j$
  Endfor
If there is any label from $\{1, 2, \ldots, d\}$ that has not been excluded then
  - Assign a nonexcluded label to $I_j$
Else
  - Leave $I_j$ unlabeled
Endif
Endfor

- Claim: Every interval gets a label and no pair of overlapping intervals get the same label.
- Claim: The greedy algorithm is optimal.
- The running time of the algorithm is $O(n \log n)$. Can modify algorithm for computing depth to maintain set of available labels and to assign them efficiently.
Scheduling to Minimise Lateness

- Study different model: job $i$ has a length $t(i)$ and a deadline $d(i)$.
- We want to schedule all $n$ jobs on one resource.
- Our goal is to assign a starting time $s(i)$ to each job such that each job is delayed as little as possible.
Scheduling to Minimise Lateness

- Study different model: job $i$ has a length $t(i)$ and a deadline $d(i)$.
- We want to schedule all $n$ jobs on one resource.
- Our goal is to assign a starting time $s(i)$ to each job such that each job is delayed as little as possible.
- A job $i$ is *delayed* if $f(i) > d(i)$; the *lateness of the job* is
  \[ \max(0, f(i) - d(i)) \]

- The *lateness of a schedule* is
  \[ \max_{1 \leq i \leq n} \left( \max(0, f(i) - d(i)) \right) \]
Examples of Lateness
Examples of Lateness
Examples of Lateness

- $0 - 1 = 1$
- $6 - 6 = 0$
- $10 - 2 = 8$
- $16 - 1 = 15$
Examples of Lateness
Examples of Lateness

<table>
<thead>
<tr>
<th>Time</th>
<th>Lateness</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2 - 1 = 1</td>
</tr>
<tr>
<td>4</td>
<td>6 - 6 = 0</td>
</tr>
<tr>
<td>6</td>
<td>12 - 1 = 11</td>
</tr>
<tr>
<td>10</td>
<td>16 - 2 = 14</td>
</tr>
</tbody>
</table>

T. M. Murali
February 10, 15, 2021
CS 4104: Greed is Good
Scheduling to Minimise Lateness

INSTANCE: Set \( \{(t(i), d(i)), 1 \leq i \leq n\} \) of lengths and deadlines of \( n \) jobs.

SOLUTION: Set \( \{s(i), 1 \leq i \leq n\} \) of start times such that \( \max_{1 \leq i \leq n} \left( \max \left( 0, s(i) + t(i) - d(i) \right) \right) \) is as small as possible.
Scheduling to Minimise Lateness

**INSTANCE:** Set \( \{(t(i), d(i)), 1 \leq i \leq n\} \) of lengths and deadlines of \( n \) jobs.

**SOLUTION:** Set \( \{s(i), 1 \leq i \leq n\} \) of start times such that

\[
\max_{1 \leq i \leq n} \left( \max \left(0, s(i) + t(i) - d(i)\right) \right)
\]

is as small as possible.
Template for Greedy Algorithm

Key question: In what order should we schedule the jobs?
Template for Greedy Algorithm

- Key question: In what order should we schedule the jobs?
  - **Shortest length** Increasing order of length $t(i)$.
  
  **Shortest slack time** Increasing order of $d(i) - t(i)$.

- **Earliest deadline** Increasing order of deadline $d(i)$. 
Template for Greedy Algorithm

Key question: In what order should we schedule the jobs?

**Shortest length**  Increasing order of length $t(i)$. Ignores deadlines completely!
Shortest job may have a very late deadline.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t(i)$</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>$d(i)$</td>
<td>100</td>
<td>10</td>
</tr>
</tbody>
</table>

**Shortest slack time**  Increasing order of $d(i) - t(i)$.

**Earliest deadline**  Increasing order of deadline $d(i)$. 
Template for Greedy Algorithm

Key question: In what order should we schedule the jobs?

Shortest length Increasing order of length $t(i)$. Ignores deadlines completely! Shortest job may have a very late deadline.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t(i)$</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>$d(i)$</td>
<td>100</td>
<td>10</td>
</tr>
</tbody>
</table>

Shortest slack time Increasing order of $d(i) - t(i)$. Job with smallest slack may take a long time.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t(i)$</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>$d(i)$</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

Earliest deadline Increasing order of deadline $d(i)$. 

Correct? Does it make sense to tackle jobs with earliest deadlines first?
Template for Greedy Algorithm

Key question: In what order should we schedule the jobs?

**Shortest length** Increasing order of length $t(i)$. Ignores deadlines completely! Shortest job may have a very late deadline.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t(i)$</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>$d(i)$</td>
<td>100</td>
<td>10</td>
</tr>
</tbody>
</table>

**Shortest slack time** Increasing order of $d(i) - t(i)$. Job with smallest slack may take a long time.

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t(i)$</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>$d(i)$</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

**Earliest deadline** Increasing order of deadline $d(i)$. Correct? Does it make sense to tackle jobs with earliest deadlines first?
Minimising Lateness: Earliest Deadline First

Order the jobs in order of their deadlines
Assume for simplicity of notation that $d_1 \leq \ldots \leq d_n$
Initially, $f = s$
Consider the jobs $i = 1, \ldots, n$ in this order
   Assign job $i$ to the time interval from $s(i) = f$ to $f(i) = f + t_i$
   Let $f = f + t_i$
End
Return the set of scheduled intervals $[s(i), f(i)]$ for $i = 1, \ldots, n$
Minimising Lateness: Earliest Deadline First

Order the jobs in order of their deadlines
Assume for simplicity of notation that \( d_1 \leq \ldots \leq d_n \)
Initially, \( f = s \)
Consider the jobs \( i = 1, \ldots, n \) in this order
  Assign job \( i \) to the time interval from \( s(i) = f \) to \( f(i) = f + t_i \)
  Let \( f = f + t_i \)
End
Return the set of scheduled intervals \([s(i), f(i)]\) for \( i = 1, \ldots, n\)
Minimising Lateness: Earliest Deadline First

Order the jobs in order of their deadlines.
Assume for simplicity of notation that \( d_1 \leq \ldots \leq d_n \).
Initially, \( f = s \).
Consider the jobs \( i = 1, \ldots, n \) in this order:
- Assign job \( i \) to the time interval from \( s(i) = f \) to \( f(i) = f + t_i \).
- Let \( f = f + t_i \).
End
Return the set of scheduled intervals \([s(i), f(i)]\) for \( i = 1, \ldots, n \).
A schedule has an inversion if a job $i$ with deadline $d(i)$ is scheduled before a job $j$ with an earlier deadline $d(j)$, i.e., $d(j) < d(i)$ and $s(i) < s(j)$.

▶ If $i$ and $j$ have the same deadlines, they cannot cause an inversion.

▶ Examples: 2 and 1, 3 and 1, 4 and 1, 5 and 1, 4 and 3.

Claim: If a schedule has an inversion, then there is a pair of jobs $i$ and $j$ such that $j$ is scheduled immediately after $i$ and $d(j) < d(i)$.
A schedule has an inversion if a job $i$ with deadline $d(i)$ is scheduled before a job $j$ with an earlier deadline $d(j)$, i.e., $d(j) < d(i)$ and $s(i) < s(j)$.

- If $i$ and $j$ have the same deadlines, they cannot cause an inversion.
- Examples: 
  - Poll
A schedule has an inversion if a job $i$ with deadline $d(i)$ is scheduled before a job $j$ with an earlier deadline $d(j)$, i.e., $d(j) < d(i)$ and $s(i) < s(j)$.

- If $i$ and $j$ have the same deadlines, they cannot cause an inversion.
- Examples: 2 and 1, 3 and 1, 4 and 1, 5 and 1, 4 and 3.
A schedule has an *inversion* if a job \( i \) with deadline \( d(i) \) is scheduled before a job \( j \) with an earlier deadline \( d(j) \), i.e., \( d(j) < d(i) \) and \( s(i) < s(j) \).

- If \( i \) and \( j \) have the same deadlines, they cannot cause an inversion.
- Examples: 2 and 1, 3 and 1, 4 and 1, 5 and 1, 4 and 3.

Claim: If a schedule has an inversion, then there is a pair of jobs \( i \) and \( j \) such that \( j \) is scheduled immediately after \( i \) and \( d(j) < d(i) \).
Properties of Schedules

Claim 1: The algorithm produces a schedule with no inversions and no idle time.

Claim 2: All schedules with no inversions and no idle time have the same lateness.

Case 1: All jobs have distinct deadlines. There is a unique schedule with no inversions and no idle time.

Case 2: Some jobs have the same deadline. Ordering of the jobs does not change the maximum lateness of these jobs.

Claim 3: There is an optimal schedule with no idle time.

Claim 4: There is an optimal schedule with no inversions and no idle time.

Claim 5: The greedy algorithm produces an optimal schedule. Follows from Claims 1, 2 and 4.
Properties of Schedules

• Claim 1: The algorithm produces a schedule with no inversions and no idle time.

• Claim 2: All schedules with no inversions and no idle time have the same lateness.
Properties of Schedules

- Claim 1: The algorithm produces a schedule with no inversions and no idle time.
- Claim 2: All schedules with no inversions and no idle time have the same lateness.
  - Case 1: All jobs have distinct deadlines.

Claim 3: There is an optimal schedule with no idle time.
Claim 4: There is an optimal schedule with no inversions and no idle time.
Claim 5: The greedy algorithm produces an optimal schedule.
Follows from Claims 1, 2 and 4.
Claim 1: The algorithm produces a schedule with no inversions and no idle time.

Claim 2: All schedules with no inversions and no idle time have the same lateness.

Case 1: All jobs have distinct deadlines. There is a unique schedule with no inversions and no idle time.
Properties of Schedules

- Claim 1: The algorithm produces a schedule with no inversions and no idle time.
- Claim 2: All schedules with no inversions and no idle time have the same lateness.
  - Case 1: All jobs have distinct deadlines. There is a unique schedule with no inversions and no idle time.
  - Case 2: Some jobs have the same deadline.

Claim 3: There is an optimal schedule with no idle time.
Claim 4: There is an optimal schedule with no inversions and no idle time.
Claim 5: The greedy algorithm produces an optimal schedule. Follows from Claims 1, 2 and 4.
Claim 1: The algorithm produces a schedule with no inversions and no idle time.

Claim 2: All schedules with no inversions and no idle time have the same lateness.

- Case 1: All jobs have distinct deadlines. There is a unique schedule with no inversions and no idle time.
- Case 2: Some jobs have the same deadline. Ordering of the jobs does not change the maximum lateness of these jobs.
• Claim 1: The algorithm produces a schedule with no inversions and no idle time.

• Claim 2: All schedules with no inversions and no idle time have the same lateness.
  ▶ Case 1: All jobs have distinct deadlines. There is a unique schedule with no inversions and no idle time.
  ▶ Case 2: Some jobs have the same deadline. Ordering of the jobs does not change the maximum lateness of these jobs.

• Claim 3: There is an optimal schedule with no idle time.
Claim 1: The algorithm produces a schedule with no inversions and no idle time.

Claim 2: All schedules with no inversions and no idle time have the same lateness.
- Case 1: All jobs have distinct deadlines. There is a unique schedule with no inversions and no idle time.
- Case 2: Some jobs have the same deadline. Ordering of the jobs does not change the maximum lateness of these jobs.

Claim 3: There is an optimal schedule with no idle time.

Claim 4: There is an optimal schedule with no inversions and no idle time.
Claim 1: The algorithm produces a schedule with no inversions and no idle time.

Claim 2: All schedules with no inversions and no idle time have the same lateness.
  - Case 1: All jobs have distinct deadlines. There is a unique schedule with no inversions and no idle time.
  - Case 2: Some jobs have the same deadline. Ordering of the jobs does not change the maximum lateness of these jobs.

Claim 3: There is an optimal schedule with no idle time.

Claim 4: There is an optimal schedule with no inversions and no idle time. ?!
Claim 1: The algorithm produces a schedule with no inversions and no idle time.

Claim 2: All schedules with no inversions and no idle time have the same lateness.

- Case 1: All jobs have distinct deadlines. There is a unique schedule with no inversions and no idle time.
- Case 2: Some jobs have the same deadline. Ordering of the jobs does not change the maximum lateness of these jobs.

Claim 3: There is an optimal schedule with no idle time.

Claim 4: There is an optimal schedule with no inversions and no idle time. ?!

Claim 5: The greedy algorithm produces an optimal schedule.
Claim 1: The algorithm produces a schedule with no inversions and no idle time.

Claim 2: All schedules with no inversions and no idle time have the same lateness.

Case 1: All jobs have distinct deadlines. There is a unique schedule with no inversions and no idle time.

Case 2: Some jobs have the same deadline. Ordering of the jobs does not change the maximum lateness of these jobs.

Claim 3: There is an optimal schedule with no idle time.

Claim 4: There is an optimal schedule with no inversions and no idle time.

Claim 5: The greedy algorithm produces an optimal schedule. Follows from Claims 1, 2 and 4.
Claim 4: There is an optimal schedule with no inversions and no idle time.

Approach: Start with an optimal schedule \( O \) (that may have inversions) and use an exchange argument to convert \( O \) into a schedule that satisfies Claim 4 and has lateness not larger than \( O \).

1. If \( O \) has an inversion, let \( i \) and \( j \) be consecutive inverted jobs in \( O \). After swapping \( i \) and \( j \), we get a schedule \( O' \) with one less inversion.

2. Claim: The lateness of \( O' \) is no larger than the lateness of \( O \).

It is enough to prove the last item, since after \((n/2)\) swaps, we obtain a schedule with no inversions whose lateness is no larger than that of \( O \).
Proving Claim 4

- Claim 4: There is an optimal schedule with no inversions and no idle time.
- Approach: Start with an optimal schedule $O$ (that may have inversions) and use an *exchange argument* to convert $O$ into a schedule that satisfies Claim 4 and has lateness not larger than $O$. 

![Diagram showing intervals 0, 2, 1, 8, 15, 4, 3, 5, and 1 on a timeline from 0 to 16]
Claim 4: There is an optimal schedule with no inversions and no idle time.

Approach: Start with an optimal schedule $O$ (that may have inversions) and use an *exchange argument* to convert $O$ into a schedule that satisfies Claim 4 and has lateness not larger than $O$.

1. If $O$ has an inversion,
Claim 4: There is an optimal schedule with no inversions and no idle time.

Approach: Start with an optimal schedule \( O \) (that may have inversions) and use an *exchange argument* to convert \( O \) into a schedule that satisfies Claim 4 and has lateness not larger than \( O \).

1. If \( O \) has an inversion, let \( i \) and \( j \) be consecutive inverted jobs in \( O \). After swapping \( i \) and \( j \), we get a schedule \( O' \) with one less inversion.
Claim 4: There is an optimal schedule with no inversions and no idle time.

Approach: Start with an optimal schedule \(O\) (that may have inversions) and use an \textit{exchange argument} to convert \(O\) into a schedule that satisfies Claim 4 and has lateness not larger than \(O\).

1. If \(O\) has an inversion, let \(i\) and \(j\) be consecutive inverted jobs in \(O\). After swapping \(i\) and \(j\), we get a schedule \(O'\) with one less inversion.

2. Claim: The lateness of \(O'\) is no larger than the lateness of \(O\).
Proving Claim 4

Claim 4: There is an optimal schedule with no inversions and no idle time.

Approach: Start with an optimal schedule \( O \) (that may have inversions) and use an *exchange argument* to convert \( O \) into a schedule that satisfies Claim 4 and has lateness not larger than \( O \).

1. If \( O \) has an inversion, let \( i \) and \( j \) be consecutive inverted jobs in \( O \). After swapping \( i \) and \( j \), we get a schedule \( O' \) with one less inversion.
2. Claim: The lateness of \( O' \) is no larger than the lateness of \( O \).

It is enough to prove the last item, since after \( \binom{n}{2} \) swaps, we obtain a schedule with no inversions whose lateness is no larger than that of \( O \).
Swapping Inverted Jobs

In $O$, assume each job $r$ is scheduled for the interval $[s(r), f(r)]$ and has lateness $l(r)$. For $O'$, let the lateness of job $r$ be $l'(r)$.
Swapping Inverted Jobs

In $O$, assume each job $r$ is scheduled for the interval $[s(r), f(r)]$ and has lateness $l(r)$. For $O'$, let the lateness of job $r$ be $l'(r)$.

Claim: $l'(k) = l(k)$, for all $k \neq i, j$. 
In $O$, assume each job $r$ is scheduled for the interval $[s(r), f(r)]$ and has lateness $l(r)$. For $O'$, let the lateness of job $r$ be $l'(r)$.

Claim: $l'(k) = l(k)$, for all $k \neq i, j$.

Claim: $l'(j) \leq l(j)$. 

**Figure 4.6** The effect of swapping two consecutive, inverted jobs.
In $O$, assume each job $r$ is scheduled for the interval $[s(r), f(r)]$ and has lateness $l(r)$. For $O'$, let the lateness of job $r$ be $l'(r)$.

- **Claim:** $l'(k) = l(k)$, for all $k \neq i, j$.
- **Claim:** $l'(j) \leq l(j)$.
- **Claim:** $l'(i) \leq l(j)$.
In $O$, assume each job $r$ is scheduled for the interval $[s(r), f(r)]$ and has lateness $l(r)$. For $O'$, let the lateness of job $r$ be $l'(r)$.

- **Claim:** $l'(k) = l(k)$, for all $k \neq i, j$.
- **Claim:** $l'(j) \leq l(j)$.
- **Claim:** $l'(i) \leq l(j)$ because $l'(i) = f(j) - d_i \leq f(j) - d_j = l(j)$. 

**Figure 4.6** The effect of swapping two consecutive, inverted jobs.
Summary of Proof

1. Think of a schedule as a 2D point: $x$-coordinate is the number of inversions in the schedule and $y$-coordinate is the lateness of the schedule.
Summary of Proof

1. Think of a schedule as a 2D point: $x$-coordinate is the number of inversions in the schedule and $y$-coordinate is the lateness of the schedule.

2. Where does the schedule $A$ produced by the algorithm lie?
Summary of Proof

1. Think of a schedule as a 2D point: $x$-coordinate is the number of inversions in the schedule and $y$-coordinate is the lateness of the schedule.
2. Where does the schedule $A$ produced by the algorithm lie? Somewhere on the $y$-axis since it has no inversions, say $(0, l_A)$.
3. Where does some other schedule $B$ with no inversions lie?
Summary of Proof

1. Think of a schedule as a 2D point: $x$-coordinate is the number of inversions in the schedule and $y$-coordinate is the lateness of the schedule.

2. Where does the schedule $A$ produced by the algorithm lie? Somewhere on the $y$-axis since it has no inversions, say $(0, l_A)$.

3. Where does some other schedule $B$ with no inversions lie? Also at $(0, l_A)$ since all schedules with no inversions have the same lateness.
Summary of Proof

1. Think of a schedule as a 2D point: $x$-coordinate is the number of inversions in the schedule and $y$-coordinate is the lateness of the schedule.

2. Where does the schedule $A$ produced by the algorithm lie? Somewhere on the $y$-axis since it has no inversions, say $(0, l_A)$.

3. Where does some other schedule $B$ with no inversions lie? Also at $(0, l_A)$ since all schedules with no inversions have the same lateness.

4. Let $X$ be any schedule that is supposed to be optimal (and better than $A$). Where does $X$ lie?
Summary of Proof

1. Think of a schedule as a 2D point: $x$-coordinate is the number of inversions in the schedule and $y$-coordinate is the lateness of the schedule.

2. Where does the schedule $A$ produced by the algorithm lie? Somewhere on the $y$-axis since it has no inversions, say $(0, l_A)$.

3. Where does some other schedule $B$ with no inversions lie? Also at $(0, l_A)$ since all schedules with no inversions have the same lateness.

4. Let $X$ be any schedule that is supposed to be optimal (and better than $A$). Where does $X$ lie? At some point $(i, l_X)$, where $i > 0$ and $l_X$ are the number of inversions in and lateness of $X$, respectively. $l_X < l_A$

5. Find an inversion in $X$ and then isolate the inversion to be between consecutive jobs in $X$.

6. Swap the jobs to get a new schedule $X_{i-1}$. Where does $X_{i-1}$ lie? $X_{i-1}$ has one fewer inversion! Lateness cannot increase! So $X_{i-1}$ is at $(i-1, l_X)$ or "below."

7. Repeat until we have $X_1$ with one inversion at $(1, l_X)$ or "below", where $l_X < l_A$.

8. Repeat one more step: $X_0$ has no inversions. What is $X_0$'s location? $(0, l_X)$ or "below" because of #7 and $(0, l_A)$ because of #3.

9. We have a contradiction!

10. Lateness of $A$ cannot be larger than that of $O$!
Summary of Proof

1. Think of a schedule as a 2D point: $x$-coordinate is the number of inversions in the schedule and $y$-coordinate is the lateness of the schedule.

2. Where does the schedule $A$ produced by the algorithm lie? Somewhere on the $y$-axis since it has no inversions, say $(0, l_A)$.

3. Where does some other schedule $B$ with no inversions lie? Also at $(0, l_A)$ since all schedules with no inversions have the same lateness.

4. Let $X$ be any schedule that is supposed to be optimal (and better than $A$). Where does $X$ lie? At some point $(i, l_X)$, where $i > 0$ and $l_X$ are the number of inversions in and lateness of $X$, respectively. $l_X < l_A$

5. Find an inversion in $X$ and then isolate the inversion to be between consecutive jobs in $X$.

6. Swap the jobs to get a new schedule $X_{i-1}$. Where does $X_{i-1}$ lie?
Summary of Proof

Think of a schedule as a 2D point: \(x\)-coordinate is the number of inversions in the schedule and \(y\)-coordinate is the lateness of the schedule.

Where does the schedule \(A\) produced by the algorithm lie? Somewhere on the \(y\)-axis since it has no inversions, say \((0, l_A)\).

Where does some other schedule \(B\) with no inversions lie? Also at \((0, l_A)\) since all schedules with no inversions have the same lateness.

Let \(X\) be any schedule that is supposed to be optimal (and better than \(A\)). Where does \(X\) lie? At some point \((i, l_X)\), where \(i > 0\) and \(l_X\) are the number of inversions in and lateness of \(X\), respectively. \(l_X < l_A\)

Find an inversion in \(X\) and then isolate the inversion to be between consecutive jobs in \(X\).

Swap the jobs to get a new schedule \(X_{i-1}\). Where does \(X_{i-1}\) lie? \(X_{i-1}\) has one fewer inversion! Lateness cannot increase! So \(X_{i-1}\) is at \((i - 1, l_X)\) or “below.”

Repeat until we have \(X_1\) with one inversion at
Summary of Proof

1. Think of a schedule as a 2D point: $x$-coordinate is the number of inversions in the schedule and $y$-coordinate is the lateness of the schedule.

2. Where does the schedule $A$ produced by the algorithm lie? Somewhere on the $y$-axis since it has no inversions, say $(0, l_A)$.

3. Where does some other schedule $B$ with no inversions lie? Also at $(0, l_A)$ since all schedules with no inversions have the same lateness.

4. Let $X$ be any schedule that is supposed to be optimal (and better than $A$). Where does $X$ lie? At some point $(i, l_X)$, where $i > 0$ and $l_X$ are the number of inversions in and lateness of $X$, respectively. $l_X < l_A$

5. Find an inversion in $X$ and then isolate the inversion to be between consecutive jobs in $X$.

6. Swap the jobs to get a new schedule $X_{i-1}$. Where does $X_{i-1}$ lie? $X_{i-1}$ has one fewer inversion! Lateness cannot increase! So $X_{i-1}$ is at $(i-1, l_X)$ or “below.”

7. Repeat until we have $X_1$ with one inversion at $(1, l_X)$ or “below”, where $l_X < l_A$.

8. Repeat one more step: $X_0$ has no inversions. What is $X_0$’s location?
Summary of Proof

1. Think of a schedule as a 2D point: $x$-coordinate is the number of inversions in the schedule and $y$-coordinate is the lateness of the schedule.

2. Where does the schedule $A$ produced by the algorithm lie? Somewhere on the $y$-axis since it has no inversions, say $(0, l_A)$.

3. Where does some other schedule $B$ with no inversions lie? Also at $(0, l_A)$ since all schedules with no inversions have the same lateness.

4. Let $X$ be any schedule that is supposed to be optimal (and better than $A$). Where does $X$ lie? At some point $(i, l_X)$, where $i > 0$ and $l_X$ are the number of inversions in and lateness of $X$, respectively. $l_X < l_A$

5. Find an inversion in $X$ and then isolate the inversion to be between consecutive jobs in $X$.

6. Swap the jobs to get a new schedule $X_{i-1}$. Where does $X_{i-1}$ lie? $X_{i-1}$ has one fewer inversion! Lateness cannot increase! So $X_{i-1}$ is at $(i-1, l_X)$ or “below.”

7. Repeat until we have $X_1$ with one inversion at $(1, l_X)$ or “below”, where $l_X < l_A$.

8. Repeat one more step: $X_0$ has no inversions. What is $X_0$’s location? $(0, l_X)$ or “below” because of #7 and $(0, l_A)$ because of #3.
Summary of Proof

1. Think of a schedule as a 2D point: x-coordinate is the number of inversions in the schedule and y-coordinate is the lateness of the schedule.
2. Where does the schedule $A$ produced by the algorithm lie? Somewhere on the y-axis since it has no inversions, say $(0, l_A)$.
3. Where does some other schedule $B$ with no inversions lie? Also at $(0, l_A)$ since all schedules with no inversions have the same lateness.
4. Let $X$ be any schedule that is supposed to be optimal (and better than $A$). Where does $X$ lie? At some point $(i, l_X)$, where $i > 0$ and $l_X$ are the number of inversions in and lateness of $X$, respectively. $l_X < l_A$
5. Find an inversion in $X$ and then isolate the inversion to be between consecutive jobs in $X$.
6. Swap the jobs to get a new schedule $X_{i-1}$. Where does $X_{i-1}$ lie? $X_{i-1}$ has one fewer inversion! Lateness cannot increase! So $X_{i-1}$ is at $(i-1, l_X)$ or “below.”
7. Repeat until we have $X_1$ with one inversion at $(1, l_X)$ or “below”, where $l_X < l_A$.
8. Repeat one more step: $X_0$ has no inversions. What is $X_0$’s location? $(0, l_X)$ or “below” because of #7 and $(0, l_A)$ because of #3.
9. We have a contradiction!
10. Lateness of $A$ cannot be larger than that of $O$!
Common Mistakes with Exchange Arguments

- Wrong: start with algorithm’s schedule $A$ and argue that $A$ cannot be improved by swapping two jobs.

- Correct: Start with an arbitrary schedule $O$ (which can be the optimal one) and argue that $O$ can be converted into the schedule that is essentially the same as the one the algorithm produces without increasing the lateness.
Common Mistakes with Exchange Arguments

- **Wrong:** start with algorithm’s schedule $A$ and argue that $A$ cannot be improved by swapping two jobs.
- **Correct:** Start with an arbitrary schedule $O$ (which can be the optimal one) and argue that $O$ can be converted into the schedule that is essentially the same as the one the algorithm produces without increasing the lateness.
- **Wrong:** Swap two jobs that are not neighbouring in $O$. Pitfall is that the completion times of all intervening jobs changes.
- **Correct:** Show that an inversion exists between two neighbouring jobs and swap them.
Summary

- Greedy algorithms make local decisions.
- Three analysis strategies:
  - **Greedy algorithm stays ahead** Show that after each step in the greedy algorithm, its solution is at least as good as that produced by any other algorithm.
  - **Structural bound** First, discover a property that must be satisfied by every possible solution. Then show that the (greedy) algorithm produces a solution with this property.
  - **Exchange argument** Transform the optimal solution in steps into the solution by the greedy algorithm without worsening the quality of the optimal solution.