CS3230 Tutorial 8

Greedy Algorithms

(AY 25/26 Semester 2)

March 16, 2026

(Prepared by Benson)

Contents

Recap: Greedy Algorithms

Dynamic Programming:

Optimal Substructure:
An optimal solution to a problem (instance) contains optimal solutions to subproblems.

+

Overlapping Subproblems:
A recursive solution contains a "small" number of distinct subproblems repeated many times.

Greedy Algorithms:

Optimal Substructure:
An optimal solution to a problem (instance) contains optimal solutions to subproblems.

+

Greedy-choice Property:
There exists an optimal solution containing a specific locally optimal choice.

Recap: Optimal Substructure

The Change-Making Problem

Consider a coin system with denominations $c_1 < c_2 < \cdots < c_m$.
What is the minimum number of coins needed to pay $\$d$ **without change**?

- Example: What is the minimum number of coins needed to pay \$12, using \$1, \$3 and \$5 coins?
- Assume we have already decided for sure to take a \$3 coin.

Minimum number of coins to pay \$9

Optimal Substructure: An optimal solution to a problem (instance) contains optimal solutions to subproblems.

Recap: Optimal Substructure

Key Idea of Cut-and-Paste Argument:

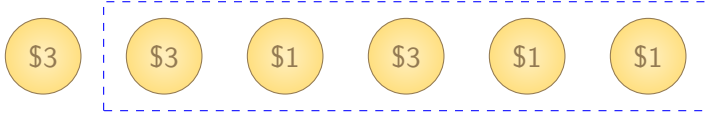
1. Suppose the solution to the subproblem is sub-optimal.
2. Cut and paste the solution to the subproblem to the whole problem.
3. Conclude that the solution to the whole problem is also sub-optimal.

Recap: Optimal Substructure

Optimal Substructure: An optimal solution to a problem (instance) contains optimal solutions to subproblems.

Proof of Optimal Substructure Property (by contradiction):

- Denote our solution by S where $\sum S = x$.
- Suppose the solution to a subproblem is suboptimal, i.e. there exists $T \subseteq S$ such that $\sum T = \sum T^* = y$, yet $|T^*| < |T|$.
- Replace the coins in T by the coins in T^*, i.e. consider $S' = (S \setminus T) \cup T^*$.
- Then, we have obtained a better solution, i.e. $\sum S' = x$ yet $|S'| < |S|$. Contradiction!



Recap: Greedy-choice Property

Optimal Substructure: An optimal solution to a problem (instance) contains optimal solutions to subproblems.

What does optimal substructure give us?

- After we fixed part of the solution, we can **continue by solving the subproblem**, as the optimal solution to the problem contains an optimal solution to the subproblem.

Minimum number of coins to pay \$9

- Enumerate the part of the solution we fix $\Rightarrow$ **Dynamic programming**.
- Is this necessary?

Recap: Greedy-choice Property

- Intuitively, we will want to use the coin with the largest denomination.
- Let $\$d$ be the amount to be paid. We will want to use the coin with the **largest** denomination c_k such that $c_k \leq d$.
- Greedy-choice Property: **There exists** an optimal solution using at least one $\$c_k$.
  - If $d \geq 5$, **there exists** an optimal solution using a \$5 coin.

> **Greedy-choice Property**: **There exists** an optimal solution containing a specific locally optimal choice.

Recap: Greedy-choice Property

Key Idea of Exchange Argument:

Take any optimal solution.
"Exchange" it for one that satisfies our property.

Recap: Greedy-choice Property

Proof of Greedy Choice Property (by exchange argument):

- Let S be any optimal solution for paying $d \geq 5$ dollars.
- If it uses a \$5 coin, we are done.
- Otherwise,
  - Case #1: If it uses more than one \$3 coin,

We obtained an equally optimal solution using a \$5 coin.

  - Case #2: If it uses one \$3 coin, it must use at least two \$1 coins (since $d \geq 5$).

We obtained a better solution, contradiction.

  - Case #3: If it uses no \$3 coins, it must use at least five \$1 coins (since $d \geq 5$).

We obtained a better solution, contradiction.

Recap: Greedy-choice Property

⚠ **There exists** an optimal solution containing x $\neq$ **All** optimal solutions contain x

- The solution 

obtained from the greedy algorithm is optimal. However, there exists a equally optimal solution 

.
- The greedy algorithm always gives *some* optimal solution, but not all optimal solutions are achievable from the greedy algorithm.

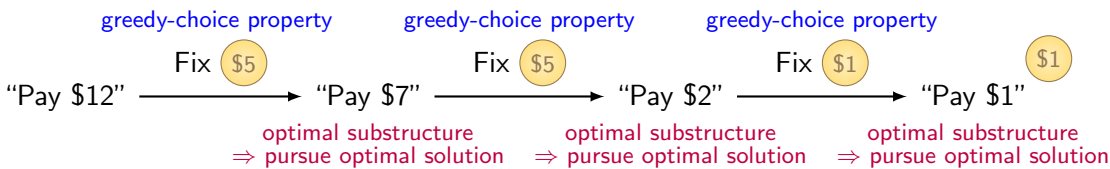
Recap: Greedy Algorithms

Optimal Substructure:
An optimal solution to a problem (instance) contains optimal solutions to subproblems.

+

Greedy-choice Property:
There exists an optimal solution containing a specific locally optimal choice.

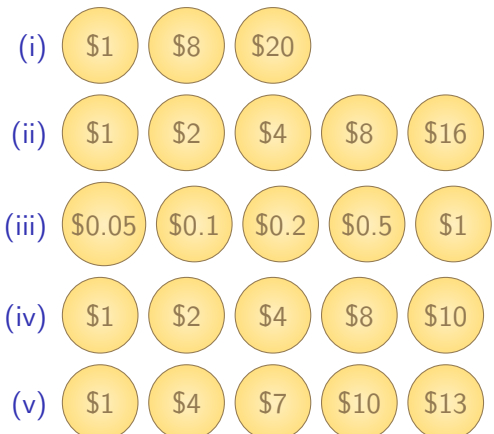
How to pay \$12?



Optimal substructure + Greedy-choice property **NATURALLY** yields a greedy algorithm.

Recap: Greedy Algorithms

Poll: Consider the following coin systems. Under how many of them does the coin change problem satisfy **(1) optimal substructure**, and **(2) greedy-choice property**?



Solution.
The proof presented for optimal substructure works for all coin systems.
Greedy-choice property:

- (i) Counter-example: \$24. We can use $3 \times$ \$8.
- (ii) Holds (convert into binary).
- (iii) Holds (Singapore coin system).
- (iv) Counter-example: \$16. We can use \$8 + \$8.
- (v) Holds (exchange argument works!).

Recap: Greedy Algorithms

Definition

A coin system is **canonical** if and only if it satisfies the greedy-choice property.

If a coin system is canonical, we can apply the **Cashier's algorithm**.

Algorithm Cashier's algorithm

```
1:  procedure CASHIER(c, d)
2:      Sort coin denominations by value: c_1 < c_2 < ··· < c_n
3:      S ← ∅
4:      while d ≠ 0 do
5:          k ← largest integer such that c_k ≤ d
6:          if k = 0 then
7:              return "no solution"
8:          d ← d − c_k
9:          S ← S ∪ {c_k}
10:     return S
```

Fitting CDs: Introduction

Suppose Bob has a collection of music files that he wants to burn into CD-s. Each CD has a storage capacity of 100 MB. In addition, Bob does not want to store more than two music files per CD. Note that each music file can't be split and hence can't be burnt on more that 1 CD.

Given a set A of file sizes (in MB), each smaller than 100MB. Let $\textsc{MinCD}(A)$ denote the minimum number of CD-s required to fit the files in A.

Q1a. Optimal Substructure

Optimal Substructure: An optimal solution to a problem (instance) contains optimal solutions to subproblems.

Describe an optimal substructure property of the problem and prove it.
Assume that at least one pair of files fit onto a CD.

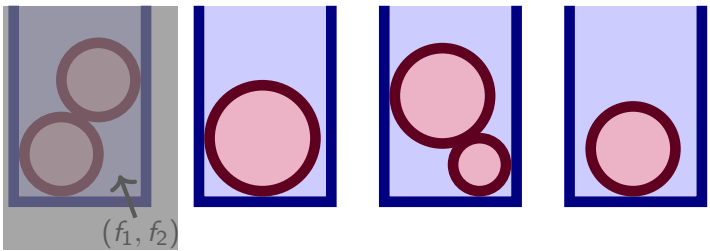
Answer:

- For any pair of files $f_1, f_2 \in A$ that belong on a single CD in an optimal solution, $\textsc{MinCD}(A) = 1 + \textsc{MinCD}(A \setminus \{f_1, f_2\})$.

Q1. Optimal Structure

For any pair of files $f_1, f_2 \in A$ that belong on a single CD in an optimal solution, $\textsc{MinCD}(A) = 1 + \textsc{MinCD}(A \setminus \{f_1, f_2\})$.

Proof:

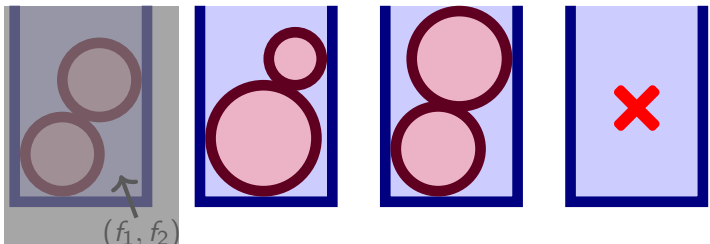


- Consider the optimal solution OPT such that (f_1, f_2) are paired onto a single CD.
- Consider $\texttt{OPT} \setminus (f_1, f_2)$. It is a valid solution to the subproblem $A \setminus \{f_1, f_2\}$.
- Hence, $\textsc{MinCD}(A \setminus \{f_1, f_2\}) \leq \textsc{MinCD}(\mathrm{A}) - 1$.

Q1. Optimal Structure

For any pair of files $f_1, f_2 \in A$ that belong on a single CD in an optimal solution, $\textsc{MinCD}(A) = 1 + \textsc{MinCD}(A \setminus \{f_1, f_2\})$.

Proof:



- Suppose $\textsc{MinCD}(A \setminus \{f_1, f_2\}) < \textsc{MinCD}(\mathrm{A}) - 1$.
  - Let $\texttt{OPT}_0$ be the optimal solution for $A \setminus \{f_1, f_2\}$.
  - Then $\texttt{OPT}_0 \cup \{f_1, f_2\}$ is a valid solution to the problem A.
  - It uses $\textsc{MinCD}(A \setminus \{f_1, f_2\}) + 1 < \textsc{MinCD}(A)$ CDs $\Rightarrow$ Contradiction.
- Therefore, it must be the case that $\textsc{MinCD}(A \setminus \{f_1, f_2\}) = \textsc{MinCD}(\mathrm{A}) - 1$.

Q1b. Greedy-choice Property (I)

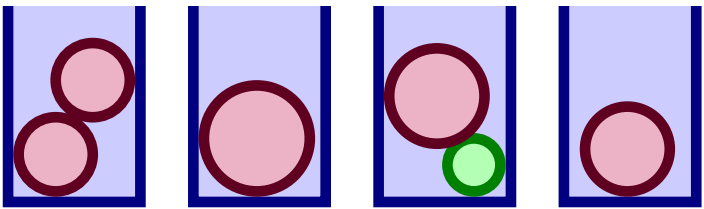
Describe a greedy-choice property of the problem and prove it.
Assume that any optimal solution always contains a pair of songs that is burned onto a CD.

Answer: The smallest file f_1 must be included in a pair in some optimal solution.

Q1b. Greedy-choice Property (I)

The smallest file f_1 must be included in a pair in some optimal solution.

Proof:

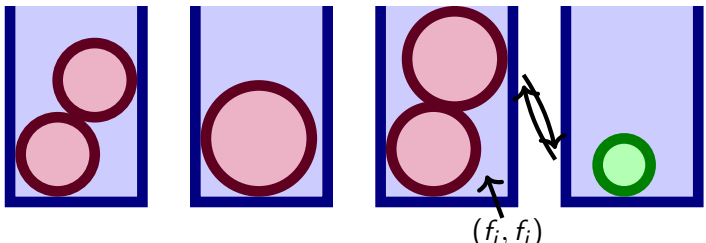


- Consider any optimal solution `OPT`.
- If f_1 is already paired in `OPT`, we are done.

Q1b. Greedy-choice Property (I)

The smallest file f_1 must be included in a pair in some optimal solution.

Proof:



- Consider any optimal solution `OPT`.
- If f_1 is not already paired in `OPT`, pick any one pair (f_i, f_j) in the optimal solution.
- Create a new solution by making f_j stand alone and pairing up (f_i, f_1). This solution is valid since $f_i + f_1 \leq f_i + f_j \leq 100$. Moreover, it is equally optimal.

Q1c. Greedy-choice Property (II)

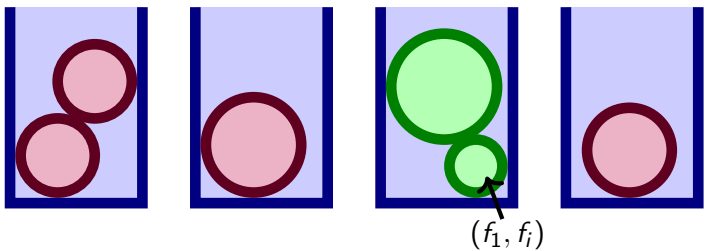
Describe *another* greedy-choice property of the problem and prove it.
Assume that any optimal solution always contains a pair of songs that is burned onto a CD.

Answer: The smallest file f_1 and $\boxed{\text{the largest file } f_2 \text{ such that } f_1 + f_2 \leq 100}$ must be included in a pair in some optimal solution.

Q1c. Greedy-choice Property (II)

The smallest file f_1 and the largest file f_2 such that $f_1 + f_2 \leq 100$ must be included in a pair in some optimal solution.

Proof:

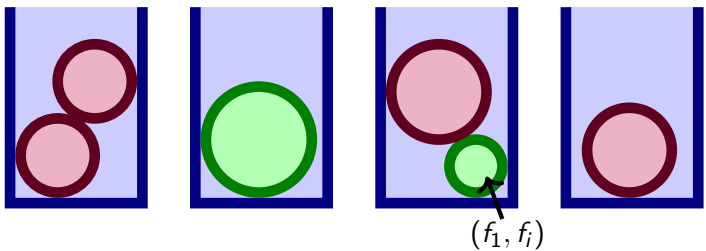


- By greedy-choice property (I), there exists an optimal solution `OPT` such that (f_1, f_i) are paired together where $i > 1$.
- If $i = 2$, we are done.

Q1c. Greedy-choice Property (II)

The smallest file f_1 and the largest file f_2 such that $f_1 + f_2 \leq 100$ must be included in a pair in some optimal solution.

Proof:

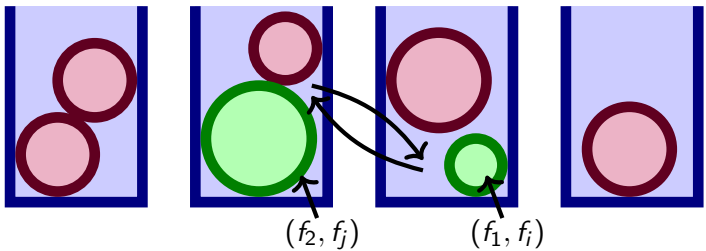


- If $i > 2$, we consider two cases.
- Case #1: f_2 is not paired with any other files.
  - Create a new solution by making f_i stand alone and pairing up (f_1, f_2). This solution is valid since $f_1 + f_2 \leq 100$. Moreover, it is equally optimal.

Q1c. Greedy-choice Property (II)

The smallest file f_1 and the largest file f_2 such that $f_1 + f_2 \leq 100$ must be included in a pair in some optimal solution.

Proof:



- If $i > 2$, we consider two cases.
- Case #2: f_2 is paired together with some other f_j.
  - Create a new solution by pairing (f_1, f_2) and (f_i, f_j). This solution is valid since $f_1 + f_2 \leq 100$ and $f_i + f_j \leq f_2 + f_j \leq 100$. It is also equally optimal.

Q1d. Greedy Algorithm

Derive a greedy algorithm for obtaining $\textsc{MinCD}$ and apply it to the following set:

$$A = \{89, 74, 81, 12, 7, 91\}$$

What is the minimum number of CD-s required?

Q1d. Greedy Algorithm

Optimal Substructure:
For any pair of files $f_1, f_2 \in A$ that belong on a single CD in an optimal solution, $\textsc{MinCD}(A) = 1 + \textsc{MinCD}(A \setminus \{f_1, f_2\})$.

+

Greedy-choice Property:
The smallest file f_1 and the largest file f_2 such that $f_1 + f_2 \leq 100$ must be included in a pair in some optimal solution.

$O(n^2)$ Algorithm:

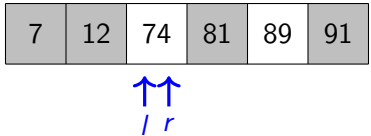
- Find smallest integer i in A.
- Find the largest integer j in A such that $i + j \leq 100$.
- If such an integer doesn't exist, let the rest of the integers in A be burnt to a CD by itself.
- Otherwise, burn both songs onto a CD (by greedy-choice property). Solve the subproblem $A \setminus \{i, j\}$ (by optimal substructure).

Q1d. Greedy Algorithm

Improved $O(n \log n)$ algorithm:

- Sort the array A.
- Let's have a left pointer l denoting the current smallest integer in A and a right pointer r denoting the largest integer in A such that you can pair it with A_l.
  - Each time l is incremented, decrement r accordingly until either you find a pair or $l \geq r$.
  - If $l < r$, pair (A_l, A_r) and increment l and decrement r simultaneously.
  - Otherwise, we can no longer find pairs to match.

Q1d. Greedy Algortihm



Optimal pairings: $(7, 91)$, $(12, 81)$

Improved $O(n \log n)$ algorithm:

- Sort the array A.
- Let's have a left pointer l denoting the current smallest integer in A and a right pointer r denoting the largest integer in A such that you can pair it with A_l.
  - Each time l is incremented, decrement r accordingly until either you find a pair or $l \geq r$.
  - If $l < r$, pair (A_l, A_r) and increment l and decrement r simultaneously.
  - Otherwise, we can no longer find pairs to match.

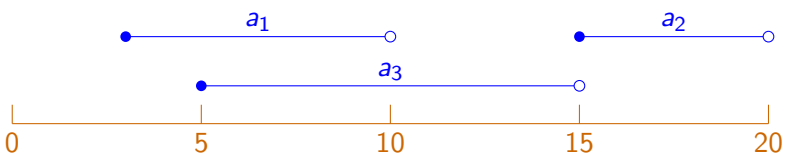
Activity Selection Problem: Introduction

Given a set of activities $S = \{a_1, a_2, \ldots, a_n\}$.

- Each activity takes place during $[s_i, f_i)$.
- Two activities are **compatible** if their time intervals do not overlap, i.e. $s_i \geq f_j$ or $s_j \geq f_i$.

Find a largest subset of mutually compatible activities.

Example: $a_1 = [3, 10)$, $a_2 = [15, 20)$, $a_3 = [5, 15)$

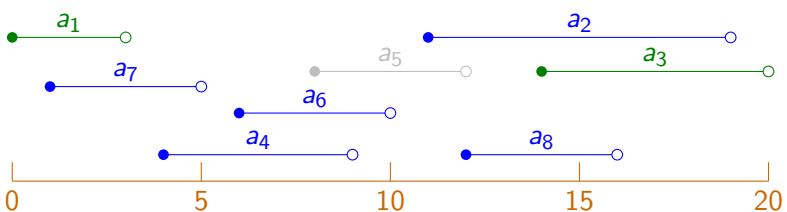


Q2a. Optimal Substructure

Describe an optimal substructure property of the problem and prove it.

Optimal Substructure Property?

Consider any optimal scheduling OPT and let $a_i = [s_i, f_i)$ be an arbitrary activity within OPT. If we remove a_i, $\texttt{OPT} \setminus \{a_i\}$ must be an optimal scheduling of the subproblem.

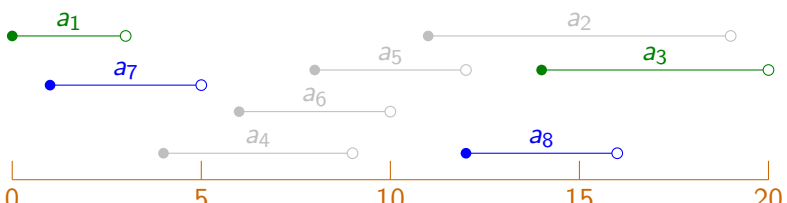


Q2a. Optimal Substructure

Describe an optimal substructure property of the problem and prove it.

Optimal Substructure Property

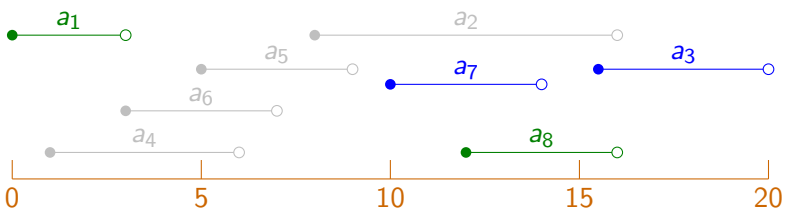
Consider any optimal scheduling OPT and let $a_i = [s_i, f_i)$ be an arbitrary activity within OPT. If we remove a_i **and all intervals conflicting with** a_i, $\texttt{OPT} \setminus \{a_i\}$ must be an optimal scheduling of the subproblem.



Q2a. Optimal Substructure

Consider any optimal scheduling $\texttt{OPT}$ and let $a_i = [s_i, f_i)$ be an arbitrary activity within $\texttt{OPT}$. If we remove a_i **and all intervals conflicting with** a_i, $\texttt{OPT} \setminus \{a_i\}$ must be an optimal scheduling of the subproblem.

Proof:



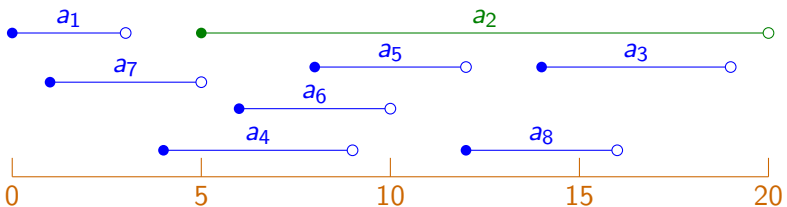
- Suppose $\texttt{OPT} \setminus \{a_i\}$ is not an optimal scheduling of the subproblem.
- Let $\texttt{OPT}'$ be the optimal scheduling of the subproblem.
- Consider the new solution $\texttt{OPT}' \cup \{a_i\}$. This is a valid solution since a_i does not conflict with any intervals in the subproblem by definition.
- Moreover, it is more optimal $\Rightarrow$ Contradiction.

Q2b. Greedy-choice Property

Describe a greedy-choice property of the problem and prove it.

Greedy-choice Property?

There exists some optimal solution including the activity that ends the latest.

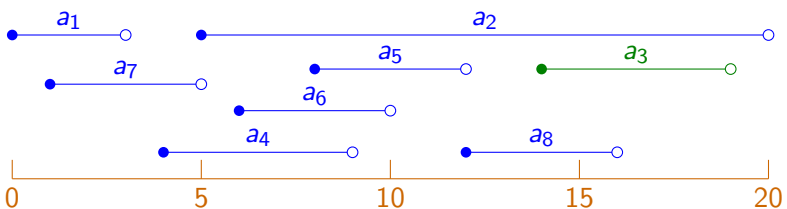


Q2b. Greedy-choice Property

Describe a greedy-choice property of the problem and prove it.

Greedy-choice Property

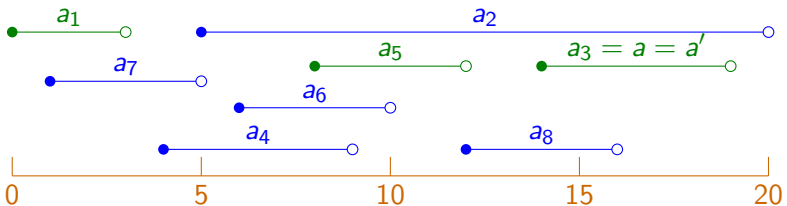
There exists some optimal solution including the activity that **starts** the latest.



Q2b. Greedy-choice Property

There exists some optimal solution including the activity that starts the latest.

Proof:

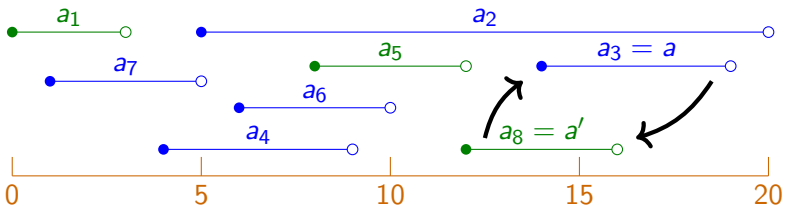


- Consider any optimal scheduling OPT. Consider the activity $a = [s, f)$ that starts last in OPT and let $a' = [s', f')$ be the activity that starts last in A.
- Case #1: If $a = a'$, we are done.

Q2b. Greedy-choice Property

There exists some optimal solution including the activity that starts the latest.

Proof:



- Consider any optimal scheduling OPT. Consider the activity $a = [s, f)$ that starts last in OPT and let $a' = [s', f')$ be the activity that starts last in A.
- Case #2: If $a \neq a'$, consider a new scheduling by replacing a with a' in OPT.
  - This scheduling is valid since for any activity $b \neq a$ in OPT, finishing time of $b \leq s \leq s'$. Moreover, this solution is equally optimal.

Q2c. Greedy Algorithm

- Sort the activities by start time in non-decreasing order.
- Inititalize $S = \varnothing$.
- Iterate through the activities in reverse order (starting from the last activity):
  - If the activity is compatible with the current schedule, add it to S.
    (We can simply check the end time against the earliest start time in S.)
  - Otherwise, skip the activity.

This runs in $O(n \log n)$.

Bonus. Concatenate Numbers

Extra Slide

There are n integers $a_1, a_2, \ldots, a_n$. **Concatenate** the n integers such that the resulting number is **maximized**.

Design a greedy algorithm which works in $O(n \log n)$ time. (Hint: Sorting them in descending order does not work.)

Solution (copy to view): If $a_i a_j > a_j a_i$, we should place a_i before a_j.