COMP 251

Algorithms & Data Structures (Winter 2021)

Algorithm Paradigms – DP3 + Greedy

School of Computer Science
McGill University

Slides of (Comp321 ,2021), Langer (2014), Kleinberg & Tardos, 2005 & Cormen et al., 2009, Jaehyun Park' slides CS 97SI, Top-coder tutorials, T-414-AFLV Course, Programming Challenges books.

Outline

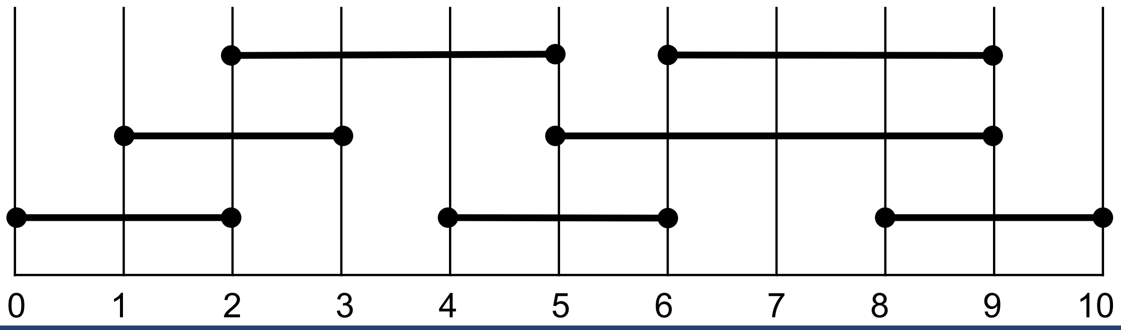
- Complete Search
- Divide and Conquer.
- Dynamic Programming.
  - Introduction.
  - Examples.
- Greedy.
  - Introduction.
  - Examples.

DP– Activity-selection Problem

- Input: Set S of n activities, $a_1, a_2, \ldots, a_n$.
  - s_i = start time of activity i.
  - f_i = finish time of activity i.
- Output: Subset A of maximum **number** of compatible activities.
  - 2 activities are compatible, if their intervals do not overlap.

Example:

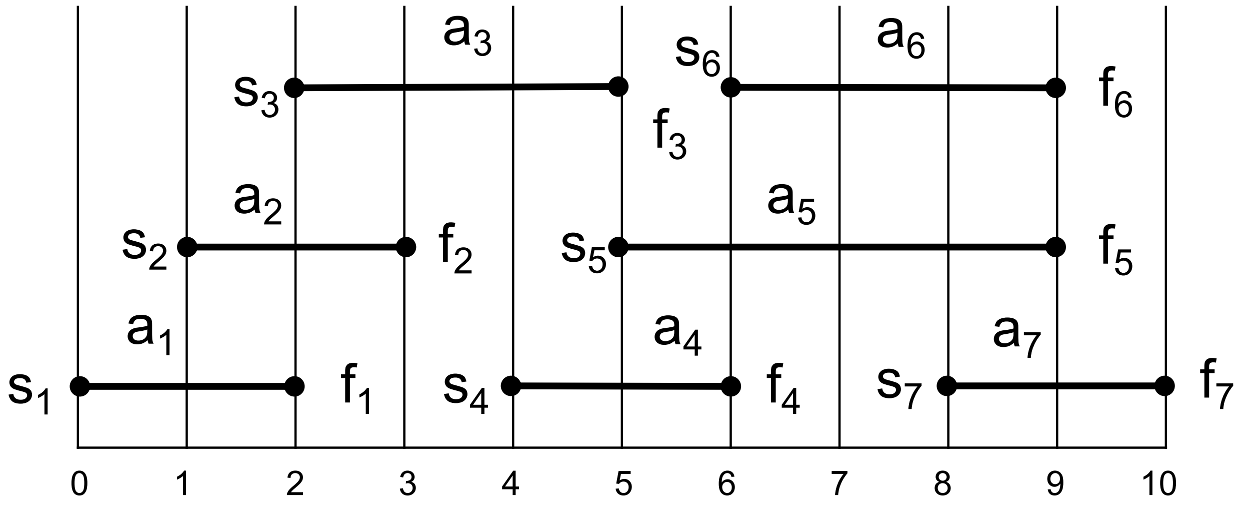
Activities in each line are compatible.



DP– Activity-selection Problem

i	1	2	3	4	5	6	7
s_i	0	1	2	4	5	6	8
f_i	2	3	5	6	9	9	10

Activities sorted by finishing time.



Optimal compatible set: $\{ a_1 , a_3 , a_5 \}$

DP– Activity-selection Problem

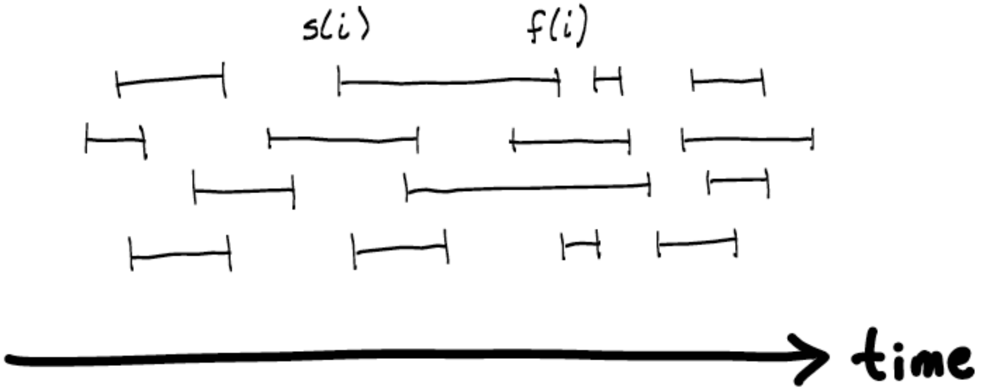
Step 1: Identify the sub-problems (in words).

Step 1.1: Identify the possible sub-problems.

- Let S_{ij} = subset of activities in S that start after a_i finishes and finish before a_j starts.

$$S_{ij} = \{ a_k \in S : \forall i,j \quad f_i \le s_k < f_k \le s_j \}$$

- A_{ij} = optimal solution to S_{ij}
- $A_{ij} = A_{ik} \cup \{ a_k \} \cup A_{kj}$
- c[i,j] = size of A_{ij}

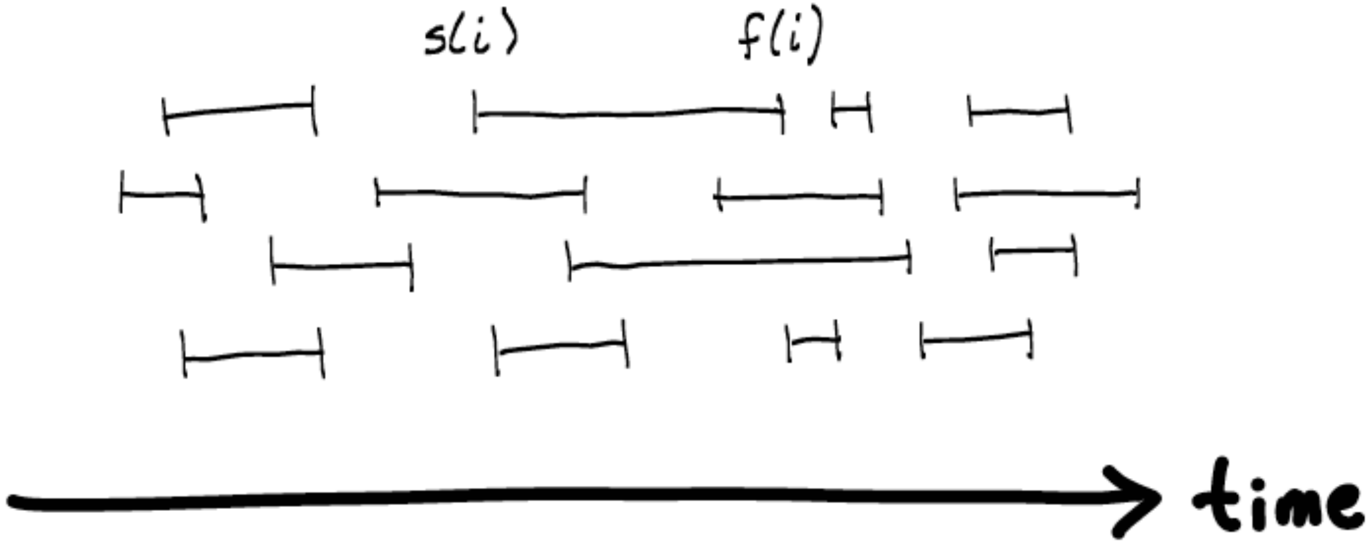


DP– Activity-selection Problem

Step 2: Find the recurrence.

Step 2.1: What decision do I make at every step?.

- Which activity a_k must I take for the optimal set.



DP– Activity-selection Problem

Step 2: Find the recurrence.

Step 2.1: What decision do I make at every step?.

- Which activity a_k must I take for the optimal set.
  - Subproblem: Selecting the maximum number of mutually compatible activities from S_{ij}.
  - Let $c[i, j]$ = size of maximum-size subset of mutually compatible activities in S_{ij}.

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset \\ \max_{k} \{c[i,k] + c[k,j] + 1\}_{i<k<j \text{ and } a_k \in S_{ij}} & \text{if } S_{ij} \neq \emptyset \end{cases}$$

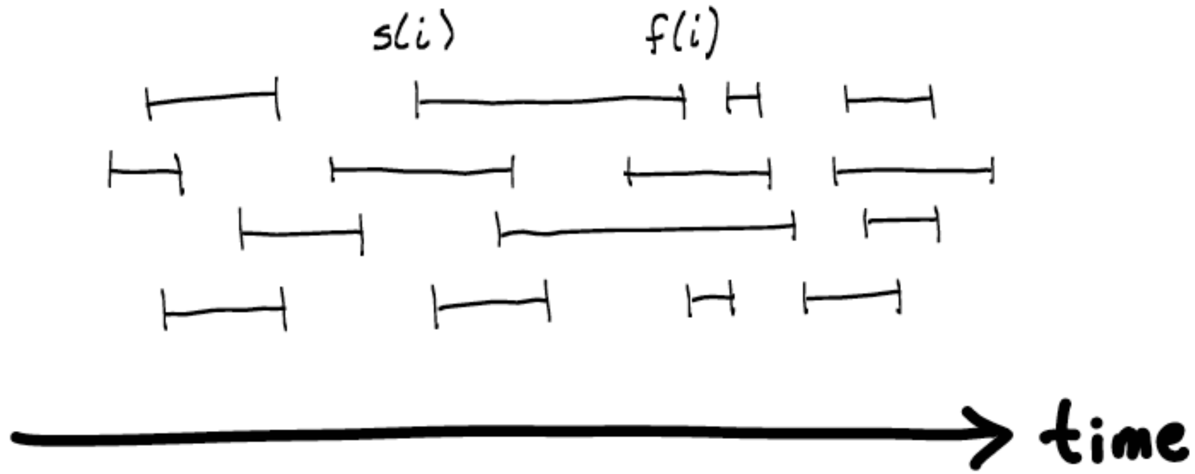
Note: We do not know (yet) which k to use for the optimal solution.

DP– Activity-selection Problem

Step 2: Find the recurrence.

- Which activity a_k must I take for the optimal set.

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset \\ \max\limits_{i<k<j \text{ and } a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \emptyset \end{cases}$$



DP– Activity-selection Problem

Step 3: Recognize and solve the base cases.

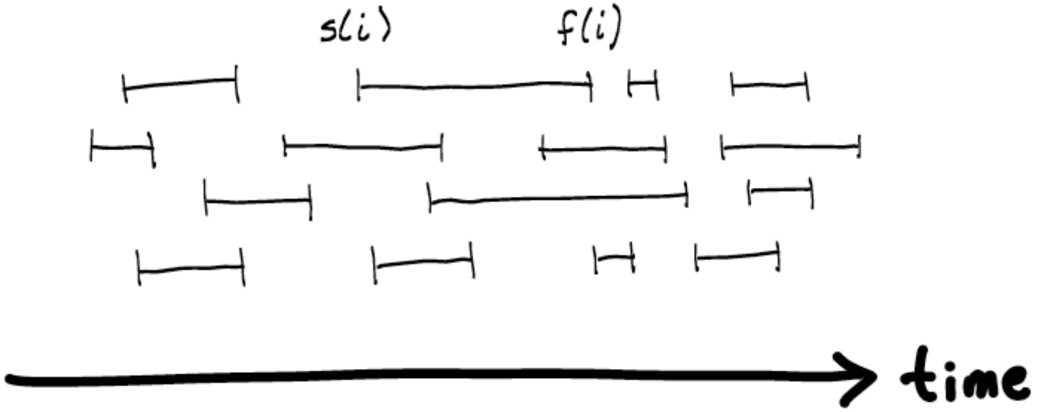
Step 4: Implement a solving methodology.

- We could then develop a recursive algorithm and memoize it, or we could work bottom-up and fill in table entries as we go along. But we would be overlooking another important characteristic of the activity-selection problem that we can use to great advantage.
  - What if we could choose an activity to add to our optimal solution without having to first solve all the subproblems? That could save us from having to consider all the choices inherent in recurrence.
    - we need consider only one choice: the greedy choice

Greedy– Activity-selection Problem

Intuition:

- We should choose an activity that leaves the resource available for as many other activities as possible.
- Of the activities we end up choosing, one of them must be the first one to finish.
- Choose the activity in S with the earliest finish time

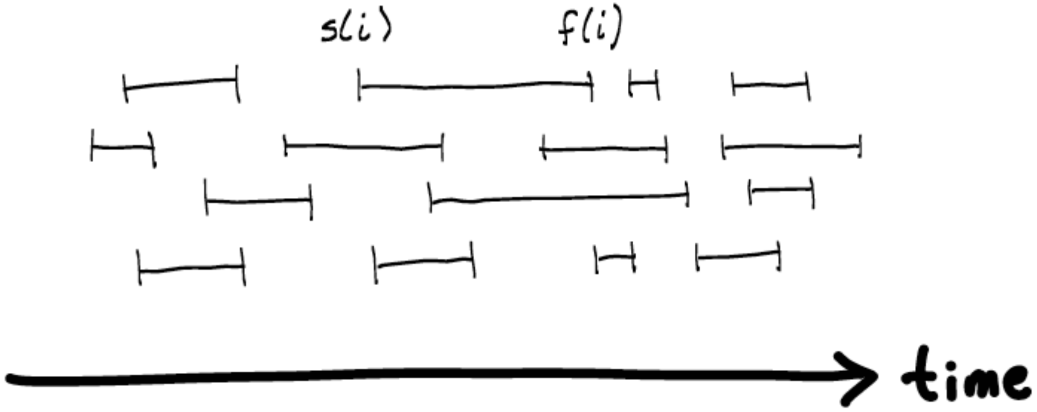


Greedy– Activity-selection Problem

Theorem:

Let $S_{ij} \neq \emptyset$, and let a_m be the activity in S_{ij} with the earliest finish time $f_m = \min\{ f_k : a_k \in S_{ij}\}$. Then:

1. a_m is used in some maximum-size subset of mutually compatible activities of S_{ij}.
2. $S_{im} = \emptyset$, so that choosing a_m leaves S_{mj} as the only nonempty subproblem.



Greedy– Activity-selection Problem

Proof:

(1) a_m is used in some maximum-size subset of mutually compatible activities of S_{ij}.

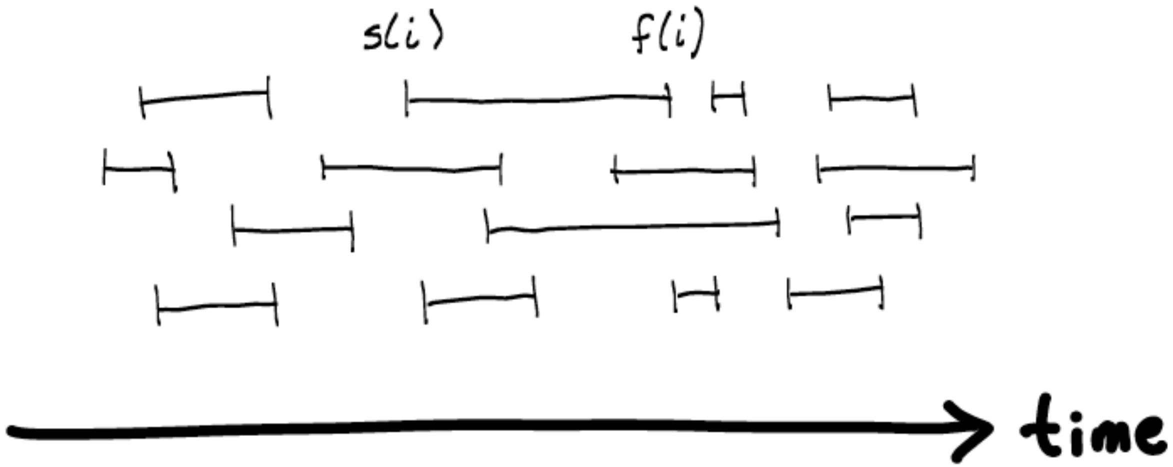
- Let A_{ij} be a maximum-size subset of mutually compatible activities in S_{ij} (i.e. an optimal solution of S_{ij}).
- Order activities in A_{ij} in monotonically increasing order of finish time, and let a_k be the first activity in A_{ij}.
- If $a_k = a_m \Rightarrow$ done.
- Otherwise, let $A'_{ij} = A_{ij} - \{ a_k \} \cup \{ a_m \}$
- A'_{ij} is valid because a_m finishes before a_k
- Since $|A_{ij}|=|A'_{ij}|$ and A_{ij} maximal $\Rightarrow A'_{ij}$ maximal too.

Greedy– Activity-selection Problem

Proof:

(2) $S_{im} = \emptyset$, so that choosing a_m leaves S_{mj} as the only nonempty subproblem.

If there is $a_k \in S_{im}$ *then* $f_i \leq s_k < f_k \leq s_m$ **$< f_m$** $\Rightarrow f_k < f_m$ which contradicts the hypothesis that a_m has the earliest finishing time.



Greedy– Activity-selection Problem

	Before theorem	After theorem
# subproblems in optimal solution	2	1
# choices to consider	j-i-1	1

$$\underbrace{\text{Before theorem}}_{A_{ij} = A_{ik} \cup \{ a_k \} \cup A_{kj}} \qquad \underbrace{\text{After theorem}}_{A_{ij} = \{ a_m \} \cup A_{mj}}$$

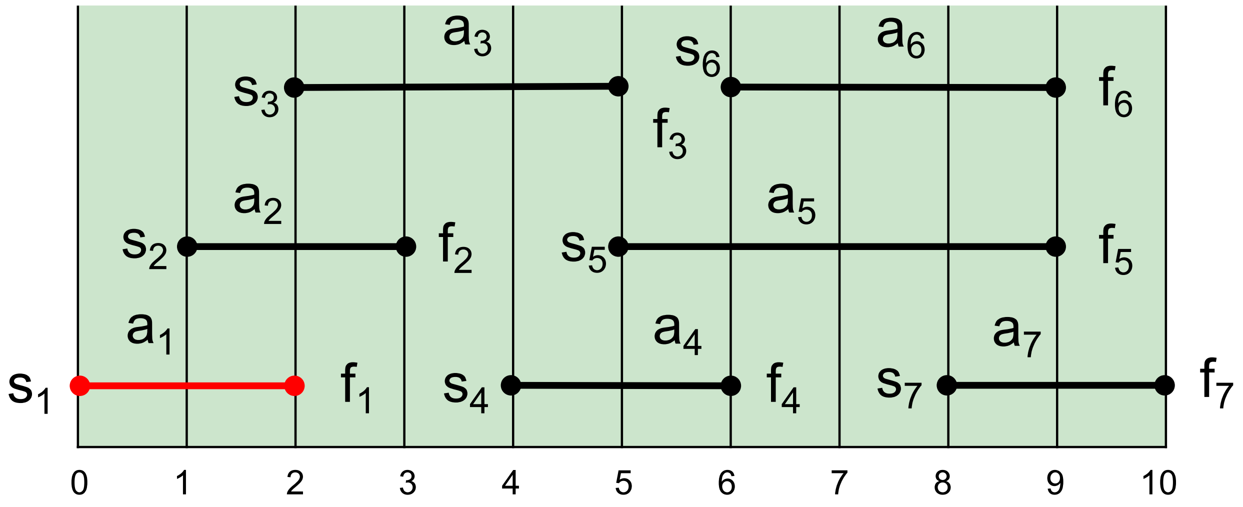
We can now solve the problem S_{ij} top-down:

- Choose $a_m \in S_{ij}$ with the earliest finish time (greedy choice).
- Solve S_{mj}.

Greedy– Activity-selection Problem

i	1	2	3	4	5	6	7
s_i	0	1	2	4	5	6	8
f_i	2	3	5	6	9	9	10

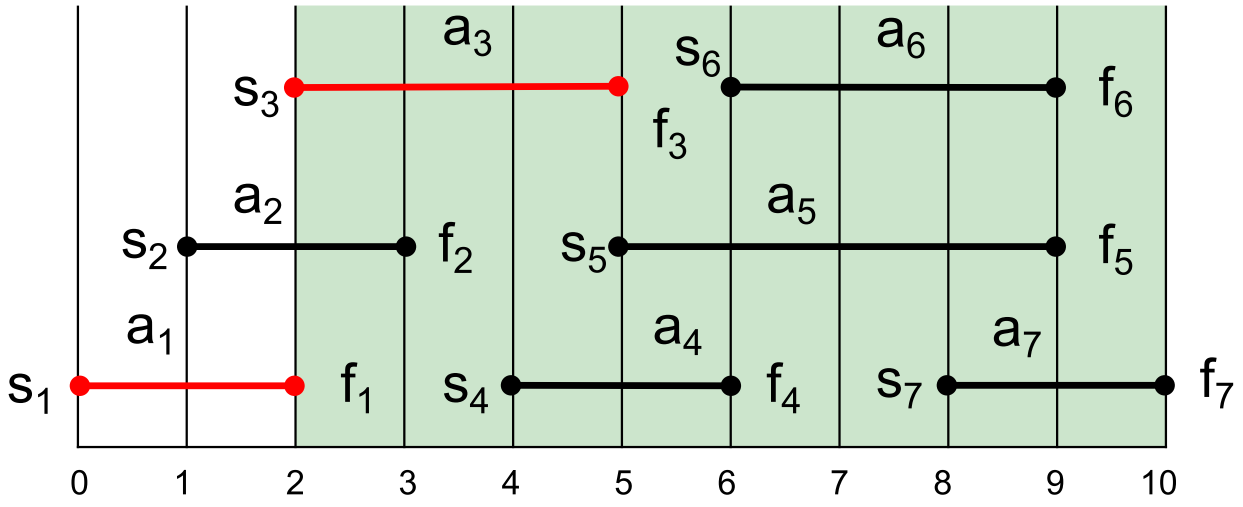
Activities sorted by finishing time.



Greedy– Activity-selection Problem

i	1	2	3	4	5	6	7
s_i	0	1	2	4	5	6	8
f_i	2	3	5	6	9	9	10

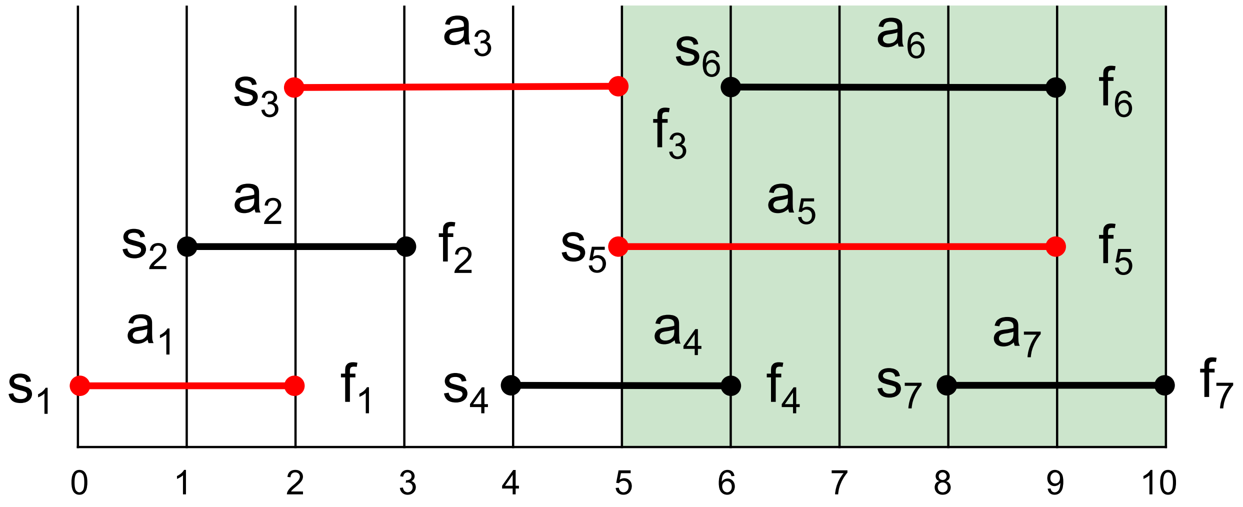
Activities sorted by finishing time.



Greedy– Activity-selection Problem

i	1	2	3	4	5	6	7
s_i	0	1	2	4	5	6	8
f_i	2	3	5	6	9	9	10

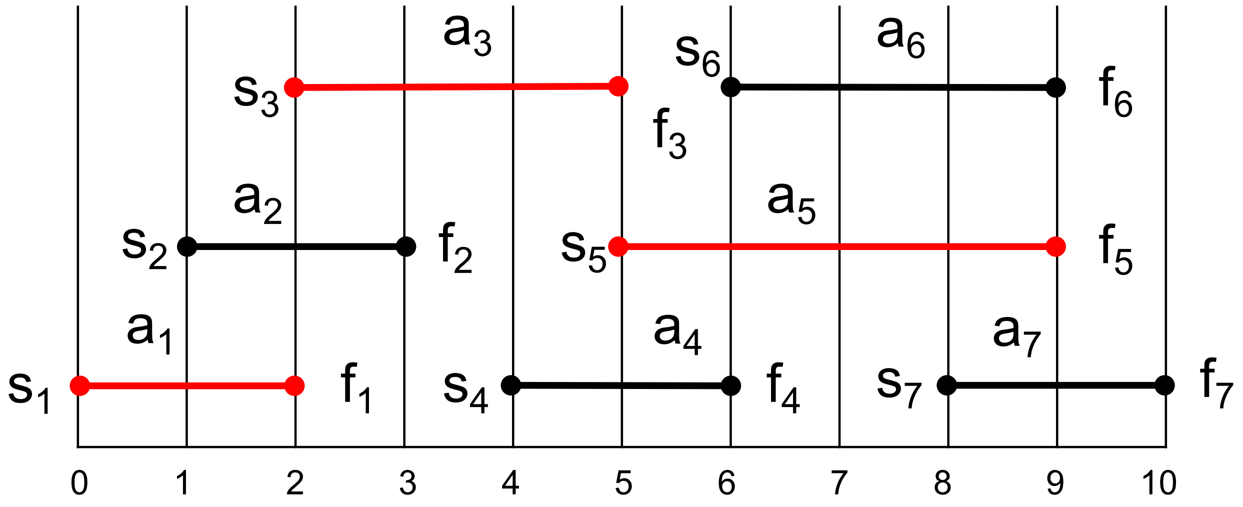
Activities sorted by finishing time.



Greedy– Activity-selection Problem

i	1	2	3	4	5	6	7
s_i	0	1	2	4	5	6	8
f_i	2	3	5	6	9	9	10

Activities sorted by finishing time.



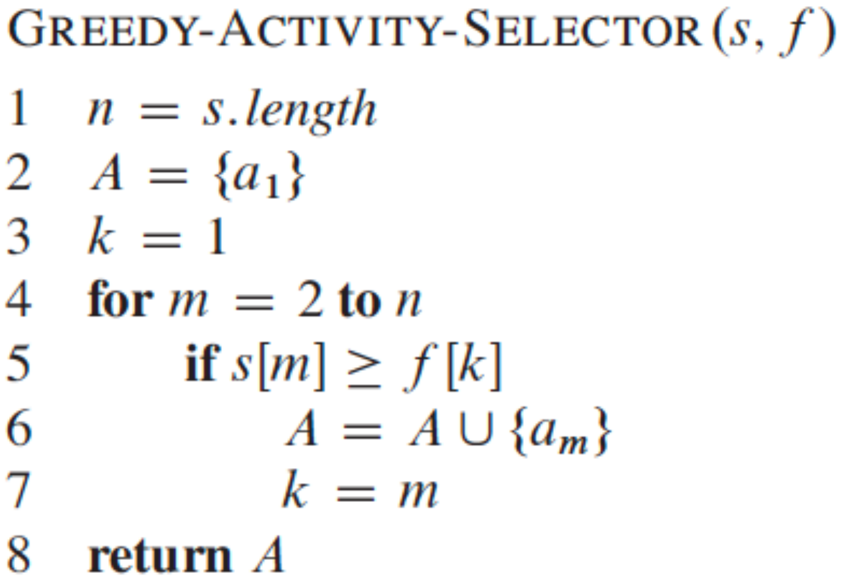
Greedy– Activity-selection Problem

```
Recursive-Activity-Selector (s, f, i, n)
1.   $m \leftarrow i+1$
2.   while $m \le n$ and $s_m < f_i$     // Find first activity in $S_{i,n+1}$
3.       do $m \leftarrow m+1$
4.   if  $m \le n$
5.       then return $\{a_m\} \cup$
                 Recursive-Activity-Selector($s, f, m, n$)
6.       else return $\emptyset$
```

Initial Call: Recursive-Activity-Selector (s, f, 0, n+1)
Complexity: $\Theta(n)$

Note 1: We assume activities are already ordered by finishing time.
Note 2: Straightforward to convert the algorithm to an iterative one.

Greedy– Activity-selection Problem

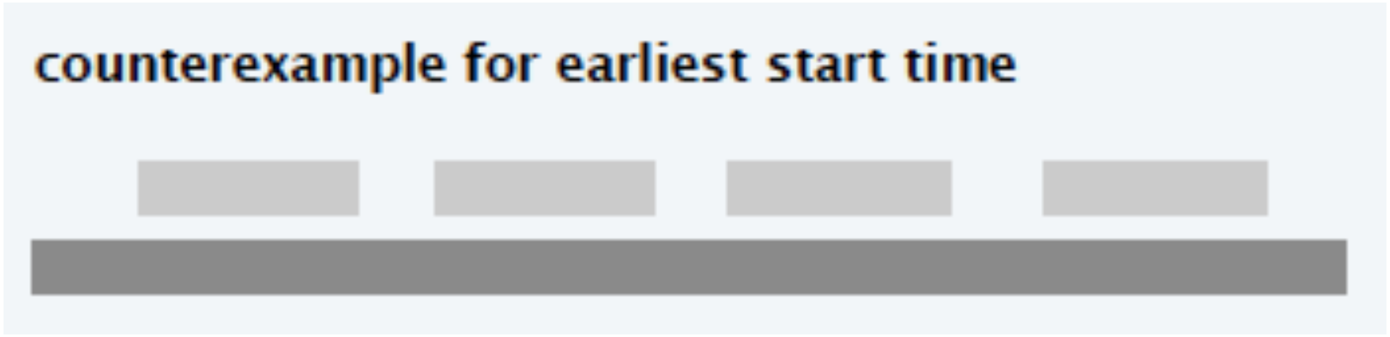


```
GREEDY-ACTIVITY-SELECTOR$(s, f)$
$n = s.length$
$A = \{a_1\}$
$k = 1$
for $m = 2$ to $n$
    if $s[m] \geq f[k]$
        $A = A \cup \{a_m\}$
        $k = m$
return $A$
```

Note 1: We assume activities are already ordered by finishing time.
Note 2: Straightforward to convert the algorithm to an iterative one.

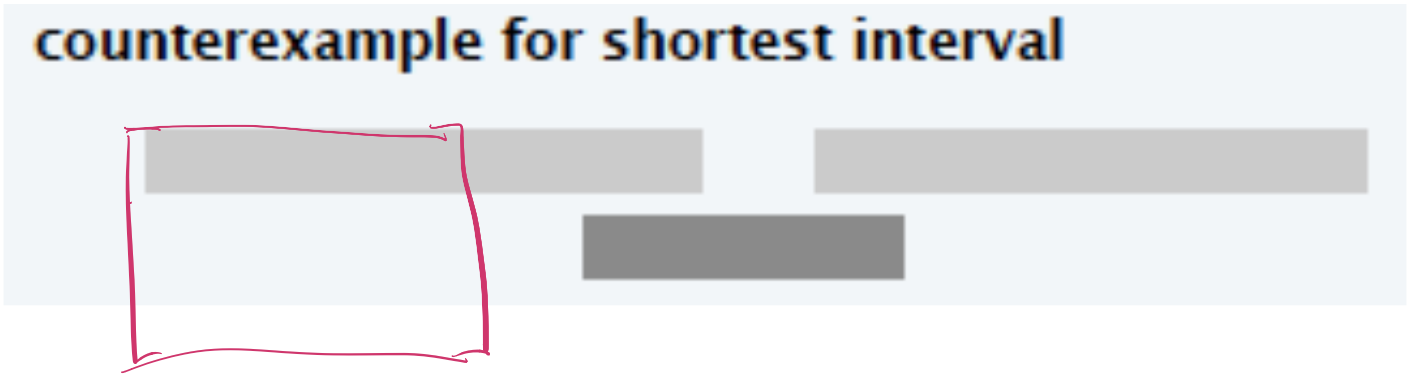
Greedy – Activity-selection Problem

- Greedy template. Consider jobs in some natural order.
- Take each job provided it's compatible with the ones already taken.
- [Earliest start time] Consider jobs in ascending order of s_j.



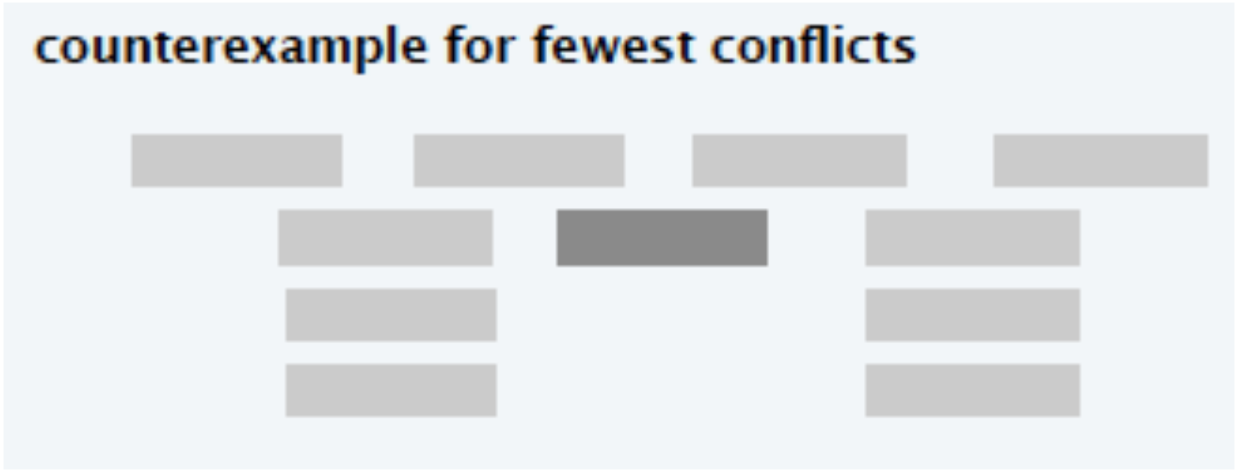
Greedy – Activity-selection Problem

- Greedy template. Consider jobs in some natural order.
- Take each job provided it's compatible with the ones already taken.
- [Shortest interval] Jobs in ascending order of $f_j – s_j$.



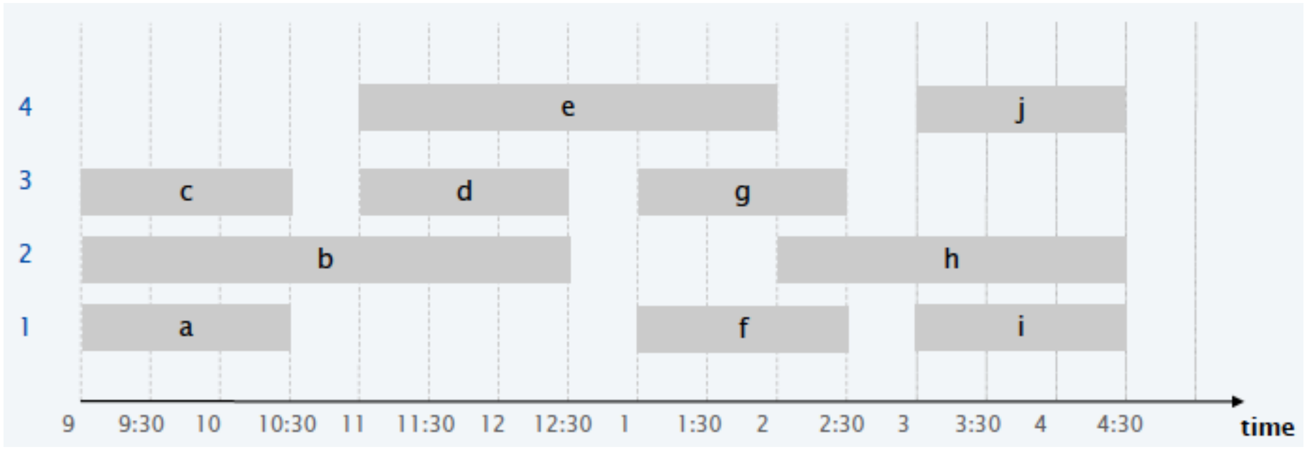
Greedy – Activity-selection Problem

- Greedy template. Consider jobs in some natural order.
- Take each job provided it's compatible with the ones already taken.
- [Fewest conflicts] For each job j, count the number of conflicting jobs c_j. Schedule in ascending order of c_j.



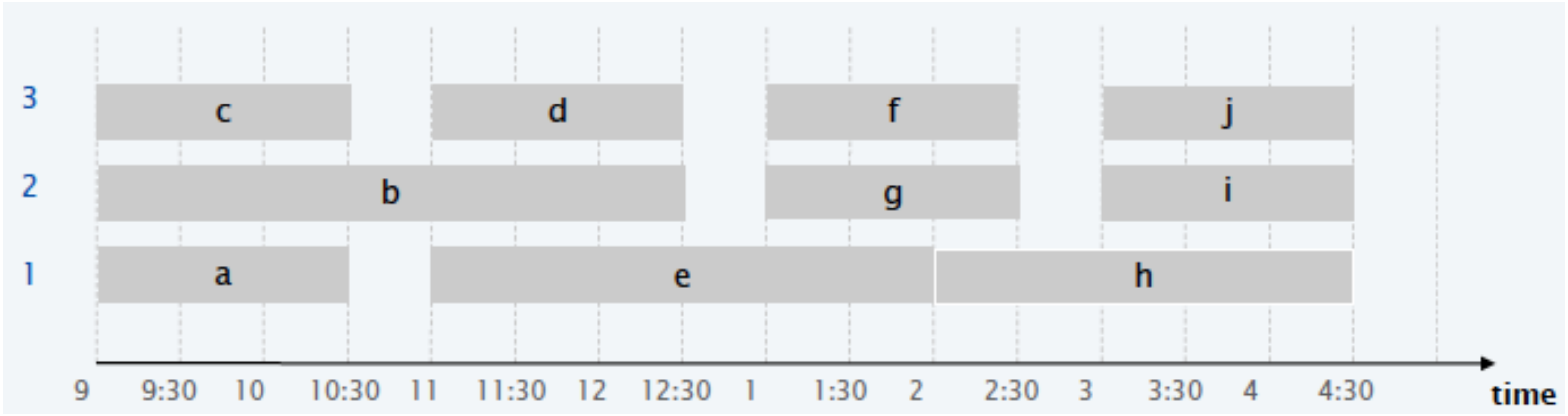
Greedy – Similar to Activity-selection Problem

- Problem:
  - Lecture j starts at s_j and finishes at f_j.
  - Goal: find minimum number of classrooms to schedule all lectures so that no two lectures occur at the same time in the same room.
  - Ex. This schedule uses 4 classrooms to schedule 10 lectures.



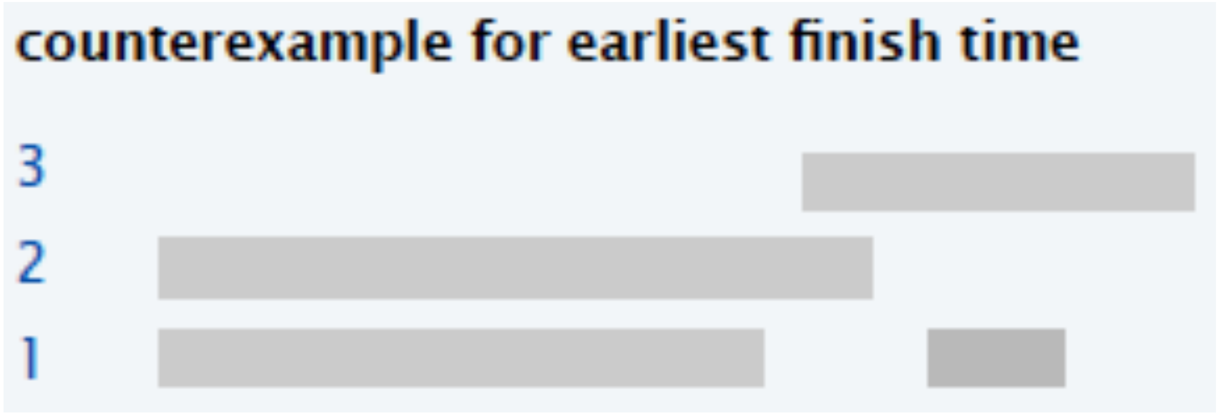
Greedy – Similar to Activity-selection Problem

- Problem:
  - Lecture j starts at s_j and finishes at f_j.
  - Goal: find minimum number of classrooms to schedule all lectures so that no two lectures occur at the same time in the same room.
  - Ex. This schedule uses 3 classrooms to schedule 10 lectures.



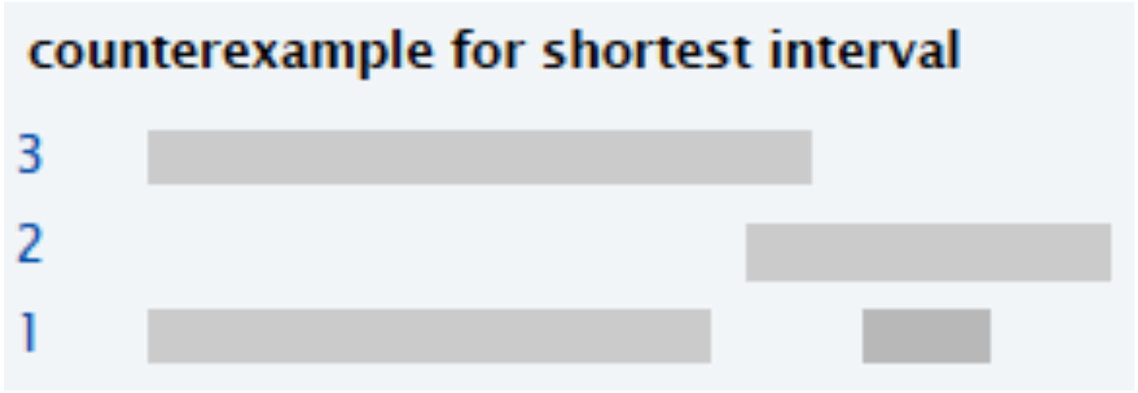
Greedy – Similar to Activity-selection Problem

- Greedy template. Consider lectures in some natural order.
- Assign each lecture to an available classroom (which one?); allocate a new classroom if none are available.
- [Earliest finish time] Consider lectures in ascending order of f_j. (solution of the previous example.)



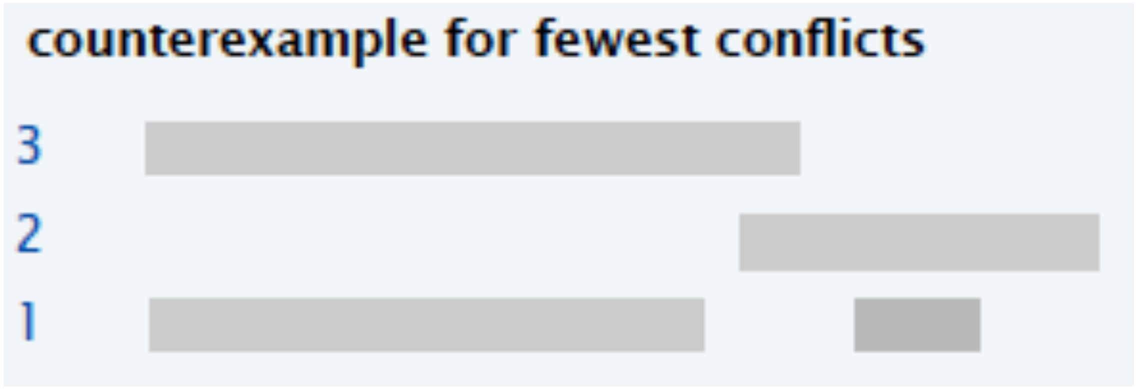
Greedy – Similar to Activity-selection Problem

- Greedy template. Consider lectures in some natural order.
- Assign each lecture to an available classroom (which one?); allocate a new classroom if none are available.
- [Shortest interval] Consider lectures in ascending order of $f_j - s_j$.



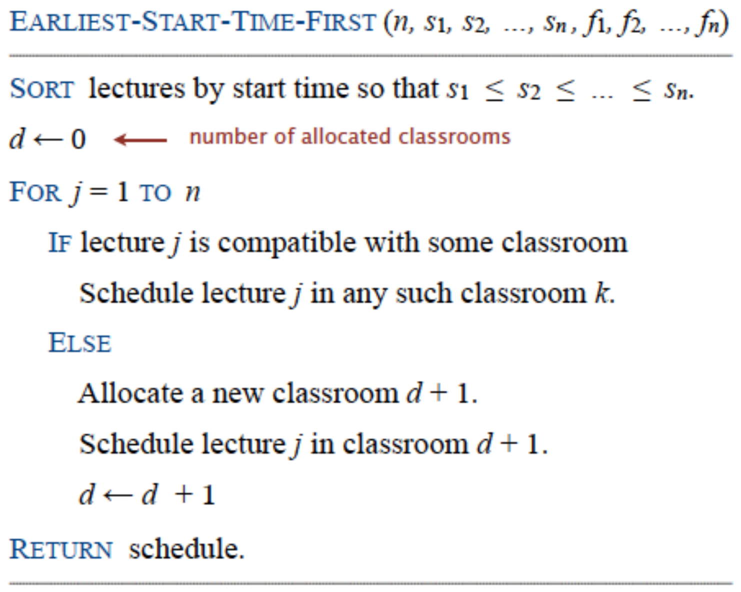
Greedy – Similar to Activity-selection Problem

- Greedy template. Consider lectures in some natural order.
- Assign each lecture to an available classroom (which one?); allocate a new classroom if none are available.
- [Fewest conflicts] For each lecture j, count the number of conflicting lectures c_j. Schedule in ascending order of c_j.



Greedy – Similar to Activity-selection Problem

- Greedy template. Consider lectures in some natural order.
- Assign each lecture to an available classroom (which one?); allocate a new classroom if none are available.
- [Earliest start time] Consider lectures in ascending order of s_j.



```
EARLIEST-START-TIME-FIRST ($n, s_1, s_2, \dots, s_n, f_1, f_2, \dots, f_n$)
SORT lectures by start time so that $s_1 \le s_2 \le \dots \le s_n$.
$d \leftarrow 0$  ⟵ number of allocated classrooms
FOR $j = 1$ TO $n$
    IF lecture $j$ is compatible with some classroom
        Schedule lecture $j$ in any such classroom $k$.
    ELSE
        Allocate a new classroom $d + 1$.
        Schedule lecture $j$ in classroom $d + 1$.
        $d \leftarrow d + 1$
RETURN schedule.
```

Greedy– Definitions

Kleinberg and Tardos: "... builds up a solution in small steps, choosing a decision at each step *myopically* (short sighted) to optimize some underlying criterion."

Cormen, Leiserson, Rivest (CLR): ".. makes the choice that looks best in the moment... it makes a locally optimal choice in the hope that the choice will lead to a globally optimal solution".

Levitin: ".. choice must be (1) feasible i.e. satisfy the problem constraints, (2) the best local choice among all feasible choices available at that step, and (3) **irrevocable**".

Hackerearth “..If we make a choice that seems best at the moment and solve the remaining subproblems later, we still reach optimal solution. We never have to reconsider our previous choices”.

Greedy– Typical Steps

- Cast the optimization problem as one in which we make a choice and are left with one subproblem to solve.
- Prove that there is always an optimal solution that makes the greedy choice (greedy choice is safe).
- Show that greedy choice and optimal solution to subproblem $\Rightarrow$ optimal solution to the problem.
- Make the greedy choice and **solve top-down**.
- You may have to preprocess input to put it into greedy order (e.g. sorting activities by finish time).

Greedy– Elements

No general way to tell if a greedy algorithm is optimal, but two key ingredients are:

- Greedy-choice Property.
  - We can build a globally optimal solution by making a locally optimal (greedy) choice.
- Optimal Substructure.

Outline

- Complete Search
- Divide and Conquer.
- Dynamic Programming.
  - Introduction.
  - Examples.
- Greedy.
  - Introduction.
  - Examples.

time for questions