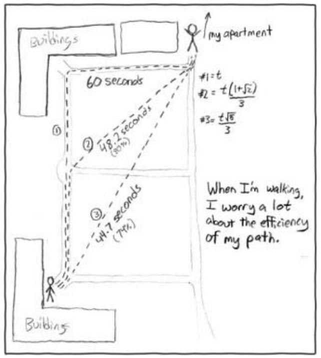
CSE 421: Algorithms

Winter 2014

Lecture 6: Greedy algorithms

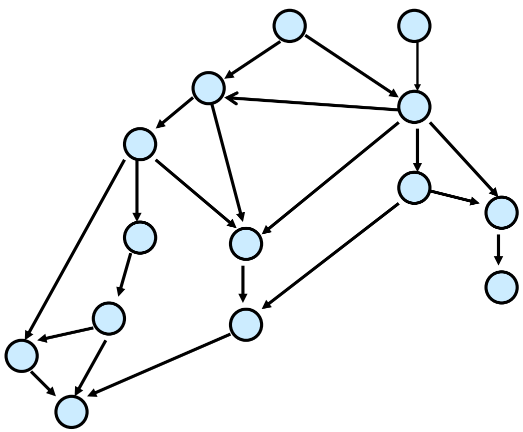
Reading: Sections 4.1-4.4



directed acyclic graphs

- A directed graph $G = (V, E)$ is **acyclic** if it has no **directed** cycles
- Terminology: A **d**irected **a**cyclic **g**raph is also called a **DAG**

directed acyclic graph



topological sort

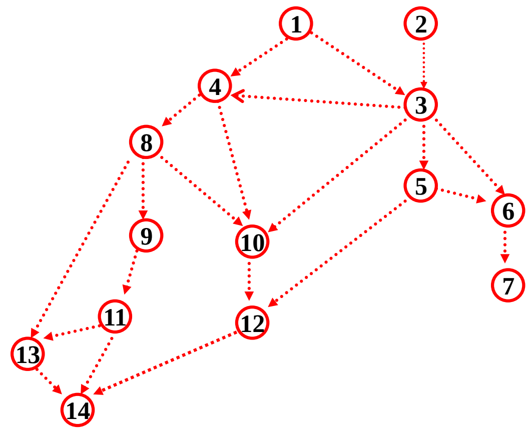
- **Given:** a directed acyclic graph (DAG) **G=(V,E)**
- **Output:** **numbering** of the vertices of **G** with **distinct** numbers from **1** to **n** so edges only go from lower number to higher numbered vertices

- Applications
  - nodes represent tasks
  - edges represent precedence between tasks
  - topological sort gives a sequential schedule for solving them

topological sort

- **Given:** a directed acyclic graph (DAG) **G=(V,E)**
- **Output:** **numbering** of the vertices of **G** with **distinct** numbers from **1** to **n** so edges only go from lower number to higher numbered vertices

topologically sorted DAG



topological sort

- Can do using DFS

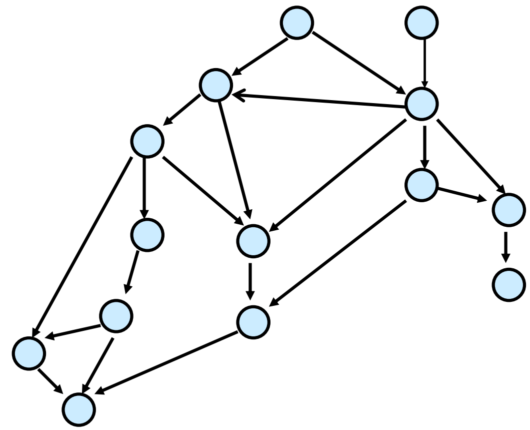
topological sort

- Can do using DFS
- Alternative simpler idea:
  - Any vertex of in-degree 0 can be given number 1 to start
  - Remove it from the graph and then give a vertex of in-degree 0 number 2, etc.

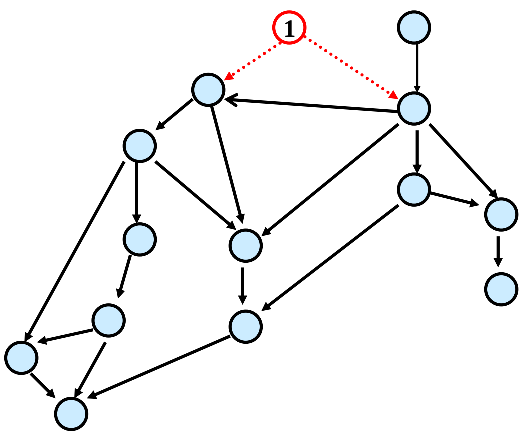
in-degree 0 vertices

Lemma: Every DAG has a vertex of in-degree 0

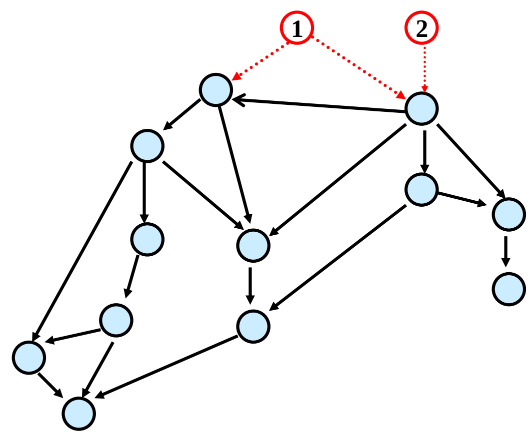
topological sort



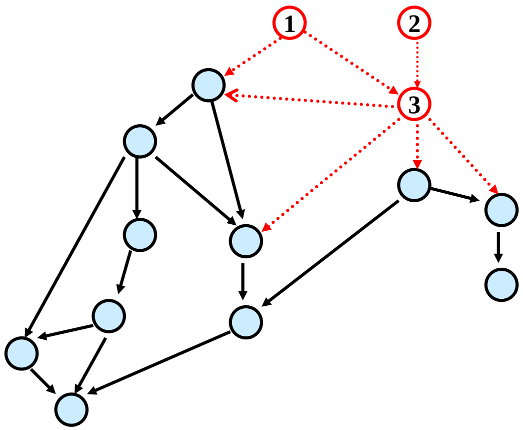
topological sort



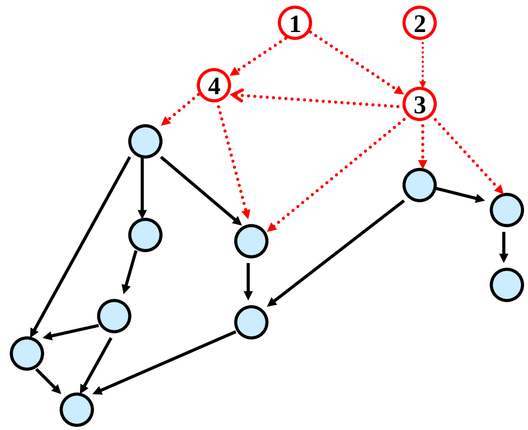
topological sort



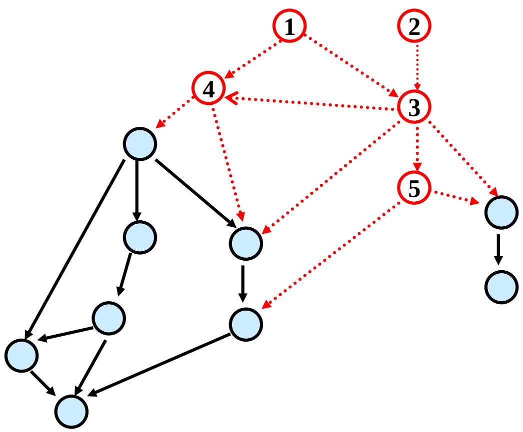
topological sort



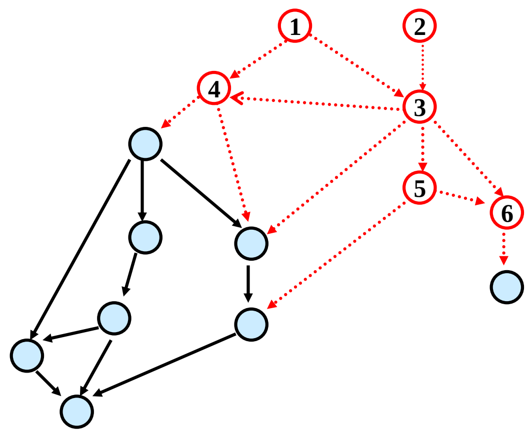
topological sort



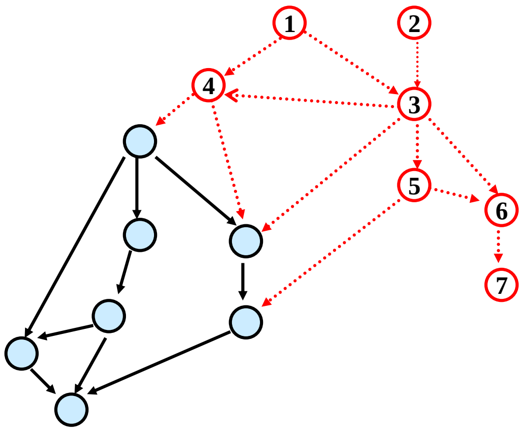
topological sort



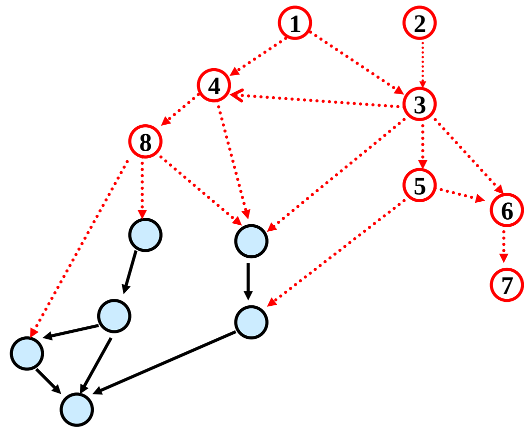
topological sort



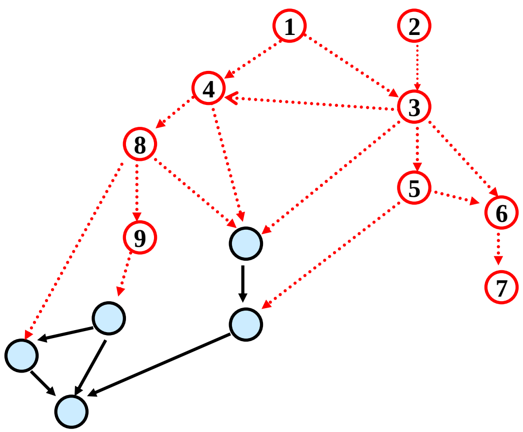
topological sort



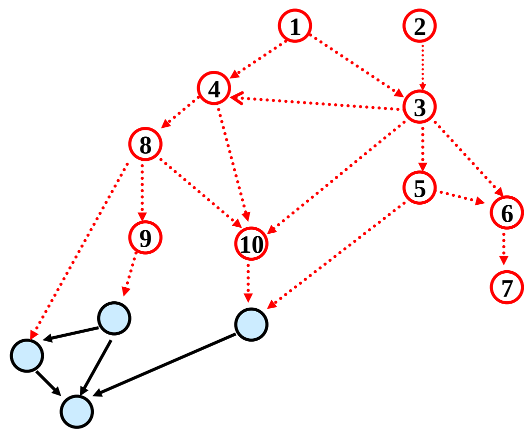
topological sort



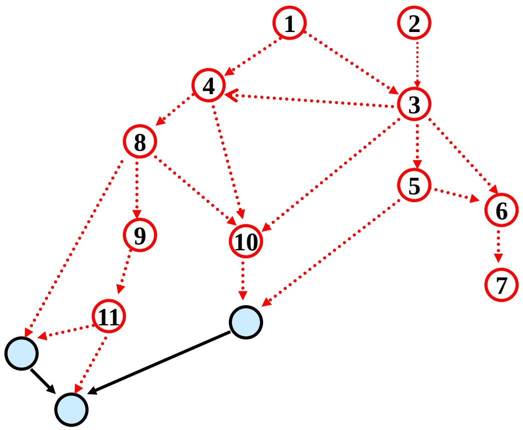
topological sort



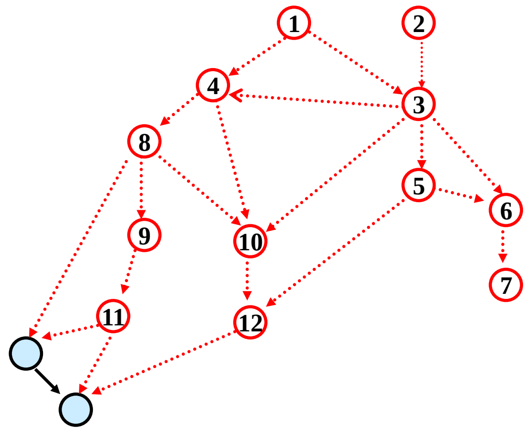
topological sort



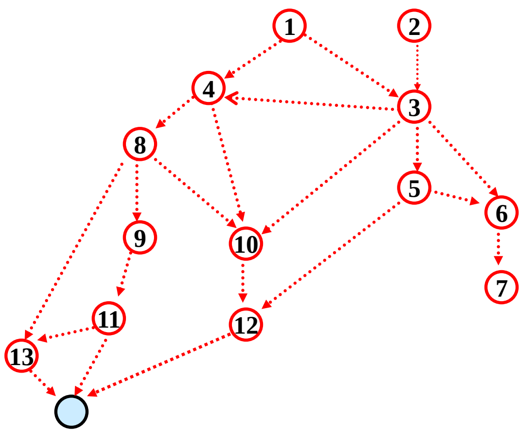
topological sort



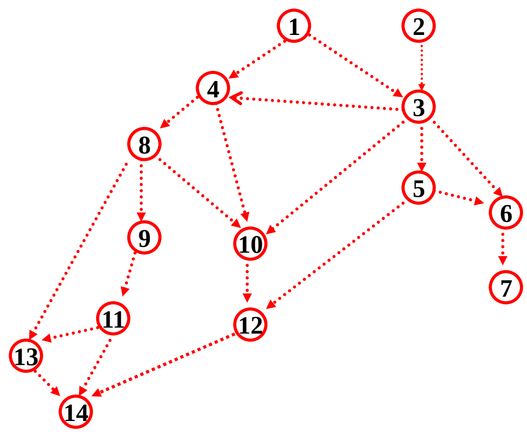
topological sort



topological sort



topological sort



implementing topological sort

- Go through all edges, computing array with in-degree for each vertex $O(m+n)$
- Maintain a queue (or stack) of vertices of in-degree **0**
- Remove any vertex in queue and number it
- When a vertex is removed, decrease in-degree of each of its neighbors by **1** and add them to the queue if their degree drops to **0**

Total cost:

greedy algorithms

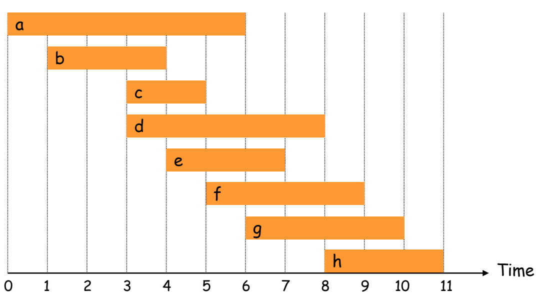
- Hard to define exactly but can give general properties
  - Solution is built in small steps
  - Decisions on how to build the solution are made to maximize some criterion without looking to the future

    Want the 'best' current partial solution as if the current step were the last step
- May be more than one greedy algorithm using different criteria to solve a given problem

interval scheduling

Interval scheduling:

- Job **j** starts at s_j and finishes at $f_j > s_j$.
- Two jobs **i** and **j** compatible if they don't overlap: $f_i \le s_j$ or $f_j \le s_i$
- **Goal:** find maximum size subset of mutually compatible jobs.



greedy algorithms

Greedy algorithms

- Easy to produce
- Fast running times
- Work only on certain classes of problems

  Hard part is showing that they are correct

Two methods for proving that greedy algorithms work

- Greedy algorithm stays ahead

  At each step any other algorithm will have a worse value for some criterion that eventually implies optimality
- Exchange Argument

  Can transform any other solution to the greedy solution at no loss in quality

interval scheduling

Single resource

Reservation requests

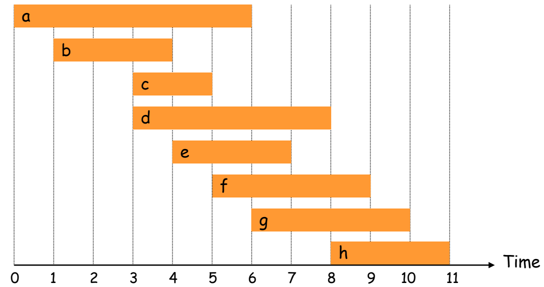
Of form "Can I reserve it from start time **s** to finish time **f**?"

$s < f$

interval scheduling

Interval scheduling:

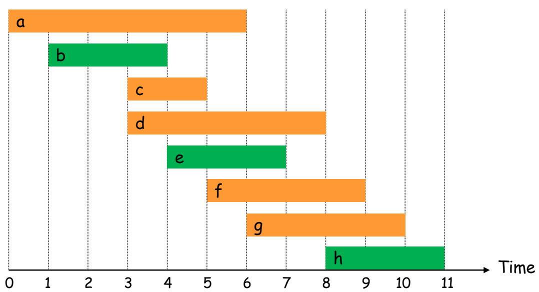
- Job j starts at s_j and finishes at $f_j > s_j$.
- Two jobs i and j compatible if they don't overlap: $f_i \leq s_j$ or $f_j \leq s_i$
- **Goal:** find maximum size subset of mutually compatible jobs.



interval scheduling

Interval scheduling:

- Job j starts at s_j and finishes at $f_j > s_j$.
- Two jobs i and j compatible if they don't overlap: $f_i \leq s_j$ or $f_j \leq s_i$
- **Goal:** find maximum size subset of mutually compatible jobs.



greedy algorithms for interval scheduling

- What criterion should we try?
  - Earliest start time s_i
  - Shortest request time f_i-s_i
  - Earliest finish fime f_i

greedy algorithms for interval scheduling

- What criterion should we try?
  - Earliest start time s_i
    Doesn't work

greedy algorithms for interval scheduling

- What criterion should we try?
  - Earliest start time s_i
    Doesn't work

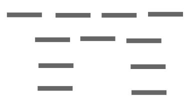
  - Shortest request time f_i-s_i
    Doesn't work

greedy algorithms for interval scheduling

- What criterion should we try?
  - Earliest start time s_i
    Doesn't work

  - Shortest request time f_i-s_i
    Doesn't work

  - Fewest conflicts doesn't work

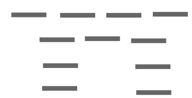


greedy algorithms for interval scheduling

- What criterion should we try?
  - Earliest start time s_i
    Doesn't work

  - Shortest request time f_i-s_i
    Doesn't work

  - Fewest conflicts doesn't work



  - Earliest finish fime f_i
    Works

greedy algorithm for interval scheduling

$R \leftarrow$ set of all requests
$A \leftarrow \varnothing$
While $R \neq \varnothing$ do
  Choose request $i \in R$ with smallest finishing time f_i
  Add request i to A
  Delete all requests in R that are not compatible with request i
Return A

greedy algorithm for interval scheduling

Claim: A is a compatible set of requests and these are added to A in order of finish time

– When we add a request to A we delete all incompatible ones from R

greedy algorithm for interval scheduling

Claim: For any other set $O \subseteq R$ of compatible requests, if we order requests in A and O by finish time then for each k:

Enough to prove that A is optimal

– If $\boxed{\text{O contains a } k^{th} \text{ request then so does A}}$ and
– the finish time of the k^{th} request in A, is $\leq$ the finish time of the k^{th} request in O, i.e. "$a_k \leq o_k$" where a_k and o_k are the respective finish times

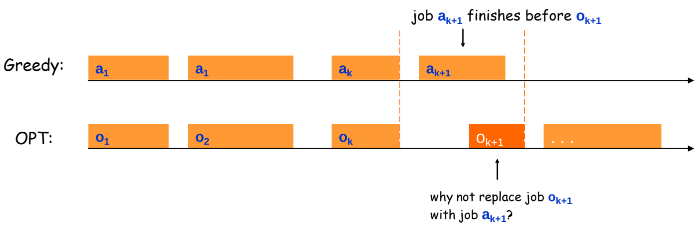
inductive proof of claim: $a_k \leq o_k$

- Base Case: This is true for the first request in A since that is the one with the smallest finish time
- Inductive Step: Suppose $a_k \leq o_k$
  - By definition of compatibility
    - If O contains a $k+1^{st}$ request r then the start time of that request must be after o_k and thus after a_k
    - Thus r is compatible with the first k requests in A
    - Therefore
      - A has at least k+1 requests since a compatible one is available after the first k are chosen
      - r was among those considered by the greedy algorithm for that $k+1^{st}$ request in A
    - Therefore by the greedy choice the finish time of r which is o_{k+1} is at least the finish time of that $k+1^{st}$ request in A which is a_{k+1}

interval scheduling: analysis

Therefore we have:

- **Theorem.** Greedy algorithm is optimal.
- Alternative Proof. (by contradiction)
  - Assume greedy is not optimal, and let's see what happens.
  - Let $a_1, a_2, \ldots a_k$ denote set of jobs selected by greedy.
  - Let $o_1, o_2, \ldots o_m$ denote set of jobs in an optimal solution with $a_1 = o_1, a_2 = o_2, \ldots, a_k = o_k$ for the largest possible value of k.



greedy algorithm implementation

O(n log n)

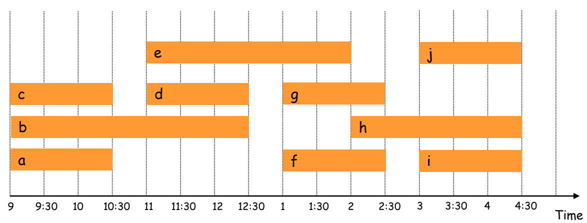
```
Sort jobs by finish times so that 0 ≤ f1 ≤ f2 ≤ ... ≤ fn.
```

O(n)

```
A ← φ
last ← 0
for j = 1 to n {
    if (last ≤ sj)
       A ← A ∪ {j}
       last ← fj
}
return A
```

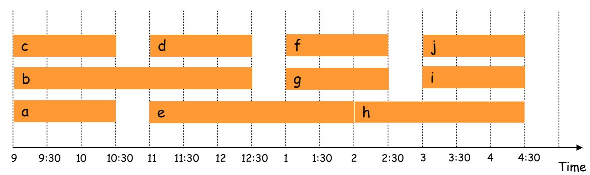
interval partitioning

- **Interval partitioning.**
  - 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 occur at the same time in the same room.
- Example: This schedule uses 4 classrooms to schedule 10 lectures.



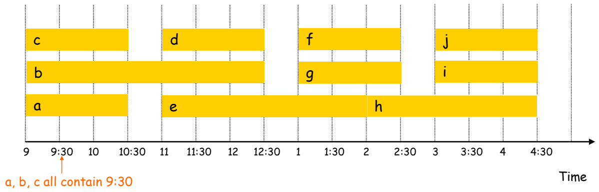
interval partitioning

- **Interval partitioning.**
  - 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 occur at the same time in the same room.
- Example: This schedule uses only 3 classrooms.



lower bound on optimal solutions

- Definition. The depth of a set of open intervals is the maximum number that contain any given time.
- Key observation. Number of classrooms needed $\geq$ depth.
- Ex: Depth of schedule below = 3 $\Rightarrow$ schedule below is optimal.



- Q. Does there always exist a schedule equal to depth of intervals?

a simple greedy algorithm

Sort requests in increasing order of start times
$(s_1,f_1),\ldots,(s_n,f_n)$

For $i=1$ to n
$j \leftarrow 1$
While (request i not scheduled)
$last_j \leftarrow$ finish time of the last request currently scheduled on resource j
if $s_i \geq last_j$ then schedule request i on resource j
$j \leftarrow j+1$
End While
End For

interval partitioning: greedy analysis

- Observation. Greedy algorithm never schedules two incompatible lectures in the same classroom.

interval partitioning: greedy analysis

- Observation. Greedy algorithm never schedules two incompatible lectures in the same classroom.
- Theorem. Greedy algorithm is optimal.
- Proof.
  - Let d = number of classrooms that the greedy algorithm allocates.
  - Classroom d is opened because we needed to schedule a job, say j, that is incompatible with all $d-1$ other classrooms.
  - Since we sorted by start time, all these incompatibilities are caused by lectures that start no later than s_j.
  - Thus, we have d lectures overlapping at time $s_j + \varepsilon$.
  - Key observation $\Rightarrow$ all schedules use $\geq d$ classrooms. ▪

a simple greedy algorithm

Sort requests in increasing order of start times
$(s_1,f_1),\ldots,(s_n,f_n)$ **O(n log n)** time

For **i=1** to **n**
  $j \leftarrow 1$
  While (request **i** not scheduled)
    $last_j \leftarrow$ finish time of the last request
      currently scheduled on resource **j**
    if $s_i \geq last_j$ then schedule request **i** on resource **j**
    $j \leftarrow j+1$
  End While
End For

May be slow: **O(nd)**
which could be **Ω(n²)**

a simple greedy algorithm

Sort requests in increasing order of start times
$(s_1,f_1),\ldots,(s_n,f_n)$ **O(n log n)** time

For **i=1** to **n**
  $j \leftarrow 1$
  While (request **i** not scheduled)
    $last_j \leftarrow$ finish time of the last request
      currently scheduled on resource **j**
    if $s_i \geq last_j$ then schedule request **i** on resource **j**
    $j \leftarrow j+1$
  End While
End For

A more efficient implementation

Sort requests in increasing order of start times $(s_1,f_1),\ldots,(s_n,f_n)$ **O(n log n)**

$d \leftarrow 1$
Schedule request **1** on resource **1**
$last_1 \leftarrow f_1$
Insert **1** into priority queue **Q** with key = $last_1$
For **i=2** to **n**
  $j \leftarrow$findmin(**Q**)
  if $s_i \geq last_j$ then
    schedule request **i** on resource **j**
    $last_j \leftarrow f_i$
    Increasekey(**j**,**Q**) to $last_j$
  else
    $d \leftarrow d+1$
    schedule request **i** on resource **d**
    $last_d \leftarrow f_i$
    Insert **d** into priority queue **Q** with key = $last_d$
End For

O(n log d)

O(n log n) time

greedy analysis strategies

- **Greedy algorithm stays ahead.** Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's.
- **Exchange argument.** Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.
- **Structural.** Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound.