CSE 421
Introduction to Algorithms

Lecture 5: Greedy Algorithms

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

- Not obvious which criteria will actually work

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
- Focus on methods for proving that greedy algorithms do work

Interval Scheduling

Interval Scheduling:

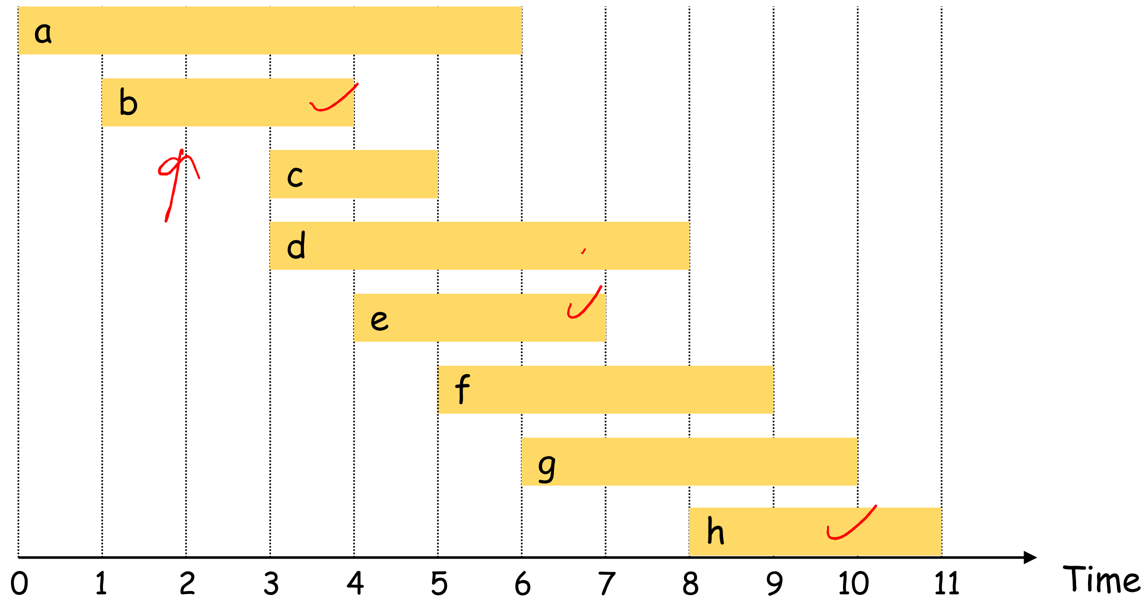
- Single resource
- Reservation requests of form:
  "Can I reserve it from start time $\boldsymbol{s}$ to finish time $\boldsymbol{f}$?"

$$\boldsymbol{s} < \boldsymbol{f}$$

Interval Scheduling

Interval scheduling:

- Job j starts at s_j and finishes at $f_j > s_j$.
- Two jobs i and j are 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?

Greedy Algorithms for Interval Scheduling

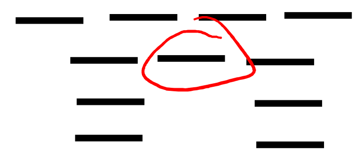
- What criterion should we try?
  - Earliest start time s_i
  - Shortest request time $f_i - s_i$
  - Fewest conflicts

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 time f_i
    - Works!

Greedy (by finish time) Algorithm for Interval Scheduling

$\boldsymbol{R} \leftarrow$ set of all requests

$\boldsymbol{A} \leftarrow \varnothing$

while $\boldsymbol{R} \neq \varnothing$ do

    Choose request $\boldsymbol{i} \in \boldsymbol{R}$ with smallest finish time $\boldsymbol{f_i}$

    Add request $\boldsymbol{i}$ to $\boldsymbol{A}$

    Delete all requests in $\boldsymbol{R}$ not compatible with request $\boldsymbol{i}$

return $\boldsymbol{A}$

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

Interval Scheduling: Analysis

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

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

Name the finish times of requests in A as $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_t$ in order.

Claim: Let $O \subseteq R$ be a set of compatible requests whose finish times in order are $\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_s$. Then for every integer $k \geq 1$ we have:

a) if O contains a k^{th} request then A does too, and

b) $\mathbf{a}_k \leq \mathbf{o}_k$ "A is ahead of O"

Note that a) alone implies that $t \geq s$ which means that A is optimal but we also need b) "stays ahead" to keep the induction going.

Inductive Proof of Claim

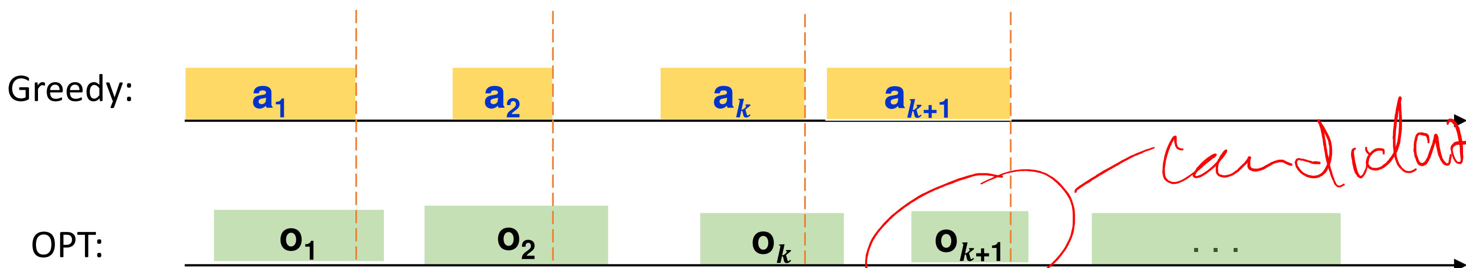
Base Case $k = 1$**:** A includes the request with smallest finish time, so if O is not empty then $\mathbf{a}_1 \le \mathbf{o}_1$

Inductive Step: Suppose that $\mathbf{a}_k \le \mathbf{o}_k$ and there is a k+1st request in O.

Then k+1st request in O is compatible with $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_k$ since $\mathbf{a}_k \le \mathbf{o}_k$ and $\mathbf{o}_k \le$ start time of k+1st request in O whose finish time is $\mathbf{o}_{k+1}$

⇒ There is a k+1st request in A whose finish time is named $\mathbf{a}_{k+1}$.

Also, since A would have considered both requests and chosen the one with the earlier finish time, $\mathbf{a}_{k+1} \le \mathbf{o}_{k+1}$.



candidate for k+1st request

■

Interval Scheduling: Greedy Algorithm Implementation

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

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

$O(n \log n)$

checks compatibility

$O(n)$

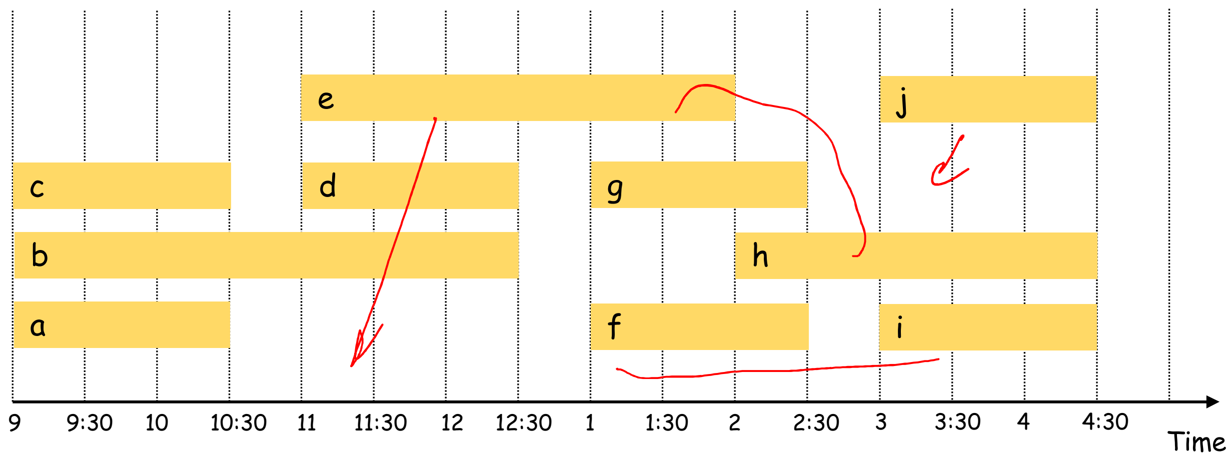
Scheduling All Intervals: Interval Partitioning

Interval Partitioning:

- Lecture j starts at s_j and finishes at f_j.

Goal: find minimum number of rooms to schedule all lectures so that no two occur at the same time in the same room.

Example: This schedule uses 4 rooms to schedule 10 lectures.



Can you do better?

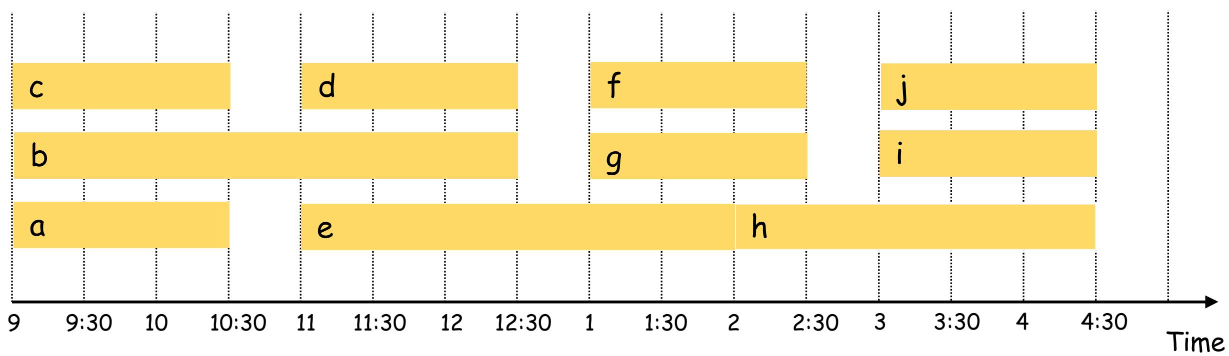
Scheduling All Intervals: Interval Partitioning

Interval Partitioning:

- Lecture j starts at s_j and finishes at f_j.

Goal: find minimum number of rooms to schedule all lectures so that no two occur at the same time in the same room.

Example: This schedule uses only 3 rooms.

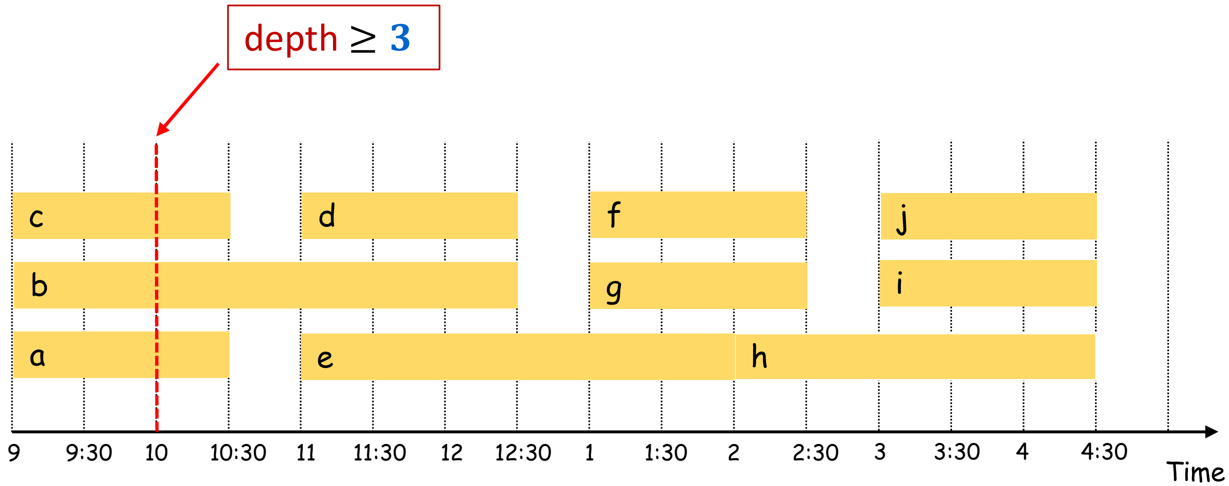


Scheduling All Intervals: Interval Partitioning

Defn: The depth of a set of open intervals is the maximum number that contain any given time.

Key observation: # of rooms needed $\geq$ depth.

Example: This schedule uses only $\mathbf{3}$ rooms. Since depth $\geq \mathbf{3}$ this is optimal.



A simple greedy algorithm

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

```
last_1 ← 0  // finish time of last request currently scheduled in room 1
for i ← 1 to n {
  j ← 1
  while (request i not scheduled) {
        if s_i ≥ last_j then
              schedule request i in room j
              last_j ← f_i
        j ← j + 1
        if last_j undefined then last_j ← 0
  }
}
```

Look for the first room where the request will fit, opening a new room if all the others used so far are full.

Interval Partitioning: Greedy Analysis

Observation: Greedy algorithm never schedules two incompatible lectures in the same room

- Only schedules request i in room j if $s_i \geq last_j$

Theorem: Greedy algorithm is optimal.

Proof:

Let d = number of rooms that the greedy algorithm allocates.

- Room d is allocated because we needed to schedule a request, say j, that is incompatible with some request in each of the other $d - 1$ rooms.
- Since we sorted by start time, these incompatibilities are caused by requests that start no later than s_j and finish after s_j.

So… we have d requests overlapping at time $s_j + \varepsilon$ for some tiny $\varepsilon > 0$.

Key observation $\Rightarrow$ all schedules use $\geq d$ rooms. ∎

A simple greedy algorithm

Runtime analysis

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

$last_1 \leftarrow 0$ // finish time of last request currently scheduled in room 1

for $i \leftarrow 1$ to n {

  $j \leftarrow 1$

  while (request i not scheduled) {

    if $s_i \geq last_j$ then

      schedule request i in room j

      $last_j \leftarrow f_i$

    $j \leftarrow j + 1$

    if $last_j$ undefined then $last_j \leftarrow 0$

  }

}

Might need to try all d rooms to schedule a request

$O(n\, d)$

d might be as big as n

Worst case $\Theta(n^2)$

A more efficient implementation: Priority queue

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 in room 1
$last_1 \leftarrow f_1$
insert 1 into priority queue Q with key = $last_1$
for $i \leftarrow 2$ to n {
  $j \leftarrow$ findmin(Q) ← $O(\log d)$ $O(1)$
  if $s_i \geq last_j$ then {
    schedule request i in room j
    $last_j \leftarrow f_i$
    increasekey(j,Q) to $last_j$ } ← $O(\log d)$ $O(\log d)$
  else {
    $d \leftarrow d + 1$
    schedule request i in room d
    $last_d \leftarrow f_i$
    insert d into priority queue Q with key = $last_d$ } $O(\log d)$
}

$O(n \log d)$

$\Theta(n \log n)$ total

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

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.

Exchange argument: Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.

Scheduling to Minimize Lateness

Scheduling to minimize lateness:

- Single resource as in interval scheduling but, instead of start and finish times, request $\boldsymbol{i}$ has
  - Time requirement $\boldsymbol{t_i}$ which must be scheduled in a contiguous block
  - Target deadline $\boldsymbol{d_i}$ by which time the request would like to be finished
- Overall start time $\boldsymbol{s}$ for all jobs

Requests are scheduled by the algorithm into time intervals $[\boldsymbol{s_i}, \boldsymbol{f_i}]$ s.t. $\boldsymbol{t_i = f_i - s_i}$

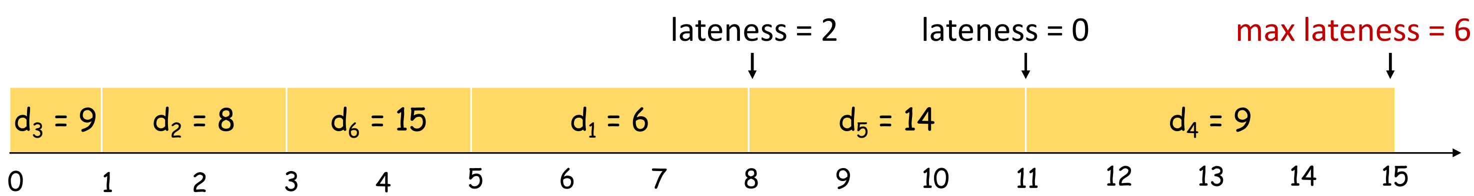
- Lateness of schedule for request $\boldsymbol{i}$ is
  - If $\boldsymbol{f_i > d_i}$ then request $\boldsymbol{i}$ is late by $\boldsymbol{L_i = f_i - d_i}$; otherwise its lateness $\boldsymbol{L_i = 0}$
- Maximum lateness $\boldsymbol{L} = \max_i \boldsymbol{Li}$

Goal: Find a schedule for **all** requests (values of $\boldsymbol{s_i}$ and $\boldsymbol{f_i}$ for each request $\boldsymbol{i}$) to minimize the maximum lateness, $\boldsymbol{L}$.

Scheduling to Minimizing Lateness

- Example:

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Minimizing Lateness: Greedy Algorithms

Greedy template: Consider jobs in some order.

[Shortest processing time first] Consider jobs in ascending order of processing time t_j.

[Earliest deadline first] Consider jobs in ascending order of deadline d_j.

[Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.

Minimizing Lateness: Greedy Algorithms

Greedy template: Consider jobs in some order.

[Shortest processing time first] Consider jobs in ascending order of processing time t_j.

	1	2
t_j	1	10
d_j	100	10

counterexample

Will schedule 1 (length 1) before 2 (length 10).
2 can only be scheduled at time 1
1 will finish at time 11 >10. Lateness 1.
Lateness 0 possible If 1 goes last.

[Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.

	1	2
t_j	1	10
d_j	2	10

counterexample

Will schedule 2 (slack 0) before 1 (slack 1).
1 can only be scheduled at time 10
1 will finish at time 11 >10. Lateness 9.
Lateness 1 possible if 1 goes first.