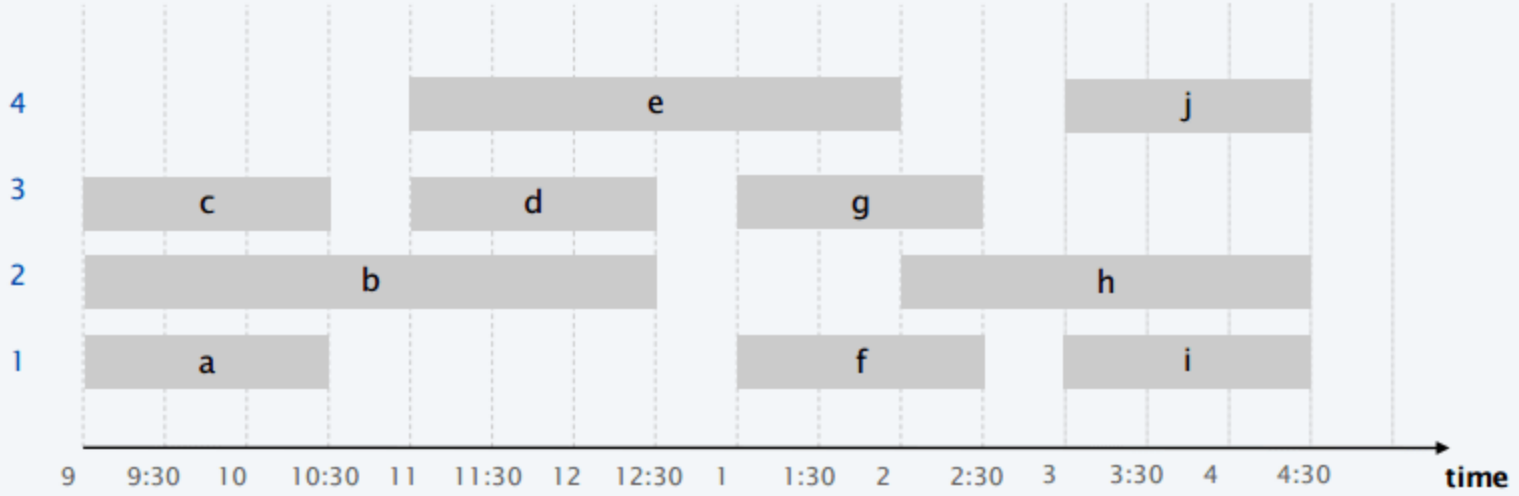
Interval partitioning

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 lectures occur at the same time in the same room.

Ex. This schedule uses 4 classrooms to schedule 10 lectures.

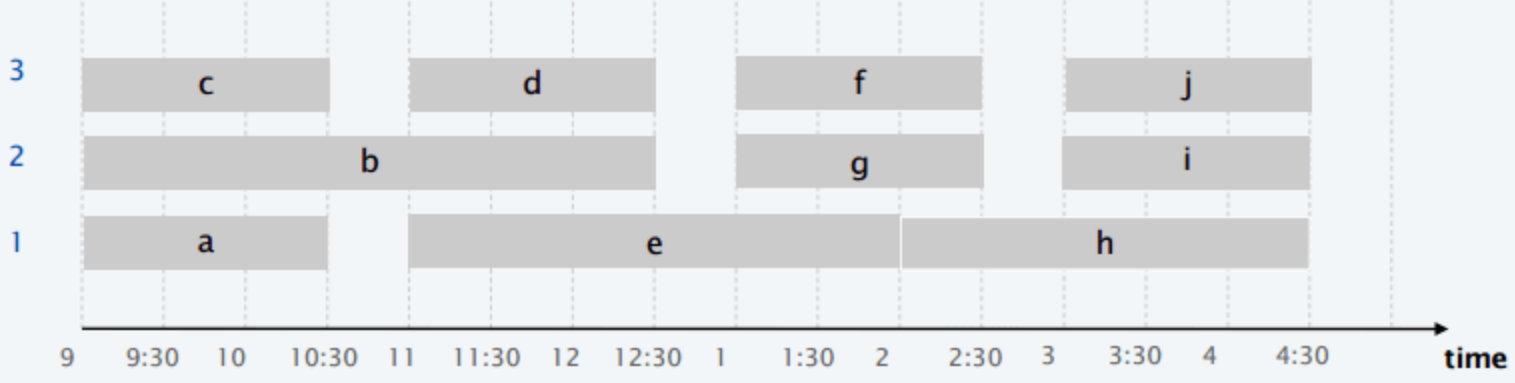


Interval partitioning

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 lectures occur at the same time in the same room.

Ex. This schedule uses 3 classrooms to schedule 10 lectures.



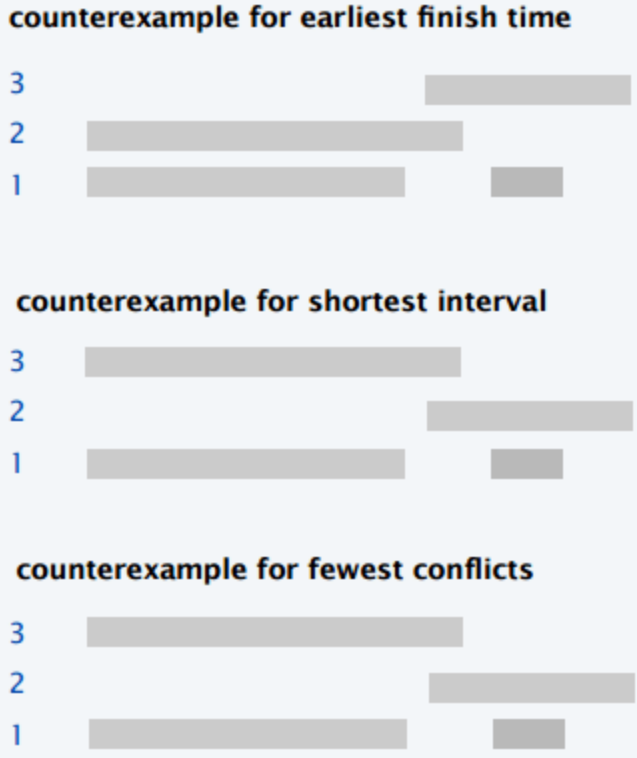
Interval partitioning: greedy algorithms

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 finish time] Consider lectures in ascending order of f_j.
- [Shortest interval] Consider lectures in ascending order of $f_j - s_j$.
- [Fewest conflicts] For each lecture j, count the number of conflicting lectures c_j. Schedule in ascending order of c_j.

Interval partitioning: greedy algorithms

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.



Interval partitioning: earliest-start-time-first algorithm

EARLIEST-START-TIME-FIRST $(n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n)$

SORT lectures by start time so that $s_1 \leq s_2 \leq \ldots \leq s_n$.

$d \leftarrow 0$ ← 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.

Interval partitioning: earliest-start-time-first algorithm

Proposition. The earliest-start-time-first algorithm can be implemented in $O(n \log n)$ time.

Pf. Store classrooms in a priority queue (key = finish time of its last lecture).

- To determine whether lecture j is compatible with some classroom, compare s_j to key of min classroom k in priority queue.
- To add lecture j to classroom k, increase key of classroom k to f_j.
- Total number of priority queue operations is $O(n)$.
- Sorting by start time takes $O(n \log n)$ time. ▪

Remark. This implementation chooses the classroom k whose finish time of its last lecture is the earliest.