Applied Algorithms

CSCI-B505 / INFO-I500

Lecture 16.

Greedy Algorithms

M. Oguzhan Kulekci

- Greedy Algorithms
  - Interval Scheduling Problem
  - Interval Partitioning Problem
  - Bin-packing Heuristics

Greedy Algorithms

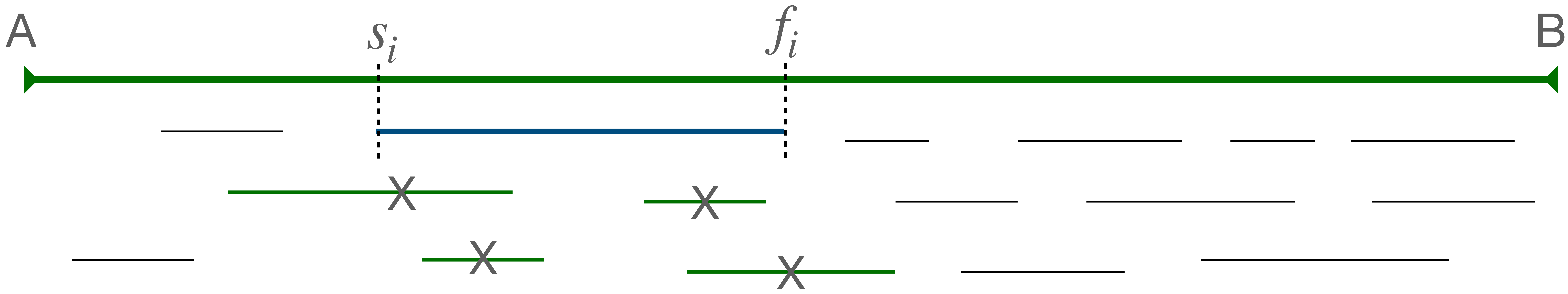
- While solving an optimization problem, we get through some steps, each of which requires a decision

- A greedy approach chooses the most appropriate decision at the moment without worrying about the future

- Greedy algorithms not always guarantee the optimum solution, but sometimes they do as well

Interval Scheduling Problem

- Given a list of n tasks, where the ith task starts at s_i and finishes at f_i, the aim is to select the largest subset of them without overlaps in time.



Select as many tasks as possible between A and B such that no two of them overlaps

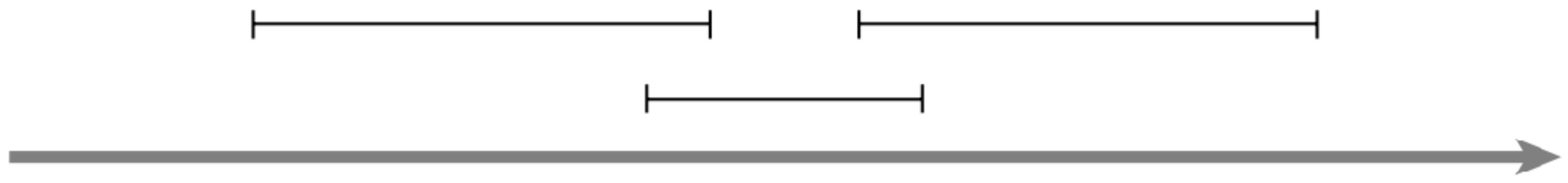
Interval Scheduling Problem

- Let's sort them according to their starting times s_i, and select accordingly.

- Problem: Some long task that starts earlier may kill many smaller items that would otherwise increase the number of fulfilled was count.
- So, it worths to give credit to task duration ?

Interval Scheduling Problem

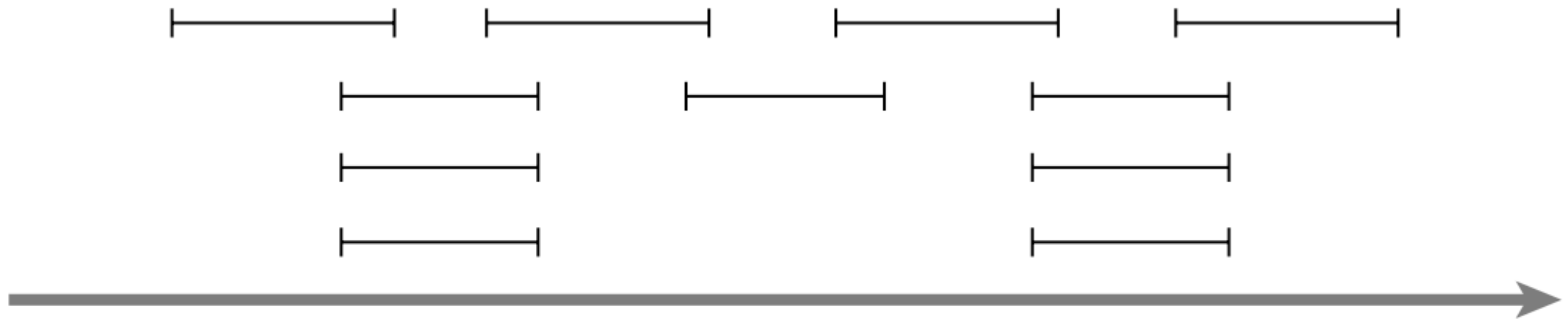
- Let's sort them according to their durations $(f_i - s_i)$, and select from shorter to longer ones.



- Problem: A short task may kill longer ones that would otherwise contribute to the number of fulfilled task count.
- So, starting earlier or being shorter seems problematic.

Interval Scheduling Problem

- How about selecting the tasks that have minimum collisions with the others?



- Problem: The optimal solution above would contain four tasks. However, the proposed heuristics selects the middle in the middle row that makes the answer three.
- Although minimal collision seems a lot better, there is a better selection strategy that guarantees the optimality !

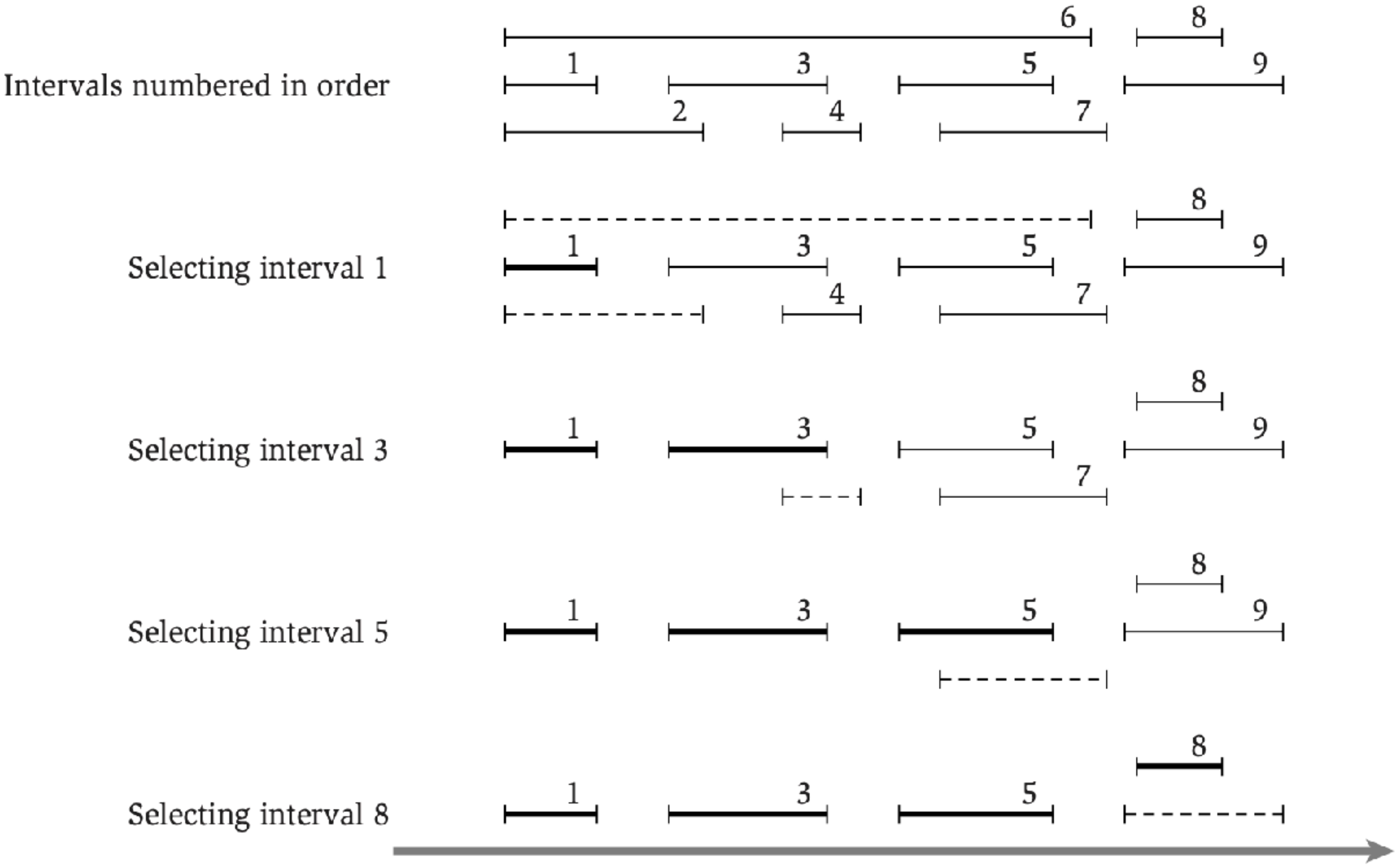
Interval Scheduling Problem

- Select the item that finishes earliest! In other words, sort the tasks according to f_i, and execute them accordingly.

```
Initially let R be the set of all requests, and let A be empty
While R is not yet empty
  Choose a request i ∈ R that has the smallest finishing time
  Add request i to A
  Delete all requests from R that are not compatible with request i
EndWhile
Return the set A as the set of accepted requests
```

- **Will that generate the optimal solution ?**

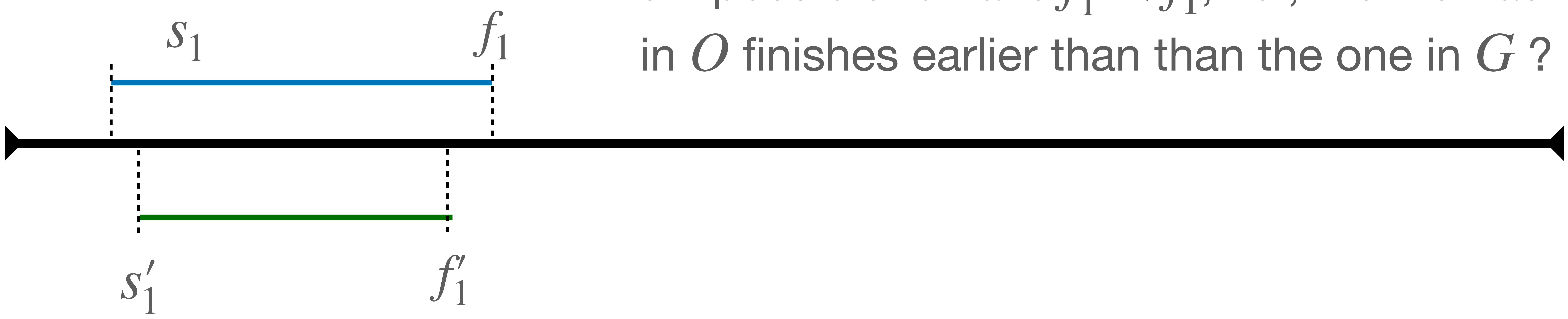
Interval Scheduling Problem



- Will that generate the optimal solution ?

Interval Scheduling Problem

- Optimal solution $O = \langle (s'_1, f'_1), (s'_2, f'_2), \ldots (s'_m, f'_m) \rangle$
- Greedy solution $G = \langle (s_1, f_1), (s_2, f_2), \ldots, (s_k, f_k) \rangle$
- If $m = k$, then G is optimal. Notice there may be multiple optimal solutions.

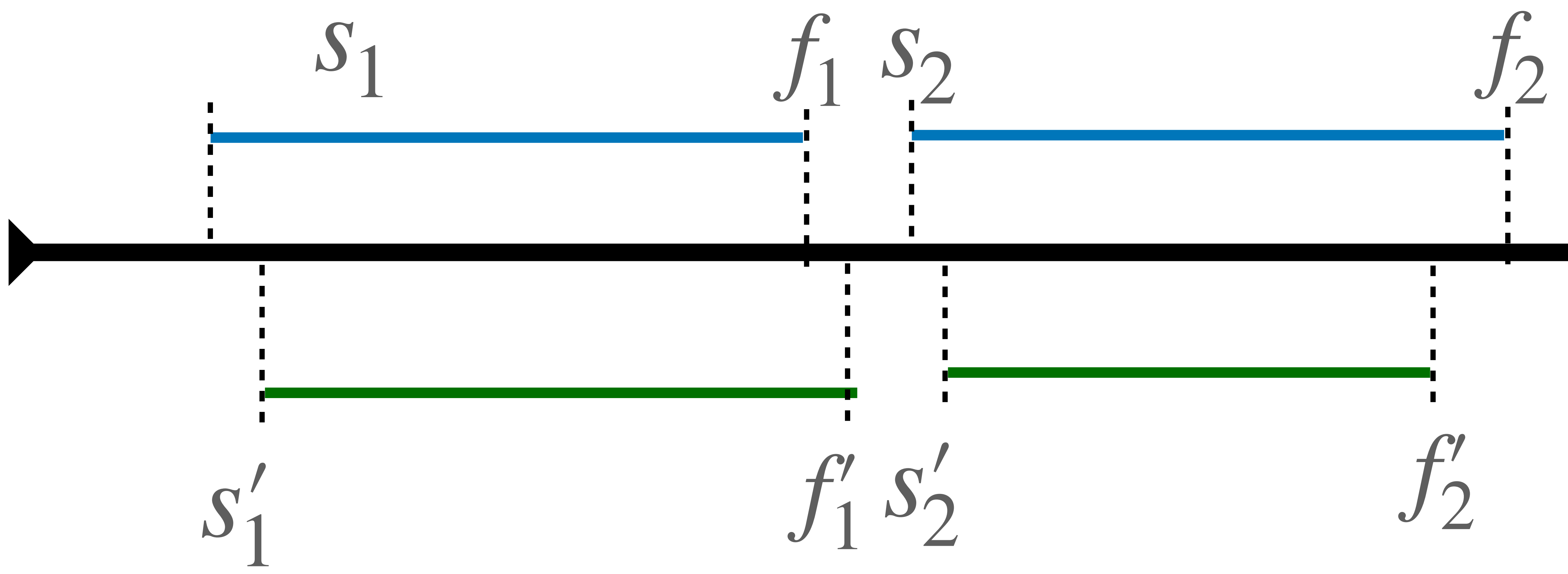


Is it possible to have $f'_1 < f_1$, i.e., the first task in O finishes earlier than than the one in G ?

No! Because the greedy algorithm chooses the one with the earliest finish time among the list always.

Interval Scheduling Problem

- Optimal solution $O = \langle (s_1', f_1'), (s_2', f_2'), \ldots (s_m', f_m') \rangle$
- Greedy solution $G = \langle (s_1, f_1), (s_2, f_2), \ldots, (s_k, f_k) \rangle$
- If $m = k$, then G is optimal. Notice there may be multiple optimal solutions.



Is it possible to have $f_2' < f_2$, i.e., the second task in O finishes earlier than than the one in G ?

- Since $s_2' > f_1' > f_1$, the task (s_2', f_2') **cannot conflict** with the first task (s_1, f_1).
- Therefore, if $f_2' < f_2$, then it should have been selected by the G.
- So, $f_2' < f_2$ is also **impossible**.
- And this is the case for all $i = 1,2,3,\ldots, k$ (induction).

Interval Scheduling Problem

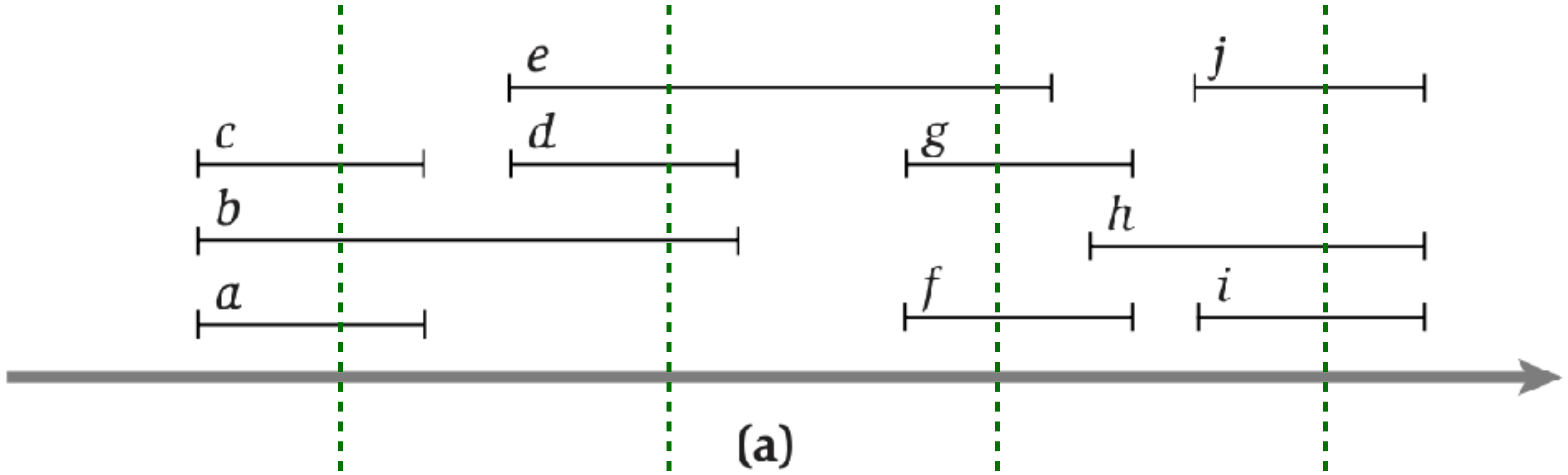
- Optimal solution $O = \langle (s'_1, f'_1), (s'_2, f'_2), \ldots (s'_m, f'_m) \rangle$
- Greedy solution $G = \langle (s_1, f_1), (s_2, f_2), \ldots, (s_k, f_k) \rangle$
- If $m = k$, then G is optimal. Notice there may be multiple optimal solutions.

Assume $m > k$, which means there is s'_{k+1}, f'_{k+1}

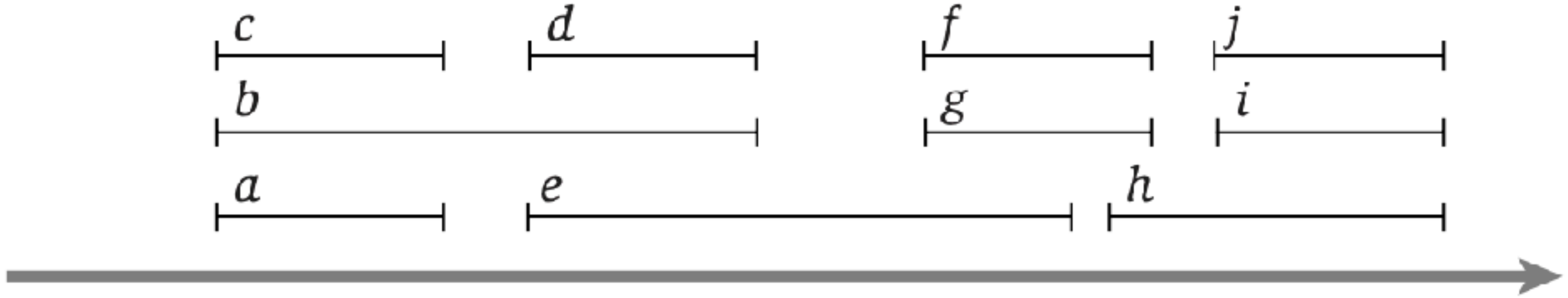
- Since $s'_{k+1} > f'_k > f_k$, the task (s'_{k+1}, f'_{k+1}) has no conflict with the task (s_k, f_k).
- Then, the task (s'_{k+1}, f'_{k+1}) should not have been eliminated from the list
- This means the greedy algorithm cannot stop at such a k value.
- This proves $(k \geq m)$, and since m is optimal, $k = m$!
- **Greedy approach is optimal…**

Interval PARTITIONING Problem

- Now, assume multiple processors are available.
- We would like to finish all tasks with **MINIMUM** number of processors.
- Given a list of n tasks, where the ith task starts at s_i and finishes at f_i, the aim is to select the largest subset of them without overlaps in time.



(a)



(b)

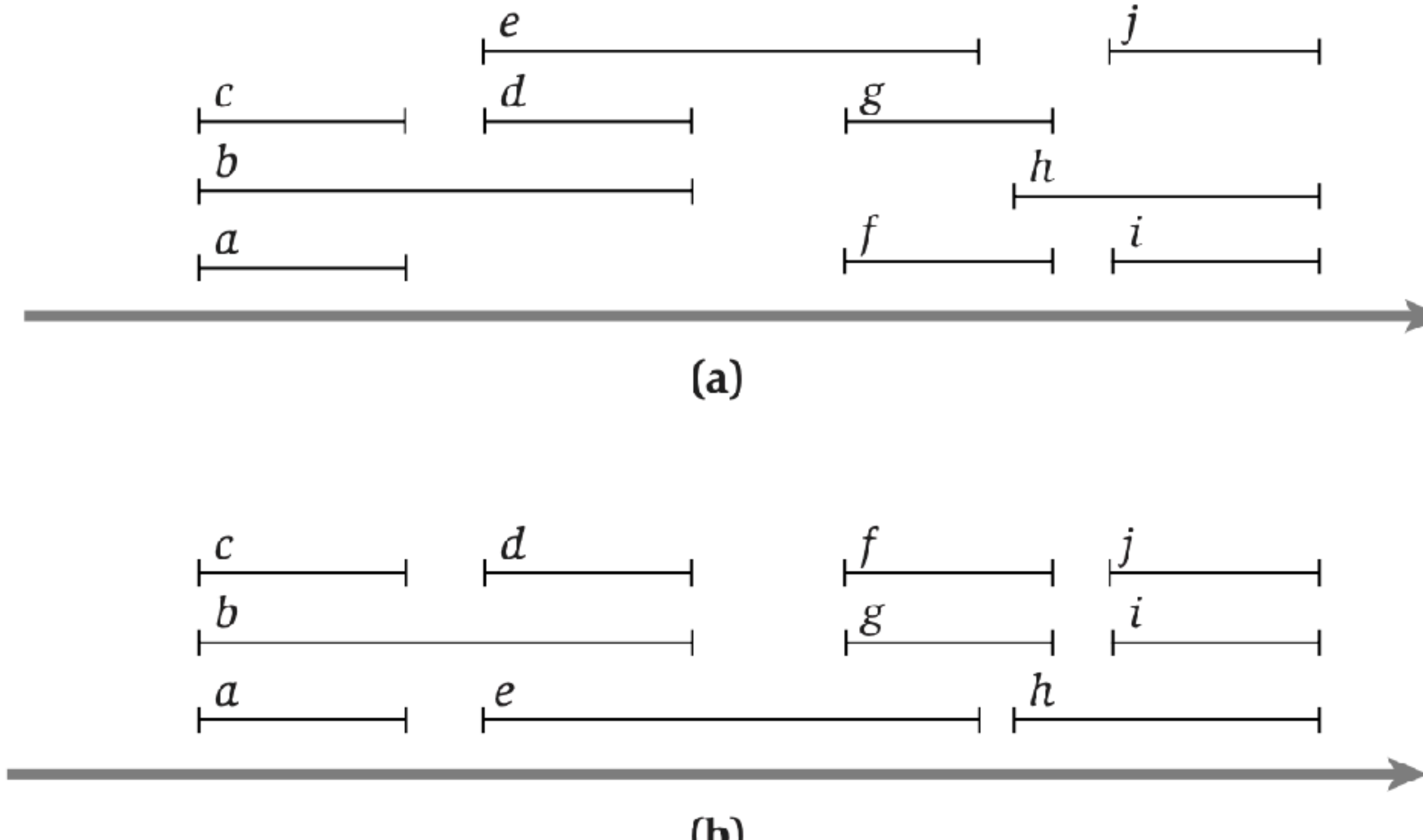
- What is the least number of processors we will need ?
- The maximum number of conflicting tasks at a moment during the whole interval!
- Call this number *depth.*

Interval PARTITIONING Problem

- Sort all tasks according to their start times.
- For each task, assign the first label that has not been assigned to previous conflicting tasks.

```
Sort the intervals by their start times, breaking ties arbitrarily
Let I_1, I_2, ..., I_n denote the intervals in this order
For j = 1, 2, 3, ..., n
  For each interval I_i that precedes I_j in sorted order and overlaps it
     Exclude the label of I_i from consideration for I_j
  Endfor
  If there is any label from {1, 2, ..., d} that has not been excluded then
    Assign a nonexcluded label to I_j
  Else
    Leave I_j unlabeled
  Endif
Endfor
```

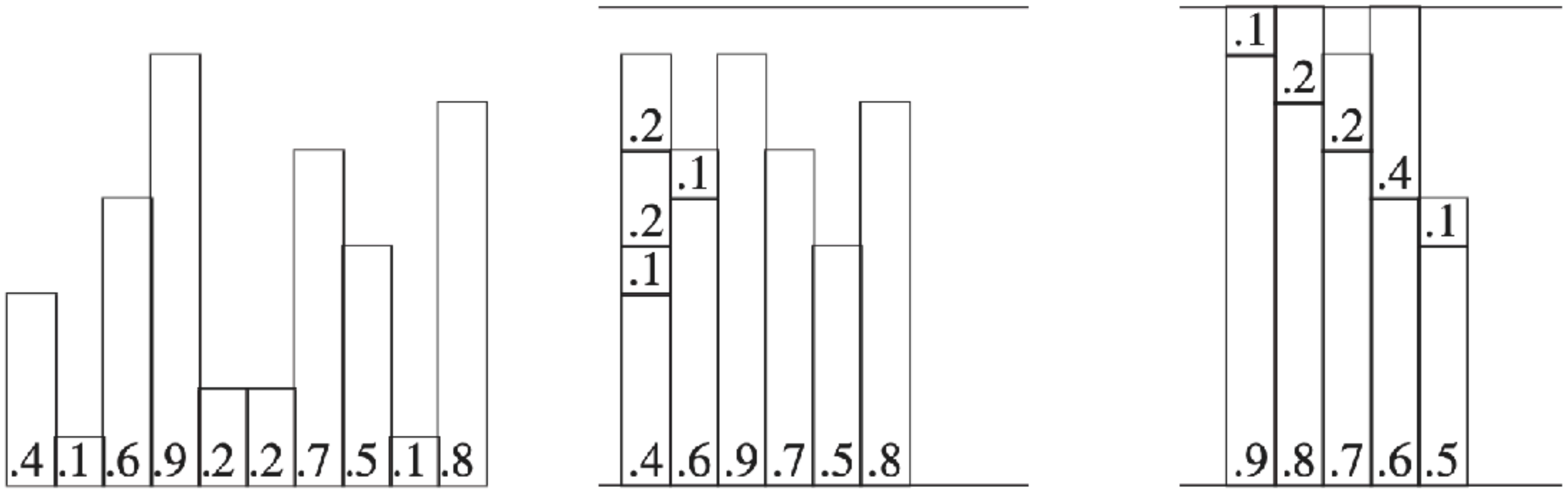
- Do we ever need labels more than the depth number?
- No! Since depth is the maximum conflicting items at a time.
- Thus, the greedy approach is optimal !



(a)

(b)

Greedy Algorithms



(a) First fit (b) Optimal

Figure 3.11. Bin packing. A list of 10 weights in the range [0, 1]. In (a) the weights are packed by the first fit algorithm, which uses 6 bins. In (b) the weights are packed optimally in 5 bins.

A.Guide.to.Experimental.Algorithmic.Catherine.C.McGeoch..2012

- Greedy algorithms cannot always produce the optimal!
- But, maybe preferred for good heuristics

Greedy Algorithms

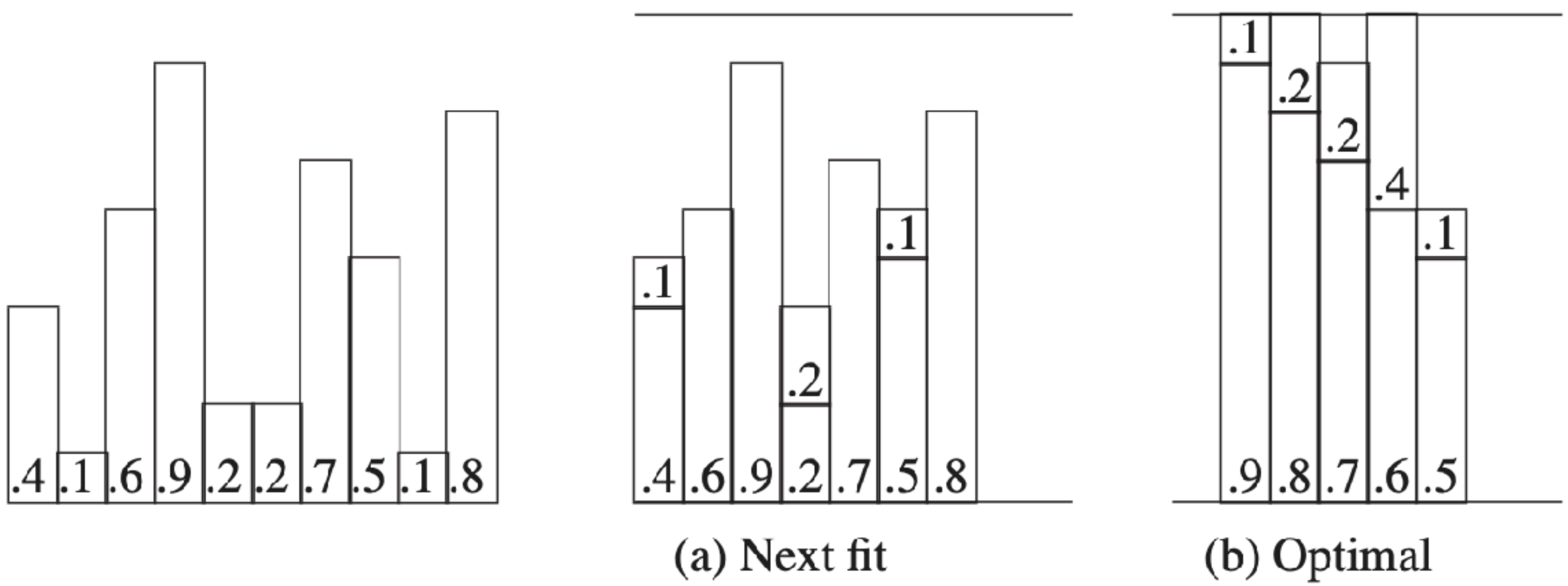


Figure 4.1. Bin packing. Two packings of the same list of 10 weights. The next fit packing uses seven bins, and the optimal packing uses five bins.

A.Guide.to.Experimental.Algorithmic.Catherine.C.McGeoch..2012

- If the sequence was in the following order, the next fit produce the optimal:
  .9, .1, .8, .2, .7, .2, .6, .4, .5, .1
- So, the issue is to find the correct permutation of the input
- Yes, it is factorial in time. However, the solution improves during the search and we can stop when we find a good-enough solution.

- Greedy algorithms cannot always produce the optimal!
- They can even be helpful in the brute force search of the optimum.

Reading assignment

- Read the chapter 4.1 and 4.2 from Kleinberg&Tardos, chapter 16.1 and 16.2 from Cormen.