CS 344: Design and Analysis of Computer Algorithms

(Spring 2022 — Sections 5,6,7,8)

Lecture 13: Greedy Algorithms: Job Scheduling, Disjoint Intervals

Summary of Greedy Algorithms

Greedy Algorithms

- Greedy algorithms allow us to bypass examining all options
- Every time you do greedy algorithms:
  - Start with building intuition why there are some options that can be ignored entirely
  - Design your algorithm based on this “greedy” observation
  - Prove correctness of the algorithm: Exchange argument
  - Analyze runtime of your algorithm
- Not every problem admits a greedy algorithm: do not force it!

The Job Scheduling Problem

The Job Scheduling Problem

- **Input:**
  - A collection of n computing jobs each with a length $L[i]$
  - A single processor that can compute job i in $L[i]$ time
- **Output:**
  - On ordering π of the jobs with minimal total delay
  - $delay(\pi) = \sum_{i=1}^{n} \sum_{j=1}^{i} L[\pi(j)]$

The Job Scheduling Problem: Example

- **Input:**

6	1	2	3	2	1	1	4

- A possible ordering $\pi = (1,2,3,4,5,6,7,8)$
  - i.e., the same ordering as the input
- Job 1 has to wait 6 unit
- Job 2 has to wait 7 unit

6	1	2	3	2	1	1	4

- Job 3 has to wait 9 unit
- …
- Total delay is 6+7+9+12+14+15+16+20 = 99 units

The Job Scheduling Problem: Example

- **Input:**

6	1	2	3	2	1	1	4

- Another possible ordering $\pi = (3,2,1,7,5,6,4,8)$

2	1	6	1	2	1	3	4

The Job Scheduling Problem: Example

- **Input:**

6	1	2	3	2	1	1	4

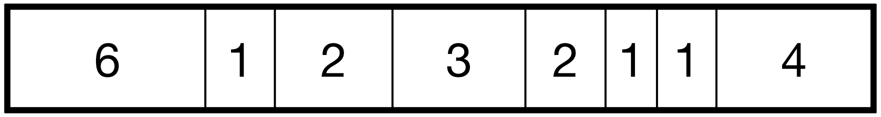
- Another possible ordering $\pi = (3,2,1,7,5,6,4,8)$
  - Job 3 has to wait 2 unit
  - Job 2 has to wait 3 unit
  - Job 1 has to wait 9 unit

2	1	6	1	2	1	3	4

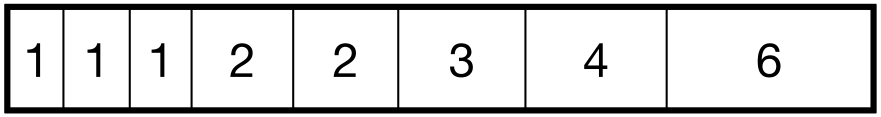
  - …
  - Total delay is 2+3+9+10+12+13+16+20 = 85 units

The Job Scheduling Problem: Example

- **Input:**



- Yet another possible ordering $\pi = (2,6,7,5,3,4,8,1)$



The Job Scheduling Problem: Example

- **Input:**

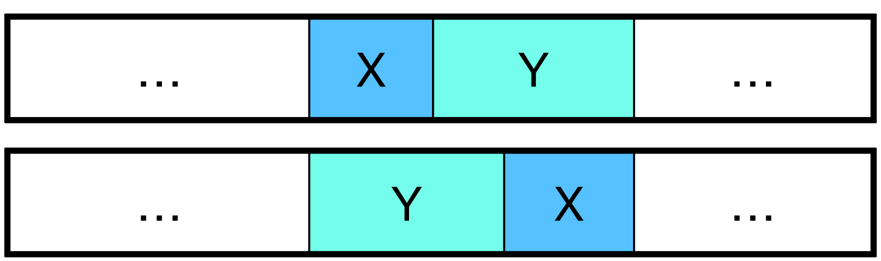
6	1	2	3	2	1	1	4

- Yet another possible ordering $\pi = (2,6,7,5,3,4,8,1)$
  - Job 2 has to wait 1 unit
  - Job 6 has to wait 2 unit
  - Job 7 has to wait 3 unit
  - ...
  - Total delay is 1+2+3+5+7+10+14+20 = 62 units

1	1	1	2	2	3	4	6

Greedy Algorithm?

- A “Greedy” Observation:
  - Suppose we have two jobs next to each other in the output ordering π
  - If we flip the order of these two jobs, the delay for remaining jobs does not change
  - What happens to delays of these two jobs?



Greedy Algorithm

- Sort the jobs in increasing (non-increasing) order of their length
- Output the resulting ordering as π
  - $\pi(i)$ is the position of i-th smallest element in the original array

Proof of Correctness

Greedy Algorithm

- Sort the jobs in increasing (non-increasing) order of their length
- Output the resulting ordering as π
  - $\pi(i)$ is the position of i-th smallest element in the original array

- Proof of Correctness? Done already
- Runtime? Only involves sorting so $O(n \log n)$ time using, say, merge sort

The Maximum Disjoint Intervals Problem

Maximum Disjoint Intervals Problem

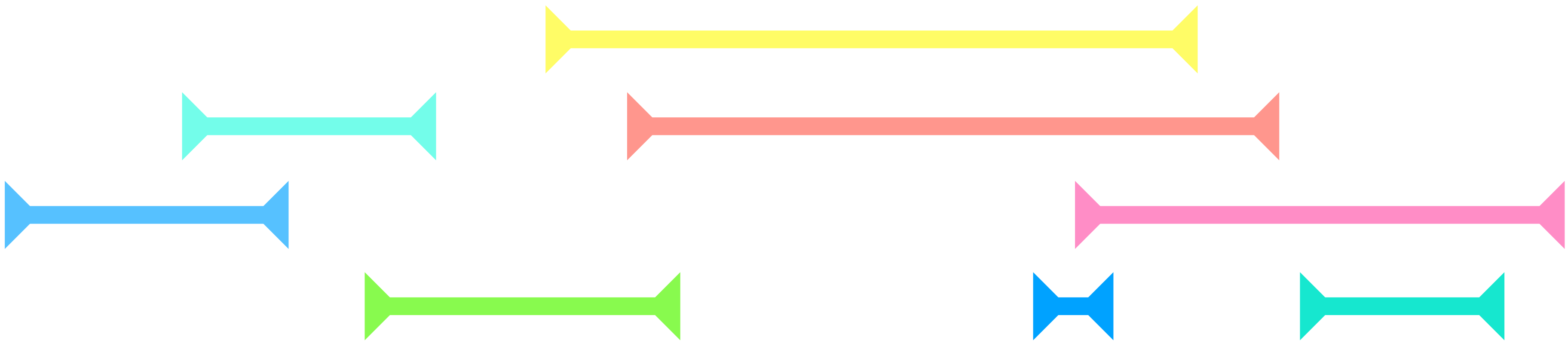
- **Input:**
  - A collection of n intervals specified by their endpoints as $[a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]$
  - Two intervals are disjoint if they do not share any points
    - E.g. [1,3] [4,5] are disjoint but [1,3] and [2,4] are not.
- **Output:**
  - Maximum number of intervals that are all disjoint from each other

Maximum Disjoint Intervals Problem

- **Example:**

Maximum Disjoint Intervals Problem

- **Example:**



Maximum Disjoint Intervals Problem

- **Example:**

Maximum Disjoint Intervals Problem

- **Example:**

- An optimal solution

Maximum Disjoint Intervals Problem

- **Example:**

- A non-optimal solution

Maximum Disjoint Intervals Problem

- **Example:**

- A wrong solution

Greedy Algorithm?

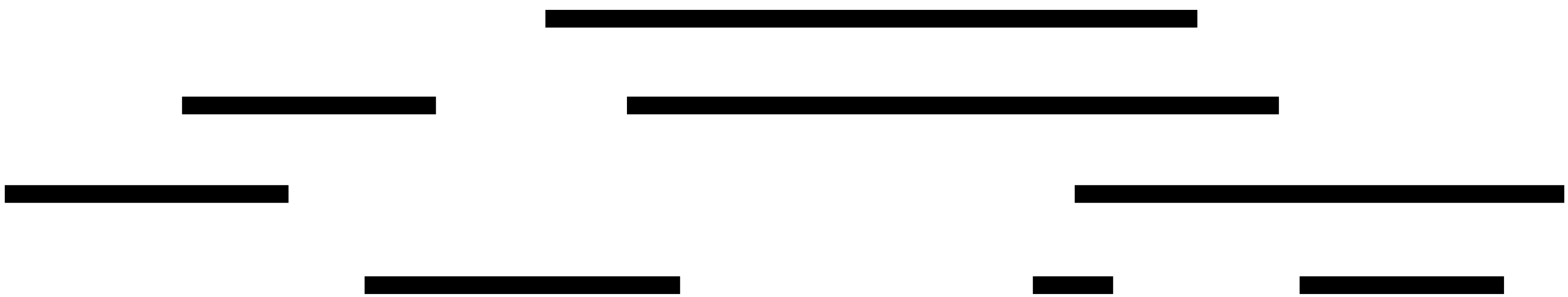
- A “Greedy” Observation:
  - The interval that finishes first can always be part of the solution
  - Why?

Greedy Algorithm

- The input is A[1],…,A[n]:
  - A[i].first gives the start point and A[i].second gives the finish
- Sort the intervals in A based on A[i].second in increasing order
- Pick A[1] in the solution and let p=1.
- For i=2 to n:
  - If A[i].first < A[p].second continue;
  - otherwise, add A[i] to the solution and update p = i.

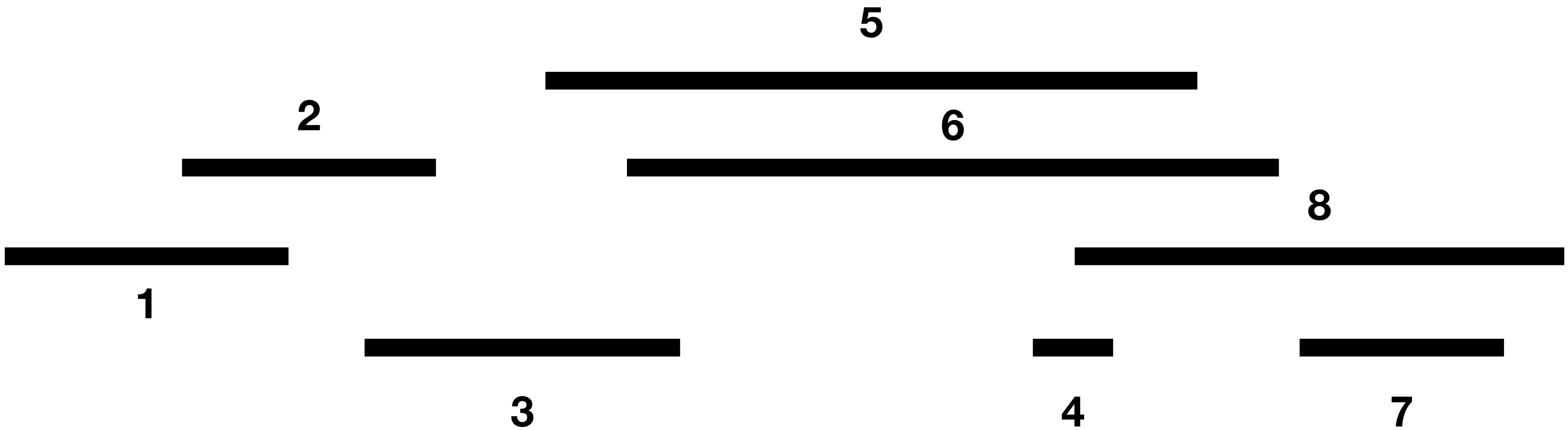
Example

- **Example:**



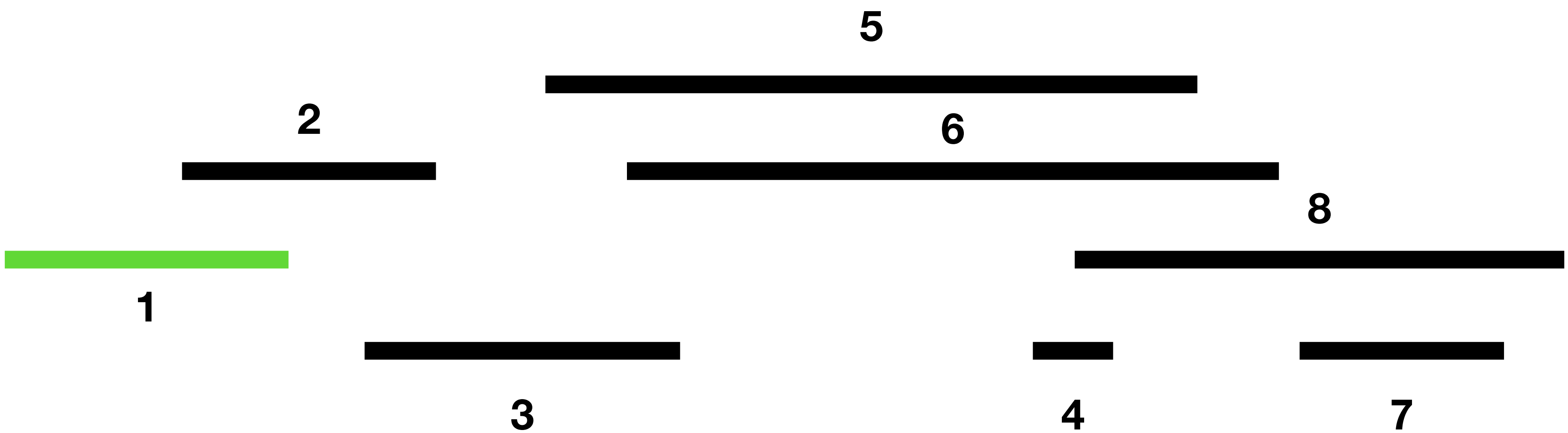
Example

- **Example:**



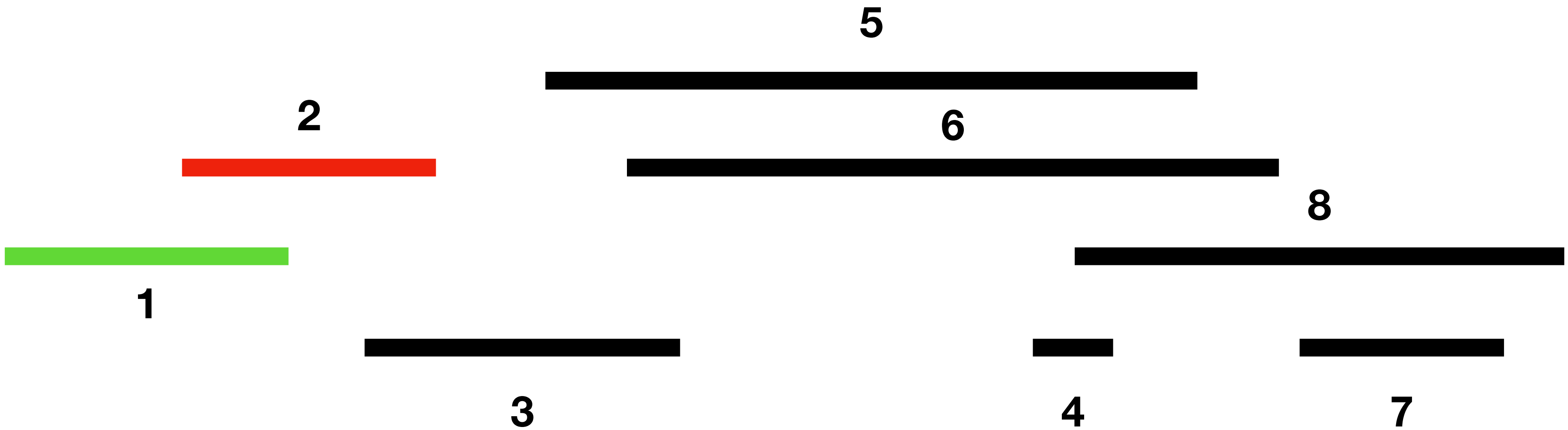
Example

- **Example:**



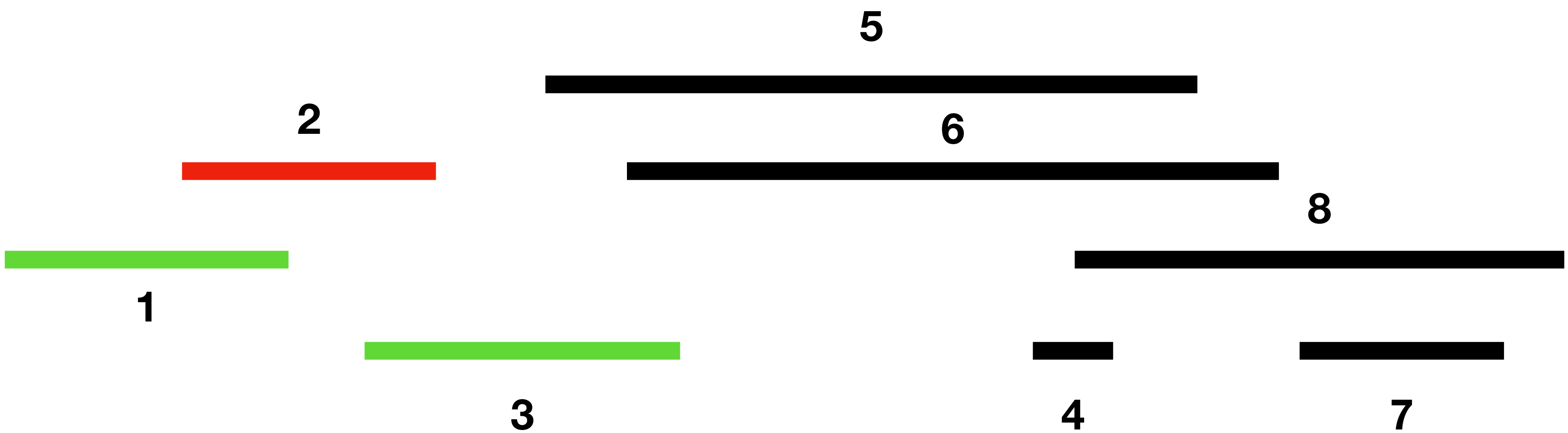
Example

- **Example:**



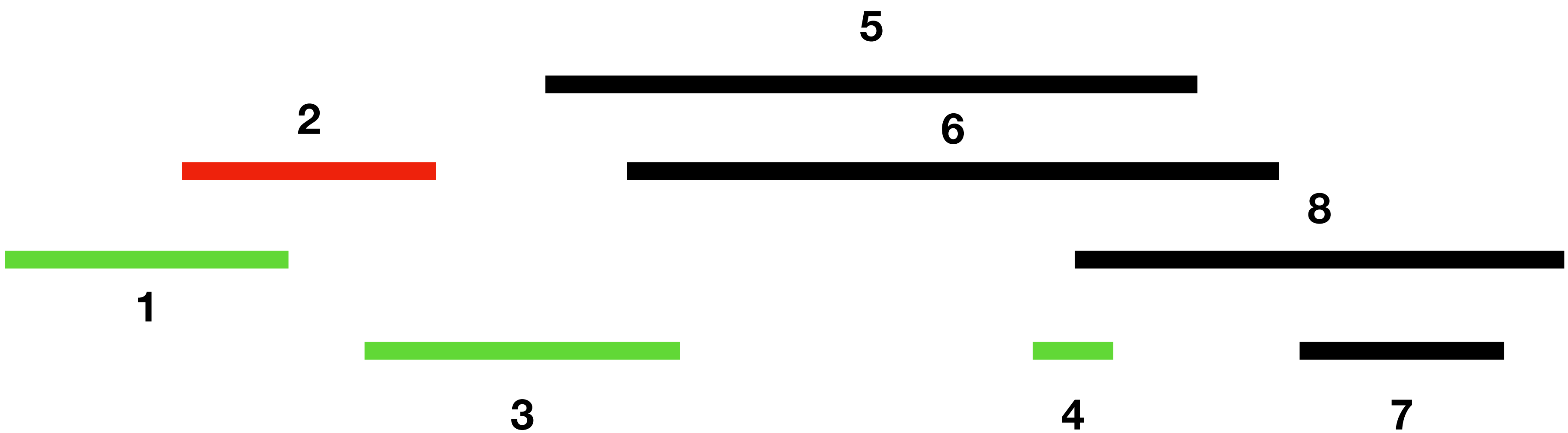
Example

- **Example:**



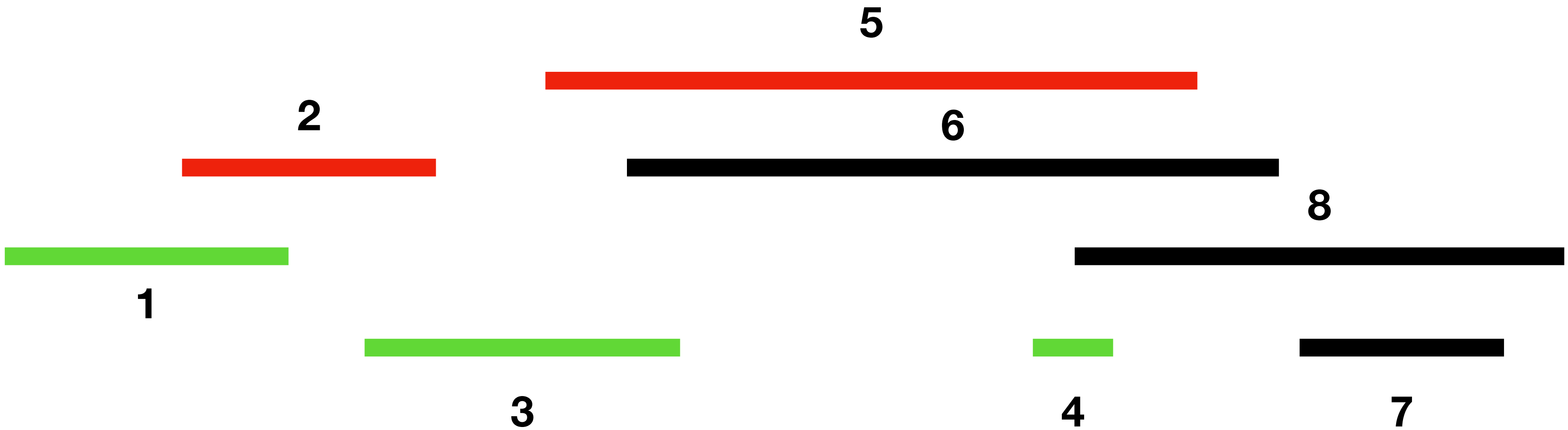
Example

- **Example:**



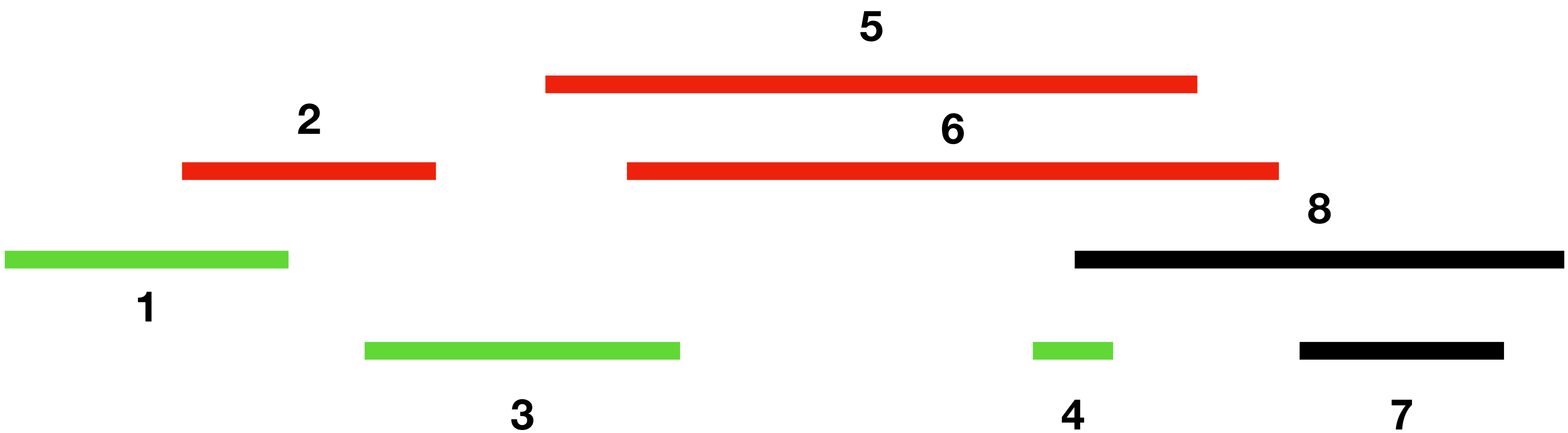
Example

- **Example:**



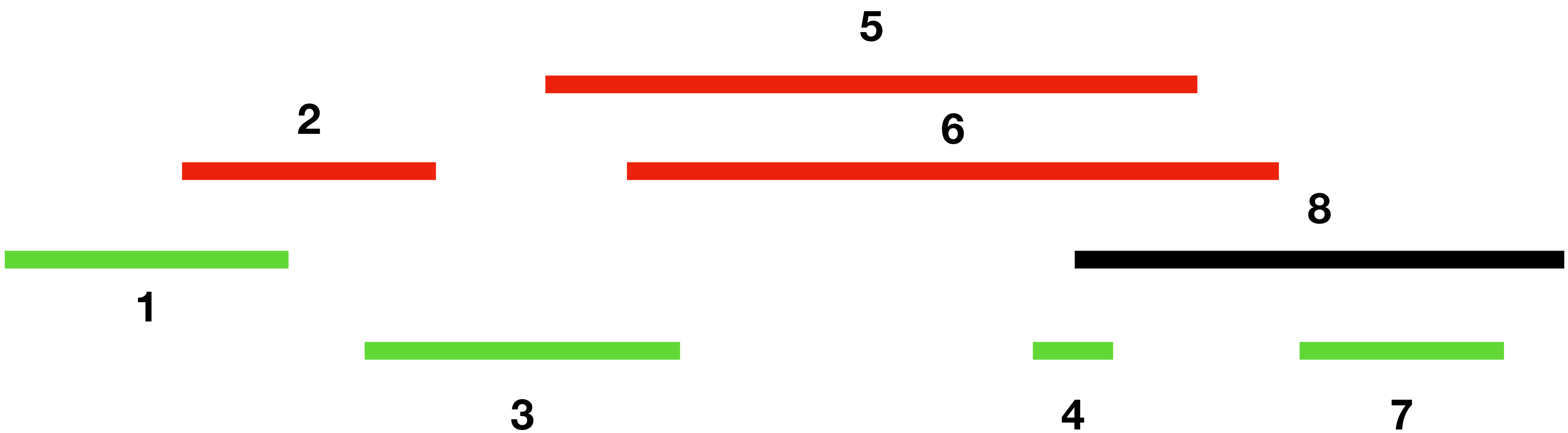
Example

- **Example:**



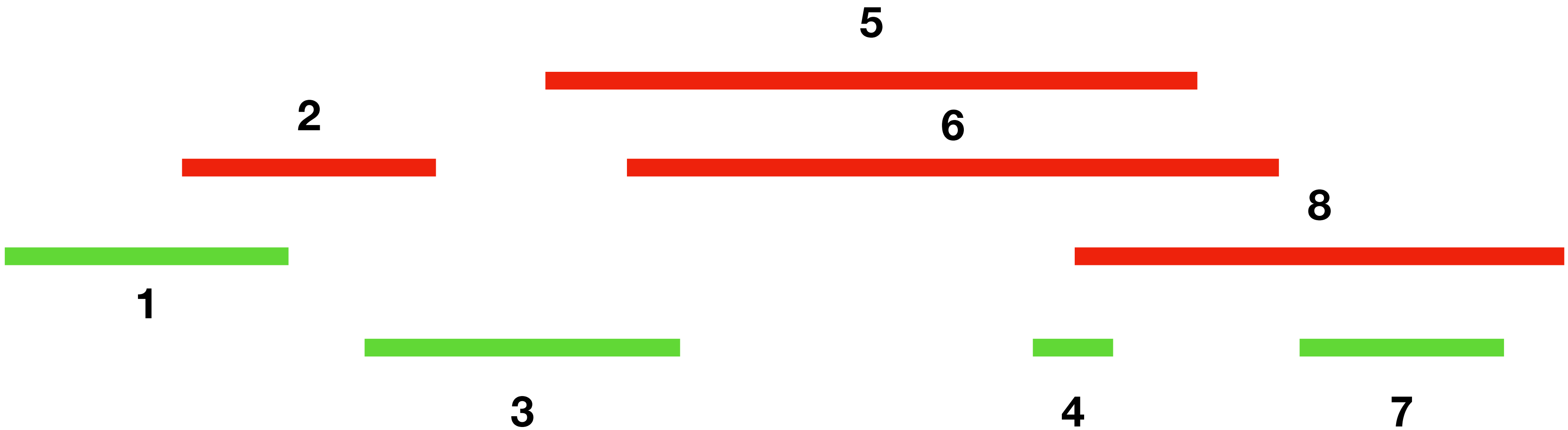
Example

- **Example:**



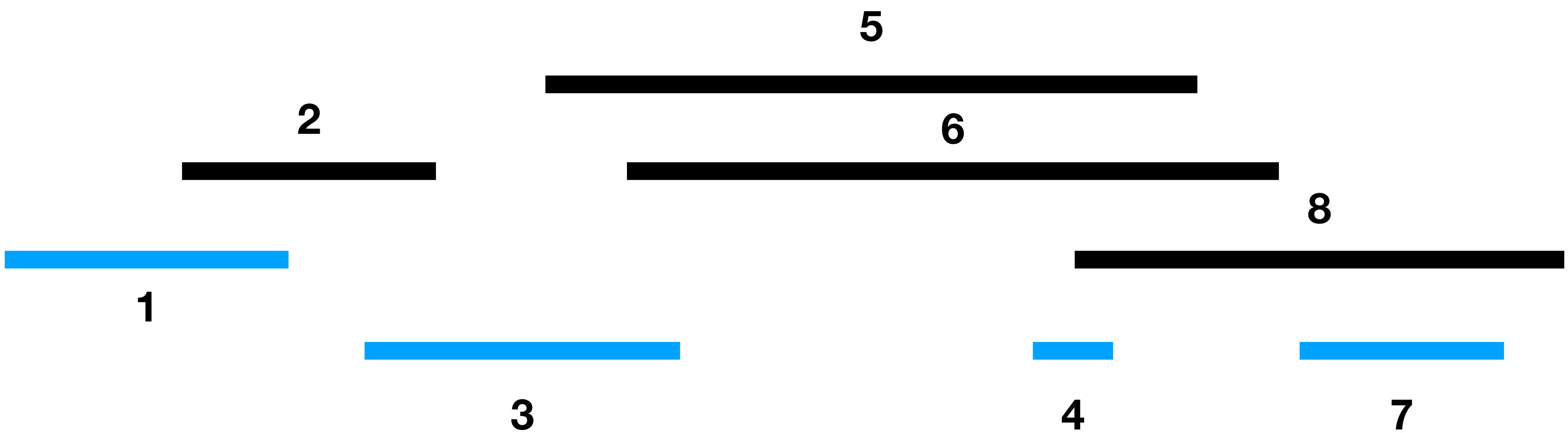
Example

- **Example:**



Example

- **Example:**



Proof of Correctness

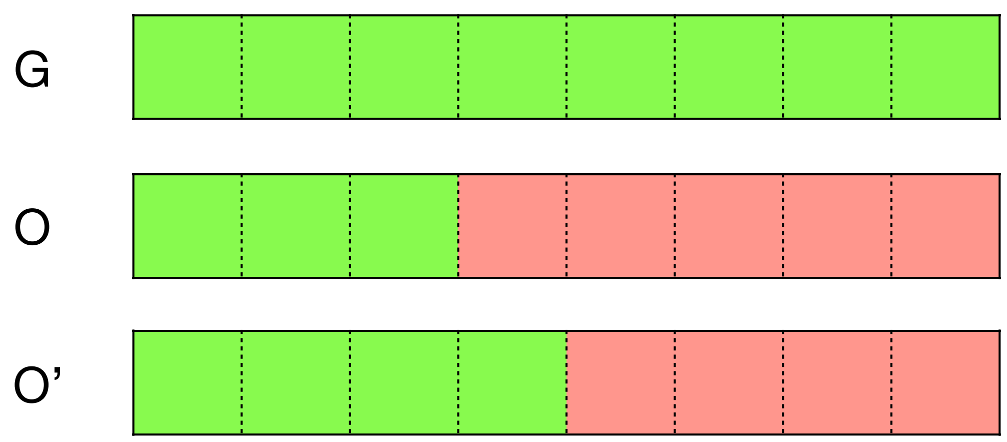
- Let $G = (g_1, g_2, \ldots, g_k)$ be the intervals output by greedy
- First, is G even a valid solution? (i.e., intervals are disjoint)

Proof of Correctness

- Let $G = (g_1, g_2, \ldots, g_k)$ be the intervals output by greedy
- Now, is G optimal?

Proof of Correctness

- Consider the intermediate solution
  $O' = (g_1 = o_1, g_2 = o_2, g_{i-1} = o_{i-1}, g_i, o_{i+1}, \ldots, o_\ell)$



Greedy Algorithm

- The input is A[1],…,A[n]:
  - A[i].first gives the start point and A[i].second gives the finish
- Sort the intervals in A based on A[i].second in increasing order
- Pick A[1] in the solution and let $p=1$.
- For $i=2$ to n:
  - If A[i].first < A[p].second continue;
  - otherwise, add A[i] to the solution and update $p = i$.