Greedy

Sep 5: Lesson 1

Greedy

“Keep picking the best one.”

Greedy

“Keep picking the best one.”

However, the “**best one**” may not always be obvious!

Event Scheduling

Event Scheduling

You just bought a new TV recorder! You have **N** ≤ 500 000 TV shows you want to record, each starting at time $\mathbf{S_i}$ and ending at time $\mathbf{E_i}$. The TV recorder can only record one show at a time. What is the maximum number of TV shows you can record such that none of them overlap timings with one another?

Event Scheduling

You just bought a new TV recorder! You have **N** ≤ 500 000 TV shows you want to record, each starting at time $\mathbf{S_i}$ and ending at time $\mathbf{E_i}$. The TV recorder can only record one show at a time. What is the maximum number of TV shows you can record such that none of them overlap timings with one another?

Discuss!

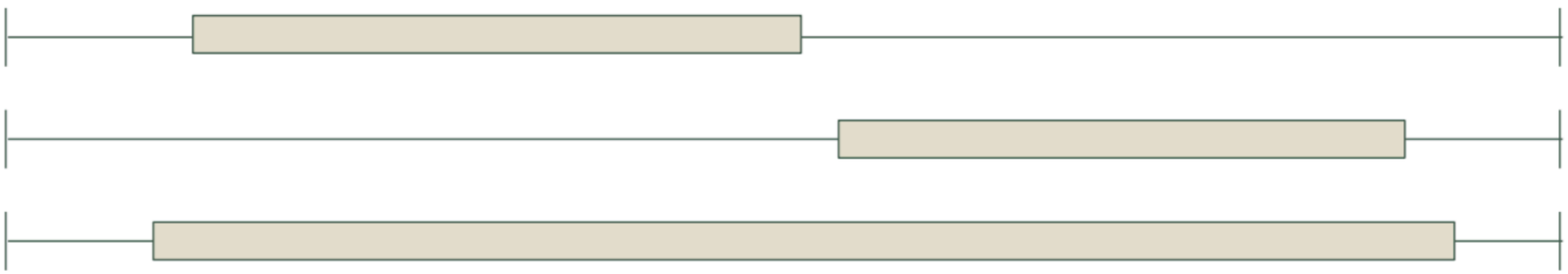
Event Scheduling

What is the greedy solution?

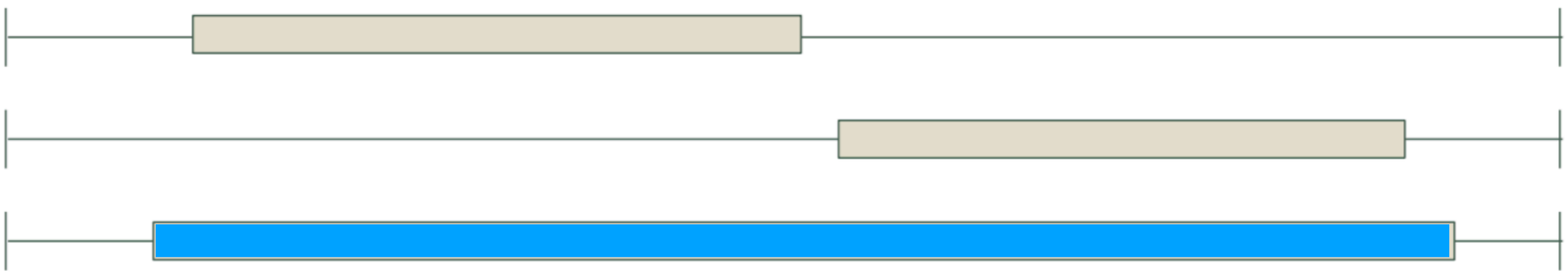
Sort by start time then end time and then process from earliest to latest start time. Each time we reach a TV show, record it if possible, else don’t record.

Does this **always work**?

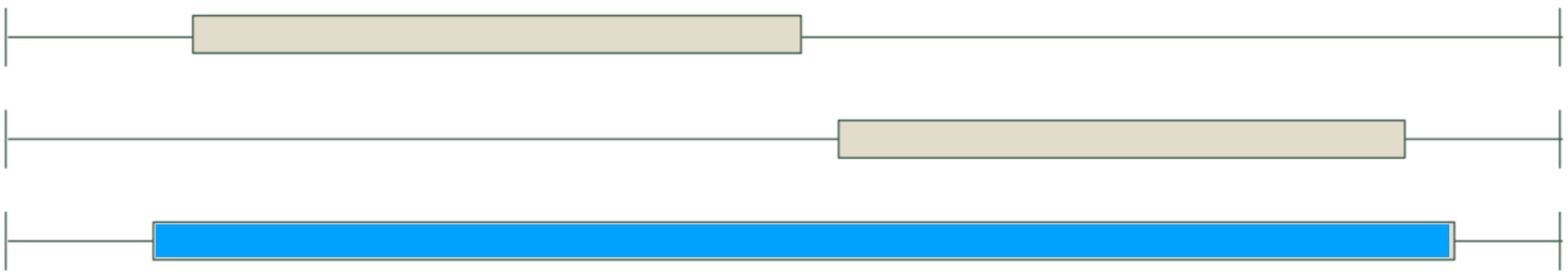
Event Scheduling



Event Scheduling



Event Scheduling



FAIL!

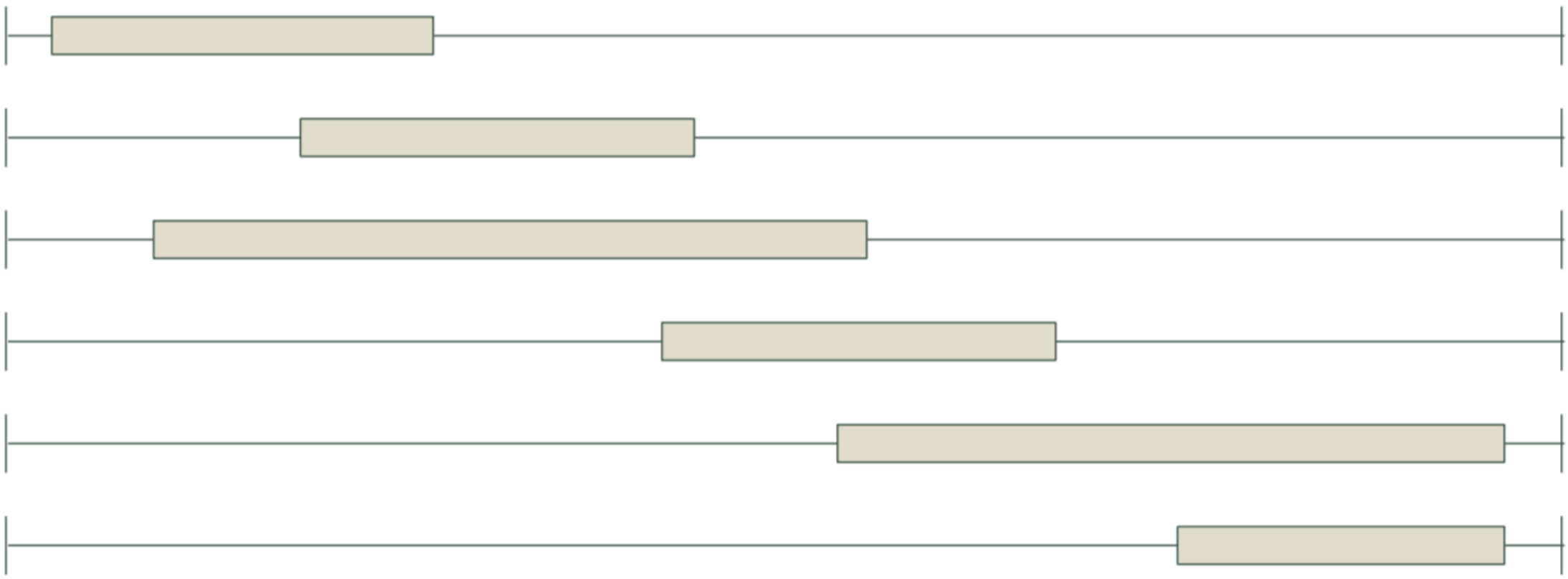
Event Scheduling

What about this?

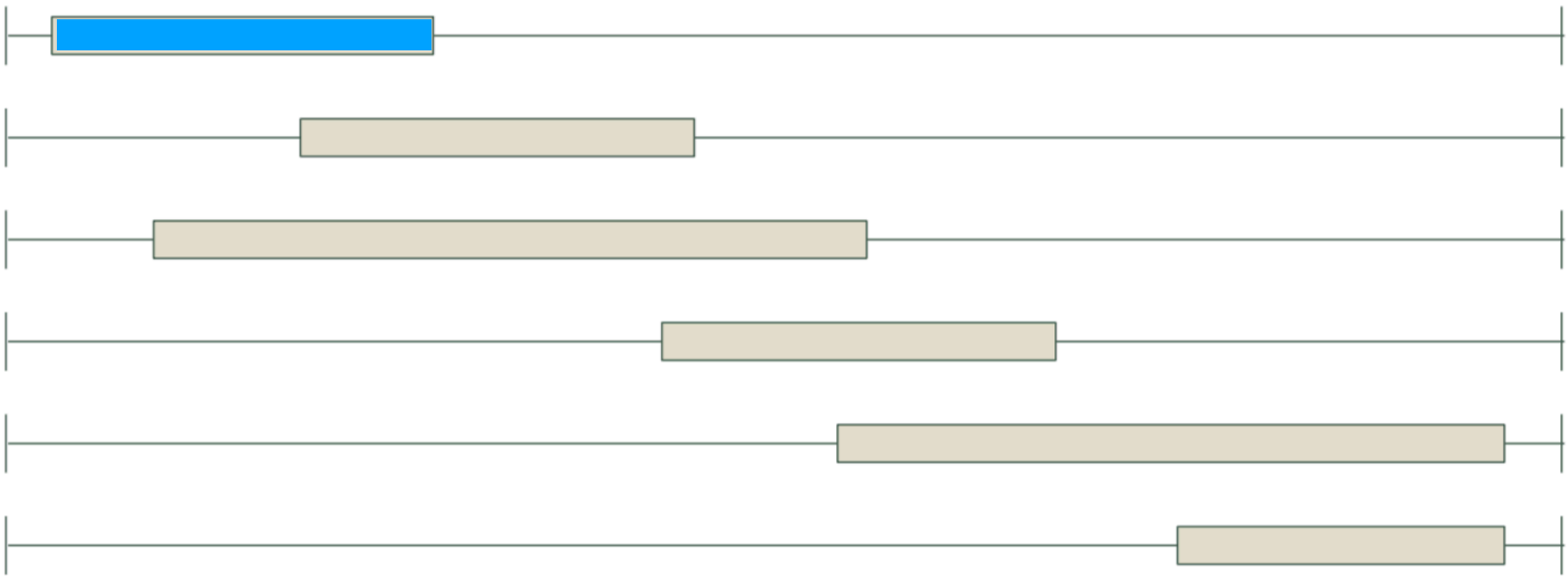
Sort by **end time** then **start time** and then process from earliest to latest end time. Each time we reach a TV show, record it if possible, else don’t record.

Does this **always work**?

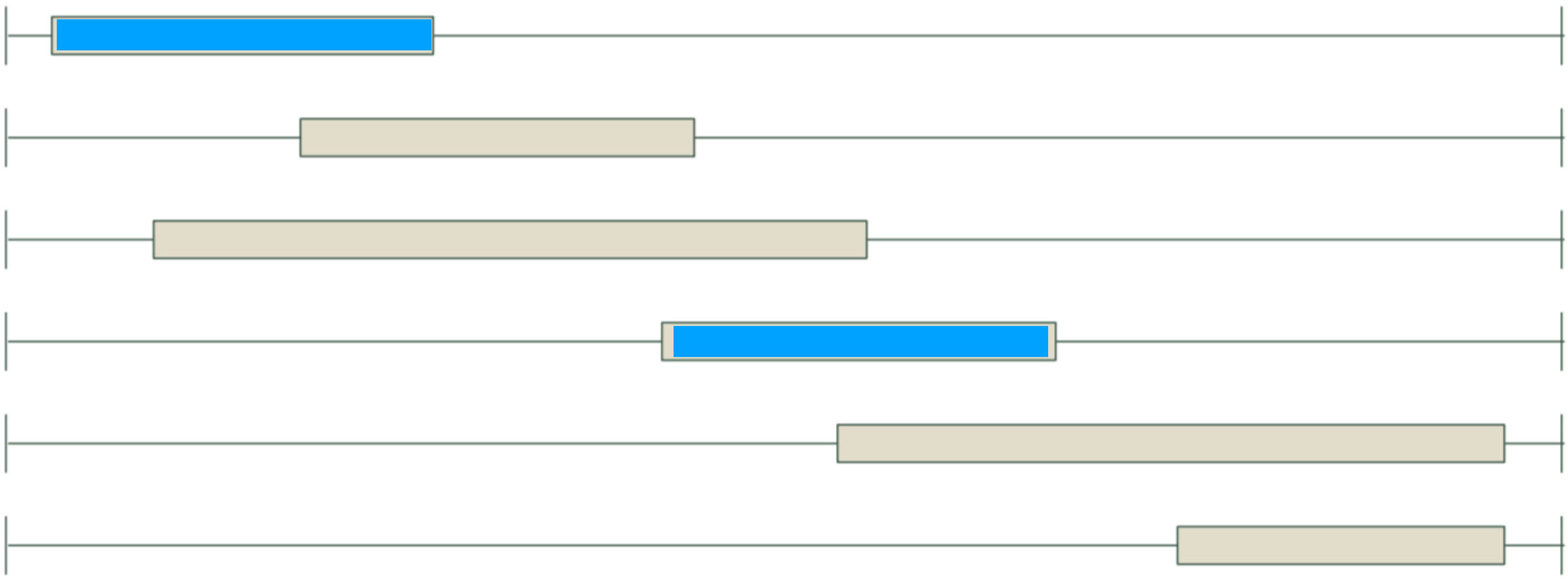
Event Scheduling



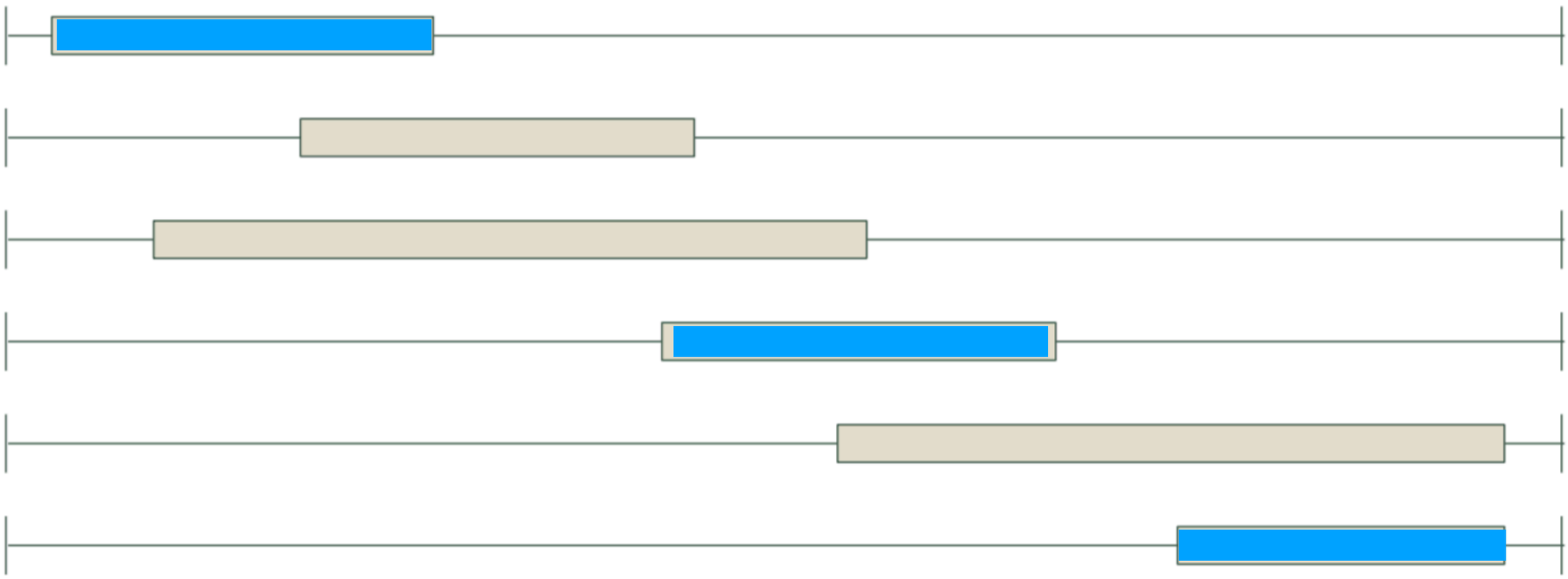
Event Scheduling



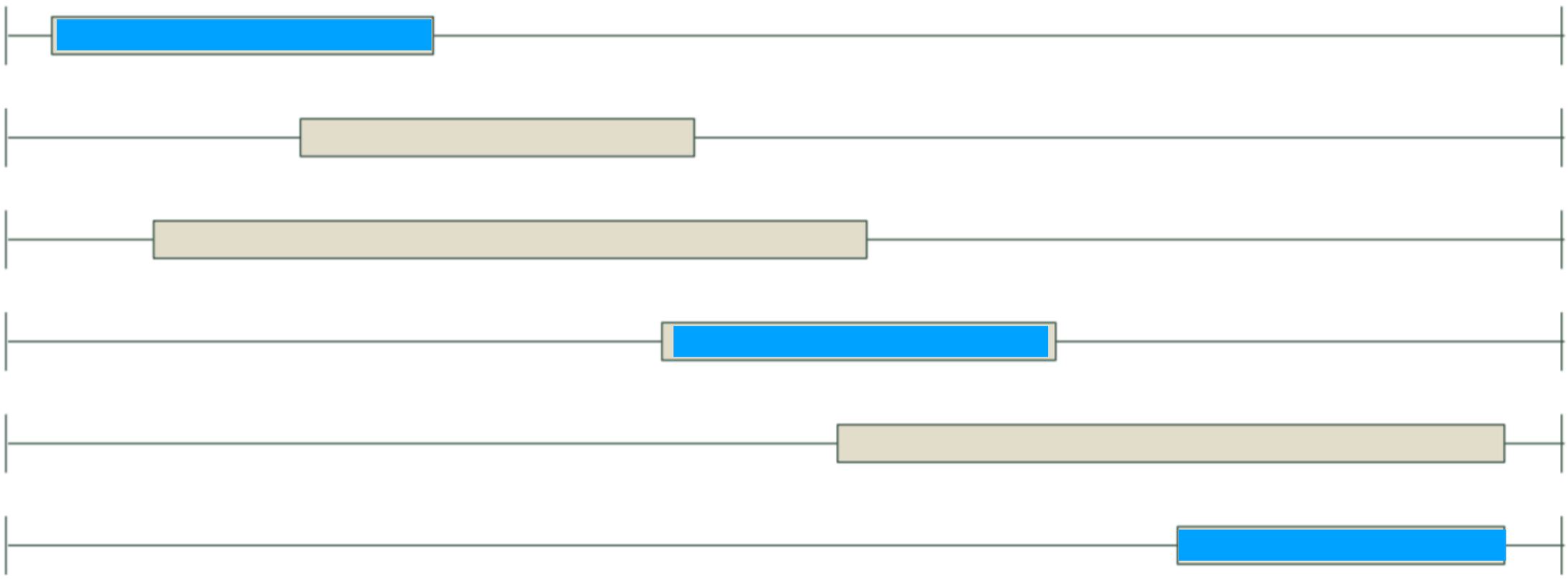
Event Scheduling



Event Scheduling



Event Scheduling



Woohoo! But does this always work?

Event Scheduling

- Proof that it **always works**: ***Exchange Argument***

Show that any solution **O** can be transformed to the greedy solution **G** by a finite series of exchanges that will not decrease the optimality of the solution.

Event Scheduling

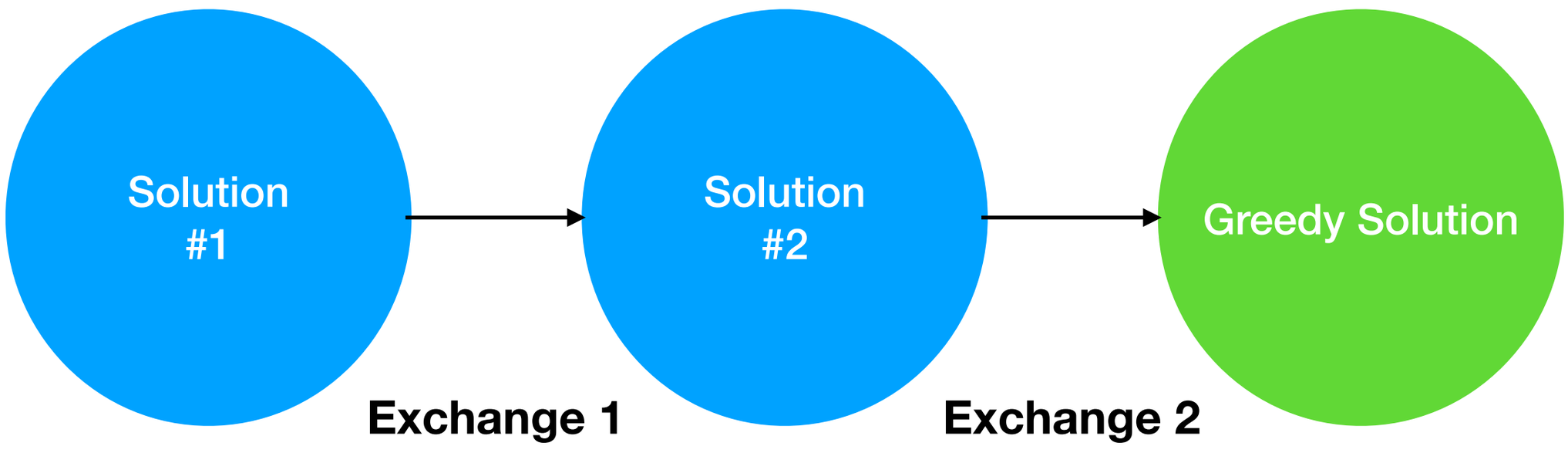
- Proof that it **always works**: ***Exchange Argument***

Show that any solution **O** can be transformed to the greedy solution **G** by a finite series of exchanges that will not decrease the optimality of the solution.

Event Scheduling

- Proof that it **always works**: ***Exchange Argument***

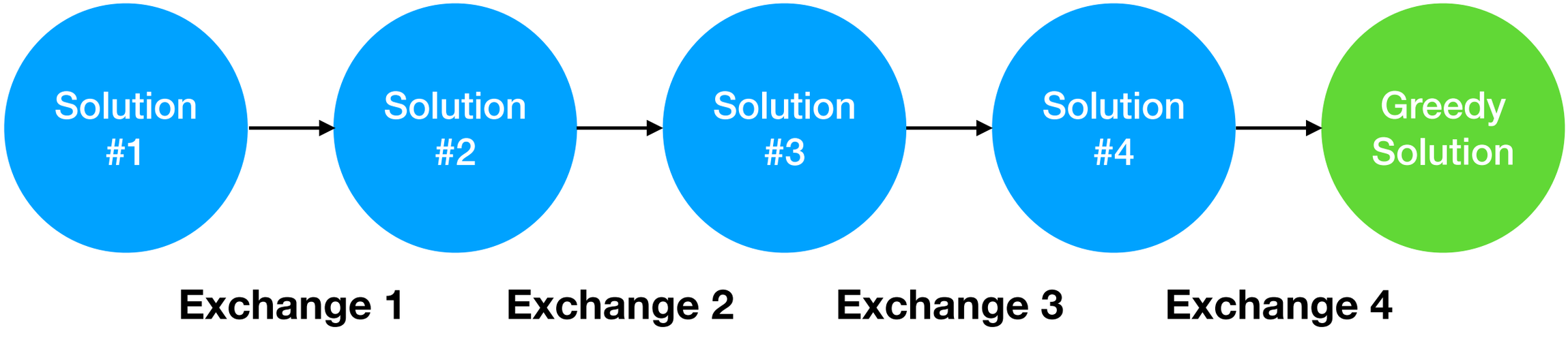
Show that any solution **O** can be transformed to the greedy solution **G** by a finite series of exchanges that will not decrease the optimality of the solution.



Event Scheduling

- Proof that it **always works**: ***Exchange Argument***

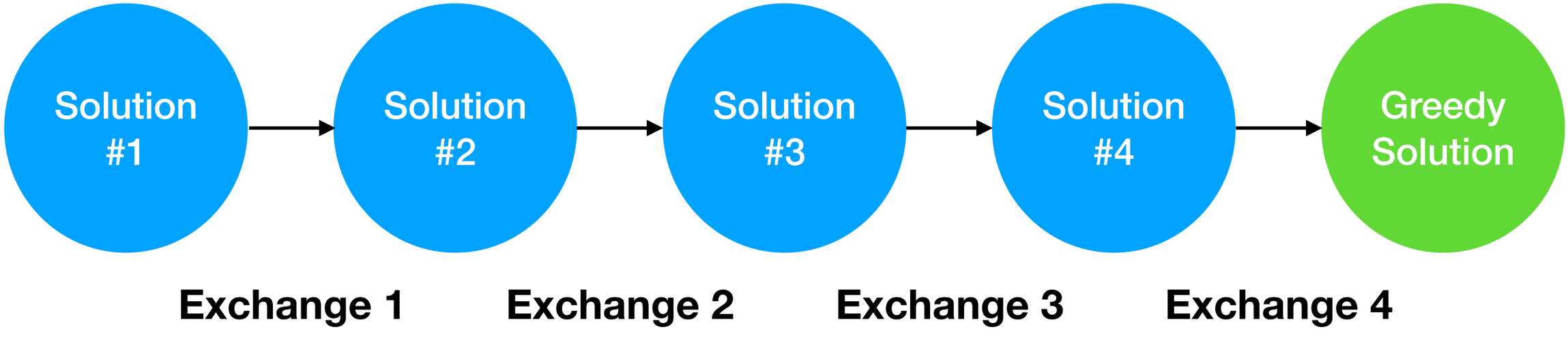
Show that any solution **O** can be transformed to the greedy solution **G** by a finite series of exchanges that will not decrease the optimality of the solution.



Event Scheduling

- Proof that it **always works**: ***Exchange Argument***

Show that any solution **O** can be transformed to the greedy solution **G** by a finite series of exchanges that will not decrease the optimality of the solution.



Event Scheduling

- Proof that it **always works**: ***Exchange Argument***

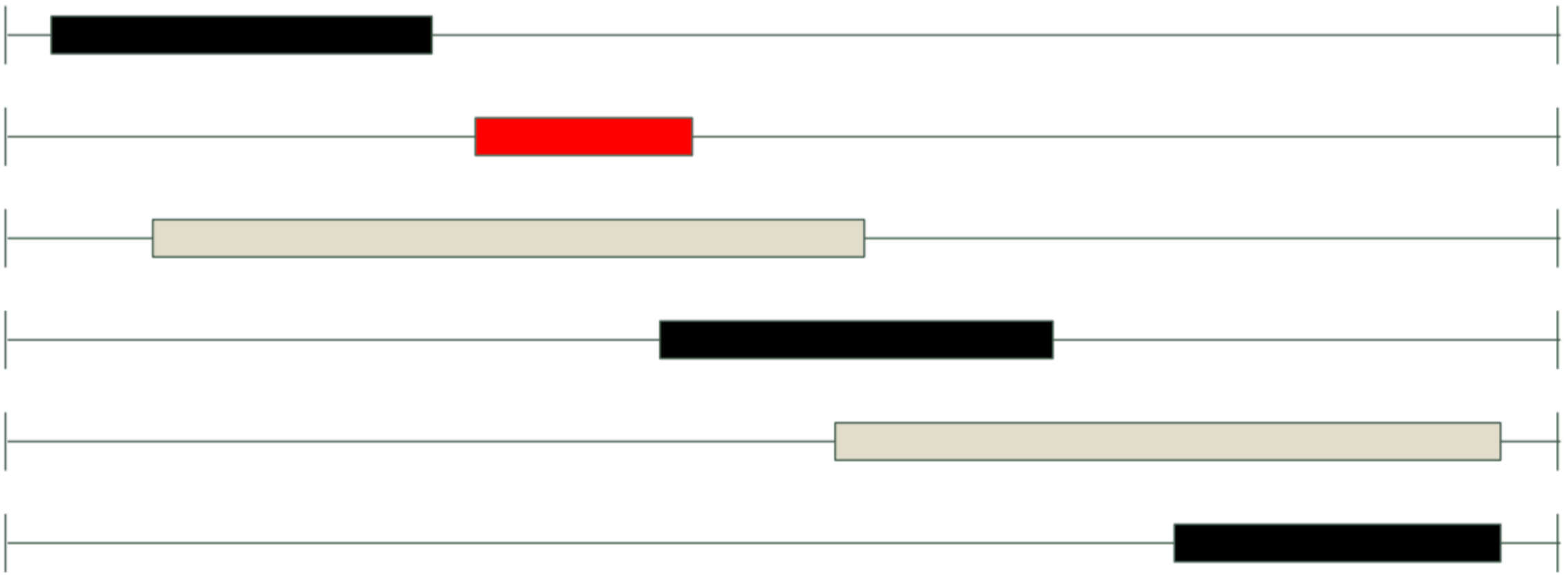
Show that any solution **O** can be transformed to the greedy solution **G** by a finite series of exchanges that will not decrease the optimality of the solution.

The important step here is to define the exchange such that

1. Eventually gives the optimal value

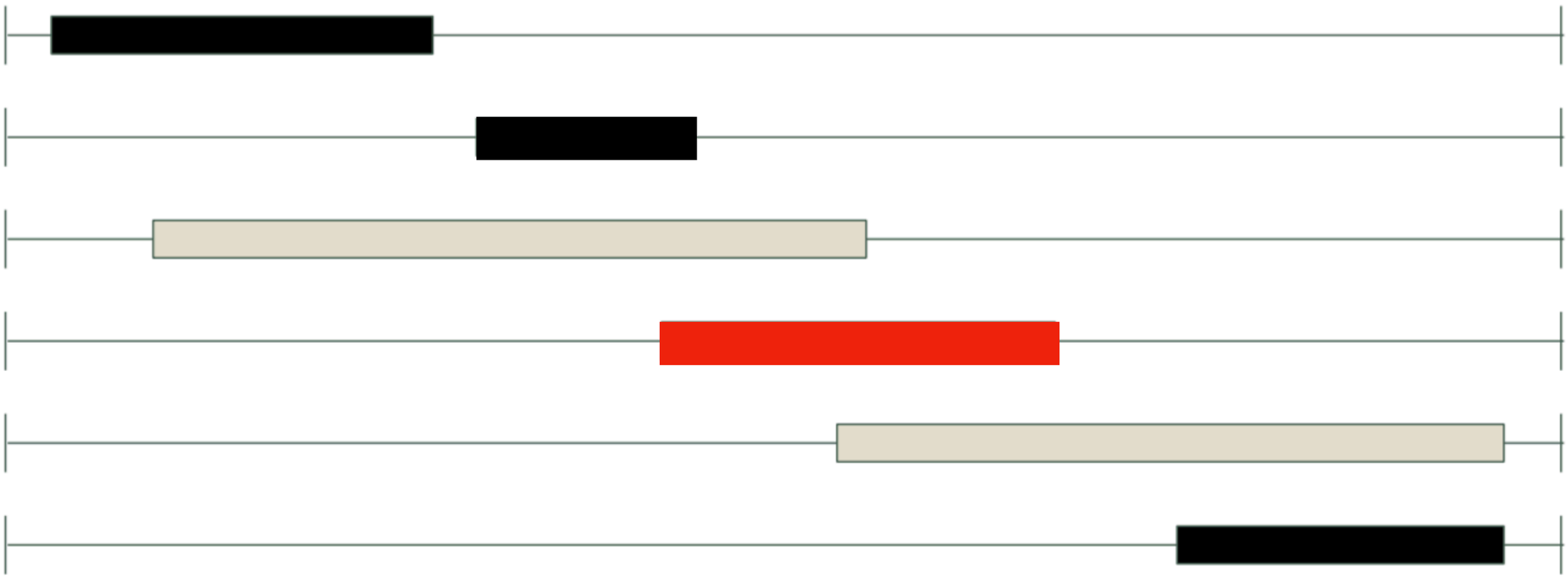
2. The current value of the solution never decreases

Event Scheduling



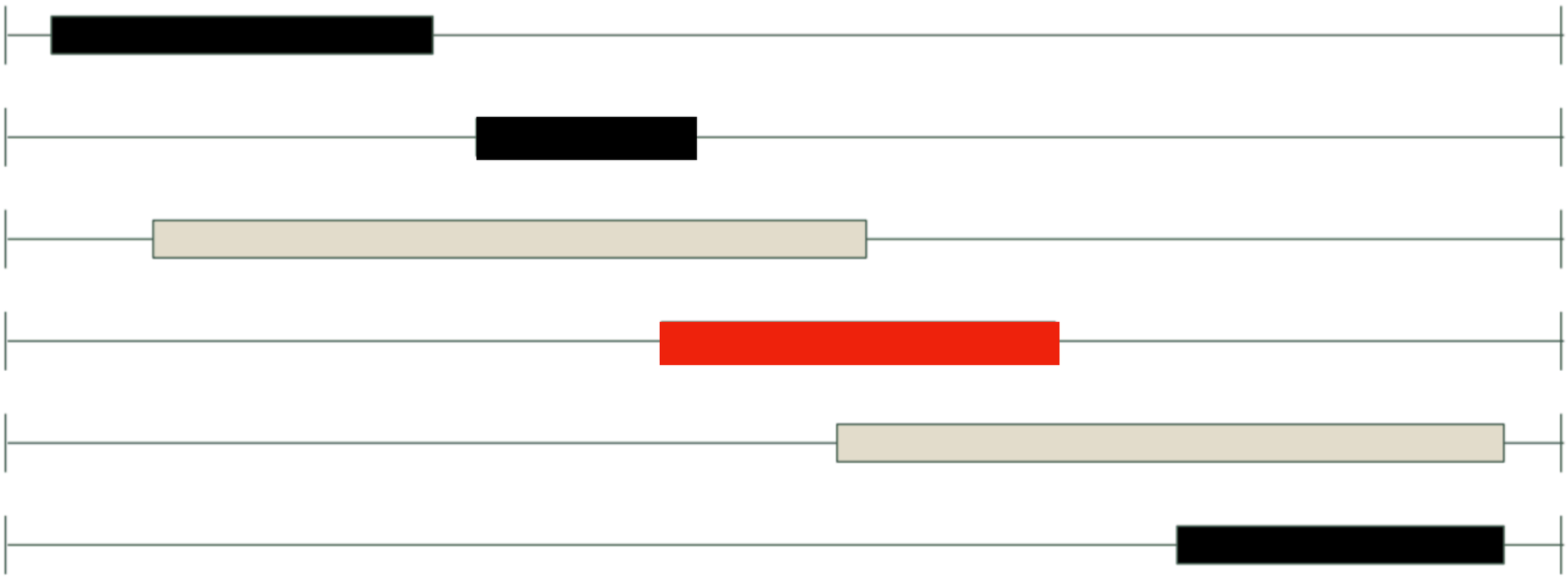
Let’s first assume there exists a solution in which you “skip” over one of the ranges you could have taken

Event Scheduling



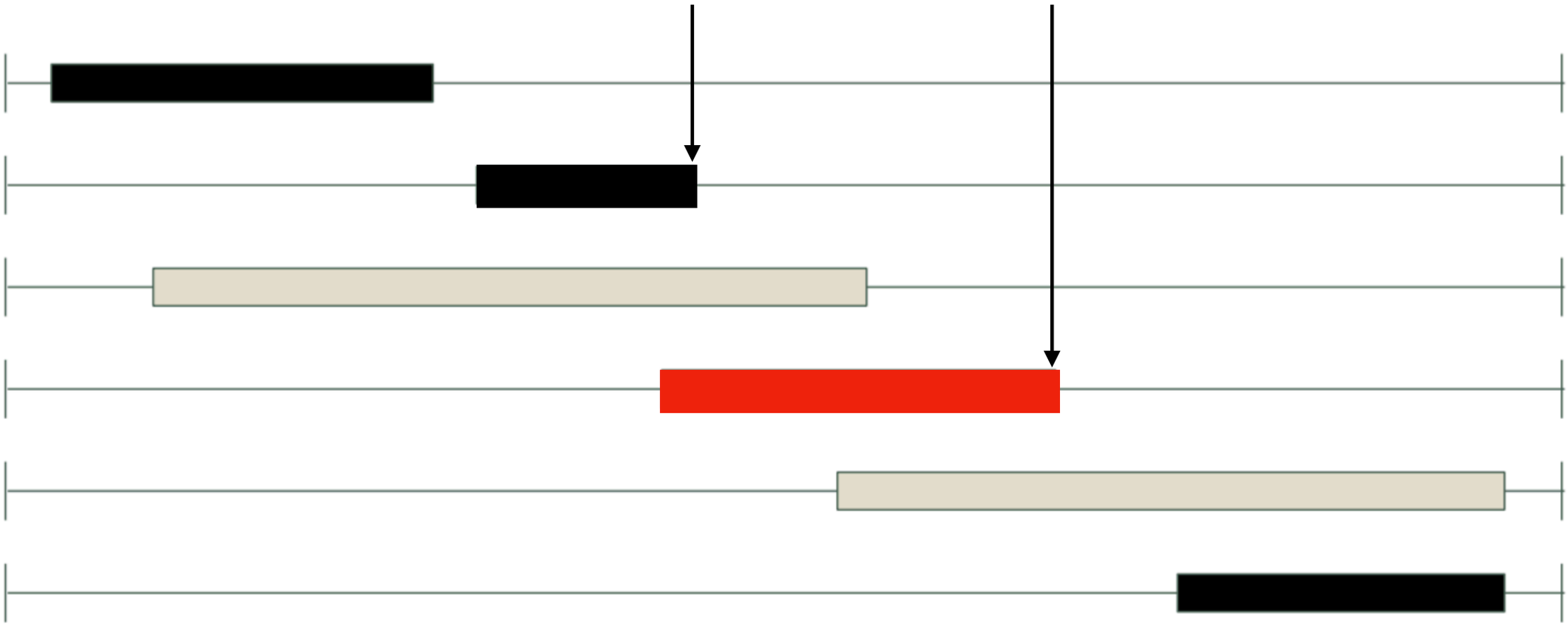
You can always “exchange” the selected range immediately after it with that “skipped” range.

Event Scheduling



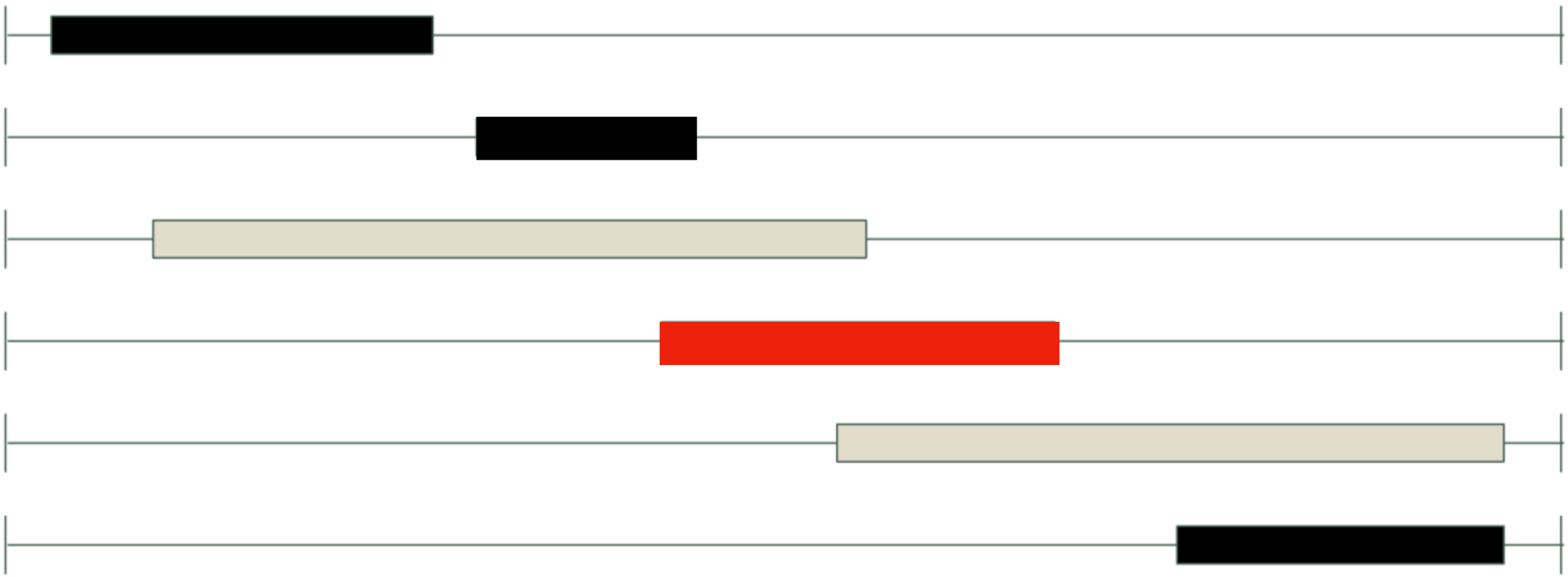
Why?

Event Scheduling



Why? Because the end time of the first one is guaranteed to be smaller than the second end time

Event Scheduling



Why? So, the skipped range will not conflict with ranges that the selected range does not conflict with.

Event Scheduling

By repeating the process of exchanging, any solution can be exchanged to form the solution where you **never** “skip” over a range, which is actually the greedy solution. Hence greedy will never give a suboptimal answer!

Event Scheduling

Questions?

Interval Partitioning

There are **N** ≤ 10^5 lectures. Lecture **i** starts at time $\mathbf{S}_i$ and ends at time $\mathbf{E}_i$. Two lectures cannot be held simultaneously in the same lecture theatre. What is the minimum number of lecture theatres the school must open?

Interval Partitioning

There are **N** ≤ 10^5 lectures. Lecture **i** starts at time $\mathbf{S}_i$ and ends at time $\mathbf{E}_i$. Two lectures cannot be held simultaneously in the same lecture theatre. What is the minimum number of lecture theatres the school must open?

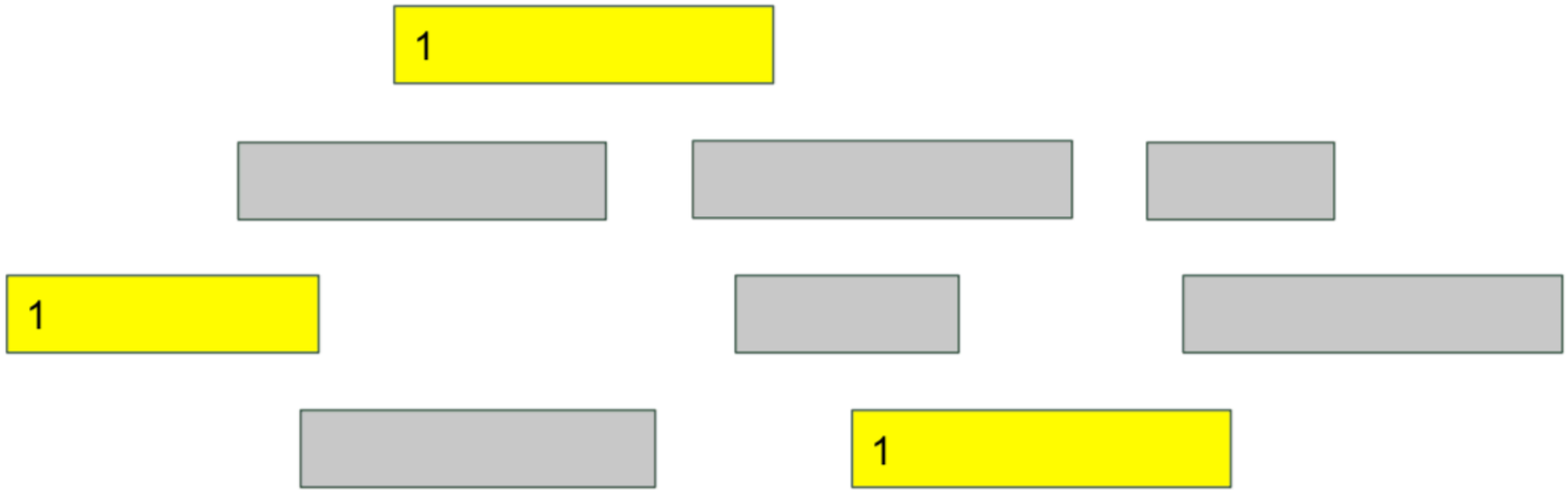
Discuss!

Interval Partitioning

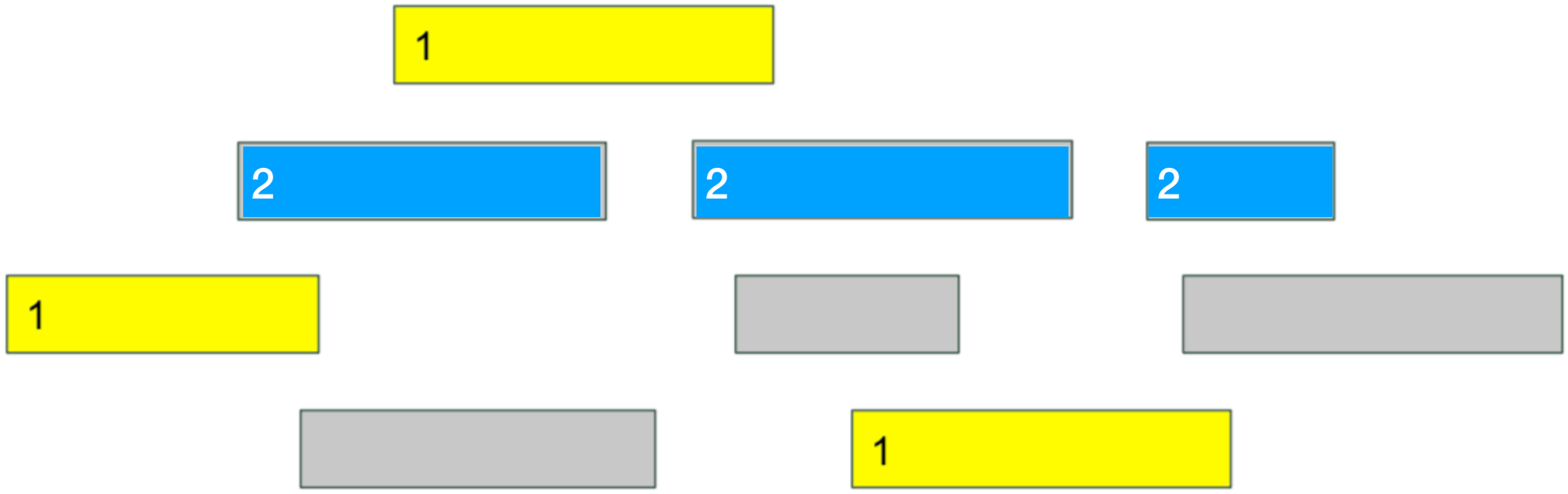
Greedy

Successively pick the interval with the first starting time greater than the current last lecture in the classroom. If there is no such interval, start a new classroom.

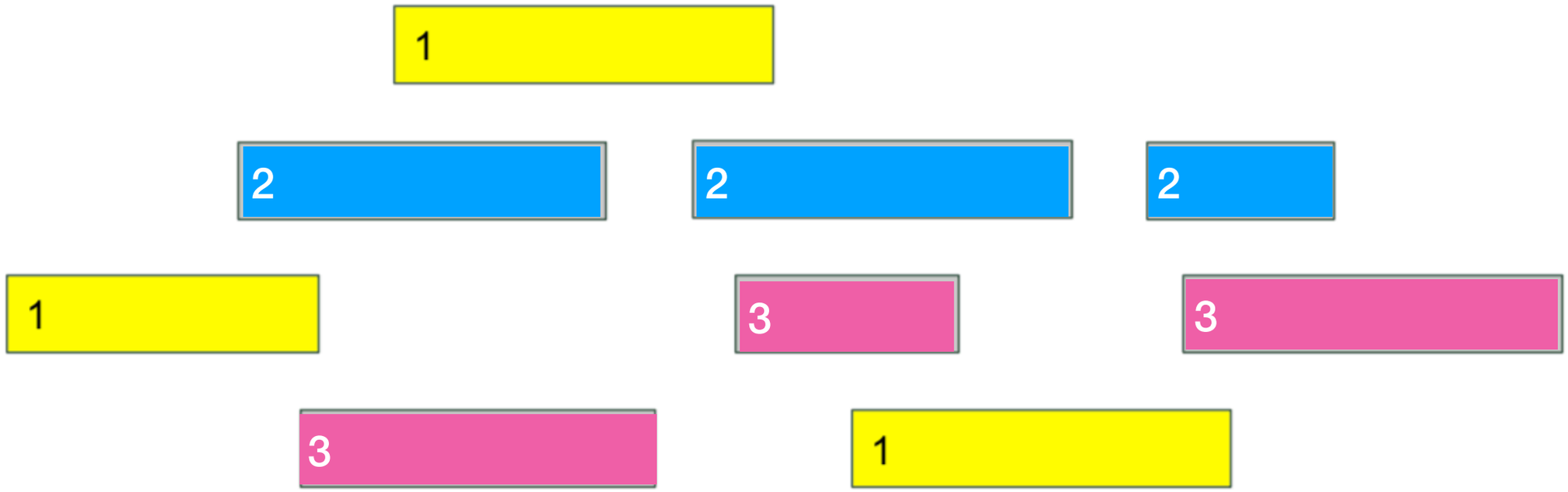
Interval Partitioning



Interval Partitioning



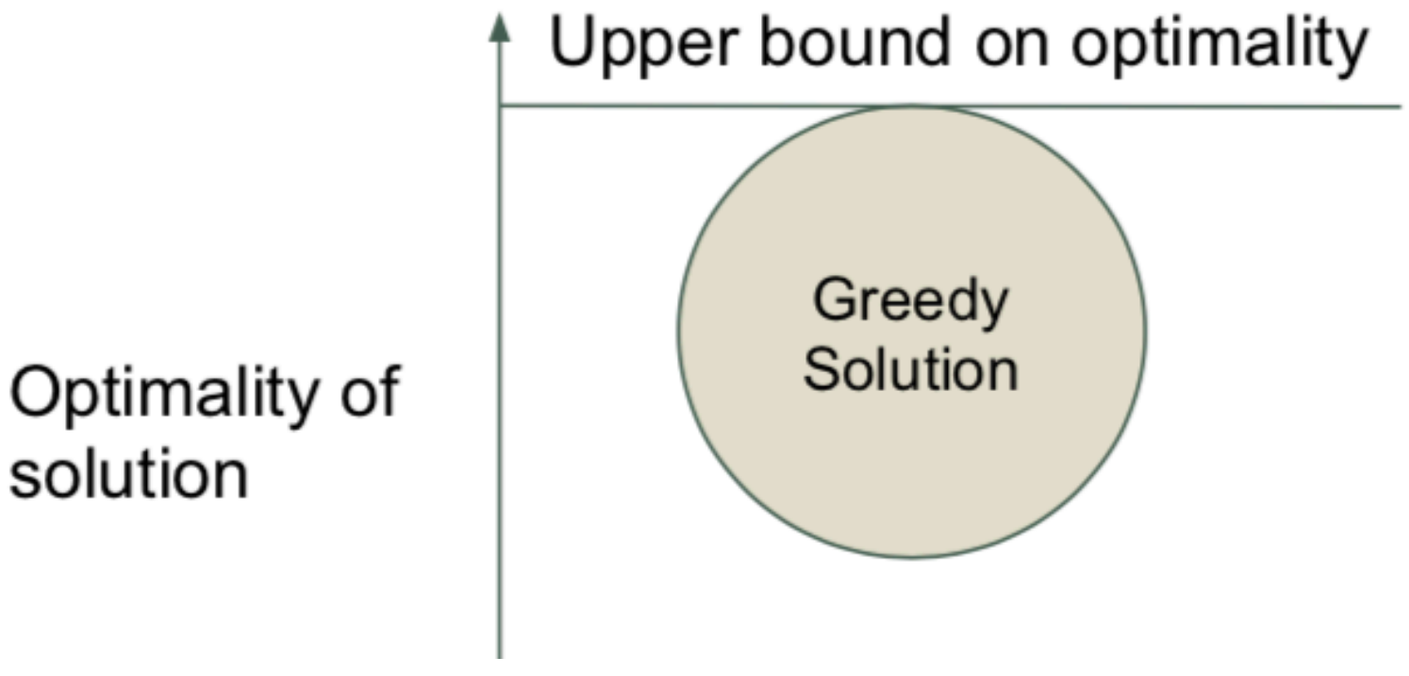
Interval Partitioning



Interval Partitioning

- Proof that it **always works**: ***Bounding Argument***

Show that any solution **O** can only attain an optimality of **V**, and the greedy solution **G** attains it.



Interval Partitioning

Lower bound: Define **depth**(x) to be the number of intervals overlapping a certain time **x**.

Let **d** be the maximum depth at all times.

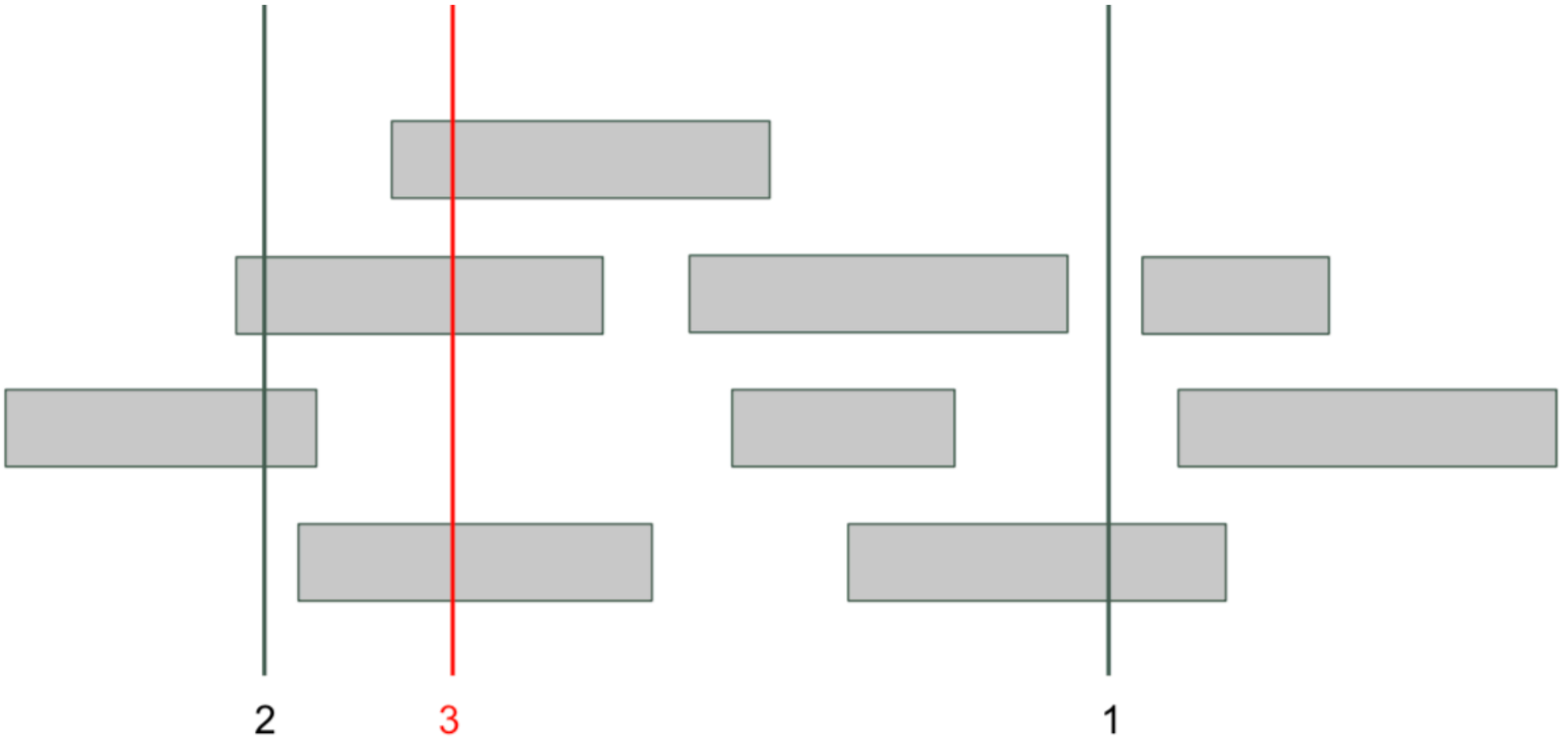
Interval Partitioning

Lower bound: Define **depth**(x) to be the number of intervals overlapping a certain time **x**.

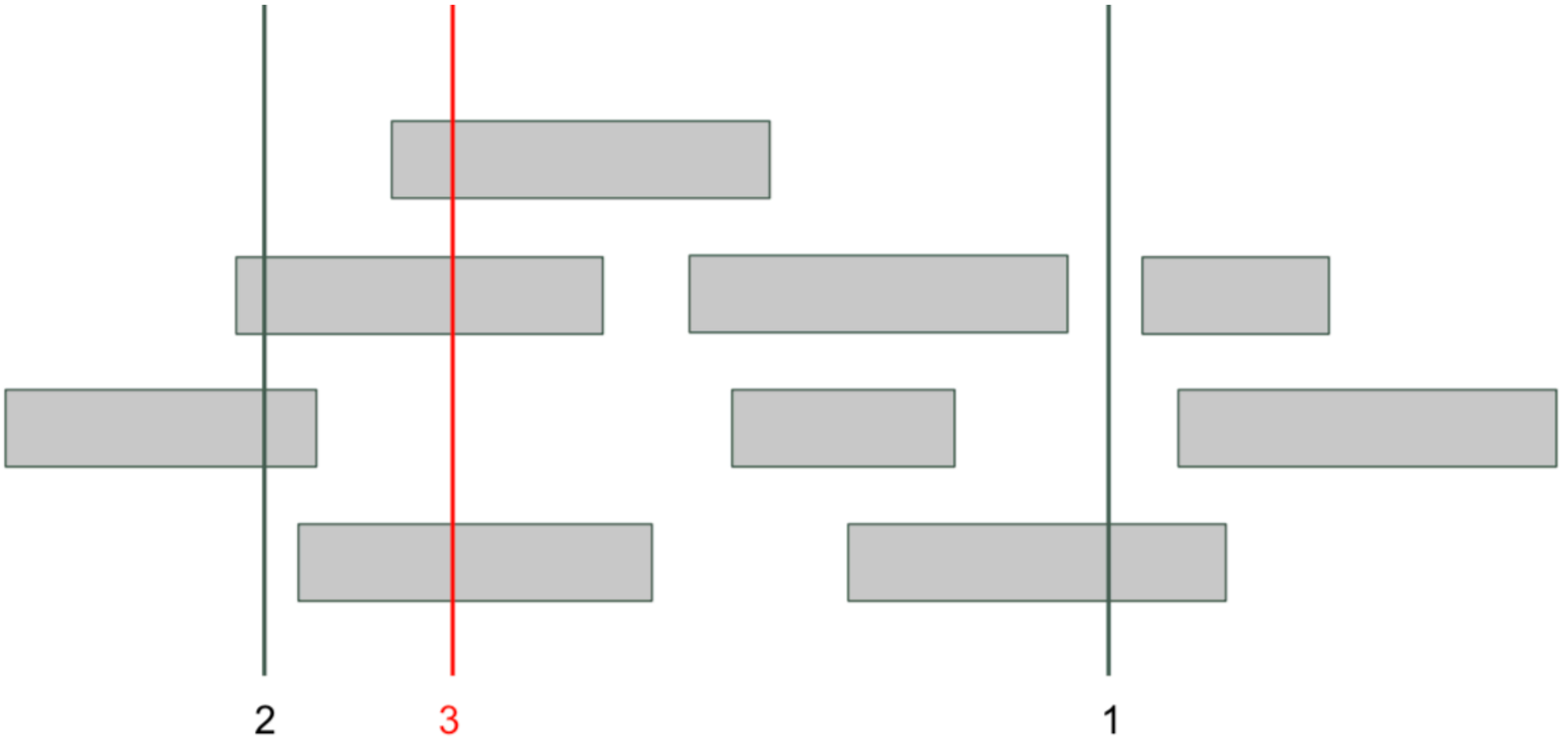
Let **d** be the maximum depth at all times.

No solution can be less than **d**.

Interval Partitioning



Interval Partitioning



$d = 3$

Interval Partitioning

Upper bound: Let **a** = number of lecture theatres the greedy algorithm allocates.

Suppose greedy algorithm is suboptimal.

Interval Partitioning

Upper bound: Let **a** = number of lecture theatres the greedy algorithm allocates.

Suppose a > d.

Let **j** be the first lecture that is scheduled in Lecture Theatre **d+1**

Interval Partitioning

Upper bound: Let **a** = number of lecture theatres the greedy algorithm allocates.

Lecture Theatre **d+1** is opened because we needed to schedule an interval **j** that is incompatible with all **d** other lecture theatres.

Interval Partitioning

Upper bound: Let **a** = number of lecture theatres the greedy algorithm allocates.

Lecture Theatre **d+1** is opened because we needed to schedule an interval **j** that is incompatible with all **d** other lecture theatres.

Since we progressively pick ranges with later start times, all these incompatibilities are caused by intervals that start earlier than $\mathbf{s}_j$ (otherwise interval **j** would be picked first instead).

Interval Partitioning

Upper bound: Let **a** = number of lecture theatres the greedy algorithm allocates.

Lecture Theatre **d+1** is opened because we needed to schedule an interval **j** that is incompatible with all **d** other lecture theatres.

Thus, at the start time of **j**, $\mathbf{s}_j$, there are at least **d+1** lectures in conflict, which contradicts the fact that the depth is **d**.

Interval Partitioning

Upper bound: Let **a** = number of lecture theatres the greedy algorithm allocates.

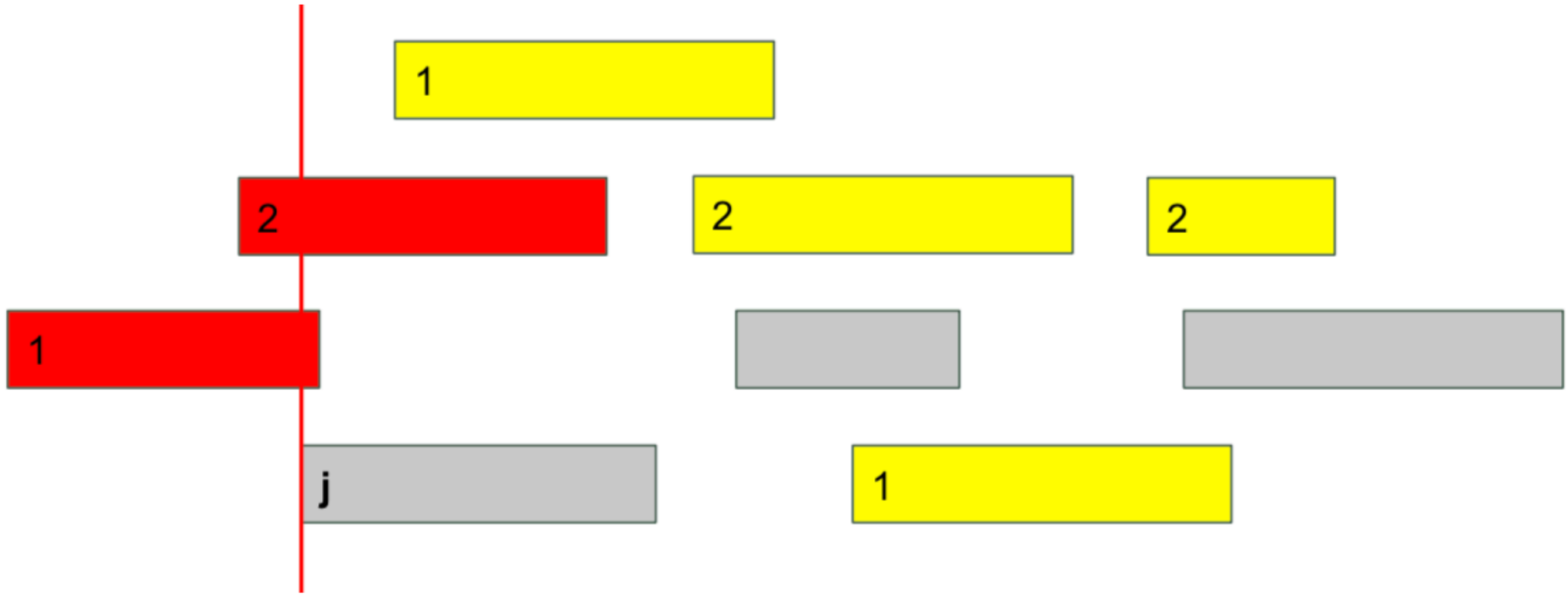
Lecture Theatre **d+1** is opened because we needed to schedule an interval **j** that is incompatible with all **d** other lecture theatres.

Thus, at the start time of **j**, $\mathbf{s}_j$, there are at least **d+1** lectures in conflict, which contradicts the fact that the depth is **d**.

Therefore, we have **a ≤ d.**

Interval Partitioning

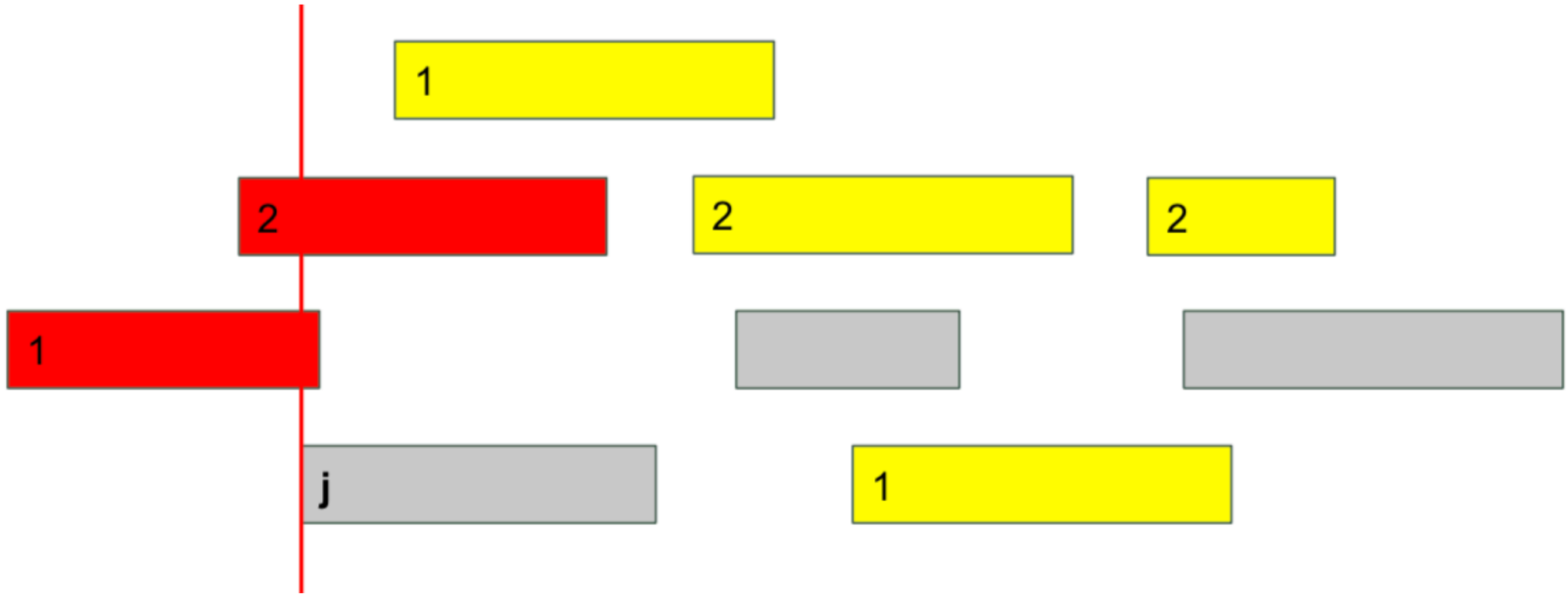
Example



LT 3 is opened because a range in LT 1 and 2 overlaps with interval j.

Interval Partitioning

Example



LT 3 is opened because a range in LT 1 and 2 overlaps with interval j.

These ranges start earlier than s_j , so depth(s_j)=3

Interval Partitioning

The greedy algorithm achieves the lower bound for the minimal solution. Hence, the greedy algorithm achieves the optimal solution.

Interval Partitioning

Definition 1 (Depth)**.** *Given a set of intervals, the* depth of this set is the maximum number of open intervals that contain a time t.

Lemma 2. *In any instance of interval partitioning we need at least depth many classrooms to schedule these intervals/courses.*

Proof This is simply because by definition of depth there is a time t and depth many courses that are all running at time t. That means that these courses are mutually in-compatible, i.e., no two of them can be scheduled at the same classroom. So, in any schedule we would need depth many classrooms. ■

Theorem 3. *In Interval Partitioning problem Greedy is optimum.*

Proof Suppose that the greedy algorithm allocates d classrooms. Our goal is to prove that $d \leq depth$. Note that this is enough to prove the theorem because by the previous lemma, $depth \leq OPT$. So, putting these together we get $d \leq OPT$. On the other hand, by definition of OPT, we know $OPT \leq d$. So, we must have $d = OPT$.

To show $d \leq depth$, by definition of $depth$, it is enough to find a time t^* such that $\geq d$ open intervals contain t^*. Let t be the time that we allocate the d-th classroom. At this time we were suppose to schedule, say j-th, course but all classrooms were already occupied so greedy had to allocate the d-th classroom. The main observation is that, by description of the algorithm, every course we have schedule so far must start before $s(j)$. Furthermore, BC all classrooms are occupied at time t there must be $d-1$ courses which are still running, i.e., $d-1$ open intervals. Now, let $t^* := t + \epsilon$ where $\epsilon > 0$ is chosen small enough such that none of those $d-1$ jobs together with job j end before or at t^*. But then we have d running courses at time t^* and this implies $depth \geq d$. ■

https://courses.cs.washington.edu/courses/cse421/21sp/notes-11.pdf

Note on Proofs

Greedy proofs may be quite complex, like in the case of the interval partitioning problem. Luckily in programming contests, you can follow your intuition!

If it **feels** true, and you can code a greedy solution quickly, then do it and see whether you get a correct answer!

Note on Proofs

AC Argument

P = The greedy choice is correct.
Q = Your solution gets full marks.

If **P**, then **Q**.
If not **Q**, then not **P**.

If **Q**, who cares?

Greedy

Practice:

- *feast*
- *lightningrod*

Follow your intuition! Prove it *if you want*

feast

feast

Problem Statement

Gug is preparing a feast for his friends. The feast consists of N plates of food arranged in a single row, with the ith plate from the left giving A_i points of satisfaction if eaten. As some plates of food might be rotten, it is possible that A_i is negative.

There are a total of K people involved in the feast, and each person will be assigned a consecutive segment of plates to consume. This segment can possibly be empty. The segments of two people cannot overlap, as food cannot be eaten twice. Gug wishes to assign the plates to his friends such that the sum of satisfaction points of all the plates of food consumed is maximised.

- https://oj.uz/problem/view/NOI19_feast

feast

Problem Statement

Input

Your program must read from standard input.

The input starts with a line with two integers N and K.

The next line will contain N integers $A_1, \ldots, A_N$.

Output

Your program must print to standard output.

The output should contain a single integer on a single line, the sum of satisfaction points in an optimal assignment.

- https://oj.uz/problem/view/NOI19_feast

feast

Problem Statement

The maximum execution time on each instance is 1.0s. For all testcases, the input will satisfy the following bounds:

- $1 \leq K \leq N \leq 3 * 10^5$
- $0 \leq |A_i| \leq 10^9$

- https://oj.uz/problem/view/NOI19_feast

feast

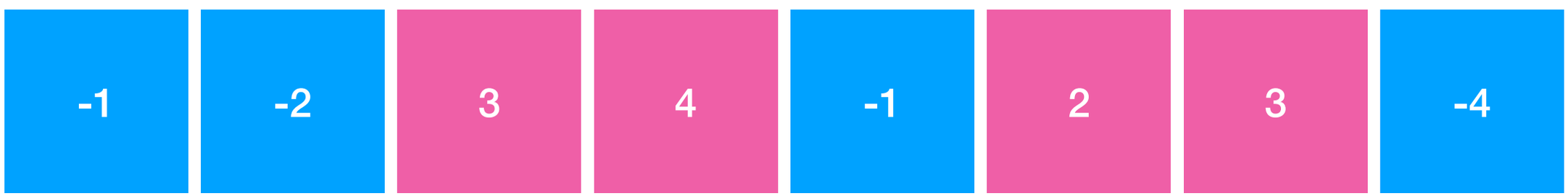
Abridged Problem Statement

- Given an array of N integers, select K non-overlapping subarrays such that their sum is maximised.

feast

Greedy Solution

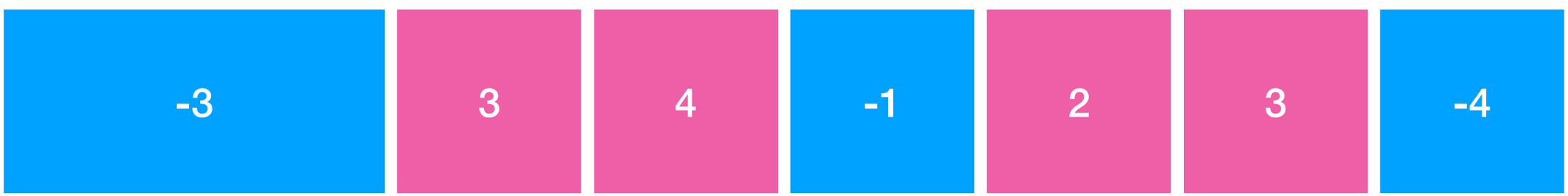
Firstly, we combine adjacent values with the same sign, and remove negative values from the two ends of the array. This creates an odd-length array where the first and last element are positive, and the sign of each integer alternates between positive and negative.



feast

Greedy Solution

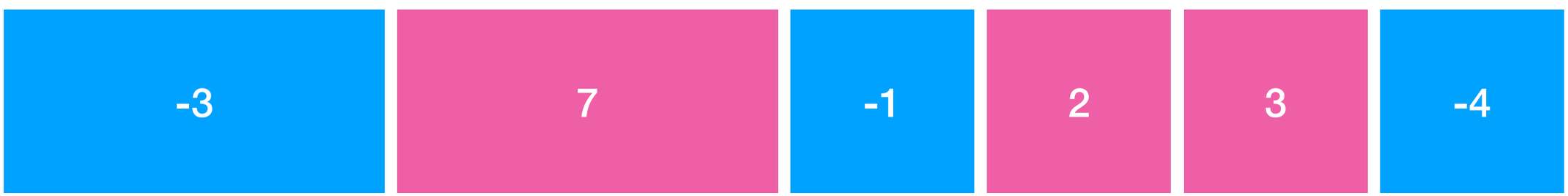
Firstly, we combine adjacent values with the same sign, and remove negative values from the two ends of the array. This creates an odd-length array where the first and last element are positive, and the sign of each integer alternates between positive and negative.



feast

Greedy Solution

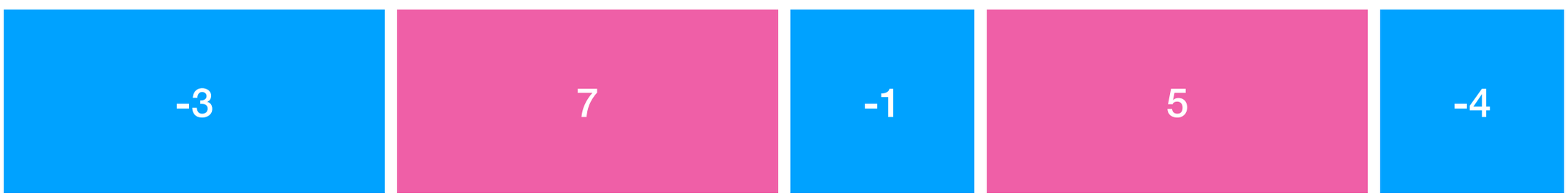
Firstly, we combine adjacent values with the same sign, and remove negative values from the two ends of the array. This creates an odd-length array where the first and last element are positive, and the sign of each integer alternates between positive and negative.



feast

Greedy Solution

Firstly, we combine adjacent values with the same sign, and remove negative values from the two ends of the array. This creates an odd-length array where the first and last element are positive, and the sign of each integer alternates between positive and negative.

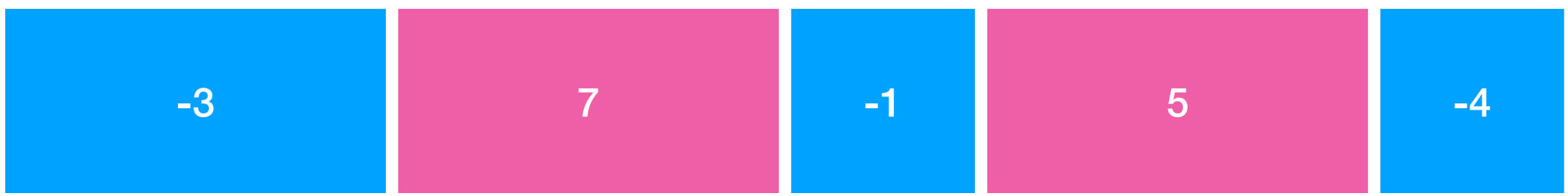


feast

Greedy Solution

If there are no more than K positive integers, then the solution would just be picking all of them.

eg, K = 2

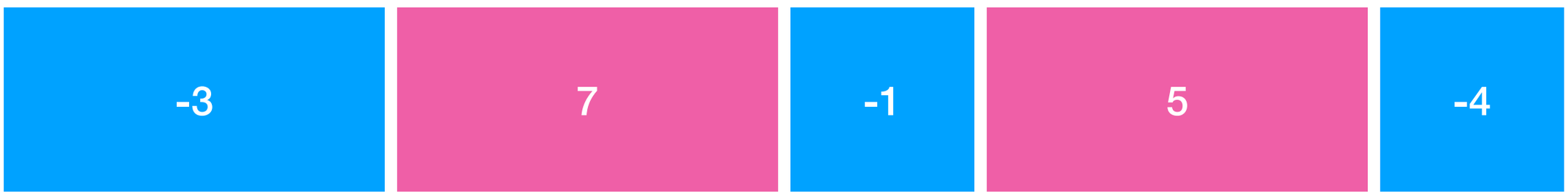


feast

Greedy Solution

If there are more than K integers, we repeat the following process until there are only K positive integers:

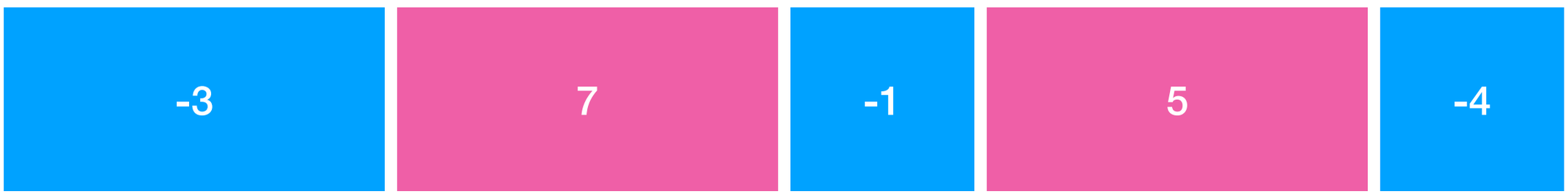
- Pick the element with the lowest absolute value.
- Combine this element with the two elements adjacent to it.



feast

Greedy Solution

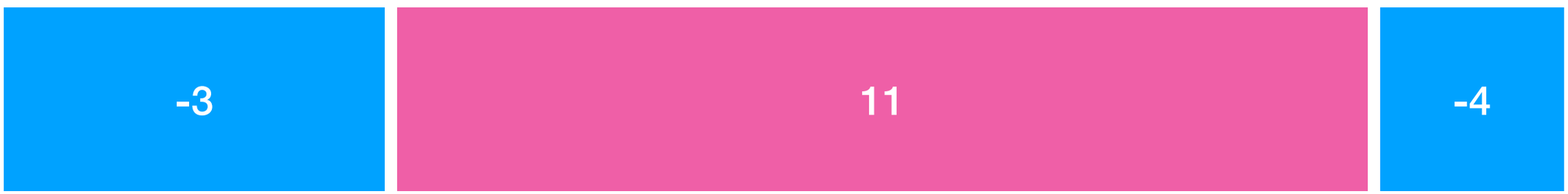
eg, K = 1



feast

Greedy Solution

eg, K = 1

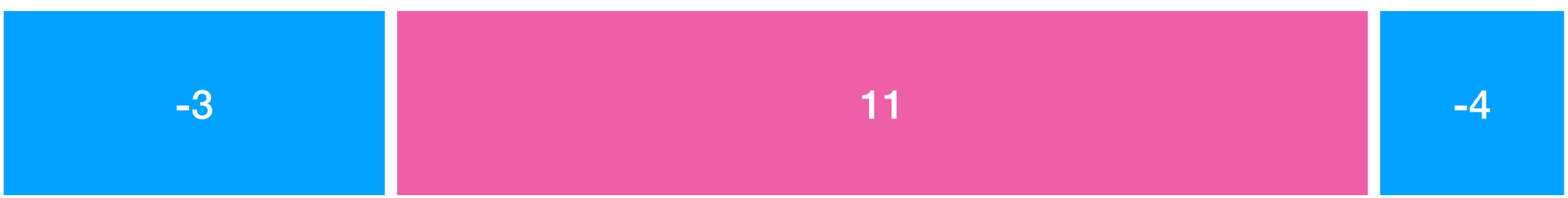


feast

Greedy Solution

eg, K = 1

Done!



feast

Complexity Analysis

Each of these operations can be done in O(logN) time, by using an STL set or priority queue. This makes the overall complexity of this solution O(N log N).

lightningrod (NOI 2018)

lightningrod

Problem Statement

Singapore has anywhere between 171 and 186 lightning days on average a year. Each square kilometer of land in Singapore can be struck up to 16 times annually. This makes Singapore one of the lightning capitals of the world.

Gug the architect surveys N buildings from left to right, and notices that the top of building i, from left to right, has coordinates (X_i, Y_i). Gug wants to protect all the buildings by planting lightning rods on top of some buildings. A lightning rod protects the building it is planted on, and all buildings that lie on or under the $45°$ line of depression leftwards and rightwards. In other words, a lightning rod on building i protects building j if and only if $|X_i - X_j| \leq Y_i - Y_j$.

Help Gug find out the minimum number of lightning rods required to protect all buildings.

https://oj.uz/problem/view/NOI18_lightningrod

lightningrod

Input format

Your program must read from standard input.

The input starts with a single integer, N, in a single line. N denotes the total number of buildings.

N lines will then follow with 2 integers each, the i^{th} line will contain X_i and Y_i. This indicates that the peak of the i^{th} building is at (X_i, Y_i). You can assume $X_i \leq X_{i+1}$, in other words, X_i is increasing.

Note: The input size for subtasks 1, 6 and 7 is extremely large, so it is only possible to obtain full credit using C++ fast input. The attachment consists of a template that uses C++ fast input to read from standard input.

Output format

Your program must print to standard output.
Output a single integer, denoting the minimum number of lightning rods required to protect all buildings.

lightningrod

Problem Statement

Subtask	Marks	N	X_i, Y_i
1	4	$2 \leq N \leq 10\,000\,000$	$0 \leq X_i \leq 10^9, Y_i = 1$
2	7	$N = 2$	$0 \leq X_i, Y_i \leq 10^9$
3	12	$2 \leq N \leq 20$	$0 \leq X_i, Y_i \leq 10^9$
4	21	$2 \leq N \leq 2\,000$	$0 \leq X_i, Y_i \leq 10^9$
5	26	$2 \leq N \leq 200\,000$	$0 \leq X_i, Y_i \leq 10^9$
6	10	$2 \leq N \leq 10\,000\,000$	$X_i = i, 0 \leq Y_i \leq 1$
7	20	$2 \leq N \leq 10\,000\,000$	$0 \leq X_i, Y_i \leq 10^9$

https://oj.uz/problem/view/NOI18_lightningrod

lightningrod

Abridged Problem Statement

- Find the minimum number of points to be chosen as lightning rods among N points, such that every point is covered by a lightning rod.

- Point i is covered by Point j if and only if $|X_i - X_j| \leq Y_j - Y_i$.

lightningrod

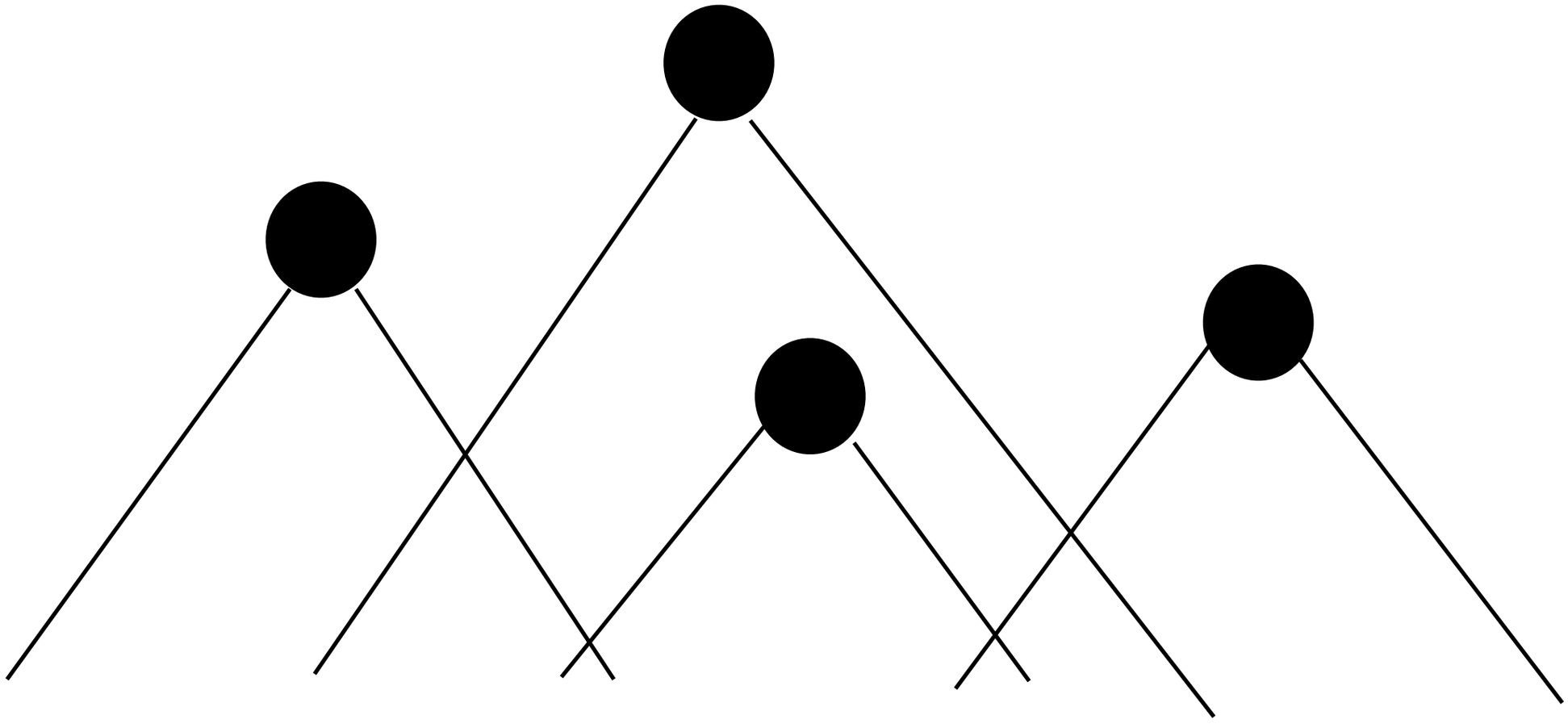
Greedy Solution

- Observe that we can first assume all points are chosen, then pick out points which are not covered by other points.

- We pick a point (X_i, Y_i) if $X_i + Y_i$ is higher than all of $X_j + Y_j$ for $i > j$, and $Y_i - X_i$ is higher than all of $Y_j - X_j$ for $i < j$. This can be done with a static prefix max of $X_i + Y_i$ and suffix max of $Y_i - X_i$

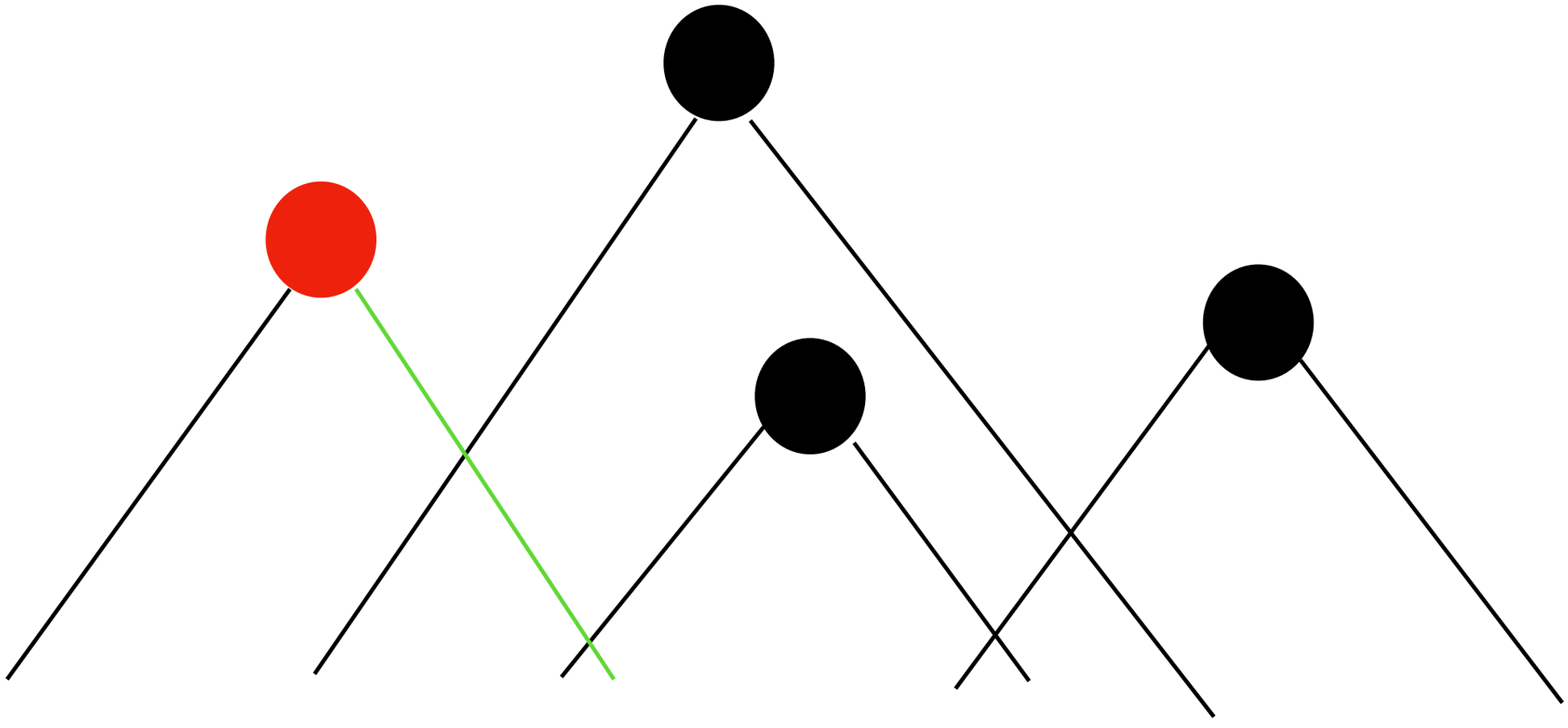
lightningrod

Greedy Solution



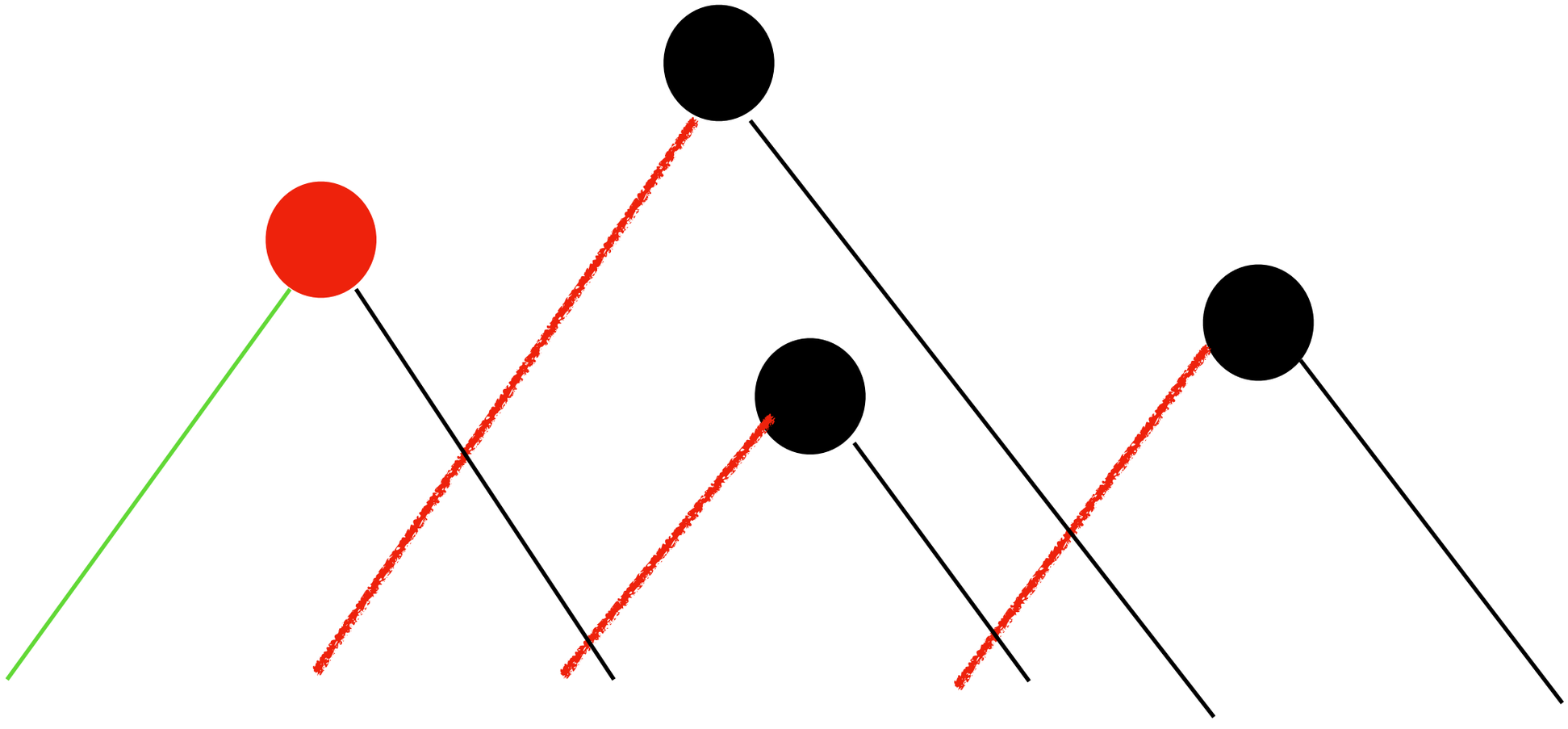
lightningrod

Greedy Solution



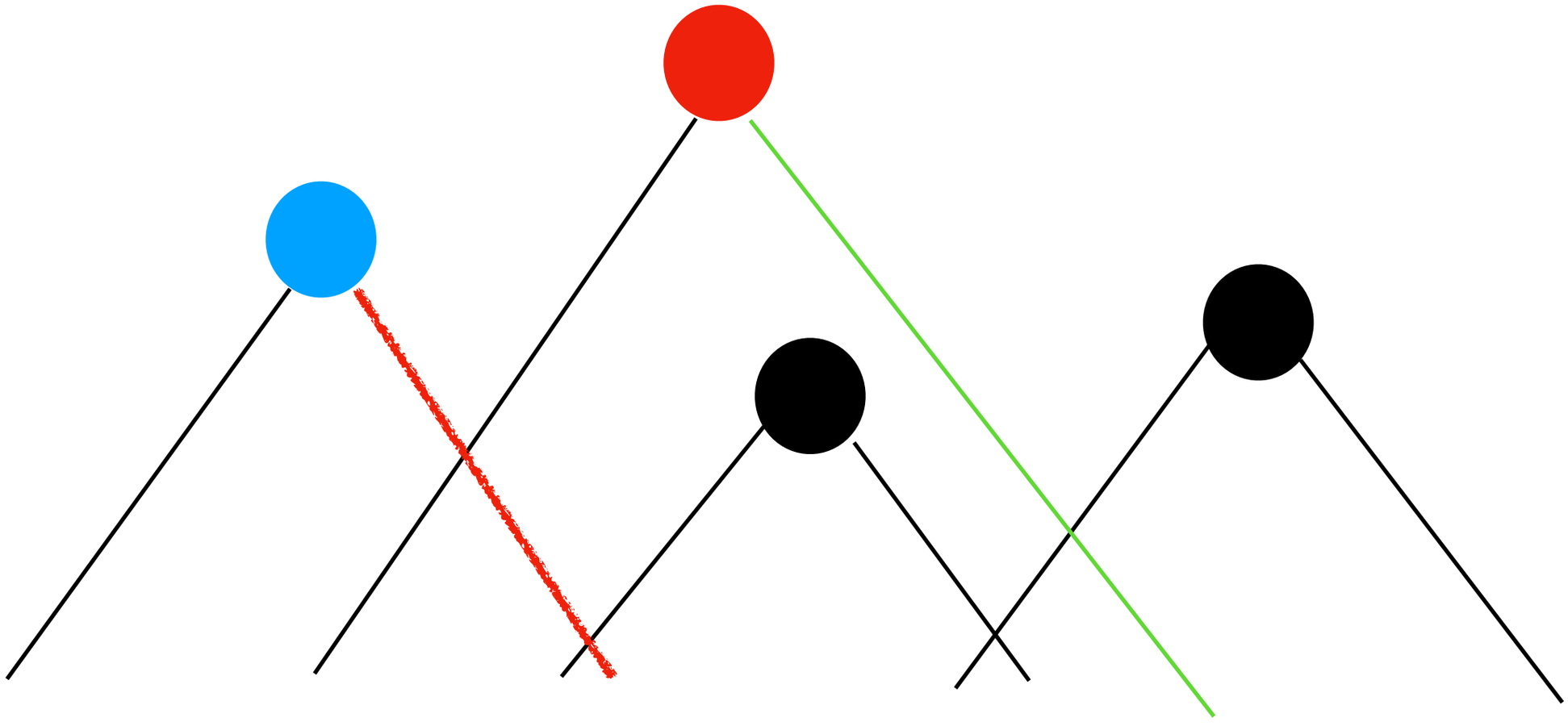
lightningrod

Greedy Solution



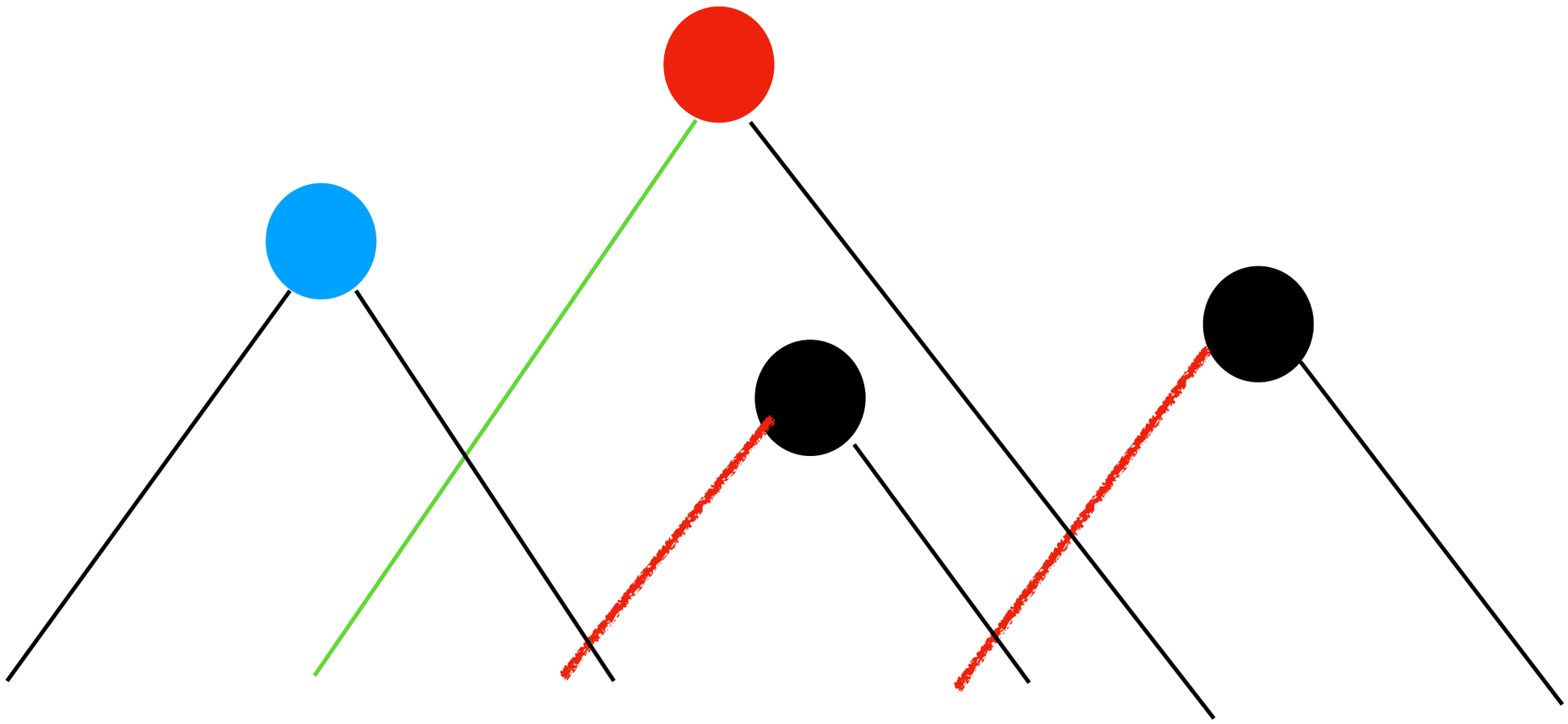
lightningrod

Greedy Solution



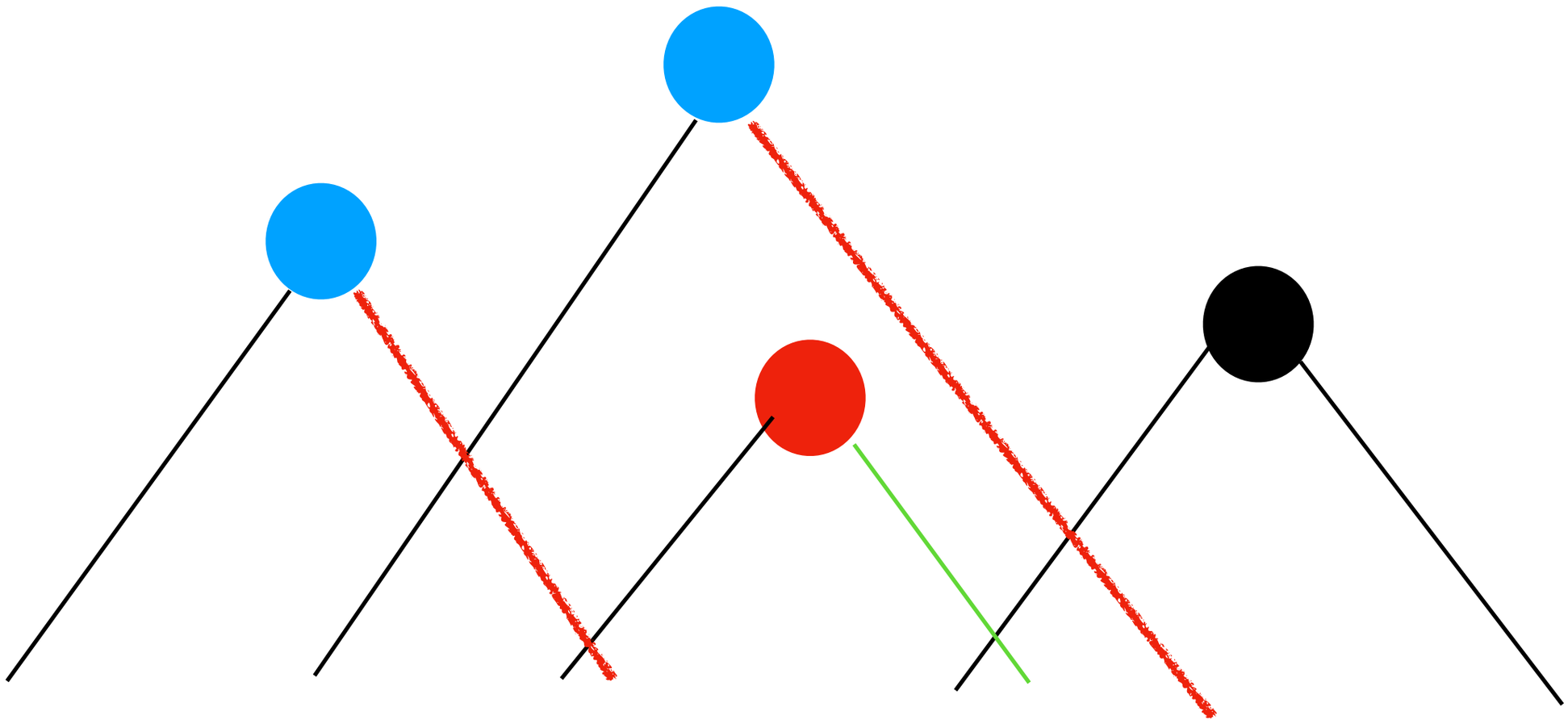
lightningrod

Greedy Solution



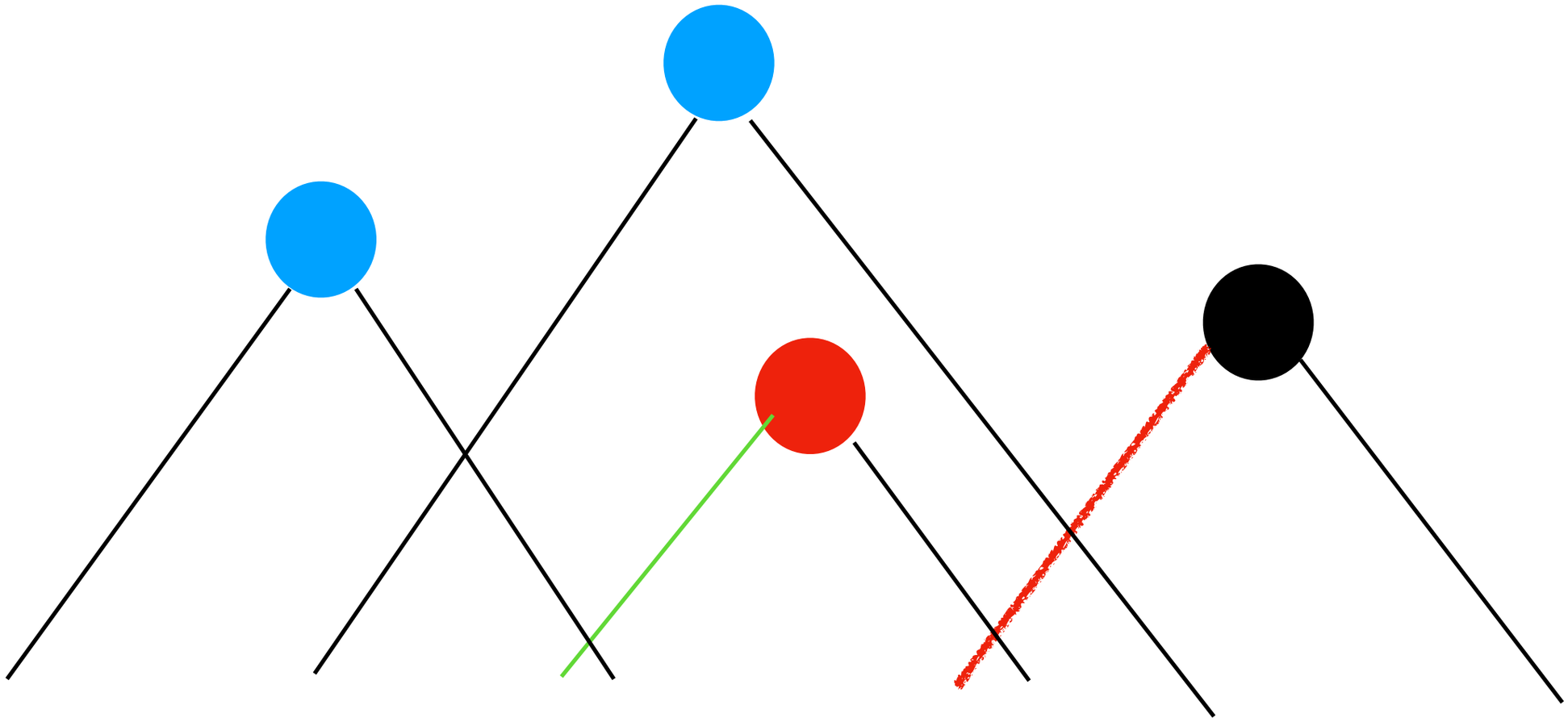
lightningrod

Greedy Solution



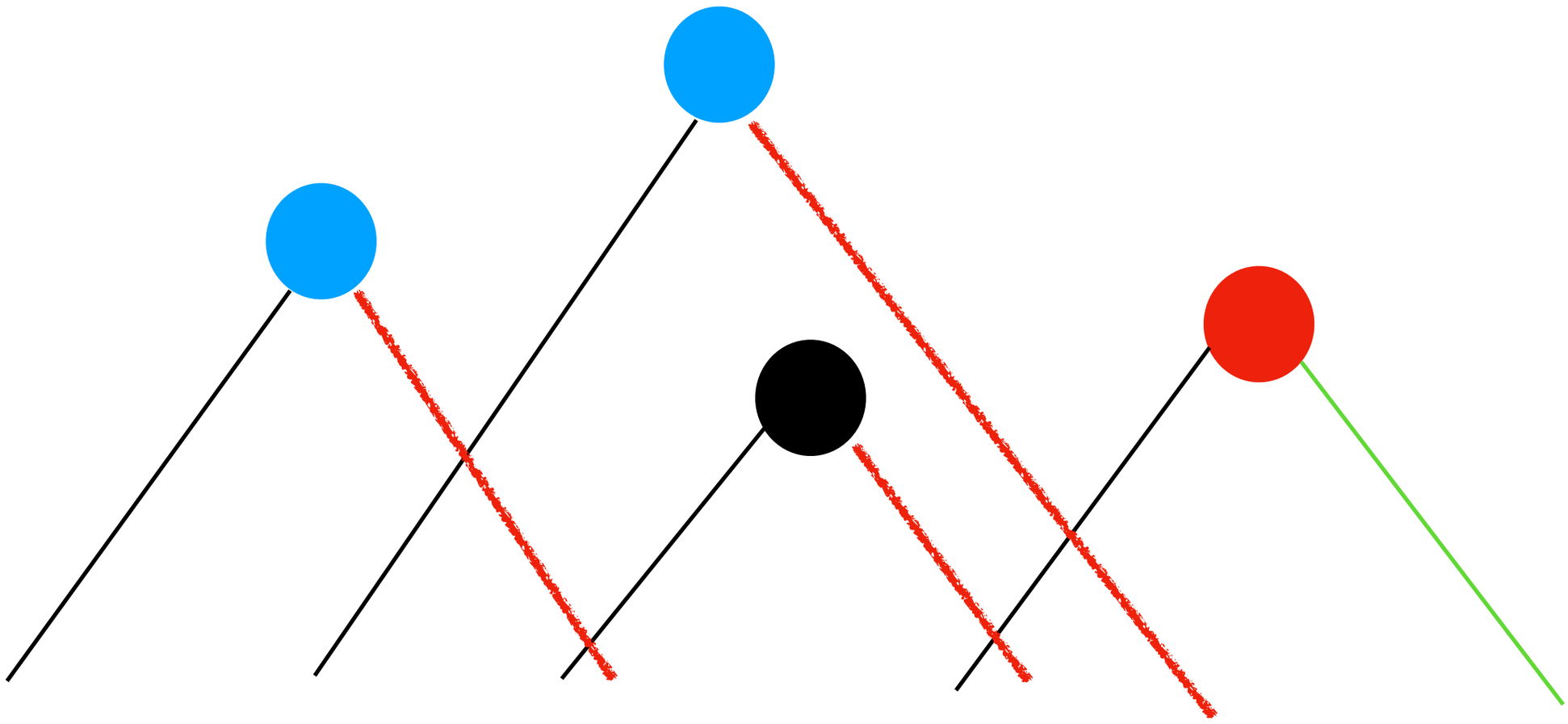
lightningrod

Greedy Solution



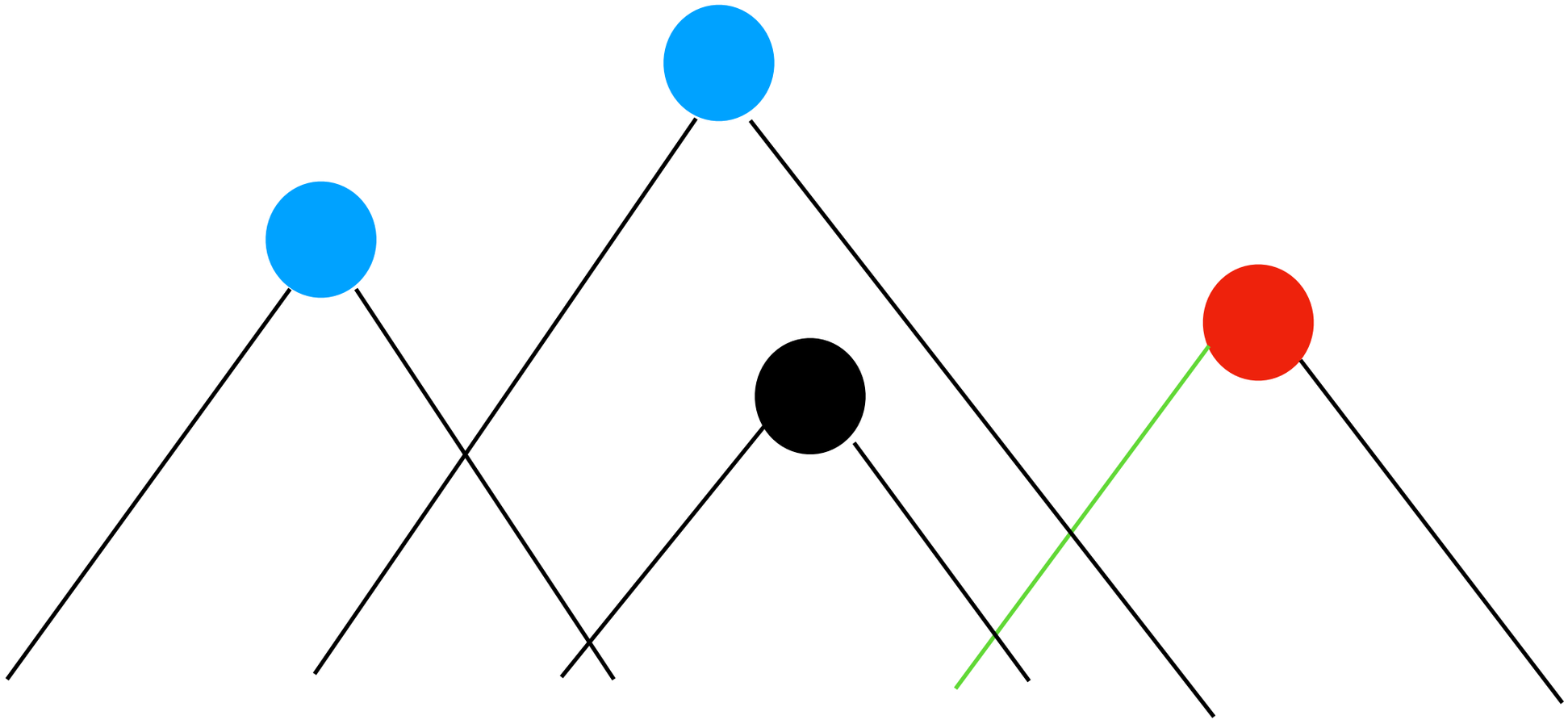
lightningrod

Greedy Solution



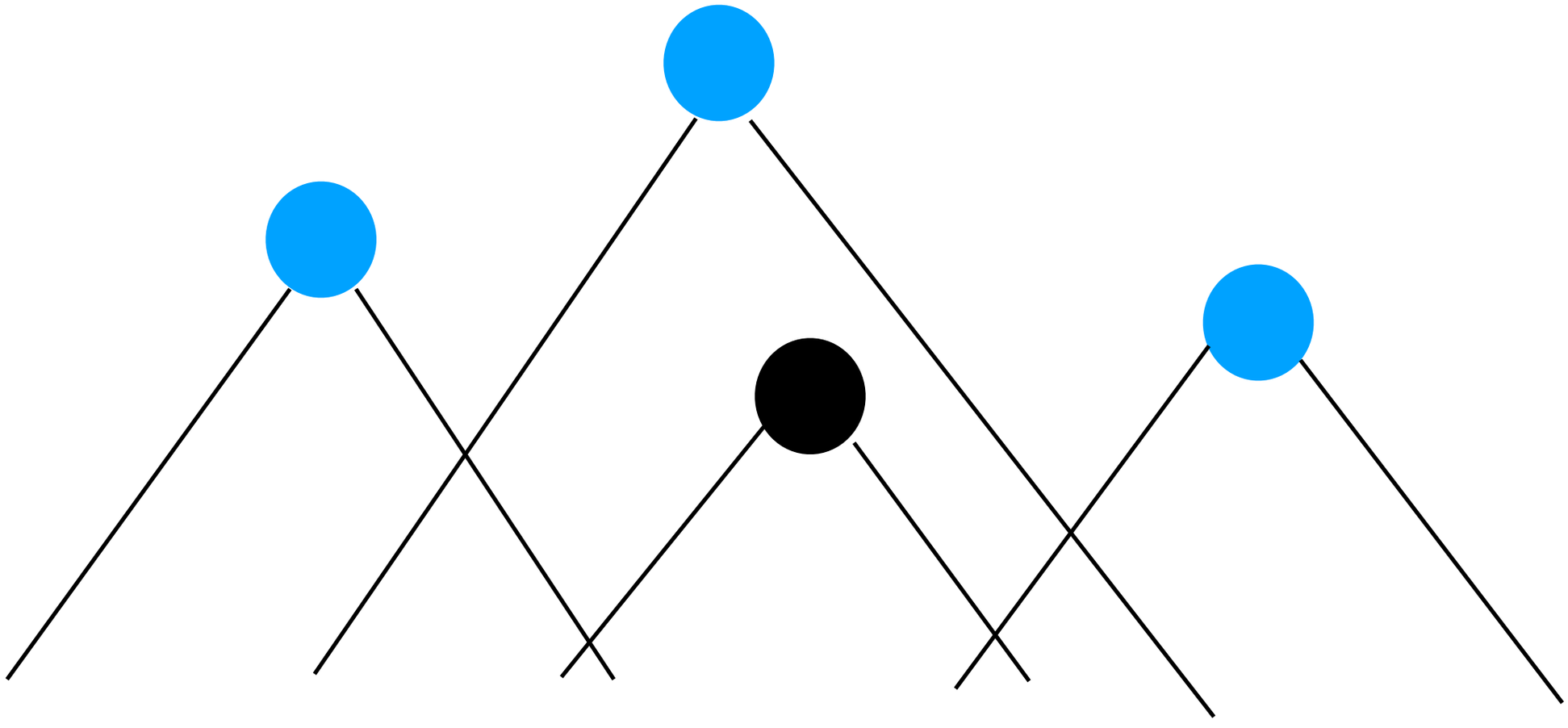
lightningrod

Greedy Solution



lightningrod

Greedy Solution



lightningrod

Complexity Analysis

O(N) time

Practice time!

Mash-up