Algorithm Problem Solving (APS): Greedy Method

Niema Moshiri
UC San Diego SPIS 2019

Example: The Change Problem (USA Currency)

- In the USA, we commonly use the following coins:
  - **C** = {1¢ (penny), 5¢ (nickel), 10¢ (dime), 25¢ (quarter)}

Example: The Change Problem (USA Currency)

- In the USA, we commonly use the following coins:
  - ***C*** = {1¢ (penny), 5¢ (nickel), 10¢ (dime), 25¢ (quarter)}
- **Input:** A non-negative integer ***x*** (in cents, not dollars)

Example: The Change Problem (USA Currency)

- In the USA, we commonly use the following coins:
  - **C** = {1¢ (penny), 5¢ (nickel), 10¢ (dime), 25¢ (quarter)}
- **Input:** A non-negative integer ***x*** (in cents, not dollars)
- **Output:** A selection of coins in **C** summing to ***x***

Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you an arcade token

Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you an arcade token
- You would probably be annoyed with me, but why?

Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you an arcade token
- You would probably be annoyed with me, but why?
  - I selected a coin that wasn't in **C**!

Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you an arcade token
- You would probably be annoyed with me, but why?
  - I selected a coin that wasn’t in **C**!
- **The issue: my solution is incorrect**

Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you 2 pennies

Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you 2 pennies
- You would probably be annoyed with me, but why?

Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you 2 pennies
- You would probably be annoyed with me, but why?
    - All coins I selected were in ***C***

Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you 2 pennies
- You would probably be annoyed with me, but why?
  - All coins I selected were in ***C***
  - The coins I selected don't sum to 42!

Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you 2 pennies
- You would probably be annoyed with me, but why?
  - All coins I selected were in **C**
  - The coins I selected don’t sum to 42!
- **The issue: my solution is incorrect**

Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you 42 pennies

Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you 42 pennies
- You would probably be annoyed with me, but why?

Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you 42 pennies
- You would probably be annoyed with me, but why?
  - All coins I selected were in ***C***

Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you 42 pennies
- You would probably be annoyed with me, but why?
  - All coins I selected were in ***C***
  - The sum of my coins equals 42

Example: The Change Problem (USA Currency)

- Imagine I owe you 42¢, so I give you 42 pennies
- You would probably be annoyed with me, but why?
  - All coins I selected were in **C**
  - The sum of my coins equals 42
- **The issue: your problem formulation was not specific!**

Optimization Problems

- In many problems, we may have *many* (even infinite) possible solutions

Optimization Problems

- In many problems, we may have *many* (even infinite) possible solutions
- In all problems, we *must* define the precise definition of **correctness**

Optimization Problems

- In many problems, we may have *many* (even infinite) possible solutions
- In all problems, we *must* define the precise definition of **correctness**
- We can also choose to define an **objective function** to optimize

Optimization Problems

- In many problems, we may have *many* (even infinite) possible solutions
- In all problems, we *must* define the precise definition of **correctness**
- We can also choose to define an **objective function** to optimize
- A solution satisfying the definition of *correctness* is **correct**

Optimization Problems

- In many problems, we may have *many* (even infinite) possible solutions
- In all problems, we *must* define the precise definition of **correctness**
- We can also choose to define an **objective function** to optimize
- A solution satisfying the definition of *correctness* is **correct**
- A *correct* solution optimizing the *objective function* is **optimal**

Revisiting the Change Problem (USA Currency)

- **C** = {1¢ (penny), 5¢ (nickel), 10¢ (dime), 25¢ (quarter)}

Revisiting the Change Problem (USA Currency)

- **C** = {1¢ (penny), 5¢ (nickel), 10¢ (dime), 25¢ (quarter)}

- **Input:** A non-negative integer ***x*** (in cents, not dollars)

Revisiting the Change Problem (USA Currency)

- **C** = {1¢ (penny), 5¢ (nickel), 10¢ (dime), 25¢ (quarter)}
- **Input:** A non-negative integer ***x*** (in cents, not dollars)
- **Output:** A selection of coins in **C** summing to ***x***

Revisiting the Change Problem (USA Currency)

- **C** = {1¢ (penny), 5¢ (nickel), 10¢ (dime), 25¢ (quarter)}
- **Input:** A non-negative integer ***x*** (in cents, not dollars)
- **Output:** A selection of coins in **C** summing to ***x*** such that the number of selected coins is minimized

Multiple Optimal Solutions

- In some problems, there may be multiple equally-optimal solutions

Multiple Optimal Solutions

- In some problems, there may be multiple equally-optimal solutions
    - Imagine if $\boldsymbol{C}$ = {1¢, 2¢, 3¢, 4¢} and $\boldsymbol{x}$ = 5¢

Multiple Optimal Solutions

- In some problems, there may be multiple equally-optimal solutions
  - Imagine if $\boldsymbol{C}$ = {1¢, 2¢, 3¢, 4¢} and $\boldsymbol{x}$ = 5¢
  - [1¢, 4¢] and [2¢, 3¢] are equally-optimal solutions

Multiple Optimal Solutions

- In some problems, there may be multiple equally-optimal solutions
  - Imagine if ***C*** = {1¢, 2¢, 3¢, 4¢} and ***x*** = 5¢
  - [1¢, 4¢] and [2¢, 3¢] are equally-optimal solutions
- You should be happy receiving any such solution

Multiple Optimal Solutions

- In some problems, there may be multiple equally-optimal solutions
  - Imagine if $\boldsymbol{C}$ = {1¢, 2¢, 3¢, 4¢} and $\boldsymbol{x}$ = 5¢
  - [1¢, 4¢] and [2¢, 3¢] are equally-optimal solutions
- You should be happy receiving any such solution
  - If not, you need to fix your objective function!

Revisiting the Change Problem (USA Currency)

- **C** = {1¢ (penny), 5¢ (nickel), 10¢ (dime), 25¢ (quarter)}
- Imagine I owe you 42¢. How should I choose the coins to give you?

Let's solve the problem!

Revisiting the Change Problem (USA Currency)

```
Algorithm change_USA(x,C):
    change ← empty list
    For each coin c in C (descending order):
        While x >= c:
            Add c to change
            x ← x - c
    Return change
```

Revisiting the Change Problem (USA Currency)

```
Algorithm change_USA(x,C):
    change ← empty list
```

Does this work for *any arbitrary* currency?

```
            Add c to change
            x ← x - c
    Return change
```

Global vs. Local Search

- There may be *many* (even infinite!) possible solutions to our problem

Global vs. Local Search

- There may be *many* (even infinite!) possible solutions to our problem
    - **Exhaustive:** Simply looking at every possible solution

Global vs. Local Search

- There may be *many* (even infinite!) possible solutions to our problem
    - **Exhaustive:** Simply looking at every possible solution
- When we try to cleverly search for an optimal solution more quickly:

Global vs. Local Search

- There may be *many* (even infinite!) possible solutions to our problem
    - **Exhaustive:** Simply looking at every possible solution
- When we try to cleverly search for an optimal solution more quickly:
    - **Global:** We can look at entire solutions at a time

Global vs. Local Search

- There may be *many* (even infinite!) possible solutions to our problem
    - **Exhaustive:** Simply looking at every possible solution
- When we try to cleverly search for an optimal solution more quickly:
    - **Global:** We can look at entire solutions at a time
    - **Local:** We can break solutions into parts and optimize part-by-part

Local Search: The Greedy Method

- **Greedy Method:** Selecting the best possible choice at each step

Local Search: The Greedy Method

- **Greedy Method:** Selecting the best possible choice at each step

- **Note that this does not always work!!!**

Local Search: The Greedy Method

- **Greedy Method:** Selecting the best possible choice at each step
- **Note that this does not always work!!!**
  - We often skip what's immediately best to improve in the long-run

Local Search: The Greedy Method

- **Greedy Method:** Selecting the best possible choice at each step
- **Note that this does not always work!!!**
    - We often skip what's immediately best to improve in the long-run
    - Example: Buying vs. leasing a car

Local Search: The Greedy Method

- **Greedy Method:** Selecting the best possible choice at each step
- **Note that this does not always work!!!**
    - We often skip what's immediately best to improve in the long-run
    - Example: Buying vs. leasing a car
- Thus, it's important to prove the correctness of a Greedy Algorithm

Revisiting the Change Problem

- **C** = {1¢, 3¢, 4¢}

Revisiting the Change Problem

- **C** = {1¢, 3¢, 4¢}
- Imagine I owe you 6¢. How should I choose the coins to give you?

Revisiting the Change Problem

- **C** = {1¢, 3¢, 4¢}
- Imagine I owe you 6¢. How should I choose the coins to give you?
    - The greedy algorithm would return [4¢, 1¢, 1¢]

Revisiting the Change Problem

- **C** = {1¢, 3¢, 4¢}
- Imagine I owe you 6¢. How should I choose the coins to give you?
  - The greedy algorithm would return [4¢, 1¢, 1¢]
  - The optimal solution is [3¢, 3¢]

Revisiting the Change Problem

- **C** = {1¢, 3¢, 4¢}
- Imagine I owe you 6¢. How should I choose the coins to give you?
  - The greedy algorithm would return [4¢, 1¢, 1¢]
  - The optimal solution is [3¢, 3¢]
  - **Our greedy algorithm doesn't work for all possible currencies!!!**

Immediate Benefit vs. Opportunity Cost

Immediate Benefit vs. Opportunity Cost

- **Immediate Benefit:** How much do I gain from this choice?

Immediate Benefit vs. Opportunity Cost

- **Immediate Benefit:** How much do I gain from this choice?
- **Opportunity Cost:** How much is the future restricted by this choice?

Immediate Benefit vs. Opportunity Cost

- **Immediate Benefit:** How much do I gain from this choice?
- **Opportunity Cost:** How much is the future restricted by this choice?
- Greedy: Take the best immediate benefit and ignore opportunity costs

Immediate Benefit vs. Opportunity Cost

- **Immediate Benefit:** How much do I gain from this choice?
- **Opportunity Cost:** How much is the future restricted by this choice?
- Greedy: Take the best immediate benefit and ignore opportunity costs
    - Optimal when immediate benefit outweighs opportunity costs

Example: The Event Scheduling Problem

- Imagine you own an event room, and you want to schedule events

Example: The Event Scheduling Problem

- Imagine you own an event room, and you want to schedule events
  - You charge a flat rate, regardless of the length of the event

Example: The Event Scheduling Problem

- Imagine you own an event room, and you want to schedule events
    - You charge a flat rate, regardless of the length of the event
    - Thus, you want to schedule as many events as possible

Example: The Event Scheduling Problem

- Imagine you own an event room, and you want to schedule events
  - You charge a flat rate, regardless of the length of the event
  - Thus, you want to schedule as many events as possible
  - However, events cannot overlap

Example: The Event Scheduling Problem

- **Input:** All $\boldsymbol{n}$ possible events $\boldsymbol{E} = [(start_1, end_1), \ldots, (start_{\boldsymbol{n}}, end_{\boldsymbol{n}})]$

Example: The Event Scheduling Problem

- **Input:** All $\boldsymbol{n}$ possible events $\boldsymbol{E} = [(start_1, end_1), \ldots, (start_{\boldsymbol{n}}, end_{\boldsymbol{n}})]$
- **Output:** A non-overlapping subset of $\boldsymbol{E}$ maximizing its size

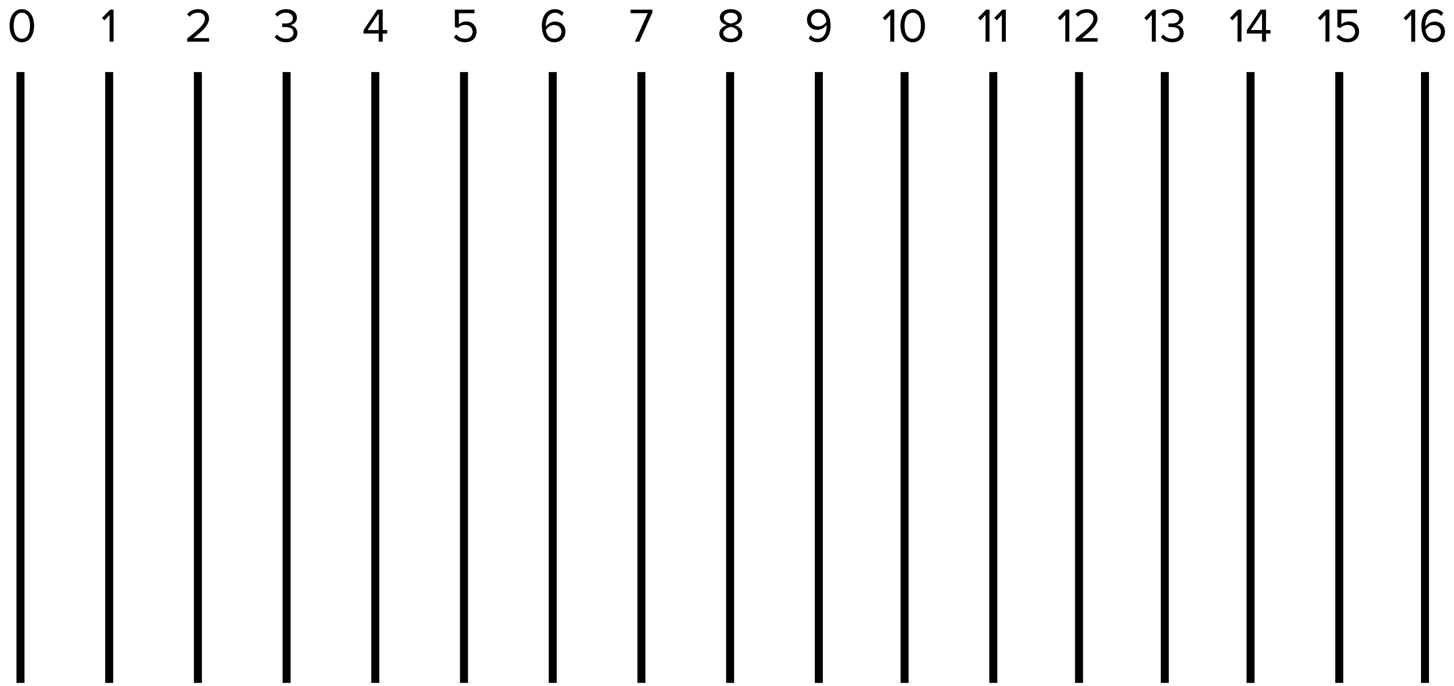
Example: The Event Scheduling Problem

- **Input:** All ***n*** possible events ***E*** = [($start_1$, end_1), ..., ($start_n$, end_n)]
- **Output:** A non-overlapping subset of ***E*** maximizing its size
- If we wanted to design a greedy algorithm, what would we optimize?

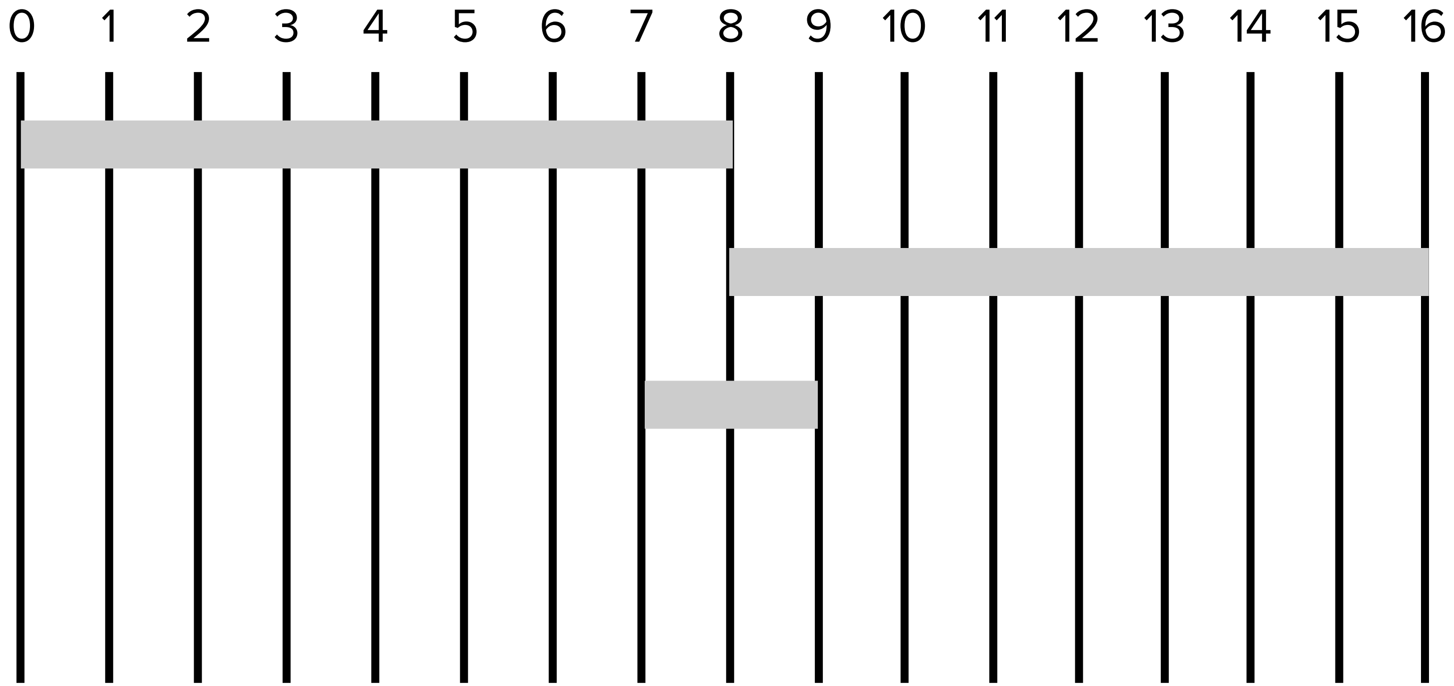
Example: The Event Scheduling Problem

- **Input:** All $\boldsymbol{n}$ possible events $\boldsymbol{E}$ = [($start_1$, end_1), ..., ($start_{\boldsymbol{n}}$, $end_{\boldsymbol{n}}$)]
- **Output:** A non-overlapping subset of $\boldsymbol{E}$ maximizing its size
- If we wanted to design a greedy algorithm, what would we optimize?
  - Shortest duration?
  - Earliest start time?
  - Fewest conflicts?
  - Earliest end time?

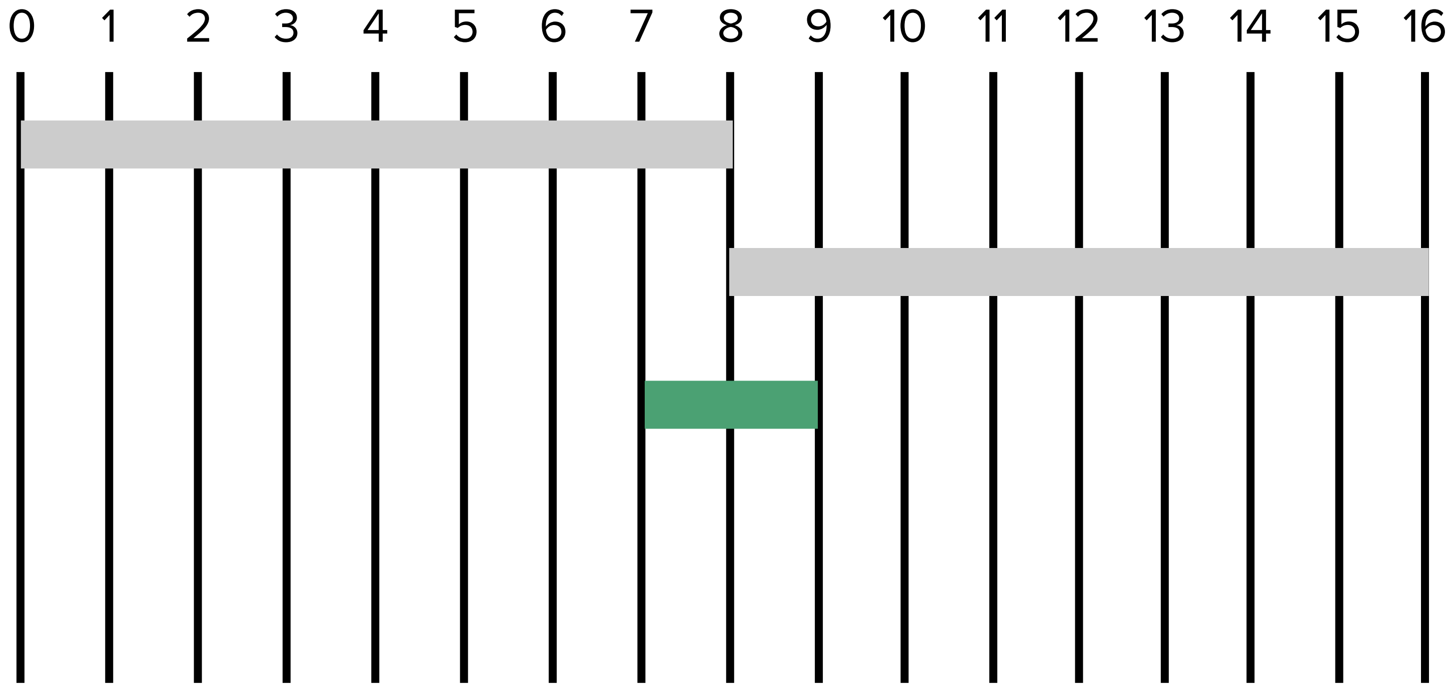
Counterexample: Shortest Duration



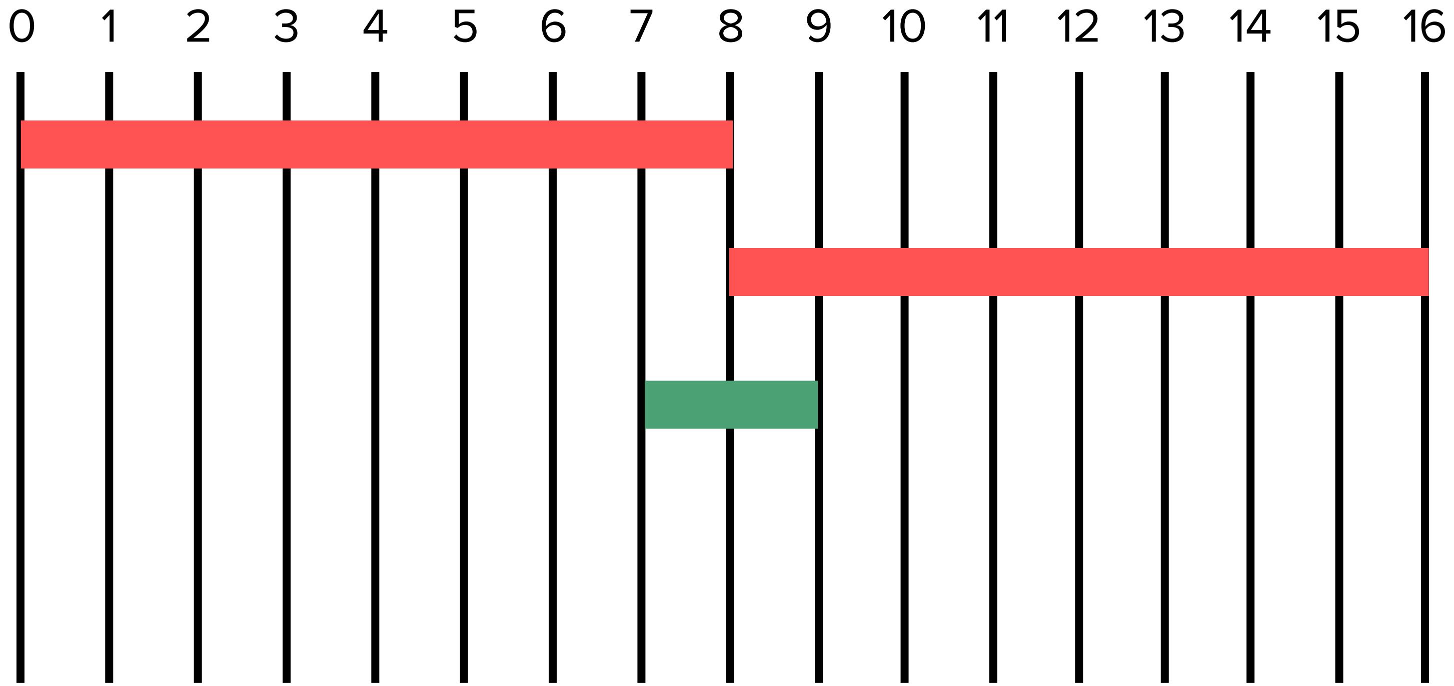
Counterexample: Shortest Duration



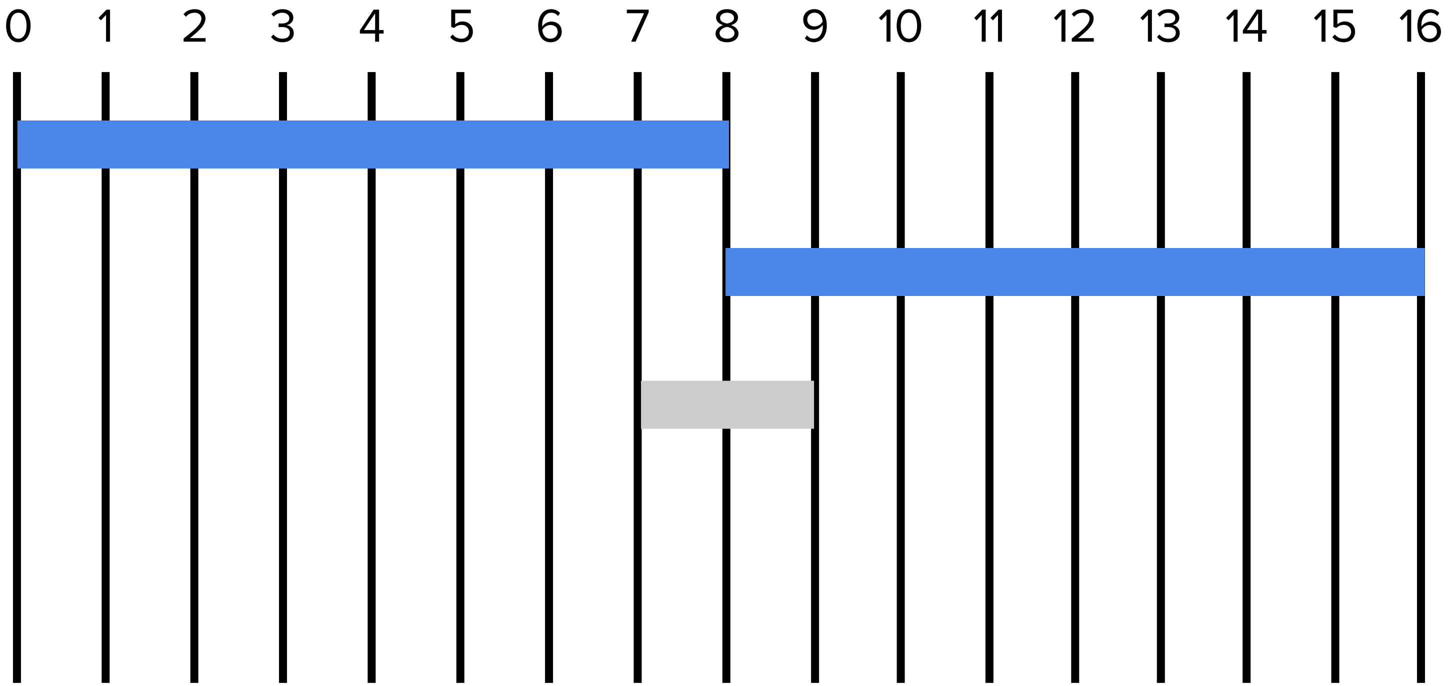
Counterexample: Shortest Duration



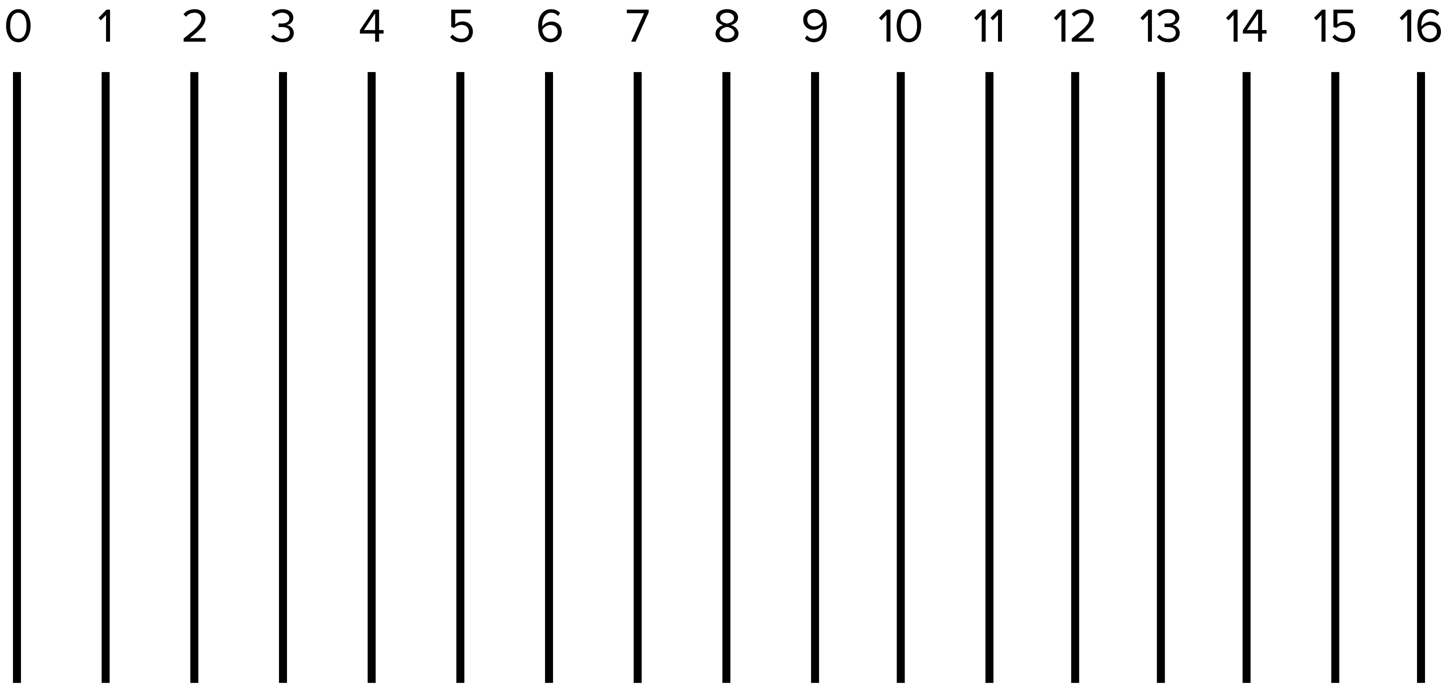
Counterexample: Shortest Duration



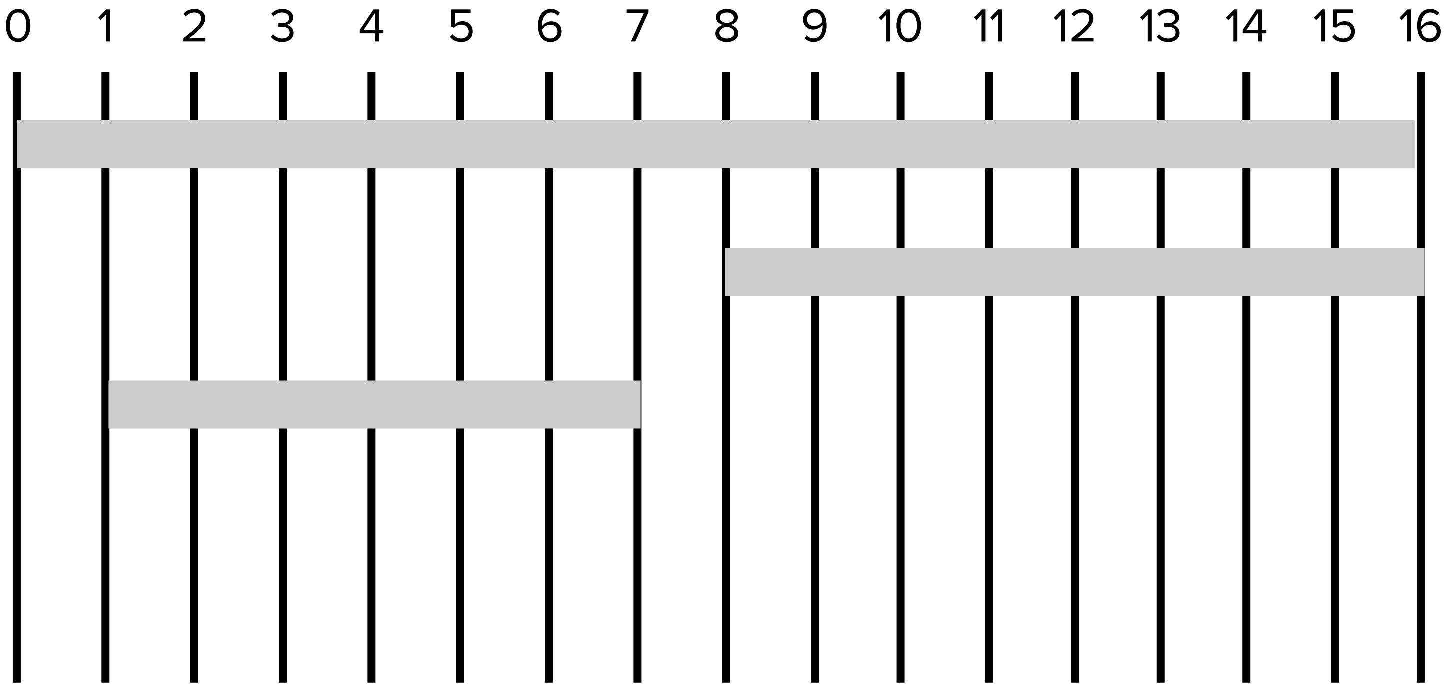
Counterexample: Shortest Duration



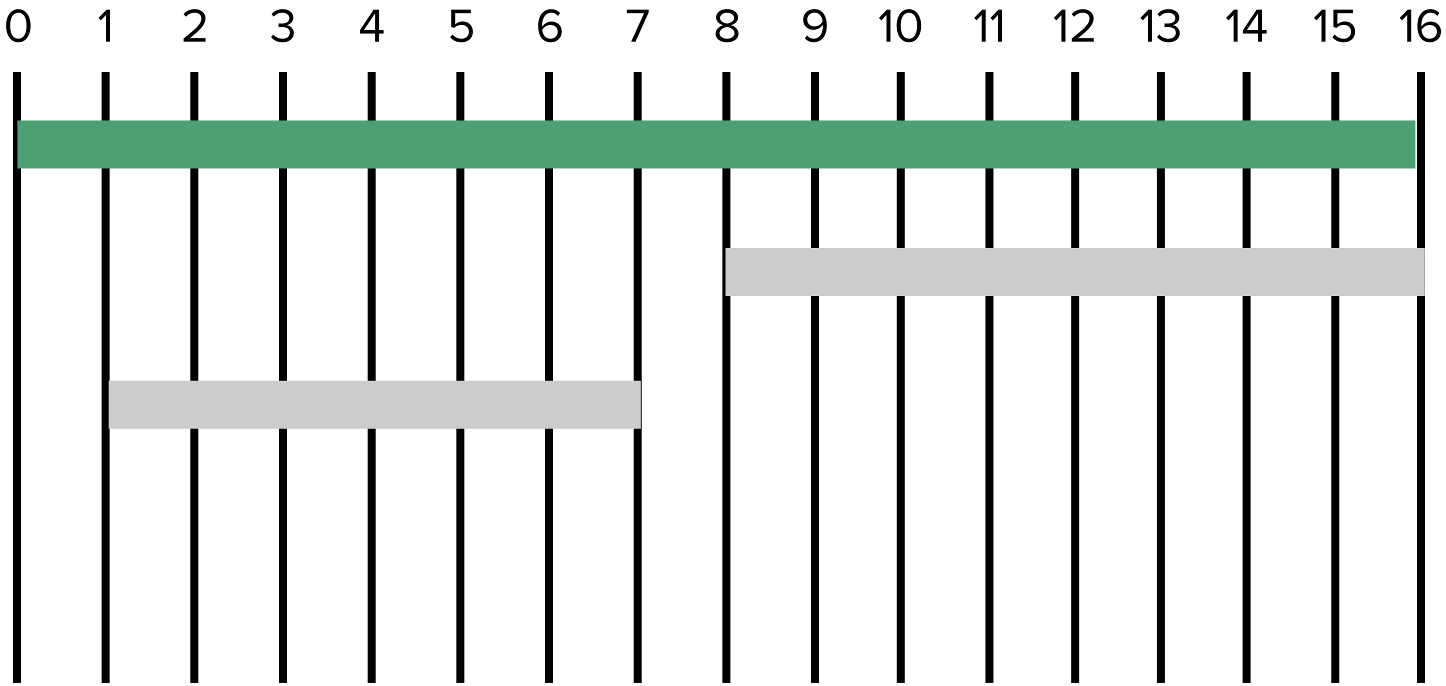
Counterexample: Earliest Start Time



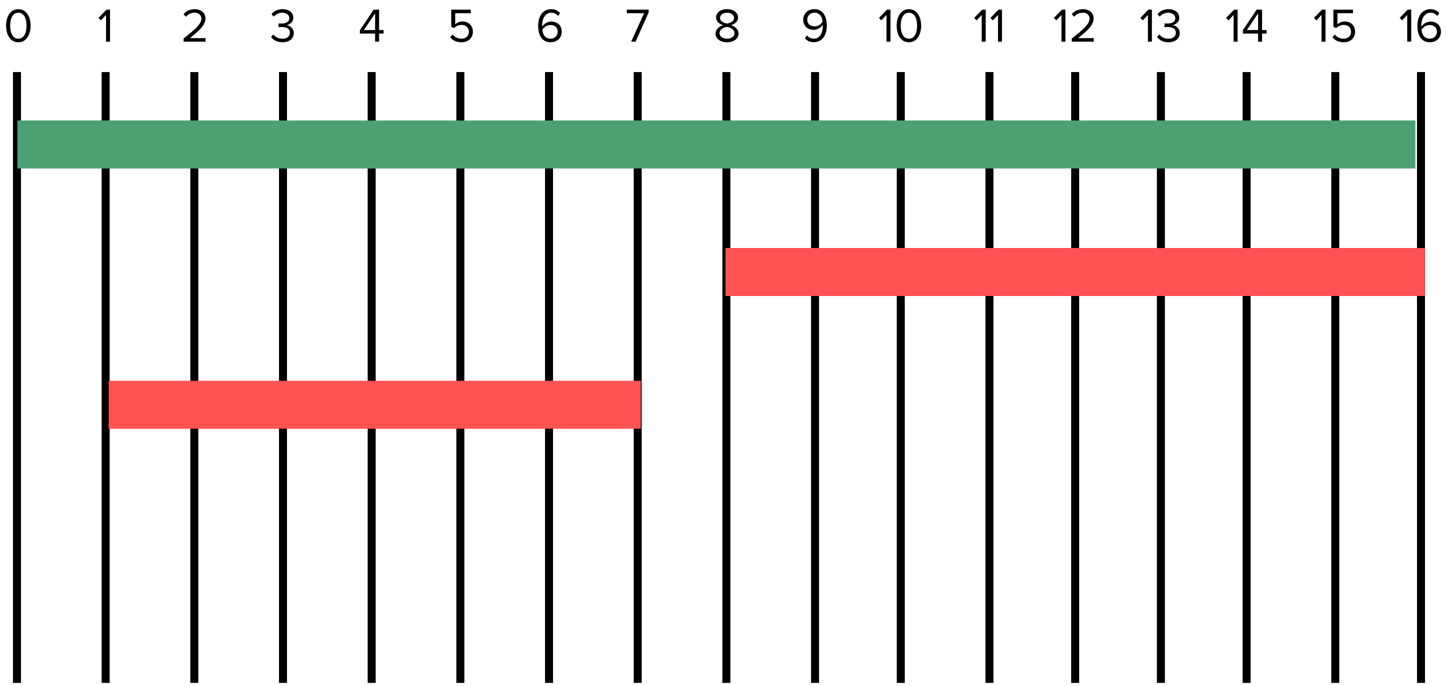
Counterexample: Earliest Start Time



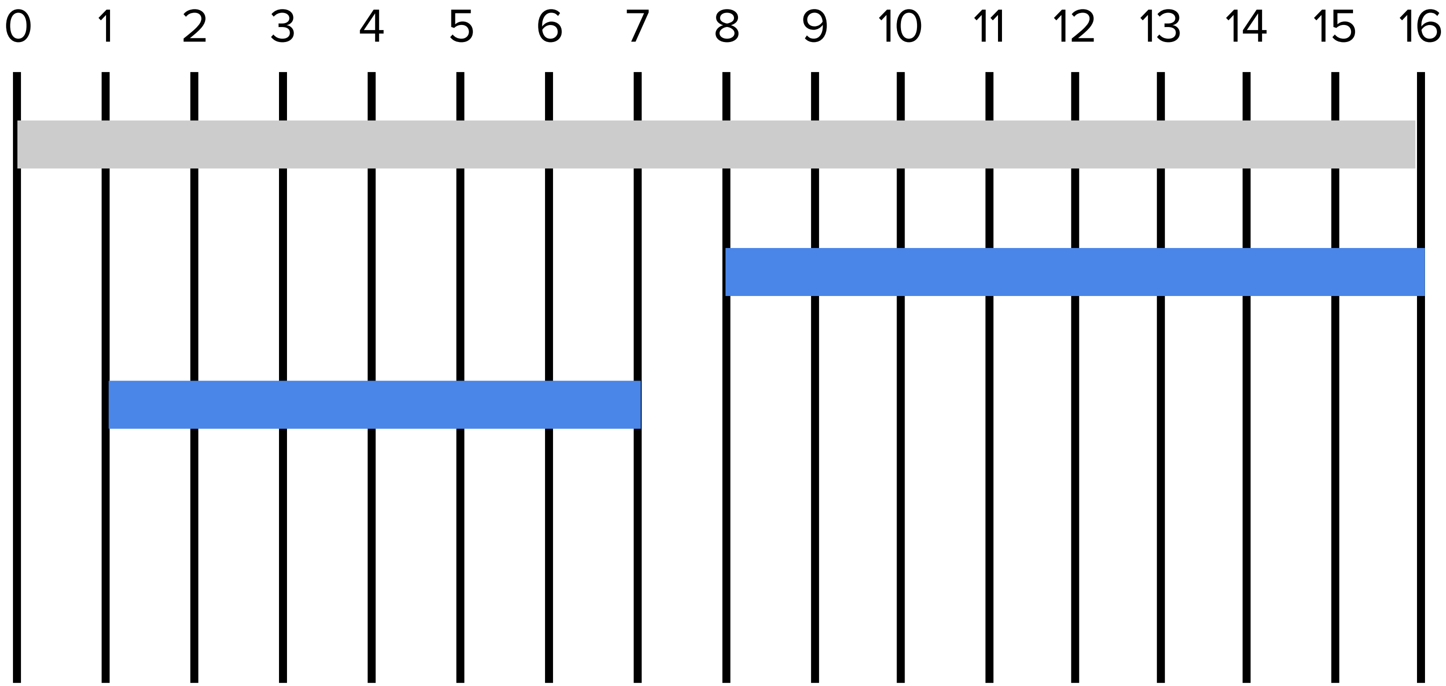
Counterexample: Earliest Start Time



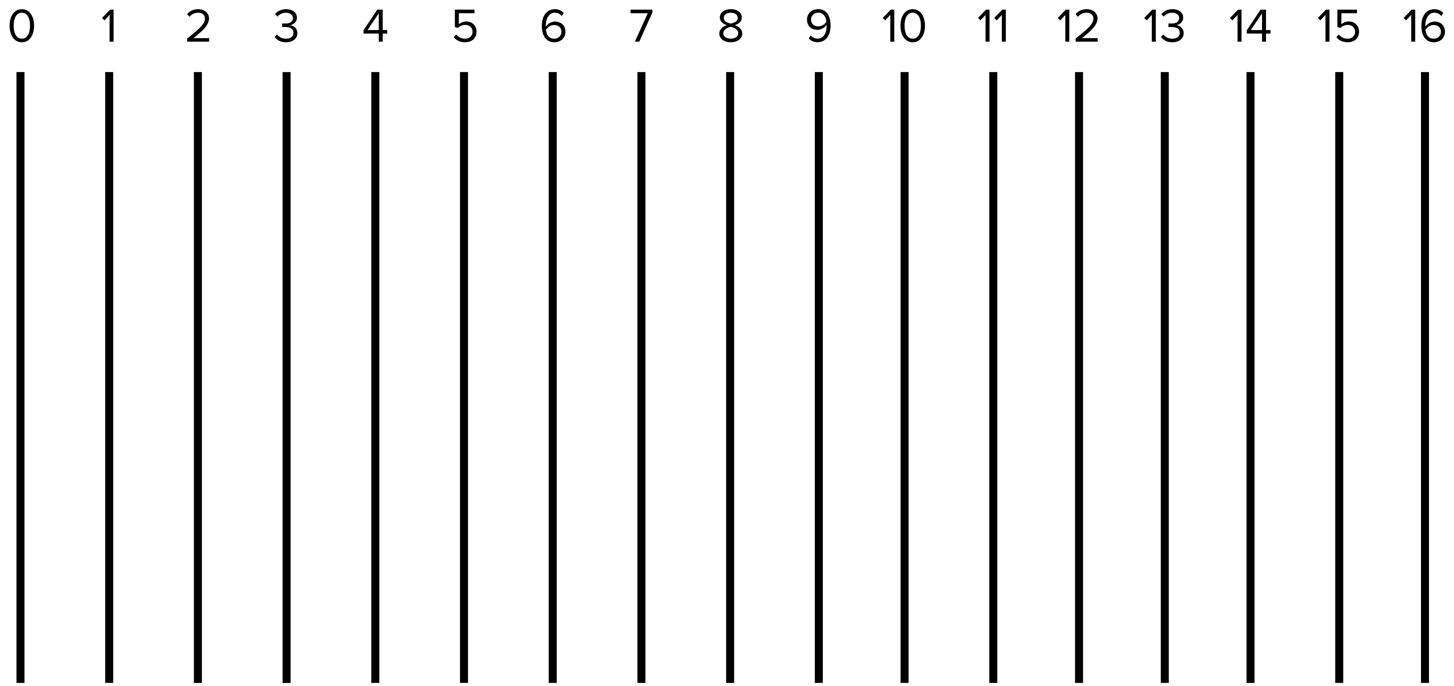
Counterexample: Earliest Start Time



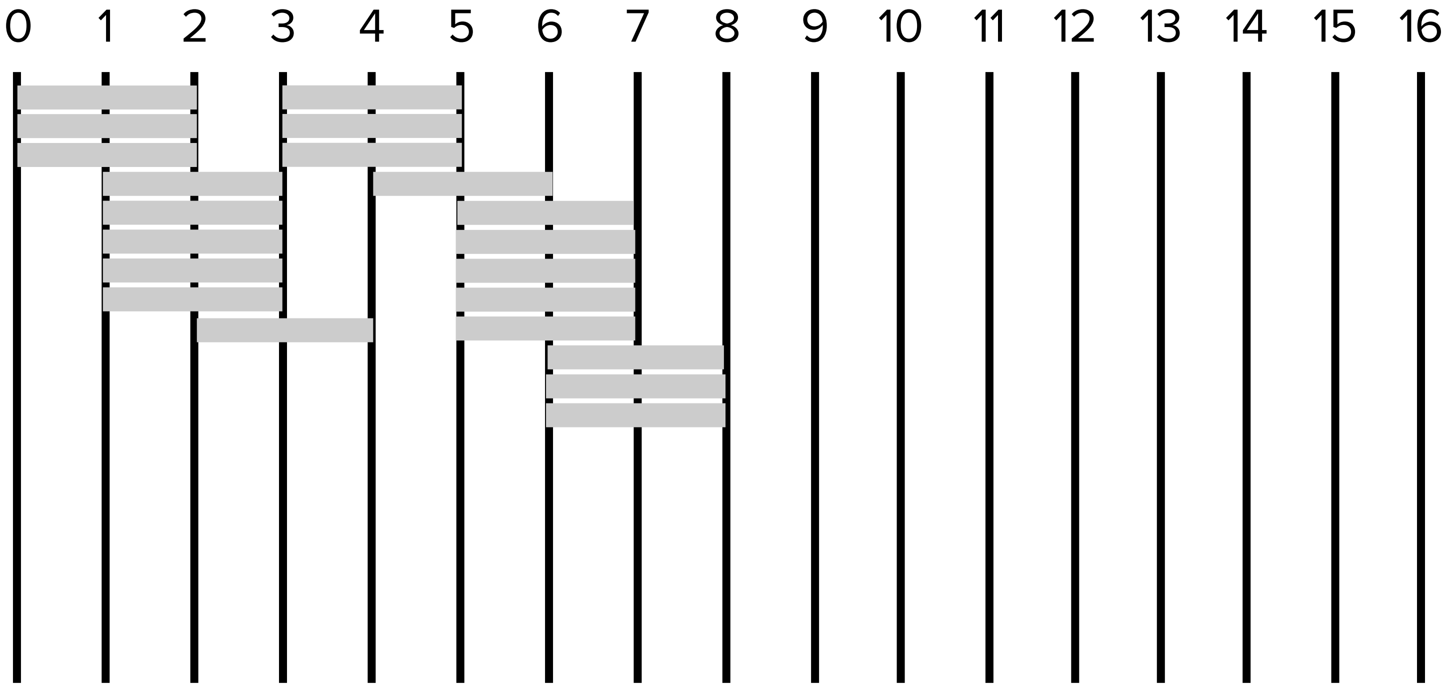
Counterexample: Earliest Start Time



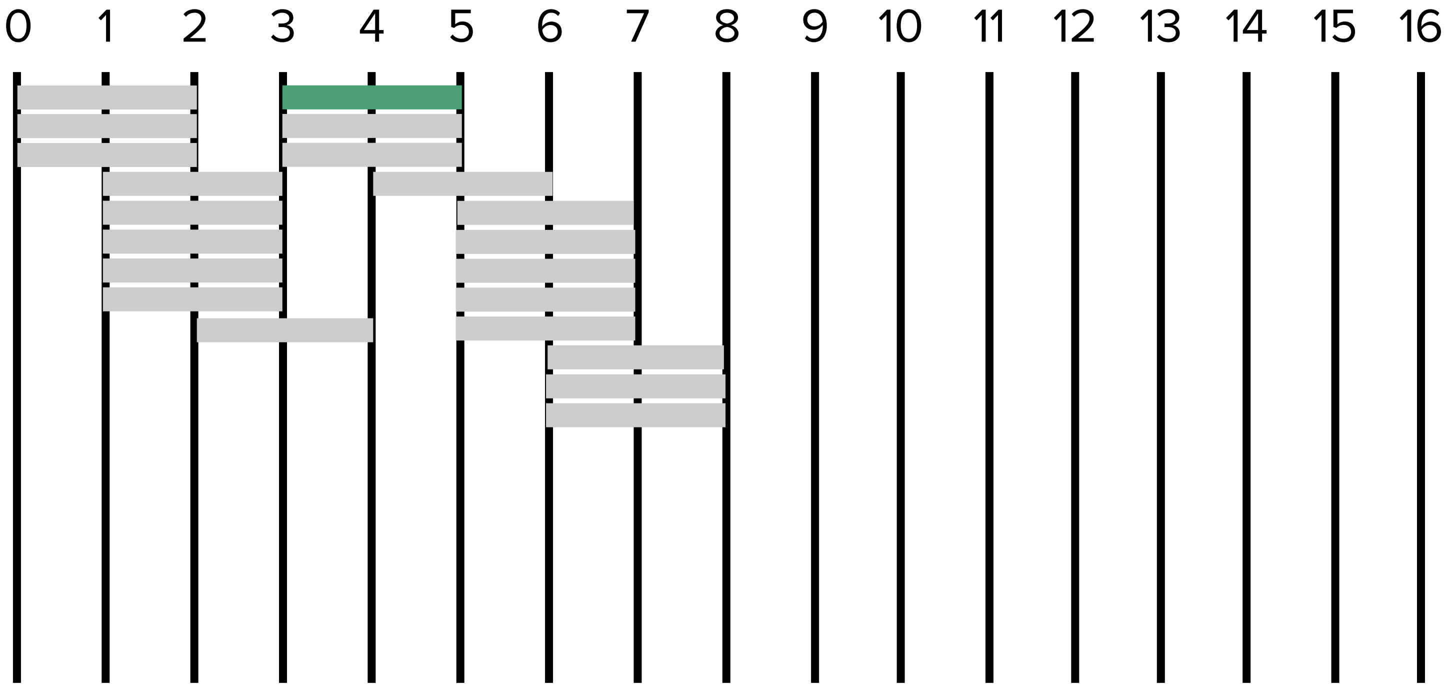
Counterexample: Fewest Conflicts



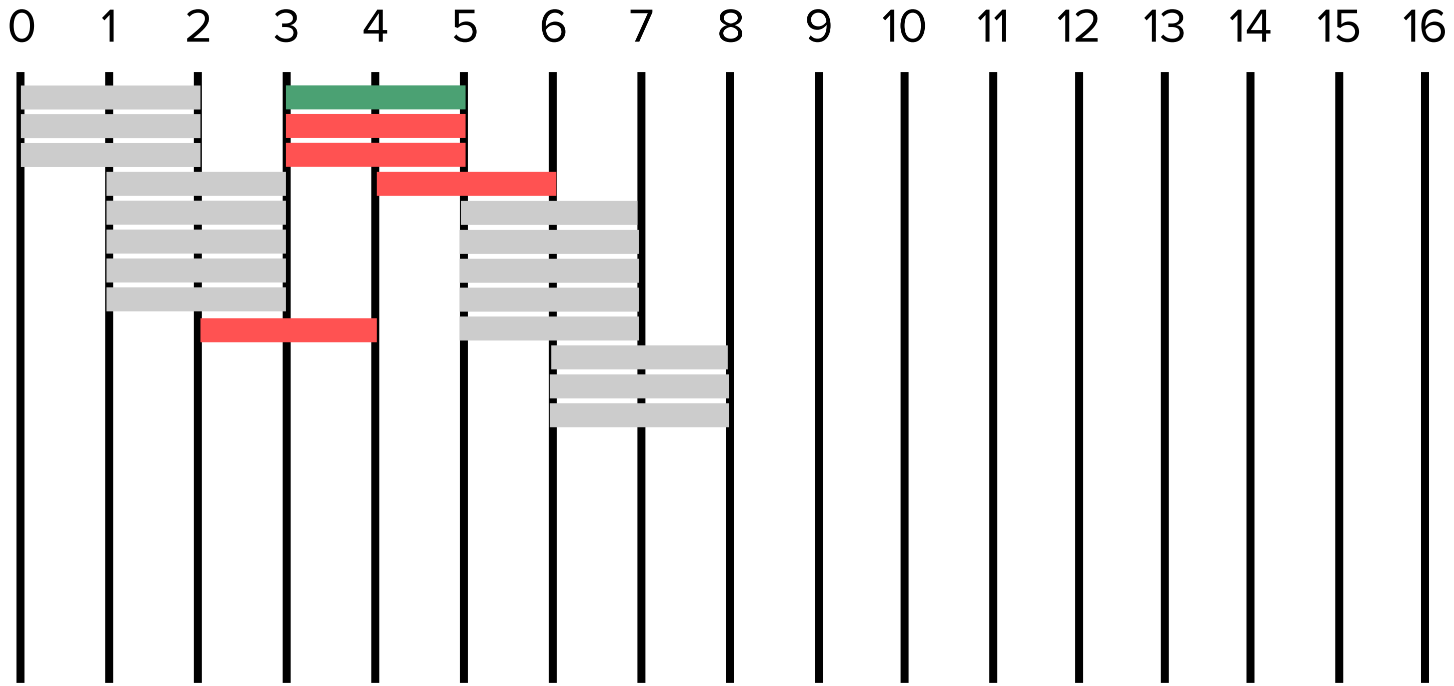
Counterexample: Fewest Conflicts



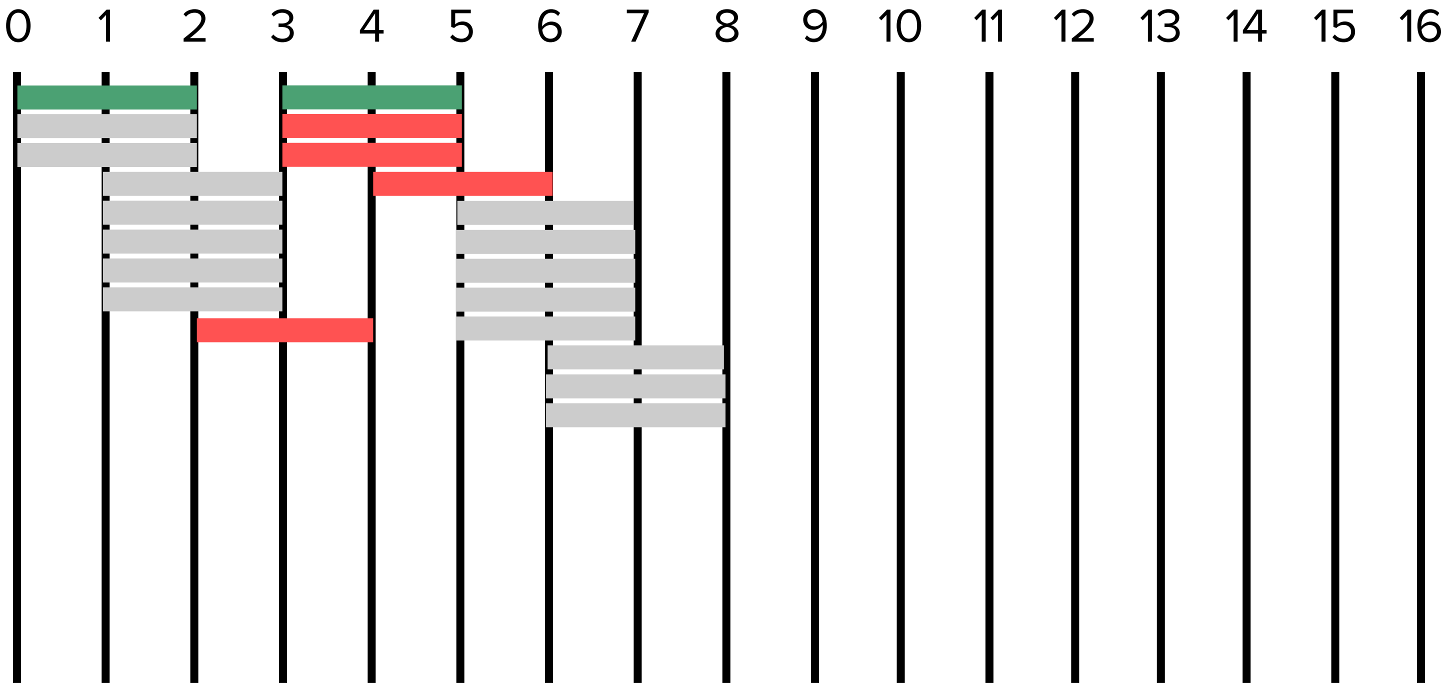
Counterexample: Fewest Conflicts



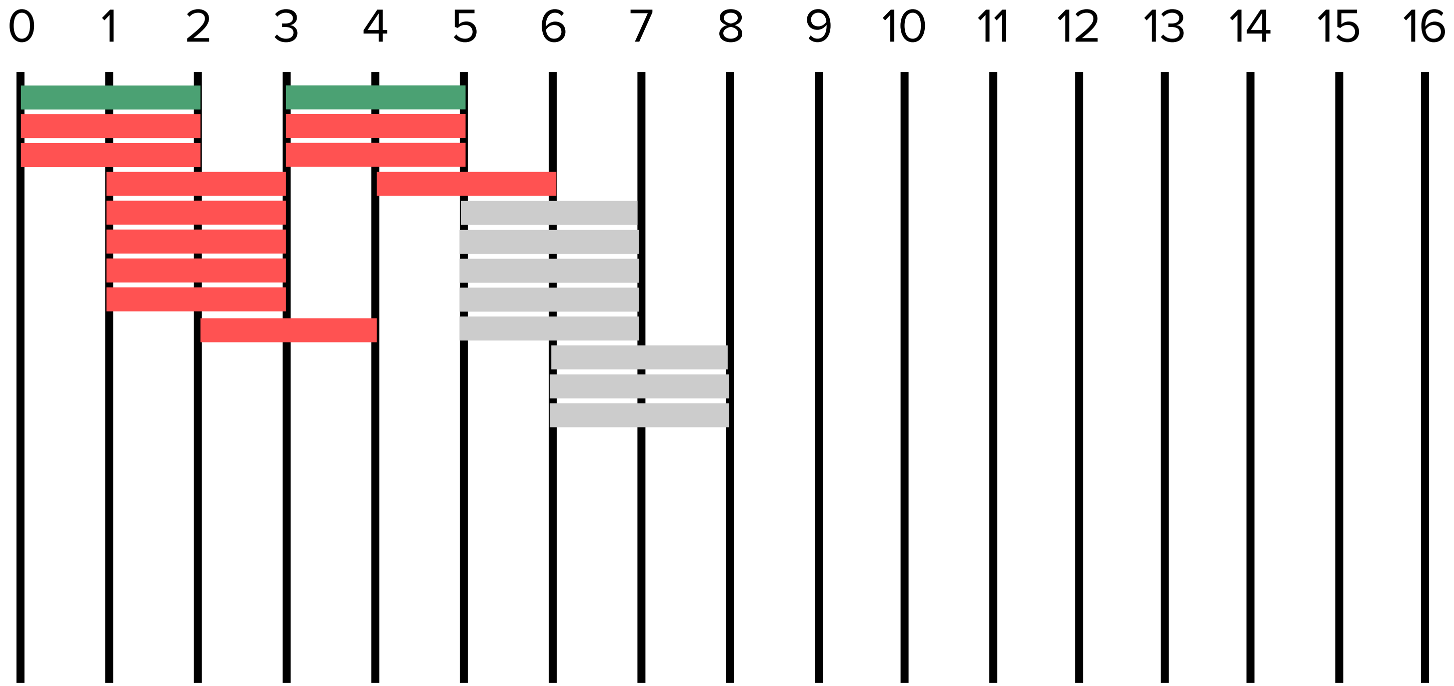
Counterexample: Fewest Conflicts



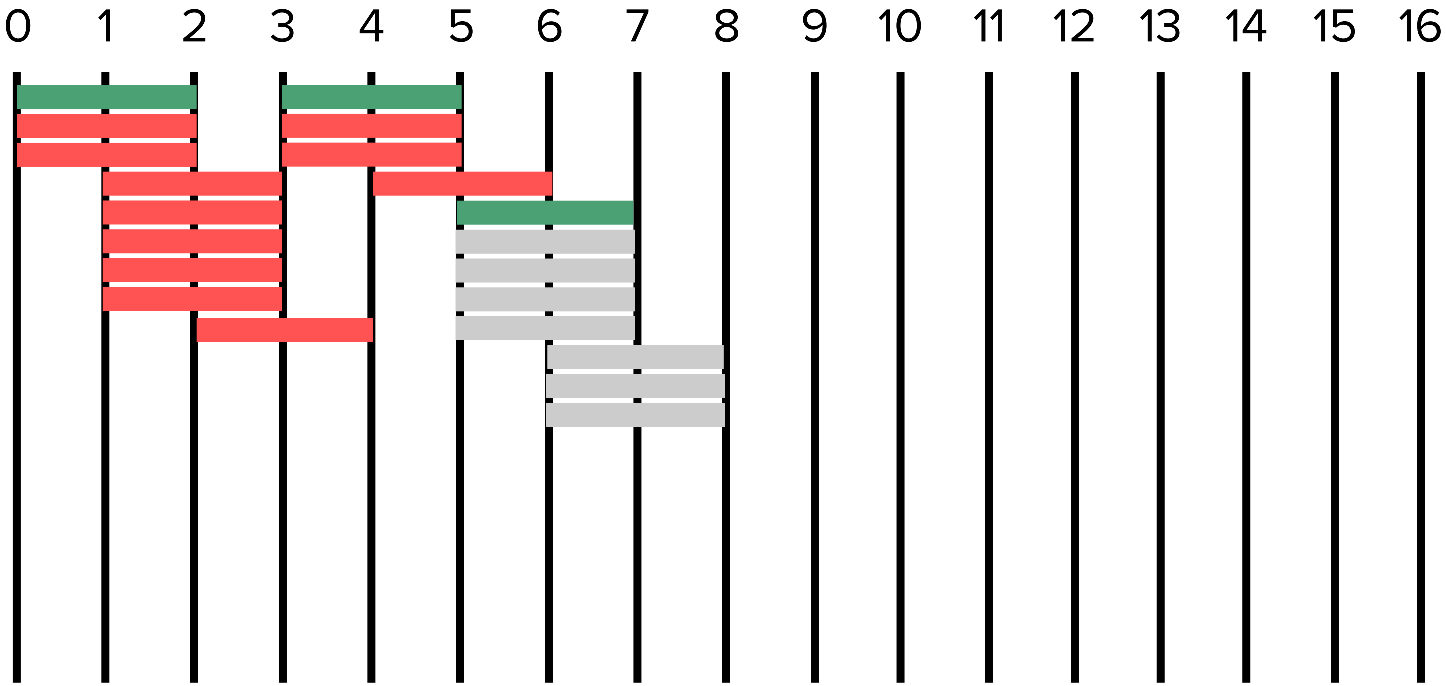
Counterexample: Fewest Conflicts



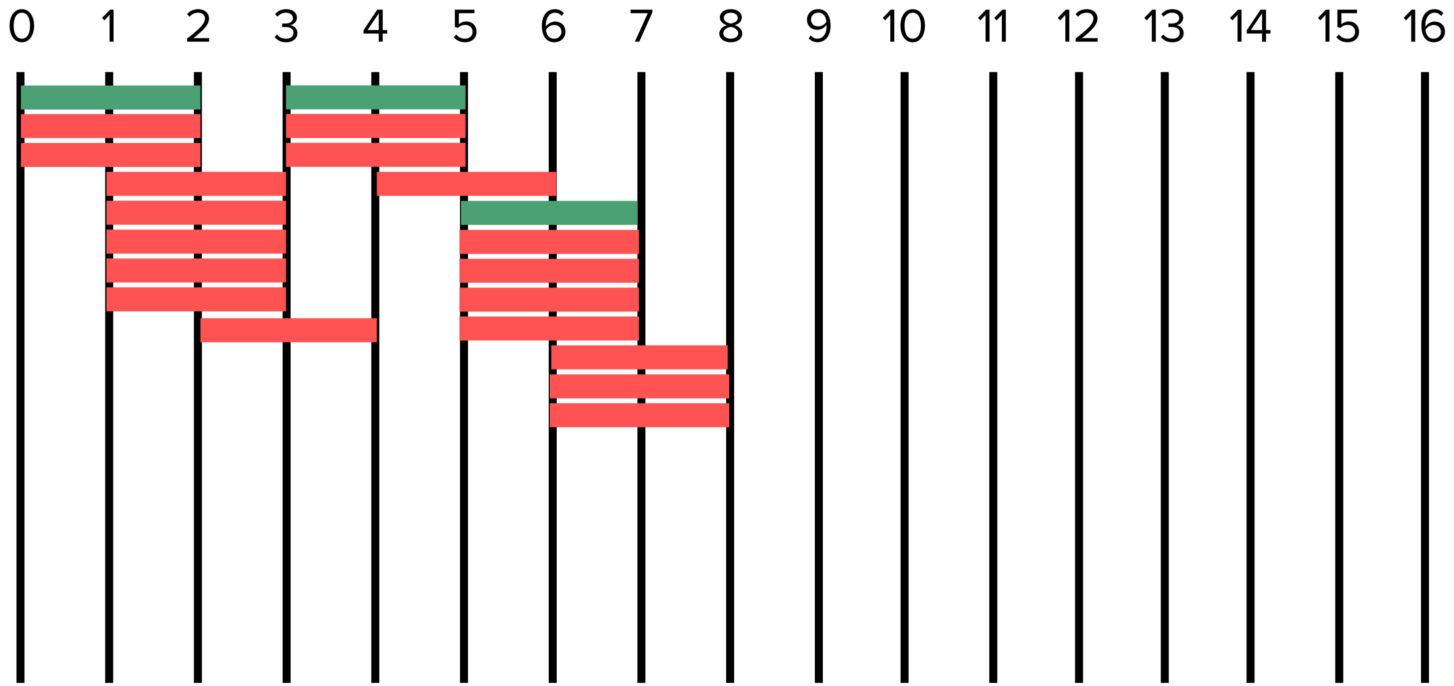
Counterexample: Fewest Conflicts



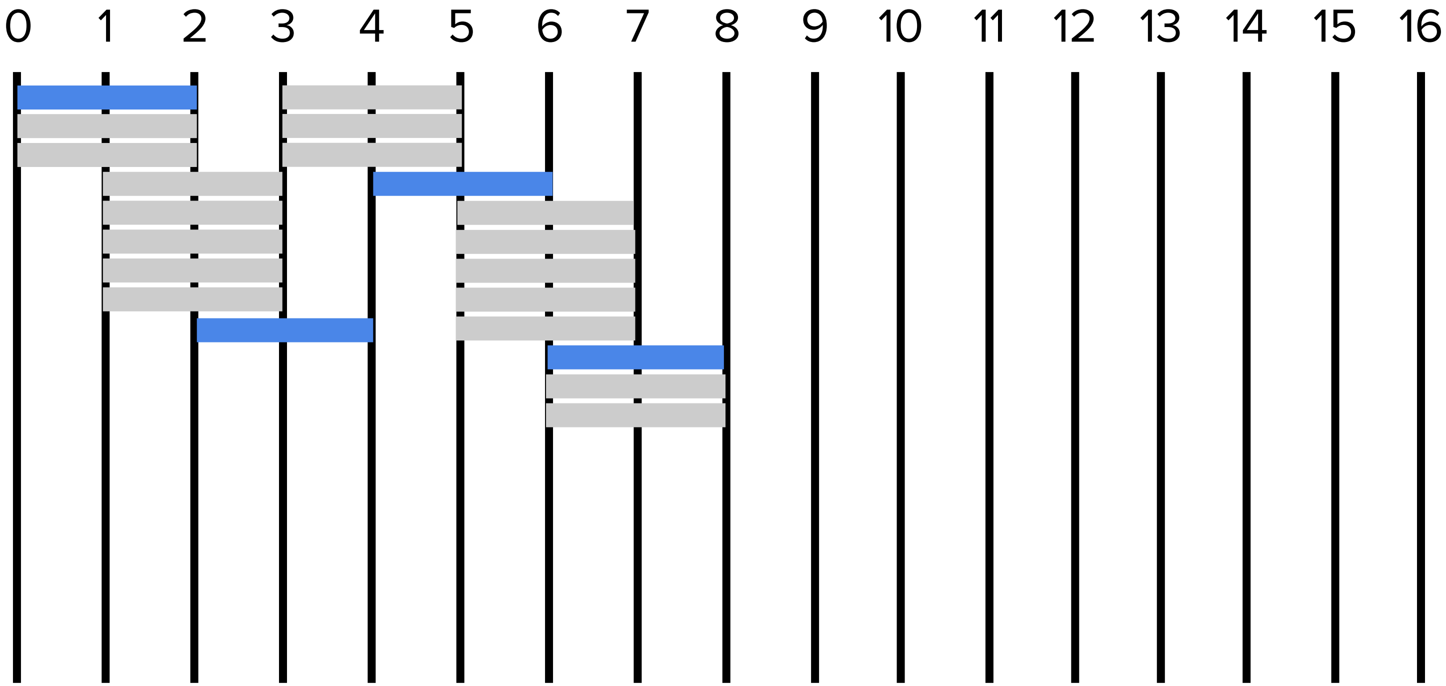
Counterexample: Fewest Conflicts



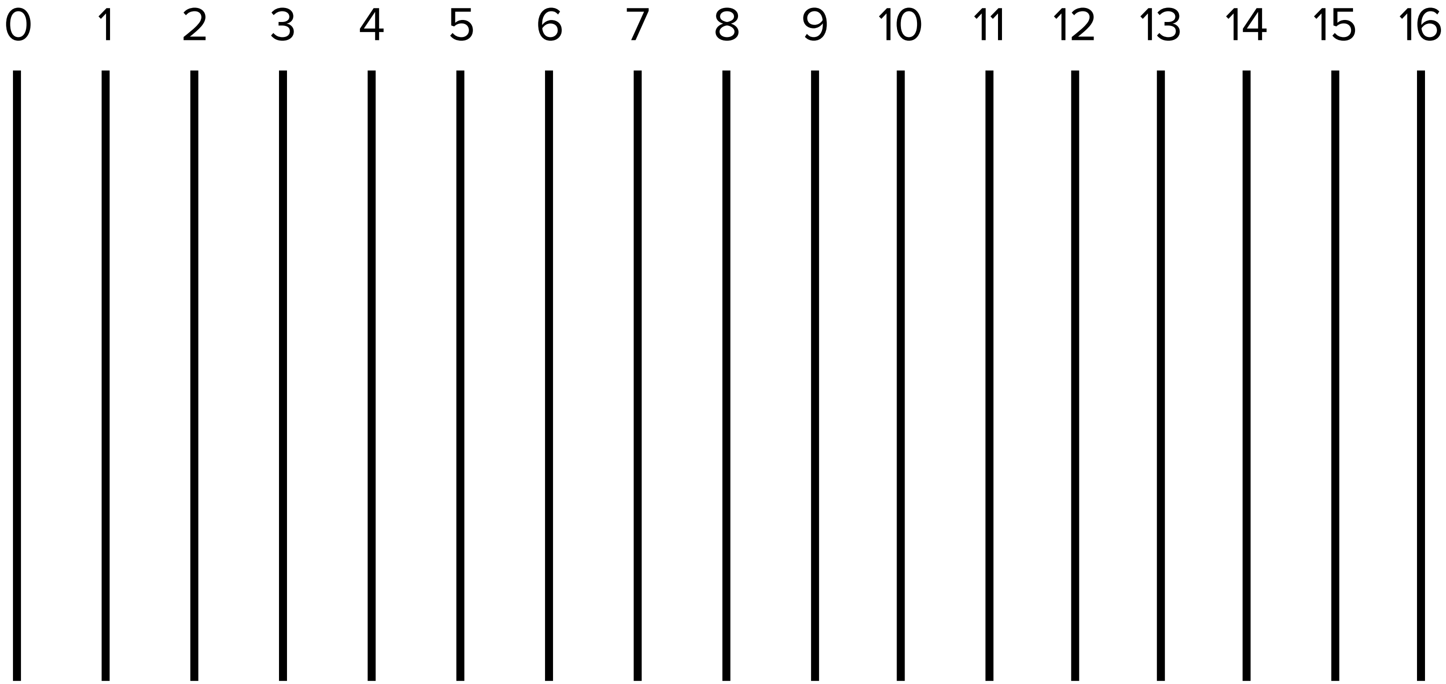
Counterexample: Fewest Conflicts



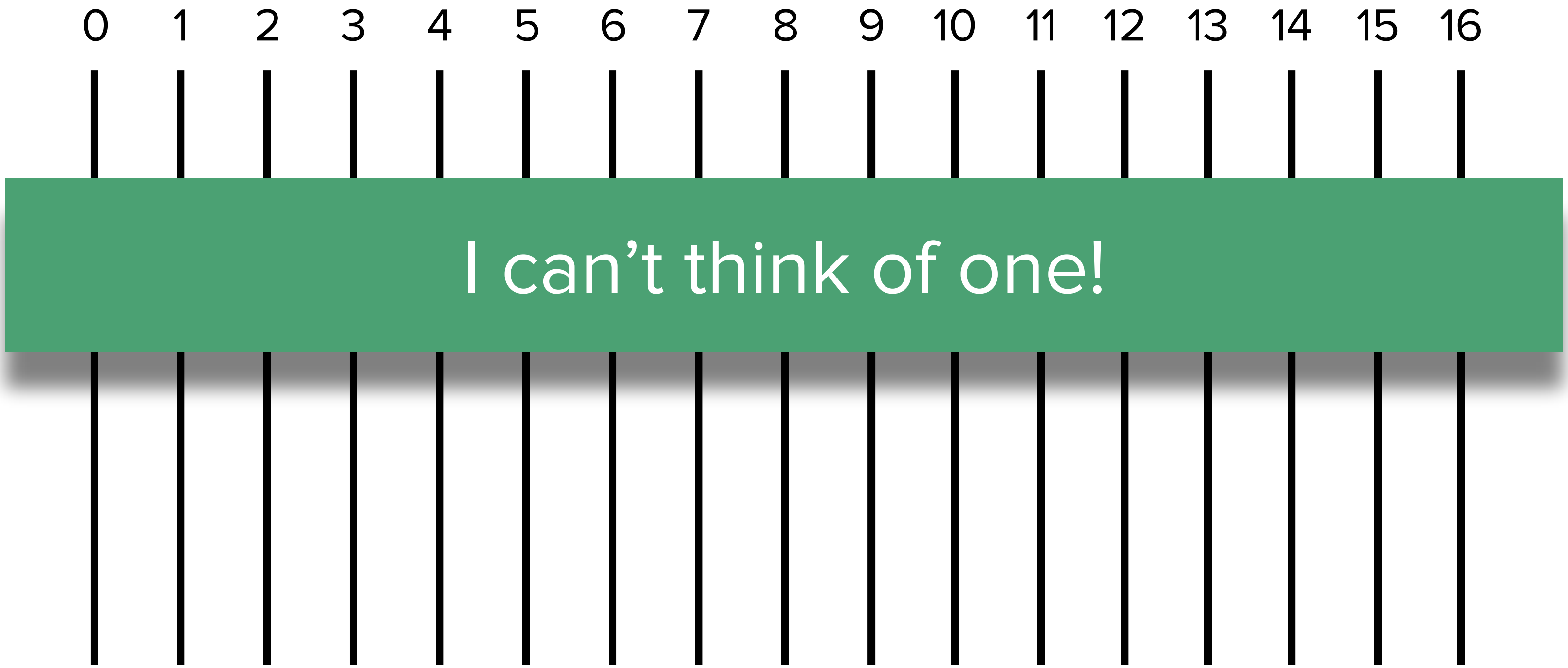
Counterexample: Fewest Conflicts



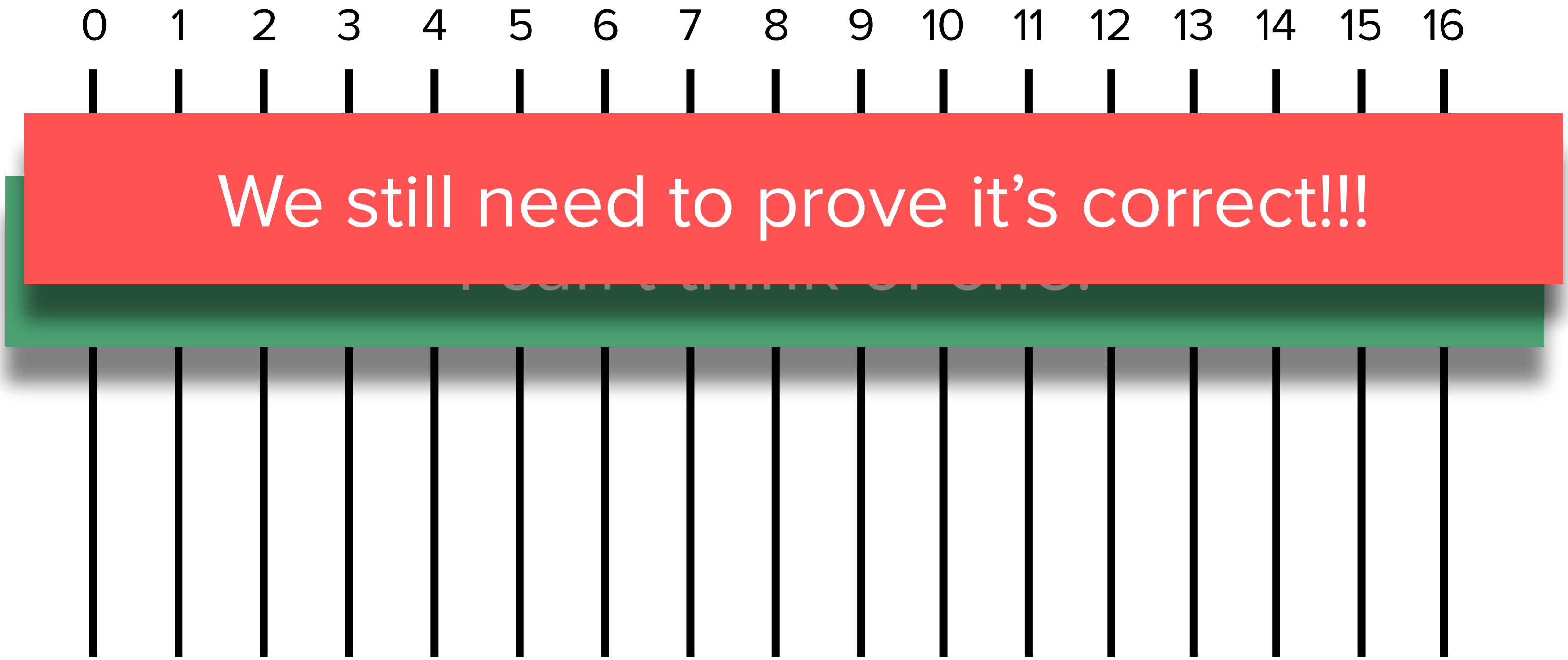
Counterexample: Earliest End Time



Counterexample: Earliest End Time



Counterexample: Earliest End Time



We still need to prove it's correct!!!

Example: The Event Scheduling Problem

```
Algorithm schedule(E):
    Sort E in ascending order of end time
    curr_time ← negative infinity
    events ← empty list
    For each event (start,end) in E:
        If start ≥ curr_time:
            Add (start,end) to events
            curr_time ← end
    Return events
```

Proofs: The Exchange Argument

- Common approach for proving greedy algorithms

Proofs: The Exchange Argument

- Common approach for proving greedy algorithms
    - Let *g* be the first greedy choice

Proofs: The Exchange Argument

- Common approach for proving greedy algorithms
  - Let g be the first greedy choice
  - Let S be any optimal solution that does not include g

Proofs: The Exchange Argument

- Common approach for proving greedy algorithms
  - Let g be the first greedy choice
  - Let S be any optimal solution that does not include g
  - Create S' by exchanging a choice in S with g and show that

Proofs: The Exchange Argument

- Common approach for proving greedy algorithms
  - Let *g* be the first greedy choice
  - Let *S* be any optimal solution that does not include *g*
  - Create *S*' by exchanging a choice in *S* with *g* and show that
    - *S*' is a valid solution

Proofs: The Exchange Argument

- Common approach for proving greedy algorithms
    - Let *g* be the first greedy choice
    - Let *S* be any optimal solution that does not include *g*
    - Create *S*’ by exchanging a choice in *S* with *g* and show that
        - *S*’ is a valid solution
        - *S*’ is just as good, or better than, *S*

Proofs: The Exchange Argument

- Common approach for proving greedy algorithms
  - Let *g* be the first greedy choice

Let's try to prove our algorithm!

  - Create *S*' by exchanging a choice in *S* with *g* and show that
    - *S*' is a valid solution
    - *S*' is just as good, or better than, *S*