# Greedy Algorithms

# Greedy Algorithms

- Usually applied to Optimization Problems:

  *Find a configuration that minimizes or maximizes an objective function.*

# Greedy Algorithms

- Usually applied to Optimization Problems:

  *Find a configuration that minimizes or maximizes an objective function.*

- A greedy algorithm. . .

# Greedy Algorithms

- Usually applied to Optimization Problems:

  *Find a configuration that minimizes or maximizes an objective function.*

- A greedy algorithm...
  - Builds up a solution in small steps

# Greedy Algorithms

- Usually applied to Optimization Problems:

  *Find a configuration that minimizes or maximizes an objective function.*

- A greedy algorithm...
  - Builds up a solution in small steps
  - At each step, makes its decision on some local criterion, myopically ignoring the global state

# Greedy Algorithms

- Usually applied to Optimization Problems:

  *Find a configuration that minimizes or maximizes an objective function.*

- A greedy algorithm. . .
  - Builds up a solution in small steps
  - At each step, makes its decision on some local criterion, myopically ignoring the global state
- Greedy algorithms that implement a greedy heuristic are easy. However,. . .

# Greedy Algorithms

- Usually applied to Optimization Problems:

  *Find a configuration that minimizes or maximizes an objective function.*

- A greedy algorithm...
  - Builds up a solution in small steps
  - At each step, makes its decision on some local criterion, myopically ignoring the global state
- Greedy algorithms that implement a greedy heuristic are easy. However,...
- Greedy algorithms that actually obtain a global optimum often do not exist.

# Greedy Algorithms

- Usually applied to Optimization Problems:

  *Find a configuration that minimizes or maximizes an objective function.*

- A greedy algorithm. . .
  - Builds up a solution in small steps
  - At each step, makes its decision on some local criterion, myopically ignoring the global state
- Greedy algorithms that implement a greedy heuristic are easy. However,. . .
- Greedy algorithms that actually obtain a global optimum often do not exist.
- Greedy algorithms are usually simple, but the proof of correctness may not be so simple.

# Greedy Algorithms

- Usually applied to Optimization Problems:

  *Find a configuration that minimizes or maximizes an objective function.*

- A greedy algorithm. . .
  - Builds up a solution in small steps
  - At each step, makes its decision on some local criterion, myopically ignoring the global state
- Greedy algorithms that implement a greedy heuristic are easy. However,. . .
- Greedy algorithms that actually obtain a global optimum often do not exist.
- Greedy algorithms are usually simple, but the proof of correctness may not be so simple.
- We will give several examples of greedy algorithms.

# Greedy Algorithm 1: Fractional knapsack problem

# Greedy Algorithm 1: Fractional knapsack problem

- Informal description:

# Greedy Algorithm 1: Fractional knapsack problem

- Informal description:
  - There are $n$ objects. Each object has a given weight and a given value.

# Greedy Algorithm 1: Fractional knapsack problem

- Informal description:
  - There are $n$ objects. Each object has a given weight and a given value.
  - There is a knapsack that can hold a given maximum weight.

# Greedy Algorithm 1: Fractional knapsack problem

- Informal description:
  - There are $n$ objects. Each object has a given weight and a given value.
  - There is a knapsack that can hold a given maximum weight.
  - Find the most valuable collection of objects that can fit into the knapsack.

# Greedy Algorithm 1: Fractional knapsack problem

- Informal description:
  - There are $n$ objects. Each object has a given weight and a given value.
  - There is a knapsack that can hold a given maximum weight.
  - Find the most valuable collection of objects that can fit into the knapsack.
- In the fractional version of the problem, we are allowed to break items into fractional portions.

# Greedy Algorithm 1: Fractional knapsack problem

- Informal description:
  - There are $n$ objects. Each object has a given weight and a given value.
  - There is a knapsack that can hold a given maximum weight.
  - Find the most valuable collection of objects that can fit into the knapsack.
- In the fractional version of the problem, we are allowed to break items into fractional portions.
- In the zero-one version of the problem, we either include all of each object or none of it.

# Greedy Algorithm 1: Fractional knapsack problem

- Informal description:
  - There are $n$ objects. Each object has a given weight and a given value.
  - There is a knapsack that can hold a given maximum weight.
  - Find the most valuable collection of objects that can fit into the knapsack.
- In the fractional version of the problem, we are allowed to break items into fractional portions.
- In the zero-one version of the problem, we either include all of each object or none of it.
- For now, we will consider just the fractional version.

# Fractional knapsack problem: Formal problem description

# Fractional knapsack problem: Formal problem description

- Inputs:

# Fractional knapsack problem: Formal problem description

- Inputs:
  - For each object ($i = 1, \ldots, n$): weight $= w_i$, value $= v_i$

# Fractional knapsack problem: Formal problem description

- Inputs:
  - For each object ($i = 1, \ldots, n$): weight $= w_i$, value $= v_i$
  - capacity of knapsack $= W$

# Fractional knapsack problem: Formal problem description

- Inputs:
  - For each object ($i = 1, \ldots, n$): weight $= w_i$, value $= v_i$
  - capacity of knapsack $= W$
- Output: For each object $i$:

# Fractional knapsack problem: Formal problem description

- Inputs:
  - For each object ($i = 1, \ldots, n$): weight $= w_i$, value $= v_i$
  - capacity of knapsack $= W$
- Output: For each object $i$:
  - $x_i =$ the weight of object $i$ that we will put in the knapsack

# Fractional knapsack problem: Formal problem description

- Inputs:
  - For each object ($i = 1, \ldots, n$): weight $= w_i$, value $= v_i$
  - capacity of knapsack $= W$
- Output: For each object $i$:
  - $x_i =$ the weight of object $i$ that we will put in the knapsack
- The goal is to maximize the quantity

$$\sum_{i=1}^{n} \left( \frac{x_i}{w_i} \right) \cdot v_i$$

# Fractional knapsack problem: Formal problem description

- Inputs:
  - For each object ($i = 1, \ldots, n$): weight $= w_i$, value $= v_i$
  - capacity of knapsack $= W$
- Output: For each object $i$:
  - $x_i =$ the weight of object $i$ that we will put in the knapsack
- The goal is to maximize the quantity

$$\sum_{i=1}^{n} \left( \frac{x_i}{w_i} \right) \cdot v_i$$

subject to the conditions

# Fractional knapsack problem: Formal problem description

- Inputs:
  - For each object ($i = 1, \ldots, n$): weight $= w_i$, value $= v_i$
  - capacity of knapsack $= W$
- Output: For each object $i$:
  - $x_i =$ the weight of object $i$ that we will put in the knapsack
- The goal is to maximize the quantity

$$\sum_{i=1}^{n} \left( \frac{x_i}{w_i} \right) \cdot v_i$$

subject to the conditions

$$0 \leq x_i \leq w_i, i = 1, \ldots, n$$

# Fractional knapsack problem: Formal problem description

- Inputs:
  - For each object ($i = 1, \ldots, n$): weight $= w_i$, value $= v_i$
  - capacity of knapsack $= W$
- Output: For each object $i$:
  - $x_i =$ the weight of object $i$ that we will put in the knapsack
- The goal is to maximize the quantity

$$\sum_{i=1}^{n} \left(\frac{x_i}{w_i}\right) \cdot v_i$$

subject to the conditions

$$0 \leq x_i \leq w_i, i = 1, \ldots, n$$

and

$$\sum_{i=1}^{n} x_i \leq W.$$

# Fractional knapsack problem: Example

# Fractional knapsack problem: Example

Suppose we have two objects:

- $w_1 = 80, v_1 = 80$
- $w_2 = 25, v_2 = 50$
- $W = 100$

# Fractional knapsack problem: Example

Suppose we have two objects:

- $w_1 = 80, v_1 = 80$
- $w_2 = 25, v_2 = 50$
- $W = 100$

Optimal solution:

# Fractional knapsack problem: Example

Suppose we have two objects:

- $w_1 = 80, v_1 = 80$
- $w_2 = 25, v_2 = 50$
- $W = 100$

Optimal solution:

- Take all of object 2 (weight = 25, value = 50)

# Fractional knapsack problem: Example

Suppose we have two objects:

- $w_1 = 80, v_1 = 80$
- $w_2 = 25, v_2 = 50$
- $W = 100$

Optimal solution:

- Take all of object 2 (weight = 25, value = 50)
- Take 75 weight units of object 1 (weight = 75, value = 75)

# Fractional knapsack problem: Example

Suppose we have two objects:

- $w_1 = 80, v_1 = 80$
- $w_2 = 25, v_2 = 50$
- $W = 100$

Optimal solution:

- Take all of object 2 (weight = 25, value = 50)
- Take 75 weight units of object 1 (weight = 75, value = 75)
- Total weight = 100, total value = 125

# Solution of Fractional Knapsack Problem

# Solution of Fractional Knapsack Problem

- Define the value density of object $i$ to be $v_i/w_i$.

# Solution of Fractional Knapsack Problem

- Define the value density of object $i$ to be $v_i/w_i$.
- Use object of highest value density in knapsack, then next highest, etc. Continue until knapsack is filled.

# Solution of Fractional Knapsack Problem

- Define the value density of object $i$ to be $v_i/w_i$.
- Use object of highest value density in knapsack, then next highest, etc. Continue until knapsack is filled.
- Formal proof of correctness is in [GT] (page 288).

# Solution of Fractional Knapsack Problem

- Define the value density of object $i$ to be $v_i/w_i$.
- Use object of highest value density in knapsack, then next highest, etc. Continue until knapsack is filled.
- Formal proof of correctness is in [GT] (page 288).

Implementation detail:

# Solution of Fractional Knapsack Problem

- Define the value density of object $i$ to be $v_i/w_i$.
- Use object of highest value density in knapsack, then next highest, etc. Continue until knapsack is filled.
- Formal proof of correctness is in [GT] (page 288).

Implementation detail:

- Sort items in decreasing order of value density:

# Solution of Fractional Knapsack Problem

- Define the value density of object $i$ to be $v_i/w_i$.
- Use object of highest value density in knapsack, then next highest, etc. Continue until knapsack is filled.
- Formal proof of correctness is in [GT] (page 288).

Implementation detail:

- Sort items in decreasing order of value density:
  - Algorithm Runs in $O(n \log n)$ time.

# Solution of Fractional Knapsack Problem

- Define the value density of object $i$ to be $v_i/w_i$.
- Use object of highest value density in knapsack, then next highest, etc. Continue until knapsack is filled.
- Formal proof of correctness is in [GT] (page 288).

Implementation detail:

- Sort items in decreasing order of value density:
  - Algorithm Runs in $O(n \log n)$ time.
- Alternatively, implement using a max-heap. (Key of each object is its value density.)

# Solution of Fractional Knapsack Problem

- Define the value density of object $i$ to be $v_i/w_i$.
- Use object of highest value density in knapsack, then next highest, etc. Continue until knapsack is filled.
- Formal proof of correctness is in [GT] (page 288).

Implementation detail:

- Sort items in decreasing order of value density:
  - Algorithm Runs in $O(n \log n)$ time.
- Alternatively, implement using a max-heap. (Key of each object is its value density.)
  - Algorithm runs in $O(n + k \log n)$ time, where $k$ is the number of objects fully or partially added to the knapsack.

# Greedy Algorithm 2: Task scheduling problem

# Greedy Algorithm 2: Task scheduling problem

- Given $n$ tasks

# Greedy Algorithm 2: Task scheduling problem

- Given $n$ tasks
- Each task has a start time $s_i$ and a finishing time $f_i$

# Greedy Algorithm 2: Task scheduling problem

- Given *n* tasks
- Each task has a start time $s_i$ and a finishing time $f_i$
- Each task must run on some machine. It will occupy that machine from its start time to its finishing time.

# Greedy Algorithm 2: Task scheduling problem

- Given $n$ tasks
- Each task has a start time $s_i$ and a finishing time $f_i$
- Each task must run on some machine. It will occupy that machine from its start time to its finishing time.
- Each machine can only work on one task at a time.

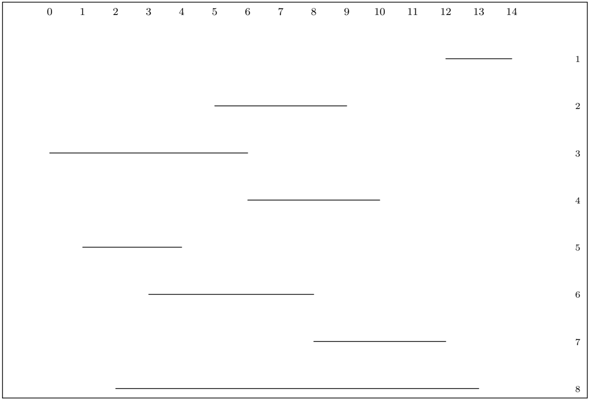
# Greedy Algorithm 2: Task scheduling problem

- Given $n$ tasks
- Each task has a start time $s_i$ and a finishing time $f_i$
- Each task must run on some machine. It will occupy that machine from its start time to its finishing time.
- Each machine can only work on one task at a time.
- Find the minimum number of required machines.

# Task scheduling problem example

# Task scheduling problem example

Example: 8 tasks:
$\{(12, 14), (5, 9), (0, 6), (6, 10), (1, 4), (3, 8), (8, 12), (2, 13)\}$



# Greedy Solution to Task scheduling problem

# Greedy Solution to Task scheduling problem

- Process tasks according to start times, earliest start time first

# Greedy Solution to Task scheduling problem

- Process tasks according to start times, earliest start time first
- For each task $T_i$:

# Greedy Solution to Task scheduling problem

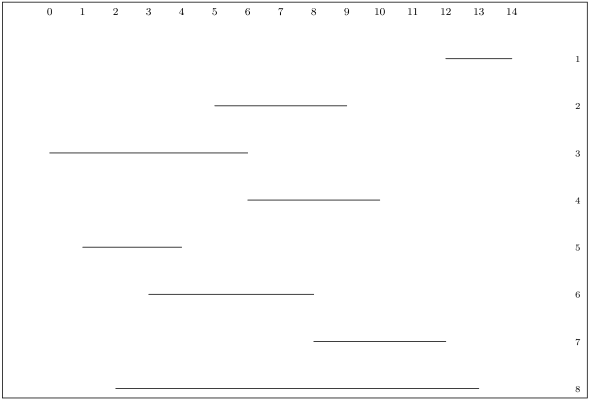
- Process tasks according to start times, earliest start time first
- For each task $T_i$:
  - If we have a machine that can handle $T_i$, schedule $T_i$ on that machine.

# Greedy Solution to Task scheduling problem

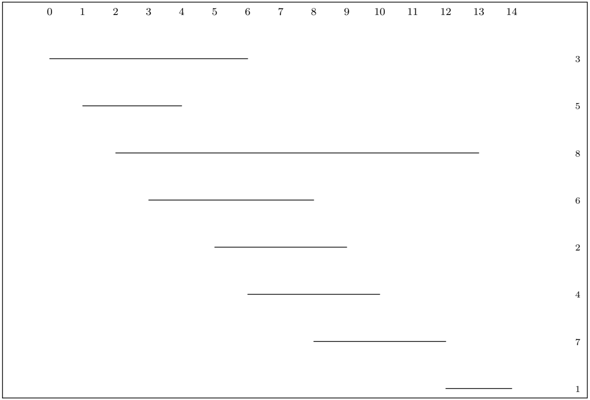
- Process tasks according to start times, earliest start time first
- For each task $T_i$:
  - If we have a machine that can handle $T_i$, schedule $T_i$ on that machine.
  - Otherwise, allocate a new machine, and schedule $T_i$ on this new machine.

# Solution to Task scheduling example

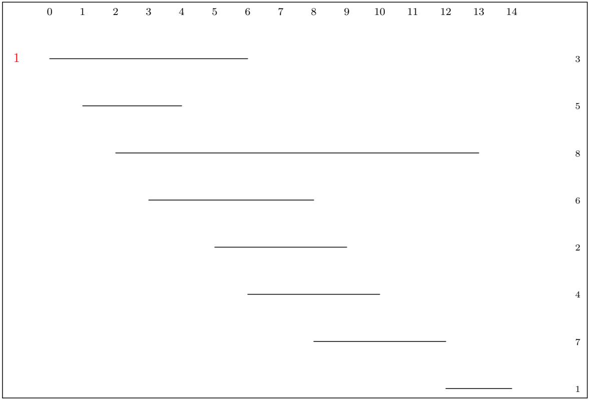
# Solution to Task scheduling example



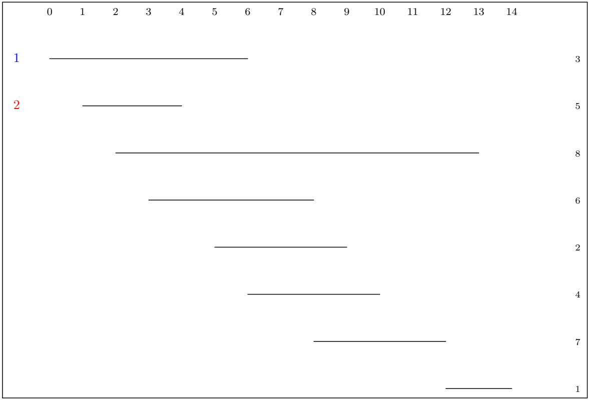
# Solution to Task scheduling example



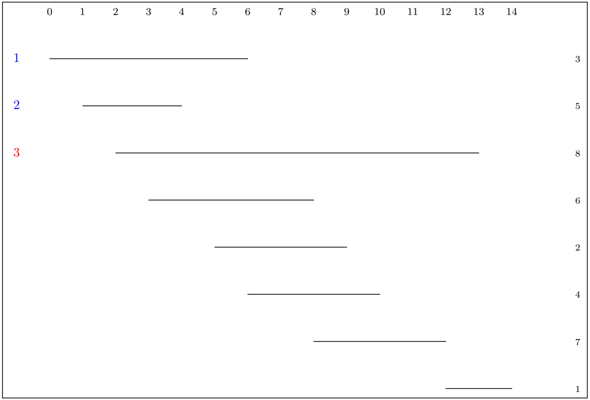
# Solution to Task scheduling example



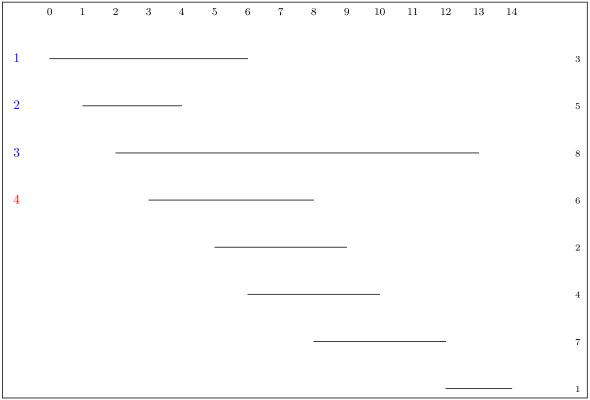
# Solution to Task scheduling example



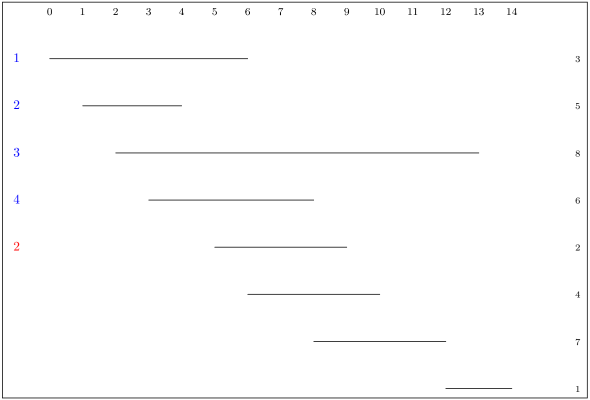
# Solution to Task scheduling example



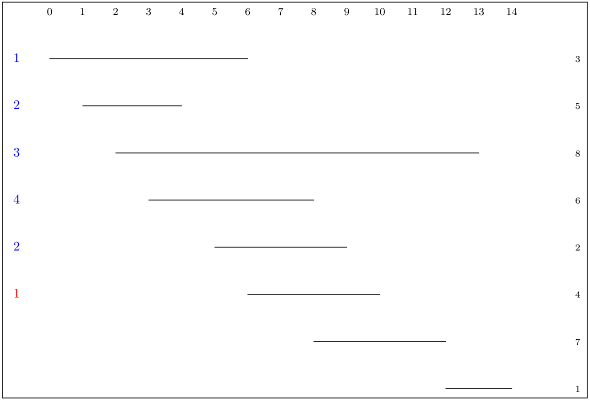
# Solution to Task scheduling example



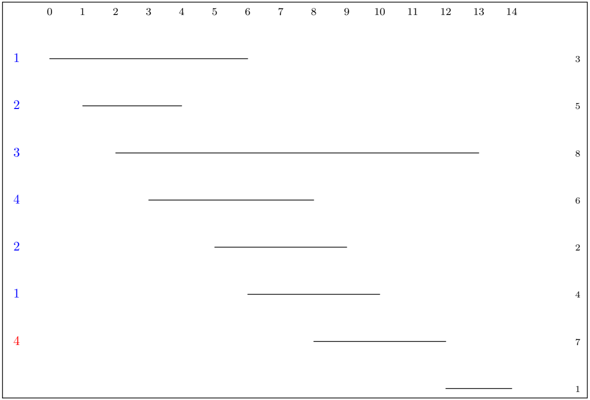
# Solution to Task scheduling example



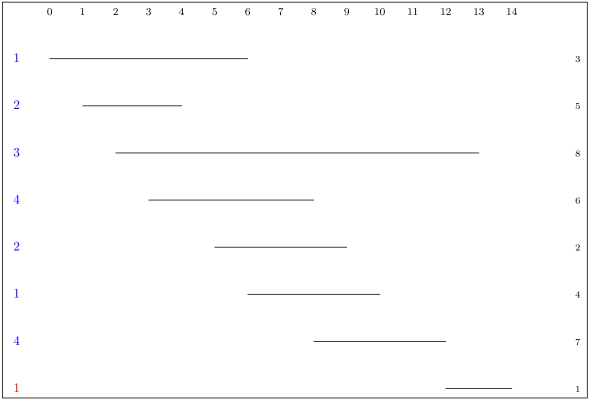
# Solution to Task scheduling example



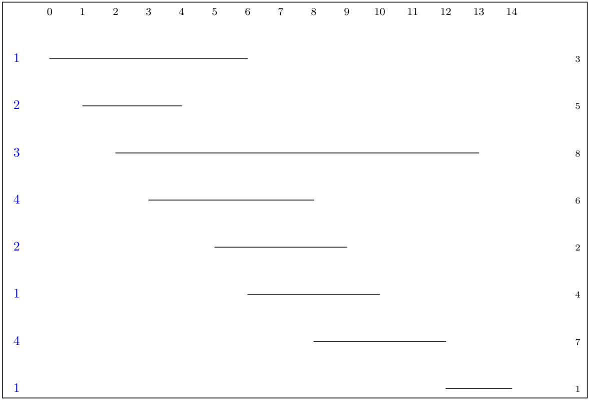
# Solution to Task scheduling example



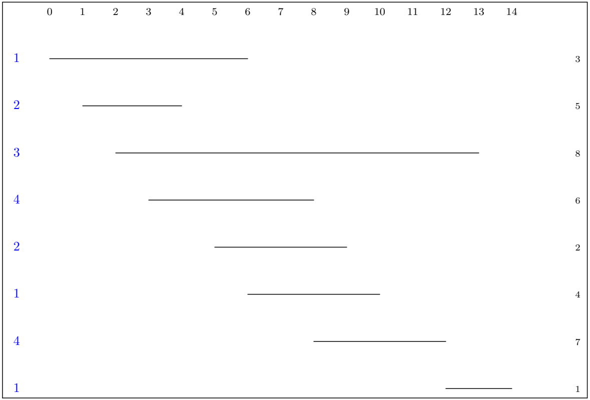
# Solution to Task scheduling example



# Solution to Task scheduling example

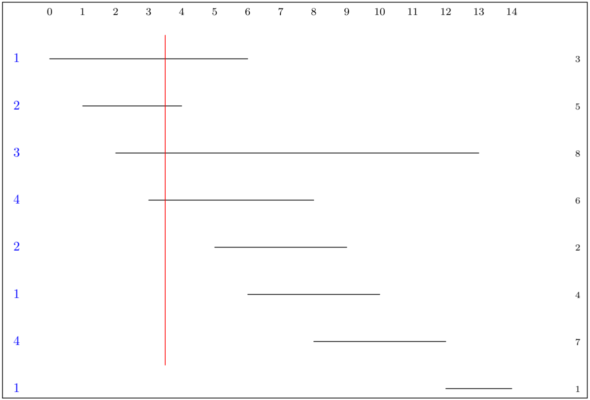


# Solution to Task scheduling example




4 machines are sufficient

# Solution to Task scheduling example




4 machines are sufficient

4 machines are necessary (so solution is optimal)

# Greedy Algorithm 3: Single-machine scheduling

# Greedy Algorithm 3: Single-machine scheduling

([GT] problem A–10.2)

# Greedy Algorithm 3: Single-machine scheduling

([GT] problem A–10.2)

- Given $n$ tasks and only 1 machine.

# Greedy Algorithm 3: Single-machine scheduling

([GT] problem A–10.2)

- Given $n$ tasks and only 1 machine.
- Each task has a start time $s_i$ and a finish time $f_i$.

# Greedy Algorithm 3: Single-machine scheduling

([GT] problem A–10.2)

- Given $n$ tasks and only 1 machine.
- Each task has a start time $s_i$ and a finish time $f_i$.
- If a task is selected, it will occupy the machine from its start time to its finishing time.

# Greedy Algorithm 3: Single-machine scheduling

([GT] problem A–10.2)

- Given $n$ tasks and only 1 machine.
- Each task has a start time $s_i$ and a finish time $f_i$.
- If a task is selected, it will occupy the machine from its start time to its finishing time.
- The machine can only work on one task at a time.

# Greedy Algorithm 3: Single-machine scheduling

([GT] problem A–10.2)

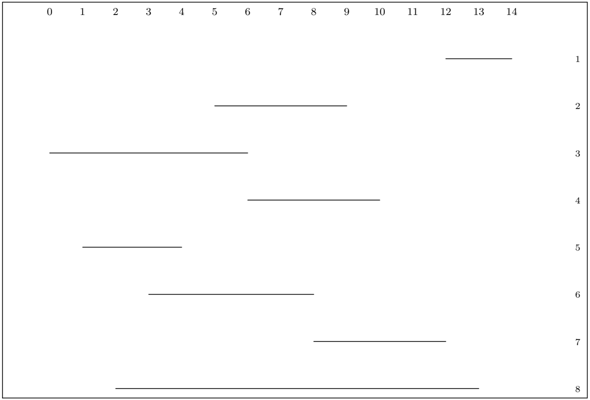
- Given $n$ tasks and only 1 machine.
- Each task has a start time $s_i$ and a finish time $f_i$.
- If a task is selected, it will occupy the machine from its start time to its finishing time.
- The machine can only work on one task at a time.
- Find the maximum number of tasks that can be run.

# Single-Machine scheduling problem example

# Single-Machine scheduling problem example

Example: 8 tasks (same tasks as previous problem):

$$\{(12,14),(5,9),(0,6),(6,10),(1,4),(3,8),(8,12),(2,13)\}$$



# Greedy Solution to Single-Machine scheduling problem

# Greedy Solution to Single-Machine scheduling problem

- Process tasks according to finishing times, earliest finishing time first.

# Greedy Solution to Single-Machine scheduling problem

- Process tasks according to finishing times, earliest finishing time first.
- For each task $T_i$:

# Greedy Solution to Single-Machine scheduling problem

- Process tasks according to finishing times, earliest finishing time first.
- For each task $T_i$:
  - If task $T_i$ does not conflict with tasks already chosen, select it.

# Greedy Solution to Single-Machine scheduling problem

- Process tasks according to finishing times, earliest finishing time first.
- For each task $T_i$:
  - If task $T_i$ does not conflict with tasks already chosen, select it.
  - Otherwise, do not select task $T_i$.

# Greedy Solution to Single-Machine scheduling problem

- Process tasks according to finishing times, earliest finishing time first.
- For each task $T_i$:
  - If task $T_i$ does not conflict with tasks already chosen, select it.
  - Otherwise, do not select task $T_i$.
- [Kleinberg-Tardos] gives correctness proof and presents some plausible alternative greedy heuristics that do not work.

# Greedy Solution to Single-Machine scheduling problem

- Process tasks according to finishing times, earliest finishing time first.
- For each task $T_i$:
  - If task $T_i$ does not conflict with tasks already chosen, select it.
  - Otherwise, do not select task $T_i$.
- [Kleinberg-Tardos] gives correctness proof and presents some plausible alternative greedy heuristics that do not work.
- This algorithm is optimal because the set of tasks it chooses "stays ahead" of any other set of tasks.

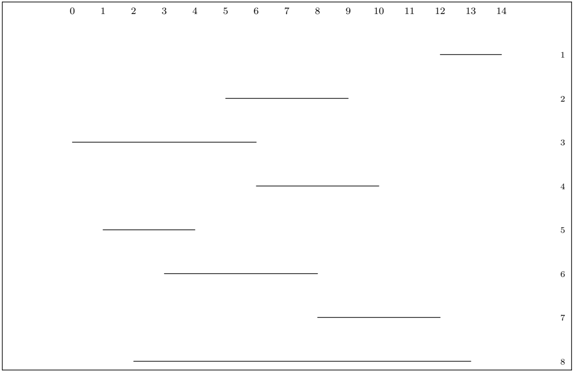
# Greedy Solution to Single-Machine scheduling problem

- Process tasks according to finishing times, earliest finishing time first.
- For each task $T_i$:
  - If task $T_i$ does not conflict with tasks already chosen, select it.
  - Otherwise, do not select task $T_i$.
- [Kleinberg-Tardos] gives correctness proof and presents some plausible alternative greedy heuristics that do not work.
- This algorithm is optimal because the set of tasks it chooses "stays ahead" of any other set of tasks.

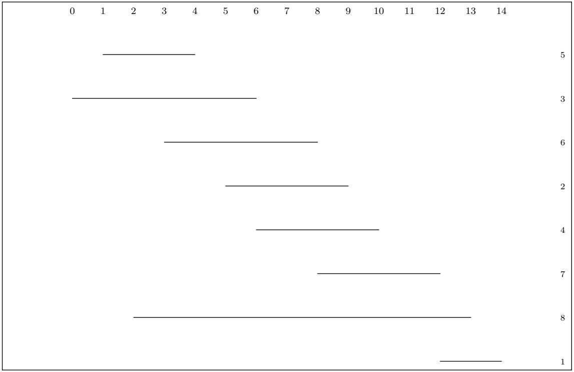
[Kleinberg-Tardos] Jon Kleinberg and Éva Tardos, Algorithm Design, Addison-Wesley, 2006.

# Solution to Single-Machine scheduling Example

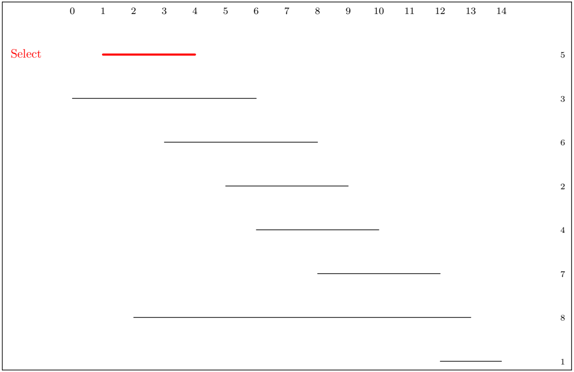
# Solution to Single-Machine scheduling Example



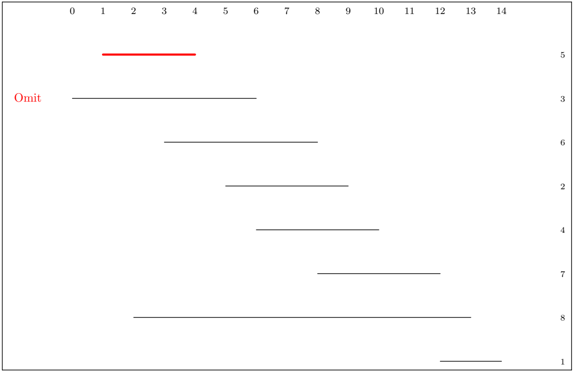
# Solution to Single-Machine scheduling Example



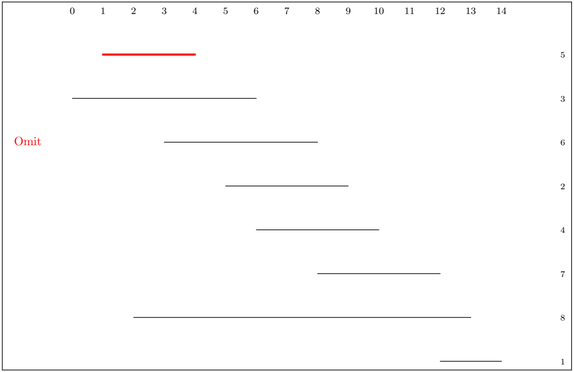
# Solution to Single-Machine scheduling Example



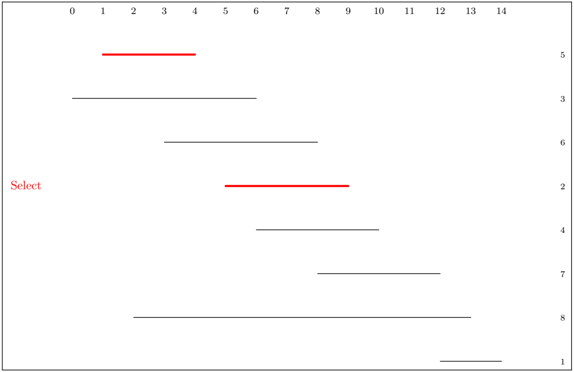
# Solution to Single-Machine scheduling Example



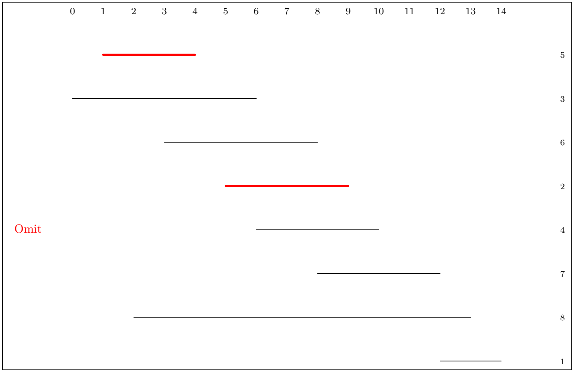
# Solution to Single-Machine scheduling Example



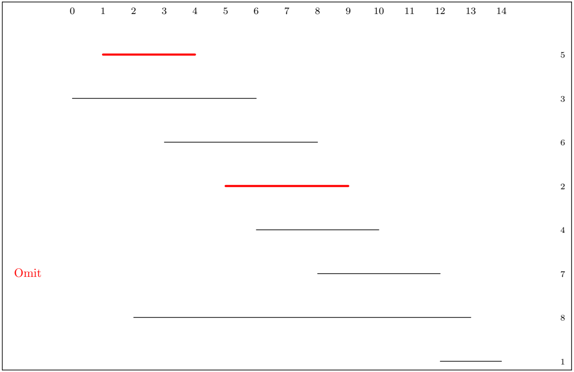
# Solution to Single-Machine scheduling Example



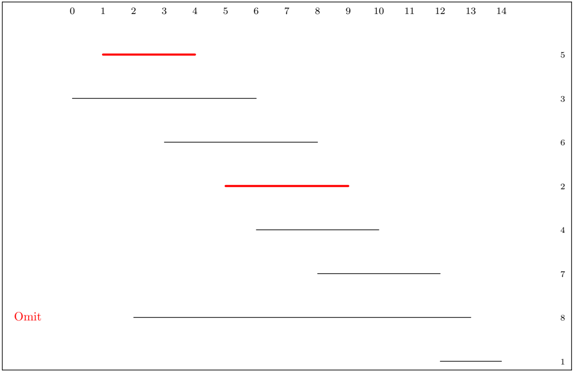
# Solution to Single-Machine scheduling Example



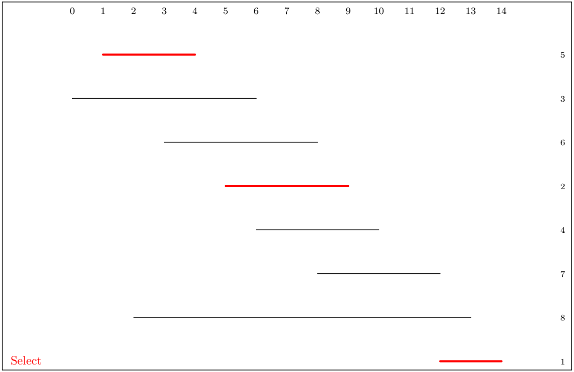
# Solution to Single-Machine scheduling Example



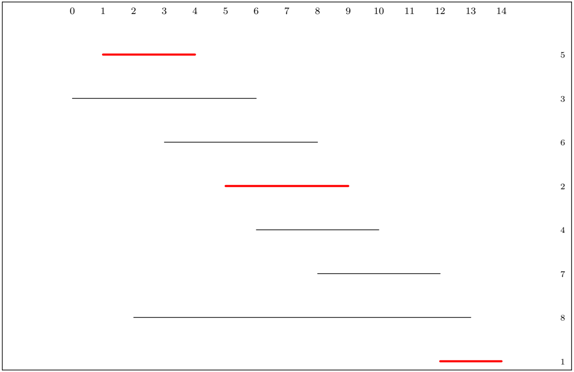
# Solution to Single-Machine scheduling Example



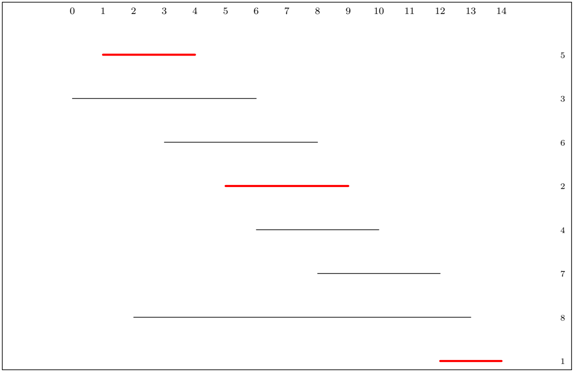
# Solution to Single-Machine scheduling Example



# Solution to Single-Machine scheduling Example



# Solution to Single-Machine scheduling Example




Choose 3 jobs: 5, 2, and 1

# Final notes on Greedy Algorithms

# Final notes on Greedy Algorithms

- Greedy algorithms appear in various contexts.

# Final notes on Greedy Algorithms

- Greedy algorithms appear in various contexts. For example:
  - Graph problems: shortest paths, minimum spanning tree

# Final notes on Greedy Algorithms

- Greedy algorithms appear in various contexts. For example:
  - Graph problems: shortest paths, minimum spanning tree
  - String compression: Huffman coding

# Final notes on Greedy Algorithms

- Greedy algorithms appear in various contexts. For example:
  - Graph problems: shortest paths, minimum spanning tree
  - String compression: Huffman coding
- Given an optimization problem, it is always worth asking: will a greedy algorithm work?

# Final notes on Greedy Algorithms

- Greedy algorithms appear in various contexts. For example:
  - Graph problems: shortest paths, minimum spanning tree
  - String compression: Huffman coding
- Given an optimization problem, it is always worth asking: will a greedy algorithm work?
  - If the answer is yes, you have an easy solution.

# Final notes on Greedy Algorithms

- Greedy algorithms appear in various contexts. For example:
  - Graph problems: shortest paths, minimum spanning tree
  - String compression: Huffman coding
- Given an optimization problem, it is always worth asking: will a greedy algorithm work?
  - If the answer is yes, you have an easy solution.
  - If the answer is no (which it often is), seeing why the greedy approach fails may give you some insight into the structure of the problem.