# Introduction to Algorithms

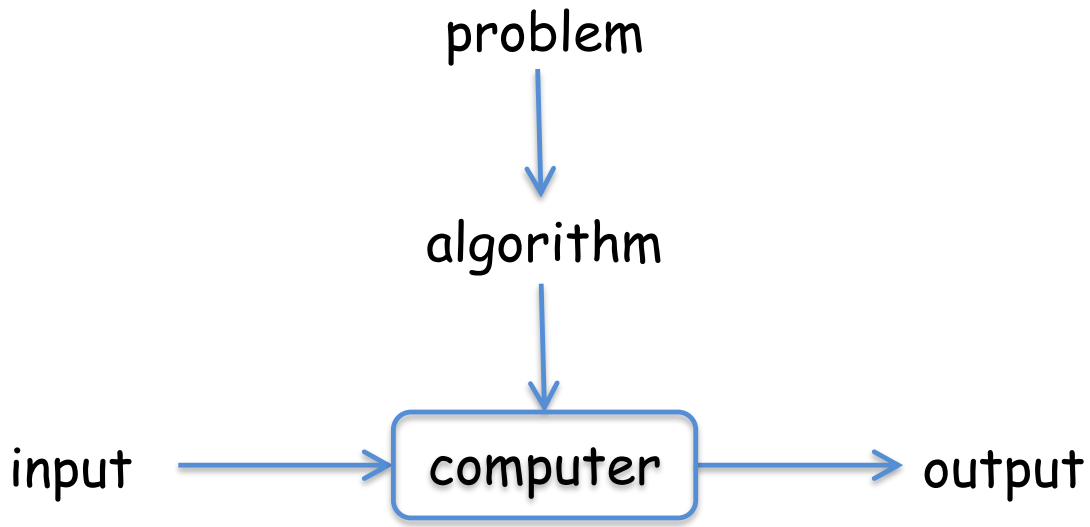
Murat Osmanoglu

# Introduction

- 'a sequence of unambiguous instructions for solving a given problem' (Levitin) – obtaining a required output for any legitimate input in a finite amount of time

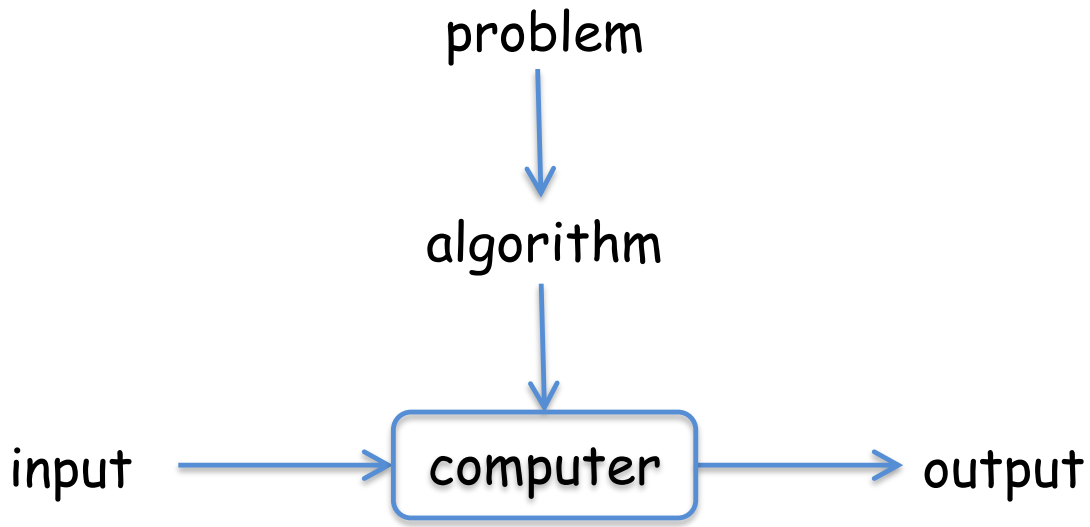
# Introduction

- 'a sequence of unambiguous instructions for solving a given problem' (Levitin) – obtaining a required output for any legitimate input in a finite amount of time



# Introduction

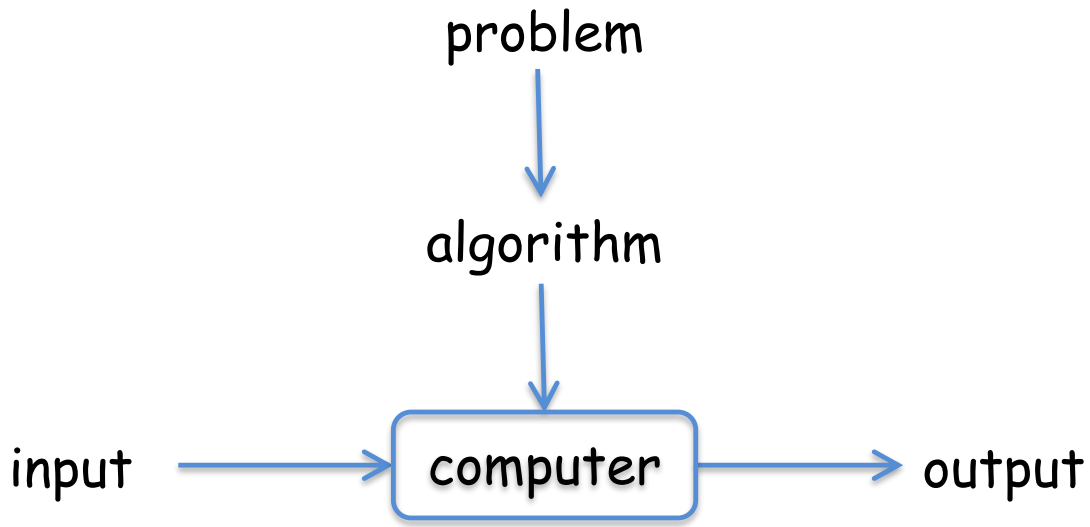
- 'a sequence of unambiguous instructions for solving a given problem' (Levitin) – obtaining a required output for any legitimate input in a finite amount of time




- each step should be expressed in a clear way

# Introduction

- 'a sequence of unambiguous instructions for solving a given problem' (Levitin) – obtaining a required output for any legitimate input in a finite amount of time




- each step should be expressed in a clear way
- the nature of the input should be specified carefully

# Fundamentals of Algorithmic Problem Solving

## Understanding the Problem

# Fundamentals of Algorithmic Problem Solving

## Understanding the Problem

- design an algorithm to find the maximum number of a finite sequence of integers

# Fundamentals of Algorithmic Problem Solving

## Understanding the Problem

- design an algorithm to find the maximum number of a finite sequence of integers
  - input : {34, 23, 2, 101, 5, 98, 43}, output : 101

# Fundamentals of Algorithmic Problem Solving

## Understanding the Problem

- design an algorithm to find the maximum number of a finite sequence of integers
  - input : {34, 23, 2, 101, 5, 98, 43}, output : 101
- an algorithmic problem is specified by describing the set of instances (input) it must work on, and what desired properties the output must have

# Fundamentals of Algorithmic Problem Solving

## Understanding the Problem

- design an algorithm to find the maximum number of a finite sequence of integers
  - input : {34, 23, 2, 101, 5, 98, 43}, output : 101
- an algorithmic problem is specified by describing the set of instances (input) it must work on, and what desired properties the output must have
- design an algorithm to determine how 'similar' two given DNA sequences are

# Fundamentals of Algorithmic Problem Solving

## Understanding the Problem

- design an algorithm to find the maximum number of a finite sequence of integers
  - input : {34, 23, 2, 101, 5, 98, 43}, output : 101
- an algorithmic problem is specified by describing the set of instances (input) it must work on, and what desired properties the output must have
- design an algorithm to determine how 'similar' two given DNA sequences are
  - DNA sequence is a string of arbitrary length over the alphabet {A, C, G, T},

# Fundamentals of Algorithmic Problem Solving

## Understanding the Problem

- design an algorithm to find the maximum number of a finite sequence of integers
  - input : {34, 23, 2, 101, 5, 98, 43}, output : 101
- an algorithmic problem is specified by describing the set of instances (input) it must work on, and what desired properties the output must have
- design an algorithm to determine how 'similar' two given DNA sequences are
  - DNA sequence is a string of arbitrary length over the alphabet {A, C, G, T},
  - the degree of the similarity may be measured by the length of their longest common subsequence

# Fundamentals of Algorithmic Problem Solving

## Understanding the Problem

- design an algorithm to find the maximum number of a finite sequence of integers
  - input : {34, 23, 2, 101, 5, 98, 43}, output : 101
- an algorithmic problem is specified by describing the set of instances (input) it must work on, and what desired properties the output must have
- design an algorithm to determine how 'similar' two given DNA sequences are
  - DNA sequence is a string of arbitrary length over the alphabet {A, C, G, T},
  - the degree of the similarity may be measured by the length of their longest common subsequence

input :
output :

# Fundamentals of Algorithmic Problem Solving

## Understanding the Problem

- design an algorithm to find the maximum number of a finite sequence of integers
  - input : {34, 23, 2, 101, 5, 98, 43}, output : 101
- an algorithmic problem is specified by describing the set of instances (input) it must work on, and what desired properties the output must have
- design an algorithm to determine how 'similar' two given DNA sequences are
  - DNA sequence is a string of arbitrary length over the alphabet {A, C, G, T},
  - the degree of the similarity may be measured by the length of their longest common subsequence

input : ACCACTGGT, ACTATCGAG

output :

# Fundamentals of Algorithmic Problem Solving

## Understanding the Problem

- design an algorithm to find the maximum number of a finite sequence of integers
  - input : {34, 23, 2, 101, 5, 98, 43}, output : 101
- an algorithmic problem is specified by describing the set of instances (input) it must work on, and what desired properties the output must have
- design an algorithm to determine how 'similar' two given DNA sequences are
  - DNA sequence is a string of arbitrary length over the alphabet {A, C, G, T},
  - the degree of the similarity may be measured by the length of their longest common subsequence

input : ACCACTGGT, ACTATCGAG

output :

# Fundamentals of Algorithmic Problem Solving

## Understanding the Problem

- design an algorithm to find the maximum number of a finite sequence of integers
  - input : {34, 23, 2, 101, 5, 98, 43}, output : 101
- an algorithmic problem is specified by describing the set of instances (input) it must work on, and what desired properties the output must have
- design an algorithm to determine how 'similar' two given DNA sequences are
  - DNA sequence is a string of arbitrary length over the alphabet {A, C, G, T},
  - the degree of the similarity may be measured by the length of their longest common subsequence

input : ACCACTGGT, ACTATCGAG

output : 6

# Fundamentals of Algorithmic Problem Solving

## Choosing Exact or Approximate Problem Solving

# Fundamentals of Algorithmic Problem Solving

## Choosing Exact or Approximate Problem Solving

- looking for an exact solution for the given problem, or an approximate solution

# Fundamentals of Algorithmic Problem Solving

## Choosing Exact or Approximate Problem Solving

- looking for an exact solution for the given problem, or an approximate solution

- some problems cannot be solved exactly for most of their instances (extracting square roots, solving nonlinear equations)

# Fundamentals of Algorithmic Problem Solving

## Choosing Exact or Approximate Problem Solving

- looking for an exact solution for the given problem, or an approximate solution
- some problems cannot be solved exactly for most of their instances (extracting square roots, solving nonlinear equations)
- for some problems, the existing algorithms can be very slow to generate an exact output

# Fundamentals of Algorithmic Problem Solving

## Choosing Exact or Approximate Problem Solving

- looking for an exact solution for the given problem, or an approximate solution
- some problems cannot be solved exactly for most of their instances (extracting square roots, solving nonlinear equations)
- for some problems, the existing algorithms can be very slow to generate an exact output
- approximation algorithm can be a sub-procedure of a more sophisticated algorithm

# Fundamentals of Algorithmic Problem Solving

Decide on Algorithm Design Techniques
(how do you design an algorithm to solve a given problem?)

# Fundamentals of Algorithmic Problem Solving

## Decide on Algorithm Design Techniques
## (how do you design an algorithm to solve a given problem?)

- an algorithm design technique is a general approach in solving problems algorithmically that is applicable to a variety of problems

# Fundamentals of Algorithmic Problem Solving

## Decide on Algorithm Design Techniques
## (how do you design an algorithm to solve a given problem?)

- an algorithm design technique is a general approach in solving problems algorithmically that is applicable to a variety of problems

- mastering these strategies provides guidance for developing new algorithms for new problems

# Fundamentals of Algorithmic Problem Solving

## Decide on Algorithm Design Techniques
## (how do you design an algorithm to solve a given problem?)

- an algorithm design technique is a general approach in solving problems algorithmically that is applicable to a variety of problems

- mastering these strategies provides guidance for developing new algorithms for new problems

- enable us to classify and study algorithms according to a corresponding design idea

# Fundamentals of Algorithmic Problem Solving

## Decide on Algorithm Design Techniques
## (how do you design an algorithm to solve a given problem?)

- an algorithm design technique is a general approach in solving problems algorithmically that is applicable to a variety of problems

- mastering these strategies provides guidance for developing new algorithms for new problems

- enable us to classify and study algorithms according to a corresponding design idea

- Brute-Force, Decrease-and-Conquer, Divide-and-Conquer, Transform-and-Conquer, Dynamic Programming, Greedy Techniques

# Fundamentals of Algorithmic Problem Solving

## Choosing an appropriate data structure to implement the algorithm

- data structure, a systematic way of organizing and accessing data

# Fundamentals of Algorithmic Problem Solving

## Choosing an appropriate data structure to implement the algorithm

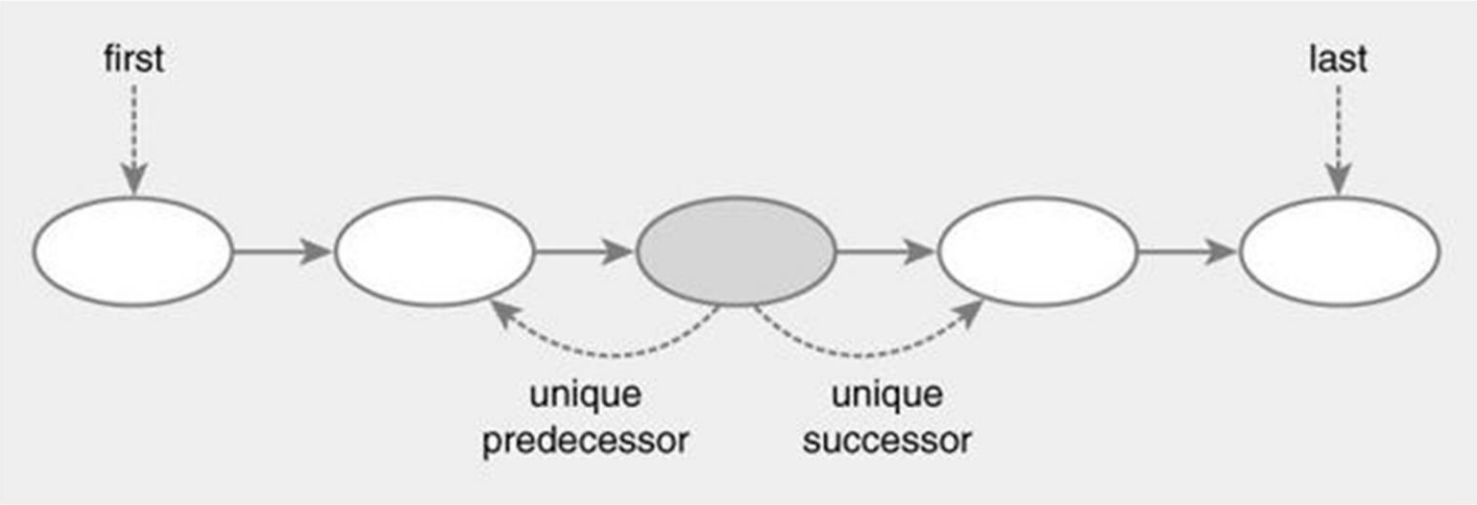
- data structure, a systematic way of organizing and accessing data

- structuring data in an effective way will improve the algorithms

# Fundamentals of Algorithmic Problem Solving

## Choosing an appropriate data structure to implement the algorithm

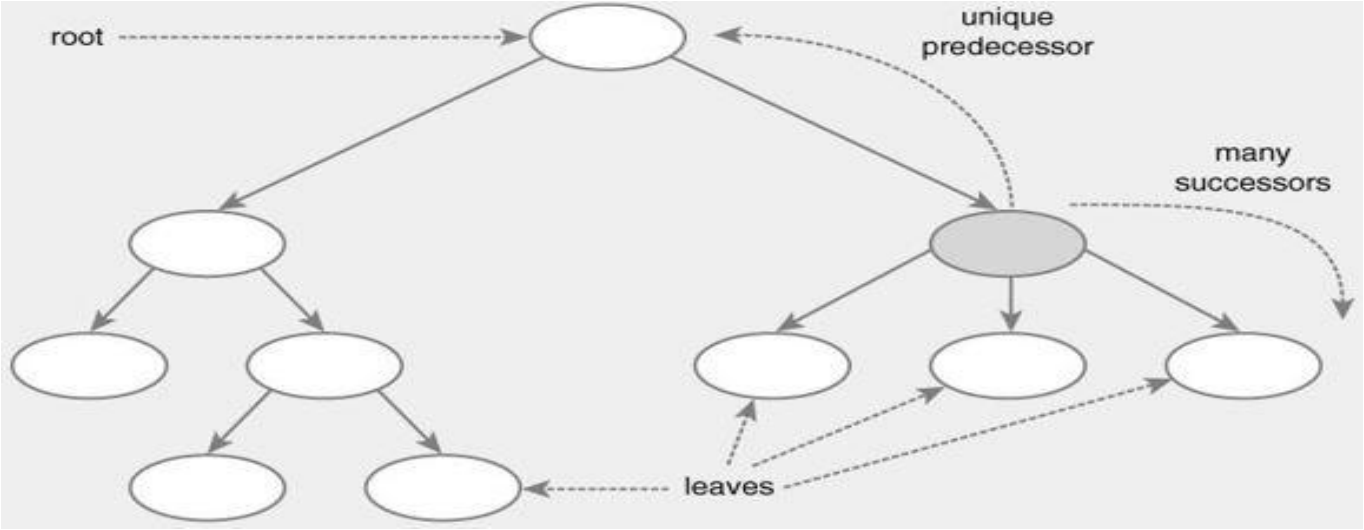
- data structure, a systematic way of organizing and accessing data
- structuring data in an effective way will improve the algorithms
- Linear Data Structures
  - one-to-one relationship between elements in the collection
  - Arrays, Linked Lists, Stacks, Queues



# Fundamentals of Algorithmic Problem Solving

## Choosing an appropriate data structure to implement the algorithm

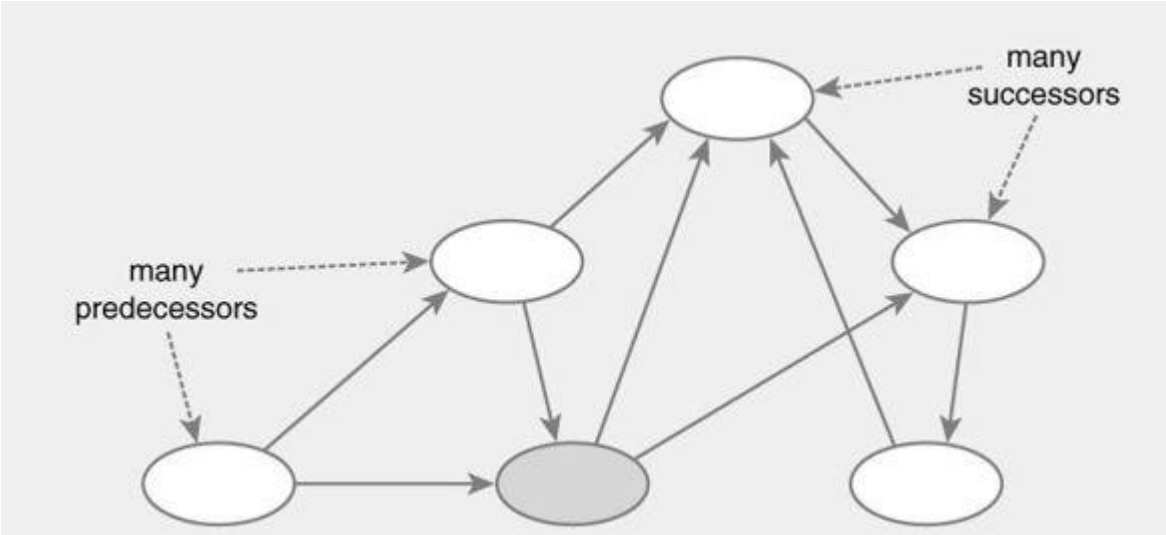
- data structure, a systematic way of organizing and accessing data
- structuring data in an effective way will improve the algorithms
- Hierarchical Data Structures
  - one-to-many relationship between elements in the collection
  - Binary Trees, AVL Trees, Splay Trees, B Trees



# Fundamentals of Algorithmic Problem Solving

## Choosing an appropriate data structure to implement the algorithm

- data structure, a systematic way of organizing and accessing data
- structuring data in an effective way will improve the algorithms
- Graph Data Structures
  - many-to-many relationship between elements in the collection



# Fundamentals of Algorithmic Problem Solving

## Choosing an appropiate methods to specify the Algorithm

# Fundamentals of Algorithmic Problem Solving

## Choosing an appropiate methods to specify the Algorithm

- two common ways of specifying an algorithm: in words, in pseudocode

# Fundamentals of Algorithmic Problem Solving

## Choosing an appropiate methods to specify the Algorithm

- two common ways of specifying an algorithm: in words, in pseudocode
- design an algorithm to find the greatest common divisor of two nonnegative (not both of them zero) integers

# Fundamentals of Algorithmic Problem Solving

## Choosing an appropiate methods to specify the Algorithm

- two common ways of specifying an algorithm: in words, in pseudocode
- design an algorithm to find the greatest common divisor of two nonnegative (not both of them zero) integers
- **Euclid's Algorithm for the integers m, n**

# Fundamentals of Algorithmic Problem Solving

## Choosing an appropiate methods to specify the Algorithm

- two common ways of specifying an algorithm: in words, in pseudocode
- design an algorithm to find the greatest common divisor of two nonnegative (not both of them zero) integers
- **Euclid's Algorithm for the integers m, n**

  **Step 1** : if n = 0, return m; otherwise go to Step 2

  **Step 2** : divide m by n and set the variable r to the remainder

  **Step 3** : set m as n and n as r, go to Step 1

# Fundamentals of Algorithmic Problem Solving

## Choosing an appropiate methods to specify the Algorithm

- two common ways of specifying an algorithm: in words, in pseudocode
- design an algorithm to find the greatest common divisor of two nonnegative (not both of them zero) integers
- **Euclid's Algorithm for the integers m, n**

**Step 1** : if n = 0, return m; otherwise go to Step 2

**Step 2** : divide m by n and set the variable r to the remainder

**Step 3** : set m as n and n as r, go to Step 1

- **Euclid(m,n)**

**input** : two non-negative, not-both-zero integers m and n
**output**: the greatest common divisor of m and n

**while** $n \neq 0$

$r \leftarrow m \ (mod \ n)$

$m \leftarrow n$

$n \leftarrow r$

**return** m

# Fundamentals of Algorithmic Problem Solving

## Choosing an appropiate methods to specify the Algorithm

- two common ways of specifying an algorithm: in words, in pseudocode
- design an algorithm to find the greatest common divisor of two nonnegative (not both of them zero) integers
- **Euclid's Algorithm for the integers m, n**

**Step 1** : if n = 0, return m; otherwise go to Step 2

**Step 2** : divide m by n and set the variable r to the remainder

**Step 3** : set m as n and n as r, go to Step 1

- **Euclid(m,n)**

**input** : two non-negati

**output**: the greatest

**while** $n \neq 0$

$r \leftarrow m \ (mod \ n)$

$m \leftarrow n$

$n \leftarrow r$

**return** m

- high-level description of algorithms that combines a natural language and familiar structures from a programming language
- use '←' for the assigments and '//' for the comments

# Fundamentals of Algorithmic Problem Solving

## Proving the correctness of the Algorithm

# Fundamentals of Algorithmic Problem Solving

## Proving the correctness of the Algorithm

- prove that the algorithm always returns the desired output for every legitimate input in a finite amount of time in a formal way

# Fundamentals of Algorithmic Problem Solving

## Proving the correctness of the Algorithm

- prove that the algorithm always returns the desired output for every legitimate input in a finite amount of time in a formal way
- use mathematical proof techniques such as proof by contradiction, induction, etc.

# Fundamentals of Algorithmic Problem Solving

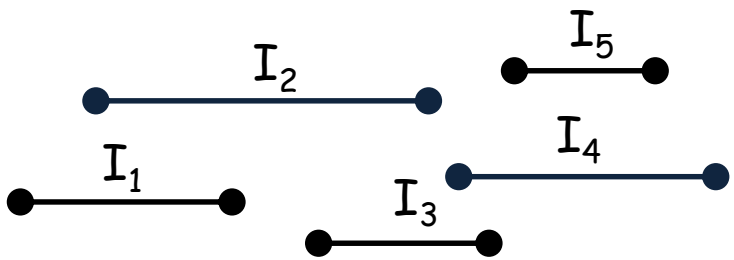
## Proving the correctness of the Algorithm

- prove that the algorithm always returns the desired output for every legitimate input in a finite amount of time in a formal way
- use mathematical proof techniques such as proof by contradiction, induction, etc.
- suppose there are n meetings requests for a meeting room. Each meeting i has a starting time $s_i$ and an ending time $t_i$. We have a constraint : no two meetings can be scheduled at same time. Design an algorithm that schedules as many meetings as possible to the room

# Fundamentals of Algorithmic Problem Solving

## Proving the correctness of the Algorithm

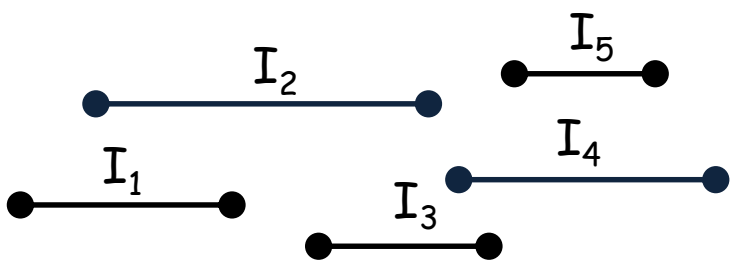
- prove that the algorithm always returns the desired output for every legitimate input in a finite amount of time in a formal way
- use mathematical proof techniques such as proof by contradiction, induction, etc.
- suppose there are n meetings requests for a meeting room. Each meeting i has a starting time $s_i$ and an ending time $t_i$. We have a constraint : no two meetings can be scheduled at same time. Design an algorithm that schedules as many meetings as possible to the room



# Fundamentals of Algorithmic Problem Solving

## Proving the correctness of the Algorithm

- prove that the algorithm always returns the desired output for every legitimate input in a finite amount of time in a formal way
- use mathematical proof techniques such as proof by contradiction, induction, etc.
- suppose there are n meetings requests for a meeting room. Each meeting i has a starting time $s_i$ and an ending time $t_i$. We have a constraint : no two meetings can be scheduled at same time. Design an algorithm that schedules as many meetings as possible to the room

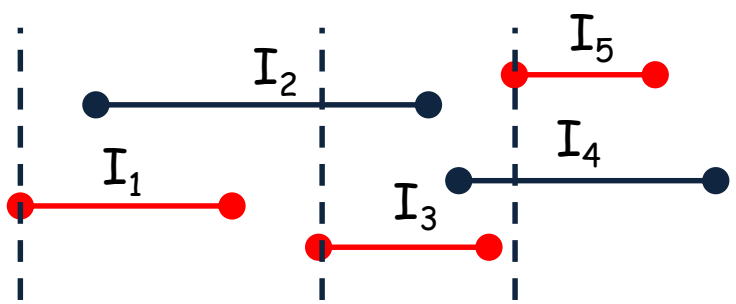



- choose the first interval as the one having the earliest start time
- remove all intervals not compatible with the chosen one

# Fundamentals of Algorithmic Problem Solving

## Proving the correctness of the Algorithm

- prove that the algorithm always returns the desired output for every legitimate input in a finite amount of time in a formal way
- use mathematical proof techniques such as proof by contradiction, induction, etc.
- suppose there are n meetings requests for a meeting room. Each meeting i has a starting time $s_i$ and an ending time $t_i$. We have a constraint : no two meetings can be scheduled at same time. Design an algorithm that schedules as many meetings as possible to the room

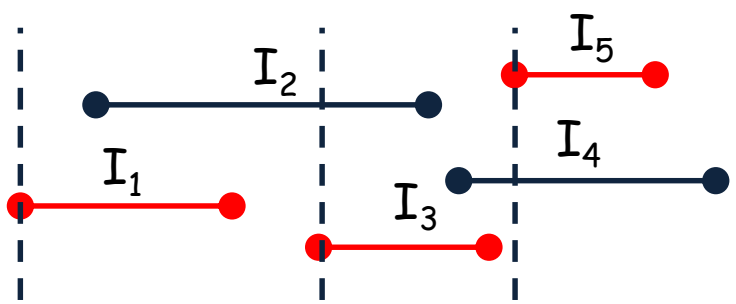



- choose the first interval as the one having the earliest start time
- remove all intervals not compatible with the chosen one

# Fundamentals of Algorithmic Problem Solving

## Proving the correctness of the Algorithm

- prove that the algorithm always returns the desired output for every legitimate input in a finite amount of time in a formal way
- use mathematical proof techniques such as proof by contradiction, induction, etc.
- suppose there are n meetings requests for a meeting room. Each meeting i has a starting time $s_i$ and an ending time $t_i$. We have a constraint : no two meetings can be scheduled at same time. Design an algorithm that schedules as many meetings as possible to the room



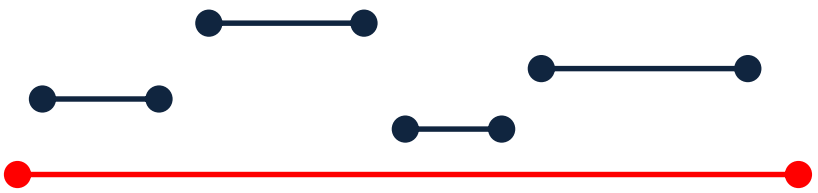

- choose the first interval as the one having the earliest start time
- remove all intervals not compatible with the chosen one

- one instance of inputs for which the algorithm works would not be enough to show the algorithm is correct

# Fundamentals of Algorithmic Problem Solving

## Proving the correctness of the Algorithm

- prove that the algorithm always returns the desired output for every legitimate input in a finite amount of time in a formal way
- use mathematical proof techniques such as proof by contradiction, induction, etc.
- suppose there are n meetings requests for a meeting room. Each meeting i has a starting time $s_i$ and an ending time $t_i$. We have a constraint : no two meetings can be scheduled at same time. Design an algorithm that schedules as many meetings as possible to the room

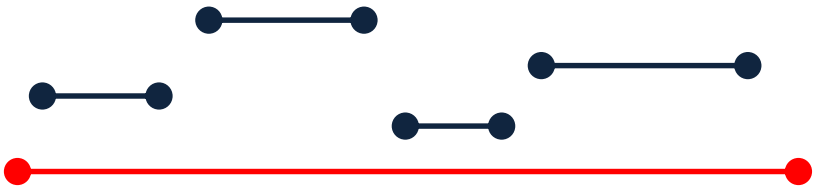


- choose the first interval as the one having the earliest start time
- remove all intervals not compatible with the chosen one

# Fundamentals of Algorithmic Problem Solving

## Proving the correctness of the Algorithm

- prove that the algorithm always returns the desired output for every legitimate input in a finite amount of time in a formal way
- use mathematical proof techniques such as proof by contradiction, induction, etc.
- suppose there are n meetings requests for a meeting room. Each meeting i has a starting time $s_i$ and an ending time $t_i$. We have a constraint : no two meetings can be scheduled at same time. Design an algorithm that schedules as many meetings as possible to the room

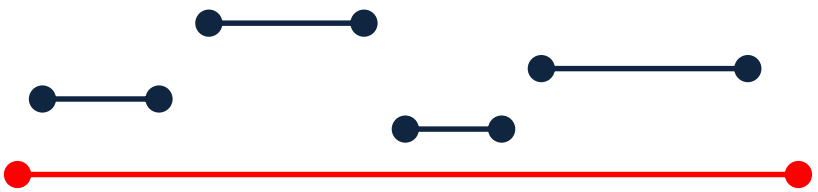


- choose the first interval as the one having the earliest start time
- remove all intervals not compatible with the chosen one
- one instance of inputs for which the algorithm fails would be enough to show the algorithm is incorrect

# Fundamentals of Algorithmic Problem Solving

## Proving the correctness of the Algorithm

- prove that the algorithm always returns the desired output for every legitimate input in a finite amount of time in a formal way
- use mathematical proof techniques such as proof by contradiction, induction, etc.
- suppose there are n meetings requests for a meeting room. Each meeting i has a starting time $s_i$ and an ending time $t_i$. We have a constraint : no two meetings can be scheduled at same time. Design an algorithm that schedules as many meetings as possible to the room



- choose the first interval as the one having the earliest start time
- remove all intervals not compatible with the chosen one

- one instance of inputs for which the algorithm fails would be enough to show the algorithm is incorrect
  (however, failure to find such instance does not mean 'it is obvious' that the algorithm is correct)

# Fundamentals of Algorithmic Problem Solving

## Evaluating the efficiency of the Algorithm

# Fundamentals of Algorithmic Problem Solving

## Evaluating the efficiency of the Algorithm

- investigate algorithm's efficiency with respect to two resources: running time and memory space
  (time complexity and space complexity)

# Fundamentals of Algorithmic Problem Solving

## Evaluating the efficiency of the Algorithm

- investigate algorithm's efficiency with respect to two resources: running time and memory space
  (time complexity and space complexity)

- how long does the algorithm take to generate a desired output as a function of input size ?

# Fundamentals of Algorithmic Problem Solving

## Evaluating the efficiency of the Algorithm

- investigate algorithm's efficiency with respect to two resources: running time and memory space
  (time complexity and space complexity)

- how long does the algorithm take to generate a desired output as a function of input size ?

- how much working memory (typically RAM) required for the algorithm to terminate as a function of input size ?