Algorithms: Basic Concepts

Analysis and Design of Algorithms

Profesores de análisis y diseño de algoritmos

Research Group on Artificial Life – Grupo de investigación en vida artificial – (Alife)
Computer and System Department
Engineering School
Universidad Nacional de Colombia

Outline

1. Basic concepts
2. Correctness and Loop Invariants
3. Analysis and Design of Algorithms

Outline

Outline

Outline

Program

Definition (Program)

A **program** is a sequence of **instructions** written to perform a specified task using a computer.

- A computer requires programs to function.
- The **source code** of a computer is written by **computer programmers**.
- The source code is written in a **programming language**.
- The source code may be converted into an **executable file** by a **compiler** or an **interpreter** (intermediate language).

Program

Definition (Program)

A **program** is a sequence of **instructions** written to perform a specified task using a computer.

- A computer requires programs to function.
- The **source code** of a computer is written by **computer programmers**.
- The source code is written in a **programming language**.
- The source code may be converted into an **executable file** by a **compiler** or an **interpreter** (intermediate language).

Program

Definition (Program)

A **program** is a sequence of **instructions** written to perform a specified task using a computer.

- A computer requires programs to function.
- The **source code** of a computer is written by **computer programmers**.
- The source code is written in a **programming language**.
- The source code may be converted into an **executable file** by a **compiler** or an **interpreter** (intermediate language).

Program

Definition (Program)

A **program** is a sequence of **instructions** written to perform a specified task using a computer.

- A computer requires programs to function.
- The **source code** of a computer is written by **computer programmers**.
- The source code is written in a **programming language**.
- The source code may be converted into an **executable file** by a **compiler** or an **interpreter** (intermediate language).

Program

Definition (Program)

A **program** is a sequence of **instructions** written to perform a specified task using a computer.

- A computer requires programs to function.
- The **source code** of a computer is written by **computer programmers**.
- The source code is written in a **programming language**.
- The source code may be converted into an **executable file** by a **compiler** or an **interpreter** (intermediate language).

Instance

Definition (Instance)

An **instance** of a problem consists of an input (satisfying any constraints imposed in the problem statement) necessary to compute a solution to the problem.

Example

The sequence $\langle 31, 41, 59, 26, 41, 58 \rangle$ is an instance of the sorting problem.

Instance

Definition (Instance)

An **instance** of a problem consists of an input (satisfying any constraints imposed in the problem statement) necessary to compute a solution to the problem.

Example

The sequence $\langle 31, 41, 59, 26, 41, 58 \rangle$ is an instance of the sorting problem.

Computational problem

Definition

- Relationship between a set of **instances** (input) and a set of **solutions** (output).
- Formally establishes the relationship between the **input** and the **output** of an algorithm.

Example

Computational problem

Definition

- Relationship between a set of **instances** (input) and a set of **solutions** (output).
- Formally establishes the relationship between the **input** and the **output** of an algorithm.

Example

Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$

Output: A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

Computational problem

Definition

- Relationship between a set of **instances** (input) and a set of **solutions** (output).
- Formally establishes the relationship between the **input** and the **output** of an algorithm.

Example

Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.

Output: A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input, such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Computational problem

Definition

- Relationship between a set of **instances** (input) and a set of **solutions** (output).
- Formally establishes the relationship between the **input** and the **output** of an algorithm.

Example

Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.

Output: A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input, such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Computational problem

Definition

- Relationship between a set of **instances** (input) and a set of **solutions** (output).
- Formally establishes the relationship between the **input** and the **output** of an algorithm.

Example

Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.

Output: A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input, such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Computational problem

Definition

- Relationship between a set of **instances** (input) and a set of **solutions** (output).
- Formally establishes the relationship between the **input** and the **output** of an algorithm.

Example

Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.

Output: A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input, such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Algorithms

Definition (Algorithm)

An **algorithm** is any well-defined computational procedure that takes some value, or set of values, as **input** and produces some value, or set of values, as **output**.

- An algorithm is, therefore, a sequence of computational steps that transform input into output.
- We can also view an algorithm as a tool for solving a well-specified **computational problem**.

Example

For the instance (input) $\langle 31, 41, 59, 26, 41, 58\rangle$, a sorting algorithm must produce (output) $\langle 26, 31, 41, 41, 58, 59\rangle$.

Algorithms

Definition (Algorithm)

An **algorithm** is any well-defined computational procedure that takes some value, or set of values, as **input** and produces some value, or set of values, as **output**.

- An algorithm is, therefore, a sequence of computational steps that transform input into output.
- We can also view an algorithm as a tool for solving a well-specified **computational problem**.

Example

For the instance (input) $\langle 31, 41, 59, 26, 41, 58 \rangle$, a sorting algorithm must produce (output) $\langle 26, 31, 41, 41, 58, 59 \rangle$.

Algorithms

Definition (Algorithm)

An **algorithm** is any well-defined computational procedure that takes some value, or set of values, as **input** and produces some value, or set of values, as **output**.

- An algorithm is, therefore, a sequence of computational steps that transform input into output.
- We can also view an algorithm as a tool for solving a well-specified **computational problem**.

Example

For the instance (input) $\langle 31, 41, 59, 26, 41, 58\rangle$, a sorting algorithm must produce (output) $\langle 26, 31, 41, 41, 58, 59\rangle$.

Algorithms

Definition (Algorithm)

An **algorithm** is any well-defined computational procedure that takes some value, or set of values, as **input** and produces some value, or set of values, as **output**.

- An algorithm is, therefore, a sequence of computational steps that transform input into output.
- We can also view an algorithm as a tool for solving a well-specified **computational problem**.

Example

For the instance (input) $\langle 31, 41, 59, 26, 41, 58\rangle$, a sorting algorithm must produce (output) $\langle 26, 31, 41, 41, 58, 59\rangle$.

Algorithms & Computers

- Algorithms are essential in Computer Science.
- They provide a step-by-step procedure that can be translated into a computer program. It can be executed by a computer.
- The efficiency of a computer program depends on the efficiency of the underlying algorithm.
- The study of algorithms does not depend on computers.
- But computation by hand can be tedious, slow, error-prone and ineffective.
- Computers can execute instructions quickly and reliably.

Algorithms & Computers

- Algorithms are essential in Computer Science.
- They provide a step-by-step procedure that can be translated into a computer program. It can be executed by a computer.
- The efficiency of a computer program depends on the efficiency of the underlying algorithm.
- The study of algorithms does not depend on computers.
- But computation by hand can be tedious, slow, error-prone and ineffective.
- Computers can execute instructions quickly and reliably.

Algorithms & Computers

- Algorithms are essential in Computer Science.
- They provide a step-by-step procedure that can be translated into a computer program. It can be executed by a computer.
- The efficiency of a computer program depends on the efficiency of the underlying algorithm.
- The study of algorithms does not depend on computers.
- But computation by hand can be tedious, slow, error-prone and ineffective.
- Computers can execute instructions quickly and reliably.

Algorithms & Computers

- Algorithms are essential in Computer Science.
- They provide a step-by-step procedure that can be translated into a computer program. It can be executed by a computer.
- The efficiency of a computer program depends on the efficiency of the underlying algorithm.
- The study of algorithms does not depend on computers.
- But computation by hand can be tedious, slow, error-prone and ineffective.
- Computers can execute instructions quickly and reliably.

Algorithms & Computers

- Algorithms are essential in Computer Science.
- They provide a step-by-step procedure that can be translated into a computer program. It can be executed by a computer.
- The efficiency of a computer program depends on the efficiency of the underlying algorithm.
- The study of algorithms does not depend on computers.
- But computation by hand can be tedious, slow, error-prone and ineffective.
- Computers can execute instructions quickly and reliably.

Algorithms & Computers

- Algorithms are essential in Computer Science.
- They provide a step-by-step procedure that can be translated into a computer program. It can be executed by a computer.
- The efficiency of a computer program depends on the efficiency of the underlying algorithm.
- The study of algorithms does not depend on computers.
- But computation by hand can be tedious, slow, error-prone and ineffective.
- Computers can execute instructions quickly and reliably.

Algorithms & Computers

- An algorithm can be specified in:
  - Text in English
  - Text in Spanish
  - Pseudocode
  - Data Flow Diagrams
  - Computer program
  - Hardware Design
- The only requirement is that the specification must provide a precise description of the computational procedure to be followed.

Algorithms & Computers

- An algorithm can be specified in:
  - Text in English
  - Text in Spanish
  - Pseudocode
  - Data Flow Diagrams
  - Computer program
  - Hardware Design
- The only requirement is that the specification must provide a precise description of the computational procedure to be followed.

Algorithms & Computers

- An algorithm can be specified in:
  - Text in English
  - Text in Spanish
  - Pseudocode
  - Data Flow Diagrams
  - Computer program
  - Hardware Design
- The only requirement is that the specification must provide a precise description of the computational procedure to be followed.

Algorithms & Computers

- An algorithm can be specified in:
  - Text in English
  - Text in Spanish
  - Pseudocode
  - Data Flow Diagrams
  - Computer program
  - Hardware Design
- The only requirement is that the specification must provide a precise description of the computational procedure to be followed.

Algorithms & Computers

- An algorithm can be specified in:
  - Text in English
  - Text in Spanish
  - Pseudocode
  - Data Flow Diagrams
  - Computer program
  - Hardware Design
- The only requirement is that the specification must provide a precise description of the computational procedure to be followed.

Algorithms & Computers

- An algorithm can be specified in:
  - Text in English
  - Text in Spanish
  - Pseudocode
  - Data Flow Diagrams
  - Computer program
  - Hardware Design
- The only requirement is that the specification must provide a precise description of the computational procedure to be followed.

Algorithms & Computers

- An algorithm can be specified in:
    - Text in English
    - Text in Spanish
    - Pseudocode
    - Data Flow Diagrams
    - Computer program
    - Hardware Design
- The only requirement is that the specification must provide a precise description of the computational procedure to be followed.

Algorithms & Computers

- An algorithm can be specified in:
  - Text in English
  - Text in Spanish
  - Pseudocode
  - Data Flow Diagrams
  - Computer program
  - Hardware Design
- The only requirement is that the specification must provide a precise description of the computational procedure to be followed.

Different Algorithms

- There can be several algorithms from different approaches to solve the same computational problem. For example:
  - Insertion-sort
  - Merge-sort
  - Selected-sort
  - Heap-sort
  - Quick-sort
- Which algorithm is better for a given application? It depends of:
  - The number of items to be sorted.
  - The extent to which the items are already somewhat sorted.
  - Possible restrictions on the item values.

Different Algorithms

- There can be several algorithms from different approaches to solve the same computational problem. For example:
  - Insertion-sort
  - Merge-sort
  - Selected-sort
  - Heap-sort
  - Quick-sort
- Which algorithm is better for a given application? It depends of:
  - The number of items to be sorted.
  - The extent to which the items are already somewhat sorted.
  - Possible restrictions on the item values.

Different Algorithms

- There can be several algorithms from different approaches to solve the same computational problem. For example:
  - Insertion-sort
  - Merge-sort
  - Selected-sort
  - Heap-sort
  - Quick-sort
- Which algorithm is better for a given application? It depends of:
  - The number of items to be sorted.
  - The extent to which the items are already somewhat sorted.
  - Possible restrictions on the item values.

Different Algorithms

- There can be several algorithms from different approaches to solve the same computational problem. For example:
  - Insertion-sort
  - Merge-sort
  - Selected-sort
  - Heap-sort
  - Quick-sort
- Which algorithm is better for a given application? It depends of:
  - The number of items to be sorted.
  - The extent to which the items are already somewhat sorted.
  - Possible restrictions on the item values.

Different Algorithms

- There can be several algorithms from different approaches to solve the same computational problem. For example:
  - Insertion-sort
  - Merge-sort
  - Selected-sort
  - Heap-sort
  - Quick-sort
- Which algorithm is better for a given application? It depends of:
  - The number of items to be sorted.
  - The extent to which the items are already somewhat sorted.
  - Possible restrictions on the item values.

Different Algorithms

- There can be several algorithms from different approaches to solve the same computational problem. For example:
  - Insertion-sort
  - Merge-sort
  - Selected-sort
  - Heap-sort
  - Quick-sort
- Which algorithm is better for a given application? It depends of:
  - The number of items to be sorted.
  - The extent to which the items are already somewhat sorted.
  - Possible restrictions on the item values.

Different Algorithms

- There can be several algorithms from different approaches to solve the same computational problem. For example:
  - Insertion-sort
  - Merge-sort
  - Selected-sort
  - Heap-sort
  - Quick-sort
- Which algorithm is better for a given application? It depends of:
  - The number of items to be sorted.
  - The extent to which the items are already somewhat sorted.
  - Possible restrictions on the item values.

Different Algorithms

- There can be several algorithms from different approaches to solve the same computational problem. For example:
  - Insertion-sort
  - Merge-sort
  - Selected-sort
  - Heap-sort
  - Quick-sort
- Which algorithm is better for a given application? It depends of:
  - The number of items to be sorted.
  - The extent to which the items are already somewhat sorted.
  - Possible restrictions on the item values.

Different Algorithms

- There can be several algorithms from different approaches to solve the same computational problem. For example:
  - Insertion-sort
  - Merge-sort
  - Selected-sort
  - Heap-sort
  - Quick-sort
- Which algorithm is better for a given application? It depends of:
  - The number of items to be sorted.
  - The extent to which the items are already somewhat sorted.
  - Possible restrictions on the item values.

Different Algorithms

- There can be several algorithms from different approaches to solve the same computational problem. For example:
  - Insertion-sort
  - Merge-sort
  - Selected-sort
  - Heap-sort
  - Quick-sort
- Which algorithm is better for a given application? It depends of:
  - The number of items to be sorted.
  - The extent to which the items are already somewhat sorted.
  - Possible restrictions on the item values.

Data structures

Definition (Data structures)

A **data structure** is a way to store and organize data in order to facilitate access and modifications.

- No single data structure works well for all purposes.
- It is important to know the strengths and limitations of several of them.

Examples

Data structures

Definition (Data structures)

A **data structure** is a way to store and organize data in order to facilitate access and modifications.

- No single data structure works well for all purposes.
- It is important to know the strengths and limitations of several of them.

Examples

- List
- Linked List
- Array
- Stack
- Queue
- Graph
- Tree
- Heap
- Matrix
- Hash table

Data structures

Definition (Data structures)

A **data structure** is a way to store and organize data in order to facilitate access and modifications.

- No single data structure works well for all purposes.
- It is important to know the strengths and limitations of several of them.

Examples

- List
- Linked List
- Array
- Stack
- Queue
- Graph
- Tree
- Heap
- Matrix
- Hash table

Data structures

Definition (Data structures)

A **data structure** is a way to store and organize data in order to facilitate access and modifications.

- No single data structure works well for all purposes.
- It is important to know the strengths and limitations of several of them.

Examples

- List
- Linked List
- Array
- Stack
- Queue
- Graph
- Tree
- Heap
- Matrix
- Hash table

Data structures

Definition (Data structures)

A **data structure** is a way to store and organize data in order to facilitate access and modifications.

- No single data structure works well for all purposes.
- It is important to know the strengths and limitations of several of them.

Examples

- List
- Linked List
- Array
- Stack
- Queue
- Graph
- Tree
- Heap
- Matrix
- Hash table

Data structures

Definition (Data structures)

A **data structure** is a way to store and organize data in order to facilitate access and modifications.

- No single data structure works well for all purposes.
- It is important to know the strengths and limitations of several of them.

Examples

- List
- Linked List
- Array
- Stack
- Queue
- Graph
- Tree
- Heap
- Matrix
- Hash table

Data structures

Definition (Data structures)

A **data structure** is a way to store and organize data in order to facilitate access and modifications.

- No single data structure works well for all purposes.
- It is important to know the strengths and limitations of several of them.

Examples

- List
- Linked List
- Array
- Stack
- Queue
- Graph
- Tree
- Heap
- Matrix
- Hash table

Data structures

Definition (Data structures)

A **data structure** is a way to store and organize data in order to facilitate access and modifications.

- No single data structure works well for all purposes.
- It is important to know the strengths and limitations of several of them.

Examples

- List
- Linked List
- Array
- Stack
- Queue
- Graph
- Tree
- Heap
- Matrix
- Hash table

Data structures

Definition (Data structures)

A **data structure** is a way to store and organize data in order to facilitate access and modifications.

- No single data structure works well for all purposes.
- It is important to know the strengths and limitations of several of them.

Examples

- List
- Linked List
- Array
- Stack
- Queue
- Graph
- Tree
- Heap
- Matrix
- Hash table

Data structures

Definition (Data structures)

A **data structure** is a way to store and organize data in order to facilitate access and modifications.

- No single data structure works well for all purposes.
- It is important to know the strengths and limitations of several of them.

Examples

- List
- Linked List
- Array
- Stack
- Queue
- Graph
- Tree
- Heap
- Matrix
- Hash table

Data structures

Definition (Data structures)

A **data structure** is a way to store and organize data in order to facilitate access and modifications.

- No single data structure works well for all purposes.
- It is important to know the strengths and limitations of several of them.

Examples

- List
- Linked List
- Array
- Stack
- Queue
- Graph
- Tree
- Heap
- Matrix
- Hash table

Data structures

Definition (Data structures)

A **data structure** is a way to store and organize data in order to facilitate access and modifications.

- No single data structure works well for all purposes.
- It is important to know the strengths and limitations of several of them.

Examples

- List
- Linked List
- Array
- Stack
- Queue
- Graph
- Tree
- Heap
- Matrix
- Hash table

Data structures

Definition (Data structures)

A **data structure** is a way to store and organize data in order to facilitate access and modifications.

- No single data structure works well for all purposes.
- It is important to know the strengths and limitations of several of them.

Examples

- List
- Linked List
- Array
- Stack
- Queue
- Graph
- Tree
- Heap
- Matrix
- Hash table

Data structures

Definition (Data structures)

A **data structure** is a way to store and organize data in order to facilitate access and modifications.

- No single data structure works well for all purposes.
- It is important to know the strengths and limitations of several of them.

Examples

- List
- Linked List
- Array
- Stack
- Queue
- Graph
- Tree
- Heap
- Matrix
- Hash table

Efficiency I

Definition (Efficiency)

The **algorithmic efficiency** is the property of an algorithm related to the number of computational resources used to solve a computational problem.

- An algorithm must be analyzed to determine the resources it uses, since different algorithms designed to solve the same problem often differ dramatically in their efficiency.
- These differences can be much more significant than differences due to hardware and software.

Examples

Insert-sort takes $c_1 n^2$ instructions; *Merge-sort* takes $c_2 n \log n$ instructions. The number of instructions is proportional to the time required. Generally, $c_1 \ll c_2$.

Efficiency I

Definition (Efficiency)

The **algorithmic efficiency** is the property of an algorithm related to the number of computational resources used to solve a computational problem.

- An algorithm must be analyzed to determine the resources it uses, since different algorithms designed to solve the same problem often differ dramatically in their efficiency.
- These differences can be much more significant than differences due to hardware and software.

Examples

Insert-sort takes $c_1 n^2$ instructions; *Merge-sort* takes $c_2 n \log n$ instructions. The number of instructions is proportional to the time required. Generally, $c_1 \ll c_2$.

Efficiency I

Definition (Efficiency)

The **algorithmic efficiency** is the property of an algorithm related to the number of computational resources used to solve a computational problem.

- An algorithm must be analyzed to determine the resources it uses, since different algorithms designed to solve the same problem often differ dramatically in their efficiency.
- These differences can be much more significant than differences due to hardware and software.

Examples

Insert-sort takes $c_1 n^2$ instructions; *Merge-sort* takes $c_2 n \log n$ instructions. The number of instructions is proportional to the time required. Generally, $c_1 \ll c_2$.

Efficiency I

Definition (Efficiency)

The **algorithmic efficiency** is the property of an algorithm related to the number of computational resources used to solve a computational problem.

- An algorithm must be analyzed to determine the resources it uses, since different algorithms designed to solve the same problem often differ dramatically in their efficiency.
- These differences can be much more significant than differences due to hardware and software.

Examples

Insert-sort takes $c_1 n^2$ instructions; *Merge-sort* takes $c_2 n \log n$ instructions. The number of instructions is proportional to the time required. Generally, $c_1 \ll c_2$.

Efficiency II

Problem

- Suppose the time required for Insertion-sort is $3.6n^2$ instructions and it is run on computer A.
- Suppose the time required for Merge-sort is $200n \log n$ instructions and it is run on computer B.
- Computer A processes 1.000.000.000 instructions per second.
- Computer B processes 10.000.000 instructions per second.
- How much time in seconds is required by each one of them, when $n = 1.000.000$?.

Efficiency II

Problem

- Suppose the time required for Insertion-sort is $3.6n^2$ instructions and it is run on computer A.
- Suppose the time required for Merge-sort is $200n \log n$ instructions and it is run on computer B.
- Computer A processes 1.000.000.000 instructions per second.
- Computer B processes 10.000.000 instructions per second.
- How much time in seconds is required by each one of them, when $n = 1.000.000$?.

Efficiency II

Problem

- Suppose the time required for Insertion-sort is $3.6n^2$ instructions and it is run on computer A.
- Suppose the time required for Merge-sort is $200n\log n$ instructions and it is run on computer B.
- Computer A processes 1.000.000.000 instructions per second.
- Computer B processes 10.000.000 instructions per second.
- How much time in seconds is required by each one of them, when $n = 1.000.000$?.

Efficiency II

Problem

- Suppose the time required for Insertion-sort is $3.6n^2$ instructions and it is run on computer A.
- Suppose the time required for Merge-sort is $200n \log n$ instructions and it is run on computer B.
- Computer A processes 1.000.000.000 instructions per second.
- Computer B processes 10.000.000 instructions per second.
- How much time in seconds is required by each one of them, when $n = 1.000.000$?.

Efficiency II

Problem

- Suppose the time required for Insertion-sort is $3.6n^2$ instructions and it is run on computer A.
- Suppose the time required for Merge-sort is $200n \log n$ instructions and it is run on computer B.
- Computer A processes 1.000.000.000 instructions per second.
- Computer B processes 10.000.000 instructions per second.
- How much time in seconds is required by each one of them, when $n = 1.000.000$?.

Efficiency II

Problem

- Suppose the time required for Insertion-sort is $3.6n^2$ instructions and it is run on computer A.
- Suppose the time required for Merge-sort is $200n \log n$ instructions and it is run on computer B.
- Computer A processes 1.000.000.000 instructions per second.
- Computer B processes 10.000.000 instructions per second.
- How much time in seconds is required by each one of them, when $n = 1.000.000$?.

Outline

Correctness

Definition (Correctness)

An algorithm is said to be **correct** if, for every input instance, it halts with the correct output.

We say that a correct algorithm solves the given computational problem.

Incorrectness

An algorithm is incorrect if:

- It does not halt at all on some valid input instances.
- It halts with an answer other than the desired one.

An **incorrect** algorithm may be useful if the error rate can be controlled.

Correctness

Definition (Correctness)

An algorithm is said to be **correct** if, for every input instance, it halts with the correct output.

We say that a correct algorithm solves the given computational problem.

Incorrectness

An algorithm is incorrect if:

- It does not halt at all on some valid input instances.
- It halts with an answer other than the desired one.

An **incorrect** algorithm may be useful if the error rate can be controlled.

Correctness

Definition (Correctness)

An algorithm is said to be **correct** if, for every input instance, it halts with the correct output.

We say that a correct algorithm solves the given computational problem.

Incorrectness

An algorithm is incorrect if:

- It does not halt at all on some valid input instances.
- It halts with an answer other than the desired one.

An **incorrect** algorithm may be useful if the error rate can be controlled.

Correctness

Definition (Correctness)

An algorithm is said to be **correct** if, for every input instance, it halts with the correct output.

We say that a correct algorithm solves the given computational problem.

Incorrectness

An algorithm is incorrect if:

- It does not halt at all on some valid input instances.
- It halts with an answer other than the desired one.

An **incorrect** algorithm may be useful if the error rate can be controlled.

Loop Invariants

Definition (Loop Invariants)

A statement (property about a loop) is said to be a **loop invariant** if we can show following three properties.

Initialization: It is true prior to the first iteration of the loop.

Maintenance: If it is true before an iteration of the loop, it remains true before the next iteration.

Termination: When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.

Loop Invariants

Definition (Loop Invariants)

A statement (property about a loop) is said to be a **loop invariant** if we can show following three properties.

Initialization: It is true prior to the first iteration of the loop.

Maintenance: If it is true before an iteration of the loop, it remains true before the next iteration.

Termination: When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.

Loop Invariants

Definition (Loop Invariants)

A statement (property about a loop) is said to be a **loop invariant** if we can show following three properties.

Initialization: It is true prior to the first iteration of the loop.

Maintenance: If it is true before an iteration of the loop, it remains true before the next iteration.

Termination: When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.

Loop Invariants

Definition (Loop Invariants)

A statement (property about a loop) is said to be a **loop invariant** if we can show following three properties.

Initialization: It is true prior to the first iteration of the loop.

Maintenance: If it is true before an iteration of the loop, it remains true before the next iteration.

Termination: When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct.

The loop invariants to help us understand why an algorithm is correct.

Outline

1 Basic concepts

2 Correctness and Loop Invariants

3 Analysis and Design of Algorithms

Analysis of Algorithms

Definition (Analysis of Algorithms)

Analyzing an algorithm is trying to predict the measurement of the resources and the performance that the algorithm has when it is running.

Resources such as memory, bandwidth, power or computer hardware are important, but most of the time what we mainly want to measure is **computational time**.

- Generally, by analyzing several candidate algorithms for a problem, a most efficient one (or some) can be easily identified.
- Requirements: combinatorics, probability theory and algebra.

Analysis of Algorithms

Definition (Analysis of Algorithms)

Analyzing an algorithm is trying to predict the measurement of the resources and the performance that the algorithm has when it is running.

Resources such as memory, bandwidth, power or computer hardware are important, but most of the time what we mainly want to measure is **computational time**.

- Generally, by analyzing several candidate algorithms for a problem, a most efficient one (or some) can be easily identified.
- Requirements: combinatorics, probability theory and algebra.

Analysis of Algorithms

Definition (Analysis of Algorithms)

Analyzing an algorithm is trying to predict the measurement of the resources and the performance that the algorithm has when it is running.

Resources such as memory, bandwidth, power or computer hardware are important, but most of the time what we mainly want to measure is **computational time**.

- Generally, by analyzing several candidate algorithms for a problem, a most efficient one (or some) can be easily identified.
- Requirements: combinatorics, probability theory and algebra.

Implementation Technology

- Before we can analyze an algorithm, we must have a model of the **implementation technology** that will be used.
- We will assume a generic one processor, **random-access machine (RAM)** model of computation (instructions are executed one after another).
- Strictly speaking, one should precisely define the instructions of the RAM model and their costs. Tedious!
- Realistic operations are arithmetic, data movement (load, store, copy), and control (conditionals, subroutine call and return).
- Our analysis must be independent of a concrete machine (physics configuration).

Implementation Technology

- Before we can analyze an algorithm, we must have a model of the **implementation technology** that will be used.
- We will assume a generic one processor, **random-access machine (RAM)** model of computation (instructions are executed one after another).
- Strictly speaking, one should precisely define the instructions of the RAM model and their costs. Tedious!
- Realistic operations are arithmetic, data movement (load, store, copy), and control (conditionals, subroutine call and return).
- Our analysis must be independent of a concrete machine (physics configuration).

Implementation Technology

- Before we can analyze an algorithm, we must have a model of the **implementation technology** that will be used.
- We will assume a generic one processor, **random-access machine (RAM)** model of computation (instructions are executed one after another).
- Strictly speaking, one should precisely define the instructions of the RAM model and their costs. Tedious!
- Realistic operations are arithmetic, data movement (load, store, copy), and control (conditionals, subroutine call and return).
- Our analysis must be independent of a concrete machine (physics configuration).

Implementation Technology

- Before we can analyze an algorithm, we must have a model of the **implementation technology** that will be used.
- We will assume a generic one processor, **random-access machine (RAM)** model of computation (instructions are executed one after another).
- Strictly speaking, one should precisely define the instructions of the RAM model and their costs. Tedious!
- Realistic operations are arithmetic, data movement (load, store, copy), and control (conditionals, subroutine call and return).
- Our analysis must be independent of a concrete machine (physics configuration).

Implementation Technology

- Before we can analyze an algorithm, we must have a model of the **implementation technology** that will be used.
- We will assume a generic one processor, **random-access machine (RAM)** model of computation (instructions are executed one after another).
- Strictly speaking, one should precisely define the instructions of the RAM model and their costs. Tedious!
- Realistic operations are arithmetic, data movement (load, store, copy), and control (conditionals, subroutine call and return).
- Our analysis must be independent of a concrete machine (physics configuration).

More about Analyzing Algorithms

- The behavior of an algorithm may be different for each possible input.
- We need a means for summarizing that behavior in simple, easily understood formulas.
- There are many choices in deciding how to express our analysis. Choose the simplest that shows important characteristics!.
- The time taken by an algorithm grows with the size of the input.

More about Analyzing Algorithms

- The behavior of an algorithm may be different for each possible input.
- We need a means for summarizing that behavior in simple, easily understood formulas.
- There are many choices in deciding how to express our analysis. Choose the simplest that shows important characteristics!.
- The time taken by an algorithm grows with the size of the input.

More about Analyzing Algorithms

- The behavior of an algorithm may be different for each possible input.
- We need a means for summarizing that behavior in simple, easily understood formulas.
- There are many choices in deciding how to express our analysis. Choose the simplest that shows important characteristics!.
- The time taken by an algorithm grows with the size of the input.

More about Analyzing Algorithms

- The behavior of an algorithm may be different for each possible input.
- We need a means for summarizing that behavior in simple, easily understood formulas.
- There are many choices in deciding how to express our analysis. Choose the simplest that shows important characteristics!.
- The time taken by an algorithm grows with the size of the input.

Input size

- For many problems, such as sorting the most natural measure is the number of items in the input.
- For multiplying two integers, the best measure of input size is the total number of bits needed to represent the input in ordinary binary notation.
- Sometimes, it is more appropriate to describe the size of the input with two numbers rather than one. For example, in graphs order $(|V|)$ and size $(|E|)$.

Input size

- For many problems, such as sorting the most natural measure is the number of items in the input.
- For multiplying two integers, the best measure of input size is the total number of bits needed to represent the input in ordinary binary notation.
- Sometimes, it is more appropriate to describe the size of the input with two numbers rather than one. For example, in graphs order $(|V|)$ and size $(|E|)$.

Input size

- For many problems, such as sorting the most natural measure is the number of items in the input.
- For multiplying two integers, the best measure of input size is the total number of bits needed to represent the input in ordinary binary notation.
- Sometimes, it is more appropriate to describe the size of the input with two numbers rather than one. For example, in graphs order $(|V|)$ and size $(|E|)$.

Running time

- Is the number of **primitive operations** or **steps** executed.
- It is convenient to define the notion of step so that it is as **machine-independent** as possible.
- Each execution of the i-th line takes time c_i where c_i is a constant. Some operations like calling a subroutine may take more than constant time.
- We count the number of times each line is executed.
- This allows us to estimate the number of primitive operations performed.

Running time

- Is the number of **primitive operations** or **steps** executed.
- It is convenient to define the notion of step so that it is as **machine-independent** as possible.
- Each execution of the i-th line takes time c_i where c_i is a constant. Some operations like calling a subroutine may take more than constant time.
- We count the number of times each line is executed.
- This allows us to estimate the number of primitive operations performed.

Running time

- Is the number of **primitive operations** or **steps** executed.
- It is convenient to define the notion of step so that it is as **machine-independent** as possible.
- Each execution of the i-th line takes time c_i where c_i is a constant. Some operations like calling a subroutine may take more than constant time.
- We count the number of times each line is executed.
- This allows us to estimate the number of primitive operations performed.

Running time

- Is the number of **primitive operations** or **steps** executed.
- It is convenient to define the notion of step so that it is as **machine-independent** as possible.
- Each execution of the i-th line takes time c_i where c_i is a constant. Some operations like calling a subroutine may take more than constant time.
- We count the number of times each line is executed.
- This allows us to estimate the number of primitive operations performed.

Running time

- Is the number of **primitive operations** or **steps** executed.
- It is convenient to define the notion of step so that it is as **machine-independent** as possible.
- Each execution of the *i*-th line takes time c_i where c_i is a constant. Some operations like calling a subroutine may take more than constant time.
- We count the number of times each line is executed.
- This allows us to estimate the number of primitive operations performed.

Why should we analyze an algorithm?

- We want to know about its scalability: Does it support large input sizes?.
- We want to know about its behavior on the best, worst and average-case.
- What mathematical functions describe its behavior?.
- We use **asymptotic notation**.
- It allows to establish what is feasible and what is not feasible.

Why should we analyze an algorithm?

- We want to know about its scalability: Does it support large input sizes?.
- We want to know about its behavior on the best, worst and average-case.
- What mathematical functions describe its behavior?.
- We use **asymptotic notation**.
- It allows to establish what is feasible and what is not feasible.

Why should we analyze an algorithm?

- We want to know about its scalability: Does it support large input sizes?.
- We want to know about its behavior on the best, worst and average-case.
- What mathematical functions describe its behavior?.
- We use **asymptotic notation**.
- It allows to establish what is feasible and what is not feasible.

Why should we analyze an algorithm?

- We want to know about its scalability: Does it support large input sizes?.
- We want to know about its behavior on the best, worst and average-case.
- What mathematical functions describe its behavior?.
- We use **asymptotic notation**.
- It allows to establish what is feasible and what is not feasible.

Why should we analyze an algorithm?

- We want to know about its scalability: Does it support large input sizes?.
- We want to know about its behavior on the best, worst and average-case.
- What mathematical functions describe its behavior?.
- We use **asymptotic notation**.
- It allows to establish what is feasible and what is not feasible.

Exercise

For each function $f(n)$ and time t in the following table, determine the largest size n of a problem that can be solved in time t, assuming that the algorithm to solve the problem takes $f(n)$ microseconds.

$f(n)$	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
$\lg n$							
$\sqrt{n}$							
n							
$n \lg n$							
n^2							
n^3							
2^n							
$n!$							

Solution

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
$\lg n$	2^{10^6}	$2^{6\times10^7}$	$2^{3.6\times10^9}$	$2^{8.64\times10^{10}}$	$2^{2.59\times10^{12}}$	$2^{3.15\times10^{13}}$	$2^{3.15\times10^{15}}$
$\sqrt{n}$	10^{12}	3.6×10^{15}	1.3×10^{19}	7.46×10^{21}	6.72×10^{24}	9.95×10^{26}	9.95×10^{30}
n	10^6	6×10^7	3.6×10^9	8.64×10^{10}	2.59×10^{12}	3.15×10^{13}	3.15×10^{15}
$n\lg n$	6.24×10^4	2.8×10^6	1.33×10^8	2.76×10^9	7.19×10^{10}	7.98×10^{11}	6.86×10^{13}
n^2	1000	7745	60000	293938	1609968	5615692	56156922
n^3	100	391	1532	4420	13736	31593	146645
2^n	19	25	31	36	41	44	51
$n!$	9	11	12	13	15	16	17

Design of Algorithms

Definition (Design of Algorithms)

Is the strategy thought, used and designed to implementing a solution of a problem by means of an algorithm. For example: Divide and conquer, Greedy programming, Dynamic programming, Linear programming, Randomized algorithms, etc.