Analysis of Algorithms

Introduction

Andres Mendez-Vazquez

September 2, 2018

Outline

Outline

Cinvestav

Introduction

Informal definition

Informally, an algorithm is any well defined computational procedure that

- It takes some value, or set of values, as input.
- Then, it produces some value, or set of values, as output.

Introduction

Informal definition

Informally, an algorithm is any well defined computational procedure that

- It takes some value, or set of values, as input.
- Then, it produces some value, or set of values, as output.

Examples

Cinvestav

Introduction

Informal definition

Informally, an algorithm is any well defined computational procedure that

- It takes some value, or set of values, as input.
- Then, it produces some value, or set of values, as output.

Examples

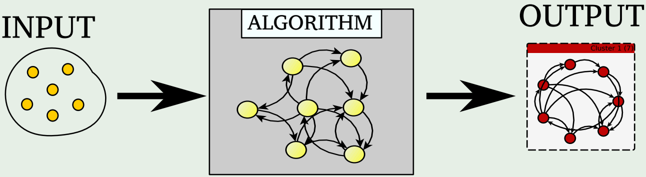
Introduction

Informal definition

Informally, an algorithm is any well defined computational procedure that

- It takes some value, or set of values, as input.
- Then, it produces some value, or set of values, as output.

Examples



Example

Sorting Problem

- **Input:** A sequence of N numbers $a_1, a_2, ..., a_N$
- **Output:** A reordering of the input sequence $a_{(1)}, a_{(2)}, ..., a_{(N)}$

Actually

We are dealing with **instances of a problem.**

Example

Sorting Problem

- **Input:** A sequence of N numbers $a_1, a_2, ..., a_N$
- **Output:** A reordering of the input sequence $a_{(1)}, a_{(2)}, ..., a_{(N)}$

Actually

We are dealing with **instances of a problem**.

Cinvestav

Stuff Like

A sequence of integer numbers

10	2	4	5	11	36	18	9	50

The Classic

- We want to order the numbers!!!

Stuff Like

A sequence of integer numbers

10	2	4	5	11	36	18	9	50

The Classic

- We want to order the numbers!!!

Outline

Cinvestav

Instance of a Problem

Instance of the problem

- For example, we have
  - 9, 8, 5, 6, 7, 4, 3, 2, 1
- Then, we finish with
  - 1, 2, 3, 4, 5, 6, 7, 8, 9

Although Instances are Important

Nevertheless

The way we use those instances is way more important

For example

Look at Recursive Fibonacci!!!

Although Instances are Important

Nevertheless

The way we use those instances is way more important

For example

Look at Recursive Fibonacci!!!

Example: Fibonacci

Fibonacci rule

- $F_n = \begin{cases} F_{n-1} + F_{n-2} & \text{if } n > 1 \\ 1 & \text{if } n = 1 \\ 0 & \text{if } n = 0 \end{cases}$

Time Complexity

1. Naive version using directly the recursion - exponential time.
2. A more elegant version - linear time.

Cinvestav

Example: Fibonacci

Fibonacci rule

- $F_n = \begin{cases} F_{n-1} + F_{n-2} & \text{if } n > 1 \\ 1 & \text{if } n = 1 \\ 0 & \text{if } n = 0 \end{cases}$

Time Complexity

1. Naive version using directly the recursion - exponential time.
2. A more elegant version - linear time.

Cinvestav

Outline

Cinvestav

Kolmogov's Definition

A bound for each sub-step

- An algorithmic process splits into steps whose complexity is bounded in advance
  - i.e., the bound is independent of the input and the current state of the computation.

Ending the Process

- The process runs until either the next step is impossible or a signal says the solution has been reached.

Kolmogov's Definition

A bound for each sub-step

- An algorithmic process splits into steps whose complexity is bounded in advance
  - i.e., the bound is independent of the input and the current state of the computation.

Transformations done at each step

- Each step consists of a direct and immediate transformation of the current state.
- This transformation applies only to the active part of the state and does not alter the remainder of the state.

Ending the Process

- The process runs until either the next step is impossible or a signal says the solution has been reached.

Kolmogov's Definition

A bound for each sub-step

- An algorithmic process splits into steps whose complexity is bounded in advance
  - i.e., the bound is independent of the input and the current state of the computation.

Transformations done at each step

- Each step consists of a direct and immediate transformation of the current state.
- This transformation applies only to the active part of the state and does not alter the remainder of the state.

Size of the steps

- The size of the active part is bounded in advance.

Ending the Process

- The process runs until either the next step is impossible or a signal says the solution has been reached.

Kolmogov's Definition

A bound for each sub-step

- An algorithmic process splits into steps whose complexity is bounded in advance
  - i.e., the bound is independent of the input and the current state of the computation.

Transformations done at each step

- Each step consists of a direct and immediate transformation of the current state.
- This transformation applies only to the active part of the state and does not alter the remainder of the state.

Size of the steps

- The size of the active part is bounded in advance.

Ending the Process

- The process runs until either the next step is impossible or a signal says the solution has been reached.

Kolmogov's Definition

A bound for each sub-step

- An algorithmic process splits into steps whose complexity is bounded in advance
  - i.e., the bound is independent of the input and the current state of the computation.

Transformations done at each step

- Each step consists of a direct and immediate transformation of the current state.
- This transformation applies only to the active part of the state and does not alter the remainder of the state.

Size of the steps

- The size of the active part is bounded in advance.

Ending the Process

- The process runs until either the next step is impossible or a signal says the solution has been reached.

Kolmogov's Definition

A bound for each sub-step

- An algorithmic process splits into steps whose complexity is bounded in advance
  - i.e., the bound is independent of the input and the current state of the computation.

Transformations done at each step

- Each step consists of a direct and immediate transformation of the current state.
- This transformation applies only to the active part of the state and does not alter the remainder of the state.

Size of the steps

- The size of the active part is bounded in advance.

Ending the Process

- The process runs until either the next step is impossible or a signal says the solution has been reached.

By The Way (BTW)

How do they look this machines, this algorithms?

After all we like to see them!!!

An Example of an Algorithm

Insertion Sort

Data: Unsorted Sequence A

Result: Sort Sequence A

`Insertion Sort(A)`

for $j \leftarrow 2$ **to** *lenght(A)* **do**

  $key \leftarrow A[j]$;

  `// Insert` $A[j]$ `Insert` $A[j]$ `into the sorted sequence` $A[1,...,j-1]$

  $i \leftarrow j-1$;

  while $i > 0$ *and* $A[i] > key$ **do**

    $A[i+1] \leftarrow A[i]$;

    $i \leftarrow i-1$;

  end

  $A[i+1] \leftarrow key$

end

Cinvestav

Outline

Cinvestav

Single-Source Shortest Path

Application → Short Paths in Maps

These algorithms allows to solve the problem of finding the shortest path in a map between two addresses.

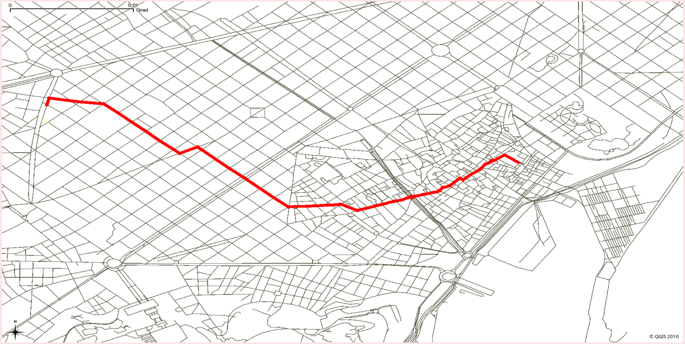
Example

Single-Source Shortest Path

Application → Short Paths in Maps

These algorithms allows to solve the problem of finding the shortest path in a map between two addresses.

Example



Solving Systems of Linear Equations

Application → Inverting Matrices

Because of stability reasons, given the system $Ax = y$, we use the the LUP decomposition or Cholensky decomposition to obtain the inverse A^{-1}.

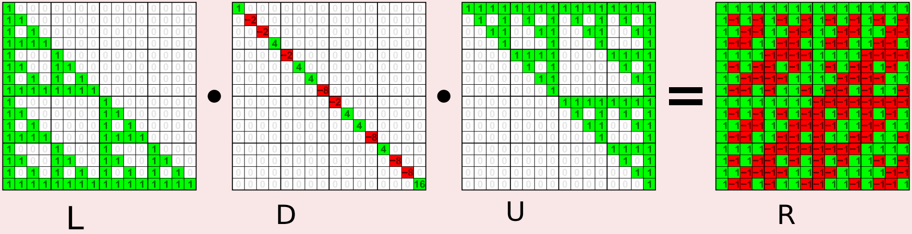
Example

Solving Systems of Linear Equations

Application → Inverting Matrices

Because of stability reasons, given the system $Ax = y$, we use the the LUP decomposition or Cholensky decomposition to obtain the inverse A^{-1}.

Example



@Copyright From Wikimedia Commons, the free media repository

Huffman Codes

Application → Compression

This method is part of the greedy methods. They are used for compression, they can achieve 20% to 90% compression in text files.

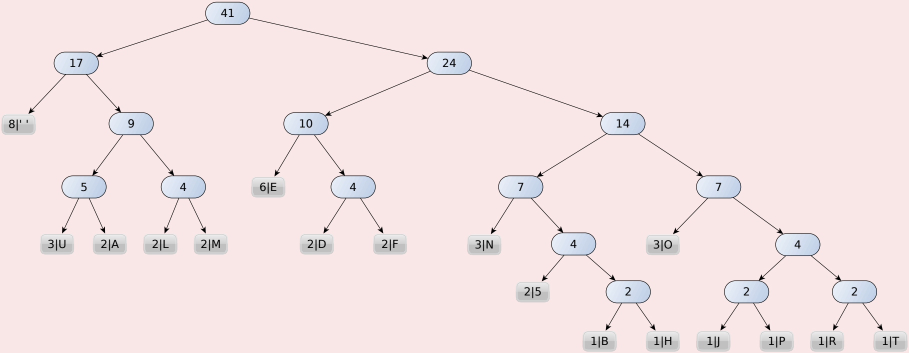
Example

Huffman Codes

Application → Compression

This method is part of the greedy methods. They are used for compression, they can achieve 20% to 90% compression in text files.

Example



Matrix Multiplication

Application → Fast Multiplication of Matrices

In many algorithms, we want to multiply different $n \times n$ matrices.

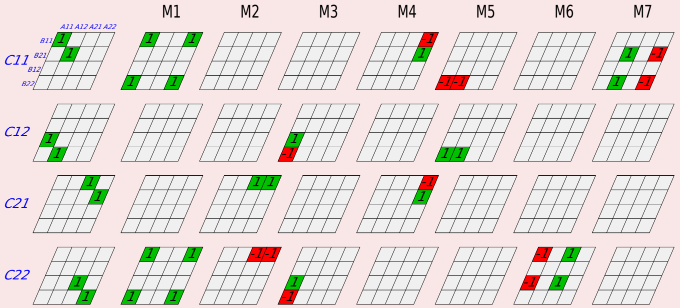
Example

Matrix Multiplication

Application → Fast Multiplication of Matrices

In many algorithms, we want to multiply different $n \times n$ matrices.

Example



STRASSEN'S ALGORITHM

@Copyright From Wikimedia Commons, the free media repository

Convex Hull

Application → Computational Geometry

Given the points in a plane, we want to find the minimum convex hull that encloses them.

Example

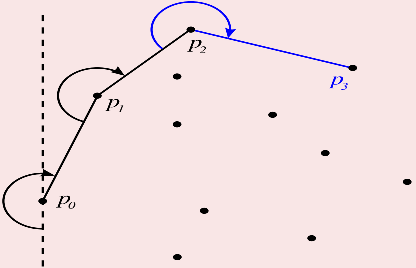
Cinvestav

Convex Hull

Application → Computational Geometry

Given the points in a plane, we want to find the minimum convex hull that encloses them.

Example



@Copyright From Wikimedia Commons, the free media repository

Synthetic Biology

Application → Computational Molecular Engineering

- In this field the engineers and biologist try to use the basis of life to create complex molecular machines.
- All these machines will requiere complex algorithms.

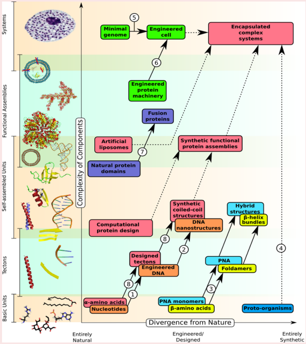
Example

Synthetic Biology

Application → Computational Molecular Engineering

- In this field the engineers and biologist try to use the basis of life to create complex molecular machines.
- All these machines will requiere complex algorithms.

Example



Pattern Recognition

Application → Machine Learning

In Machine Learning, we try to find specific patterns in data.

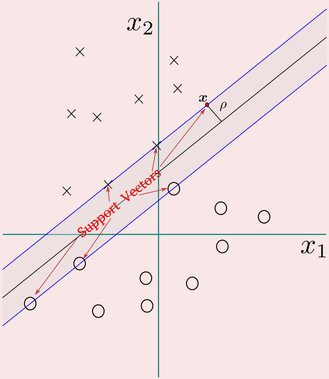
Example

Pattern Recognition

Application → Machine Learning

In Machine Learning, we try to find specific patterns in data.

Example



Databases

Application → Partition of the Database Space

For fast access Queries!!!

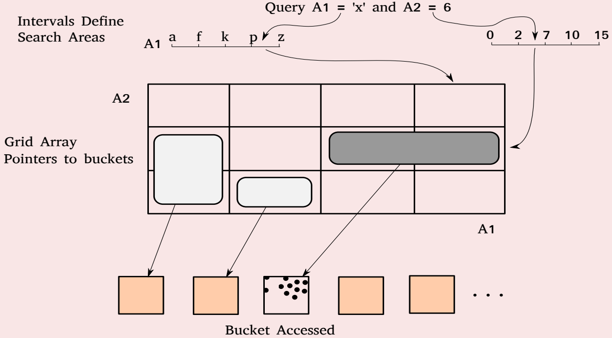
Example

Databases

Application → Partition of the Database Space

For fast access Queries!!!

Example



@Copyright Michael Unwalla: A mixed transaction cost model for coarse grained multi-column partitioning in a shared-nothing database machine

Databases

Application → Face Recognition

Facial Recognition measure face landmarks to identify different features in the face.

Example

Databases

Application → Face Recognition

Facial Recognition measure face landmarks to identify different features in the face.

Example

Outline

Cinvestav

Subjects for the Class

Growth Functions

- Asymptotic Notation - Ω, O and Θ
- Standard notation and common functions
- Solving Recursions

Cinvestav

Subjects for the Class

Growth Functions

- Asymptotic Notation - Ω, O and Θ
- Standard notation and common functions
- Solving Recursions

Divide and Conquer

- The substitution method
- The recursive three method
- The master method

Subjects for the Class

Growth Functions

- Asymptotic Notation - Ω, O and Θ
- Standard notation and common functions
- Solving Recursions

Divide and Conquer

- The substitution method
- The recursive three method
- The master method

Subjects for the Class

Growth Functions

- Asymptotic Notation - Ω, O and Θ
- Standard notation and common functions
- Solving Recursions

Divide and Conquer

- The substitution method
- The recursive three method
- The master method

Cinvestav

Subjects for the Class

Growth Functions

- Asymptotic Notation - Ω, O and Θ
- Standard notation and common functions
- Solving Recursions

Divide and Conquer

- The substitution method
- The recursive three method
- The master method

Cinvestav

Subjects for the Class

Growth Functions

- Asymptotic Notation - Ω, O and Θ
- Standard notation and common functions
- Solving Recursions

Divide and Conquer

- The substitution method
- The recursive three method
- The master method

Cinvestav

Subjects for the Class

Probabilistic Analysis

- Indicator Random Variables
- Randomization Algorithms

Subjects for the Class

Probabilistic Analysis

- Indicator Random Variables
- Randomization Algorithms

Sorting Algorithms

- Heapsort
- Quicksort
- Sorting in linear time

Cinvestav

Subjects for the Class

Probabilistic Analysis

- Indicator Random Variables
- Randomization Algorithms

Sorting Algorithms

- Heapsort
- Quicksort
- Sorting in linear time

Subjects for the Class

Probabilistic Analysis

- Indicator Random Variables
- Randomization Algorithms

Sorting Algorithms

- Heapsort
- Quicksort
- Sorting in linear time

Subjects for the Class

Probabilistic Analysis

- Indicator Random Variables
- Randomization Algorithms

Sorting Algorithms

- Heapsort
- Quicksort
- Sorting in linear time

Subject for the Class

Median Order Statistics

- Minimum and Maximum.
- Selection.
- Worst Case Selection.

- Hash Tables
- Binary Search Trees

Subject for the Class

Median Order Statistics

- Minimum and Maximum.
- Selection.
- Worst Case Selection.

Review of Basic Data Structures

- Elementary Data Structures
- Hash Tables
- Binary Search Trees

Subject for the Class

Median Order Statistics

- Minimum and Maximum.
- Selection.
- Worst Case Selection.

Review of Basic Data Structures

- Elementary Data Structures
- Hash Tables
- Binary Search Trees

Cinvestav

Subject for the Class

Median Order Statistics

- Minimum and Maximum.
- Selection.
- Worst Case Selection.

Review of Basic Data Structures

- Elementary Data Structures
- Hash Tables
- Binary Search Trees

Subject for the Class

Median Order Statistics

- Minimum and Maximum.
- Selection.
- Worst Case Selection.

Review of Basic Data Structures

- Elementary Data Structures
- Hash Tables
- Binary Search Trees

Cinvestav

Subject for the Class

Median Order Statistics

- Minimum and Maximum.
- Selection.
- Worst Case Selection.

Review of Basic Data Structures

- Elementary Data Structures
- Hash Tables
- Binary Search Trees

Subjects for the Class

Advanced Data Structures

- B-Trees
- Fibonacci Heaps
- Data Structures for Disjoint Sets

- Greedy Algorithms
- Amortized Analysis

Cinvestav

Subjects for the Class

Advanced Data Structures

- B-Trees
- Fibonacci Heaps
- Data Structures for Disjoint Sets

Advanced Techniques

- Dynamic Programming.
- Greedy Algorithms.
- Amortized Analysis.

Subjects for the Class

Advanced Data Structures

- B-Trees
- Fibonacci Heaps
- Data Structures for Disjoint Sets

Advanced Techniques

- Dynamic Programming.
- Greedy Algorithms.
- Amortized Analysis.

Subjects for the Class

Advanced Data Structures

- B-Trees
- Fibonacci Heaps
- Data Structures for Disjoint Sets

Advanced Techniques

- Dynamic Programming.
- Greedy Algorithms.
- Amortized Analysis.

Cinvestav

Subjects for the Class

Advanced Data Structures

- B-Trees
- Fibonacci Heaps
- Data Structures for Disjoint Sets

Advanced Techniques

- Dynamic Programming.
- Greedy Algorithms.
- Amortized Analysis.

Cinvestav

Subjects for the Class

Advanced Data Structures

- B-Trees
- Fibonacci Heaps
- Data Structures for Disjoint Sets

Advanced Techniques

- Dynamic Programming.
- Greedy Algorithms.
- Amortized Analysis.

Subjects for the Class

Graph Algorithms

- Elementary Graph Algorithms
- Minimum Spanning Trees
- Single-Source Shortest Paths
- All-Pairs Shortest Paths

- String Matching
- Computational Geometry

Subjects for the Class

Graph Algorithms

- Elementary Graph Algorithms
- Minimum Spanning Trees
- Single-Source Shortest Paths
- All-Pairs Shortest Paths

Selected Topics

- Multi-threaded Algorithms
- String Matching
- Computational Geometry

Subjects for the Class

Graph Algorithms

- Elementary Graph Algorithms
- Minimum Spanning Trees
- Single-Source Shortest Paths
- All-Pairs Shortest Paths

Selected Topics

- Multi-threaded Algorithms
- String Matching
- Computational Geometry

Cinvestav

Subjects for the Class

Graph Algorithms

- Elementary Graph Algorithms
- Minimum Spanning Trees
- Single-Source Shortest Paths
- All-Pairs Shortest Paths

Selected Topics

- Multi-threaded Algorithms
- String Matching
- Computational Geometry

Subjects for the Class

Graph Algorithms

- Elementary Graph Algorithms
- Minimum Spanning Trees
- Single-Source Shortest Paths
- All-Pairs Shortest Paths

Selected Topics

- Multi-threaded Algorithms
- String Matching
- Computational Geometry

Subjects for the Class

Graph Algorithms

- Elementary Graph Algorithms
- Minimum Spanning Trees
- Single-Source Shortest Paths
- All-Pairs Shortest Paths

Selected Topics

- Multi-threaded Algorithms
- String Matching
- Computational Geometry

Subjects for the Class

Graph Algorithms

- Elementary Graph Algorithms
- Minimum Spanning Trees
- Single-Source Shortest Paths
- All-Pairs Shortest Paths

Selected Topics

- Multi-threaded Algorithms
- String Matching
- Computational Geometry

Cinvestav

Subjects for the Class

NP Problems

- Encodings
- Polynomial Time Verification
- Polynomial Reduction
- NP-Hard
- NP-Complete proofs
- A family of NP-Problems

- Branch and Bound

Subjects for the Class

NP Problems

- Encodings
- Polynomial Time Verification
- Polynomial Reduction
- NP-Hard
- NP-Complete proofs
- A family of NP-Problems

Dealing with NP Problems

- Backtracking
- Branch-and-Bound

Subjects for the Class

NP Problems

- Encodings
- Polynomial Time Verification
- Polynomial Reduction
- NP-Hard
- NP-Complete proofs
- A family of NP-Problems

Dealing with NP Problems

- Backtracking
- Branch-and-Bound

Cinvestav

Subjects for the Class

NP Problems

- Encodings
- Polynomial Time Verification
- Polynomial Reduction
- NP-Hard
- NP-Complete proofs
- A family of NP-Problems

Dealing with NP Problems

- Backtracking
- Branch-and-Bound

Cinvestav

Subjects for the Class

NP Problems

- Encodings
- Polynomial Time Verification
- Polynomial Reduction
- NP-Hard
- NP-Complete proofs
- A family of NP-Problems

Dealing with NP Problems

- Backtracking
- Branch-and-Bound

Subjects for the Class

NP Problems

- Encodings
- Polynomial Time Verification
- Polynomial Reduction
- NP-Hard
- NP-Complete proofs
- A family of NP-Problems

Dealing with NP Problems

- Backtracking
- Branch-and-Bound

Subjects for the Class

NP Problems

- Encodings
- Polynomial Time Verification
- Polynomial Reduction
- NP-Hard
- NP-Complete proofs
- A family of NP-Problems

Dealing with NP Problems

- Backtracking
- Branch-and-Bound

Cinvestav

Subjects for the Class

NP Problems

- Encodings
- Polynomial Time Verification
- Polynomial Reduction
- NP-Hard
- NP-Complete proofs
- A family of NP-Problems

Dealing with NP Problems

- Backtracking
- Branch-and-Bound

Outline

Now, what we are going to look at!!!

First

Some stuff about notation!!!

Second

What abstraction of a Computer to use?

Third

A first approach to analyzing algorithms!!!

Now, what we are going to look at!!!

First

Some stuff about notation!!!

Second

What abstraction of a Computer to use?

Third

A first approach to analyzing algorithms!!!

Now, what we are going to look at!!!

First

Some stuff about notation!!!

Second

What abstraction of a Computer to use?

Third

A first approach to analyzing algorithms!!!

Cinvestav

Outline

Cinvestav

Please Follow These Simple Rules

Insertion Sort(A)

for $j \leftarrow 2$ to length(A)
    do
        $key \leftarrow A[j]$
        ▶Insert $A[j]$ into the sorted sequence $A[1, ..., j-1]\}$
        $i \leftarrow j - 1$
        while $i > 0$ and $A[i] > key$
            do
                $A[i+1] \leftarrow A[i]$
                $i \leftarrow i - 1$
        $A[i+1] \leftarrow key$

Rule

- Always put the name of the algorithm at the top together with the input.

Please Follow These Simple Rules

Insertion Sort(A)

for $j \leftarrow 2$ to length(A)
    do
        $key \leftarrow A[j]$
        ▶Insert $A[j]$ into the sorted sequence $A[1, ..., j-1]\}$
        $i \leftarrow j - 1$
        while $i > 0$ and $A[i] > key$
            do
                $A[i+1] \leftarrow A[i]$
                $i \leftarrow i - 1$
        $A[i+1] \leftarrow key$

Rule

- Always initialize all the variables.
- The $a \leftarrow b$(You also can use "$=$.") means that the value b is passed to a.

Please Follow These Simple Rules

Insertion Sort(A)

for $j \leftarrow 2$ to length(A)
    do
        $key \leftarrow A[j]$
        ▶Insert $A[j]$ into the sorted sequence $A[1, ..., j-1]\}$
        $i \leftarrow j - 1$
        while $i > 0$ and $A[i] > key$
            do
                $A[i+1] \leftarrow A[i]$
                $i \leftarrow i - 1$
        $A[i+1] \leftarrow key$

Rule

- Use indentation to preserve the block structure avoiding clutter.

Please Follow These Simple Rules

Insertion Sort(A)

```
for j ← 2 to length(A)
    do
        key ← A[j]
        ▶Insert A[j] into the sorted sequence A[1,...,j − 1]}
        i ← j − 1
        while i > 0 and A[i] > key
            do
                A[i + 1] ← A[i]
                i ← i − 1
        A[i + 1] ← key
```

Rule

- ▶ it corresponds to comments and you can also use "//"

Outline

Cinvestav

The Random-Access Machine

Definition

A Random Access Machine (RAM) is an abstract computational-machine model identical to a multiple-register counter machine with the addition of indirect addressing.

- It contains arithmetic instructions found in low level languages
- It has control instructions: Conditional and unconditional branches, return and call functions.
- It is able to do data movement: load, store, copy.
- It posses data types: integer and floating point.

The Random-Access Machine

Definition

A Random Access Machine (RAM) is an abstract computational-machine model identical to a multiple-register counter machine with the addition of indirect addressing.

Instructions

Instructions are executed one after another.

- It contains arithmetic instructions found in low level languages.
- It has control instructions: Conditional and unconditional branches, return and call functions.
- It is able to do data movement: **load, store, copy.**
- It posses data types: **integer and floating point.**

The Random-Access Machine

Definition

A Random Access Machine (RAM) is an abstract computational-machine model identical to a multiple-register counter machine with the addition of indirect addressing.

Instructions

Instructions are executed one after another.

- It contains arithmetic instructions found in low level languages.
- It has control instructions: Conditional and unconditional branches, return and call functions.
- It is able to do data movement: **load, store, copy.**
- It posses data types: **integer and floating point.**

Memory Model

A single block of memory is assumed.

The Random-Access Machine

Definition

A Random Access Machine (RAM) is an abstract computational-machine model identical to a multiple-register counter machine with the addition of indirect addressing.

Instructions

Instructions are executed one after another.

- It contains arithmetic instructions found in low level languages.
- It has control instructions: Conditional and unconditional branches, return and call functions.
- It is able to do data movement: **load, store, copy.**
- It posses data types: **integer and floating point.**

Memory Model

A single block of memory is assumed.

The Random-Access Machine

Definition

A Random Access Machine (RAM) is an abstract computational-machine model identical to a multiple-register counter machine with the addition of indirect addressing.

Instructions

Instructions are executed one after another.

- It contains arithmetic instructions found in low level languages.
- It has control instructions: Conditional and unconditional branches, return and call functions.
- It is able to do data movement: **load, store, copy.**
- It posses data types: **integer and floating point.**

Memory Model

A single block of memory is assumed.

The Random-Access Machine

Definition

A Random Access Machine (RAM) is an abstract computational-machine model identical to a multiple-register counter machine with the addition of indirect addressing.

Instructions

Instructions are executed one after another.

- It contains arithmetic instructions found in low level languages.
- It has control instructions: Conditional and unconditional branches, return and call functions.
- It is able to do data movement: **load, store, copy.**
- It posses data types: **integer and floating point.**

Memory Model

A single block of memory is assumed.

The Random-Access Machine

Definition

A Random Access Machine (RAM) is an abstract computational-machine model identical to a multiple-register counter machine with the addition of indirect addressing.

Instructions

Instructions are executed one after another.

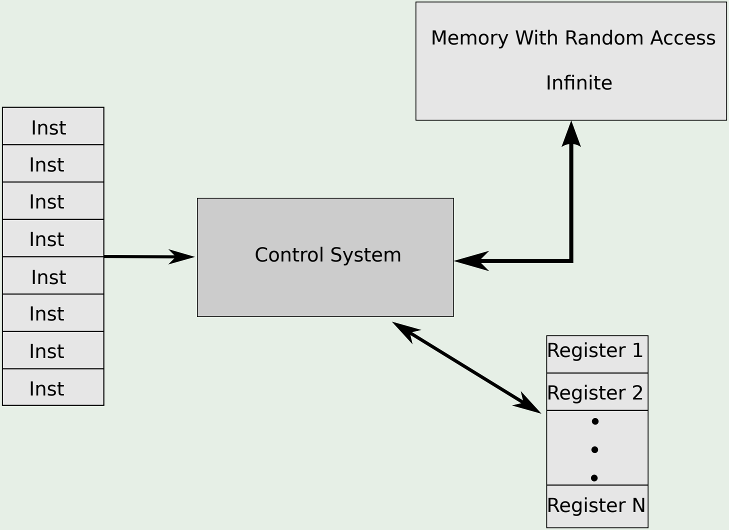
- It contains arithmetic instructions found in low level languages.
- It has control instructions: Conditional and unconditional branches, return and call functions.
- It is able to do data movement: **load, store, copy.**
- It posses data types: **integer and floating point.**

Memory Model

A single block of memory is assumed.

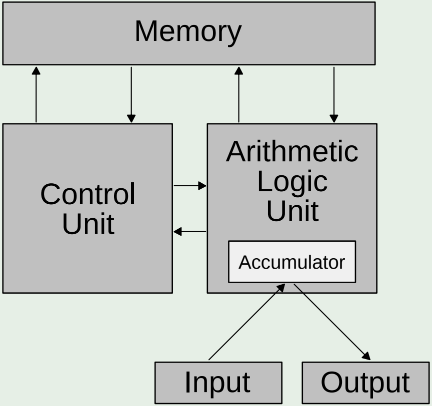
RAM Model

We have that



Although there are other equivalent models

Von Neumann architecture scheme



Outline

Cinvestav

Input Size and Running Time

Definition

The Input Size depends on the type of problem. We will indicate which input size is used per problem.

Definition

The Running Time of an algorithm is the number of of primitives operations or steps executed. For now, we will assume that each line in an algorithm takes c_i a constant time.

Input Size and Running Time

Definition

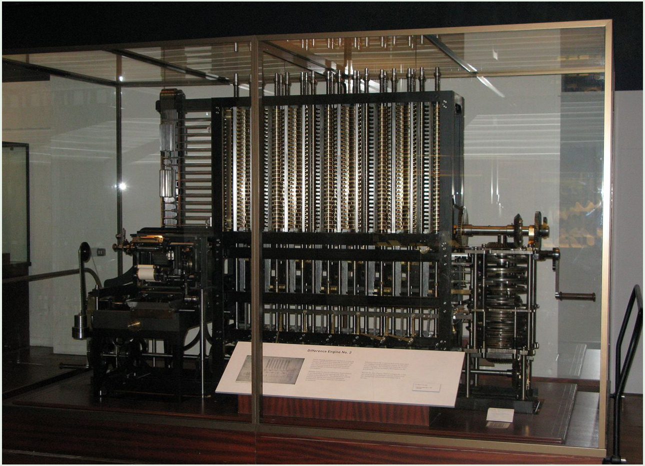
The Input Size depends on the type of problem. We will indicate which input size is used per problem.

Definition

The Running Time of an algorithm is the number of of primitives operations or steps executed. For now, we will assume that each line in an algorithm takes c_i a constant time.

Even Babbage cared about how many turns of the crank were necessary!!!

Look at the crank!!!



Outline

Cinvestav

We are going to do some quite simple

Counting the number of operations

- Therefore we have the following equivalences using algebraic sums...

We are going to do some quite simple

Counting the number of operations

- Therefore we have the following equivalences using algebraic sums...

Loops equivalent to Sums

$$\text{while } i > 0 \text{ and } A[i] > key \longleftrightarrow \sum_{j=2}^{N} 1 = N - 1$$

- Therefore the total cost of a loop would be $c_j (N-1)$

We are going to do some quite simple

Counting the number of operations

- Therefore we have the following equivalences using algebraic sums...

Loops equivalent to Sums

$$\text{while } i > 0 \text{ and } A[i] > key \longleftrightarrow \sum_{j=2}^{N} 1 = N - 1$$

Therefore

- We have that each operation i cost a certain time c_i
- Therefore the total cost of a loop would be $c_i\,(N-1)$

Cinvestav

We are going to do some quite simple

Counting the number of operations

- Therefore we have the following equivalences using algebraic sums...

Loops equivalent to Sums

$$\text{while } i > 0 \text{ and } A[i] > key \longleftrightarrow \sum_{j=2}^{N} 1 = N - 1$$

Therefore

- We have that each operation i cost a certain time c_i
- Therefore the total cost of a loop would be $c_i\,(N-1)$

Counting the Operations

Insertion Sort(A)

```
for j ← 2 to length(A)
    do
        key ← A[j]
        ▶Insert A[j] into the sorted sequence A[1, ..., j − 1]}
        i ← j − 1
        while i > 0 and A[i] > key
            do
                A[i + 1] ← A[i]
                i ← i − 1
        A[i + 1] ← key
```

Count Value

$\rightarrow \ c_1 N$

Counting the Operations

Insertion Sort(A)

```
for j ← 2 to length(A)
    do
        key ← A[j]
        ▶Insert A[j] into the sorted sequence A[1, ..., j − 1]}
        i ← j − 1
        while i > 0 and A[i] > key
            do
                A[i + 1] ← A[i]
                i ← i − 1
        A[i + 1] ← key
```

Count Value

$\rightarrow \; c_2\,(N-1)$

Counting the Operations

Insertion Sort(A)

for $j \leftarrow 2$ to length(A)
    do
        $key \leftarrow A[j]$
        ▶Insert $A[j]$ into the sorted sequence $A[1, ..., j-1]\}$
        $i \leftarrow j - 1$
        while $i > 0$ and $A[i] > key$
            do
                $A[i+1] \leftarrow A[i]$
                $i \leftarrow i - 1$
        $A[i+1] \leftarrow key$

Count Value

$\rightarrow \; c_3 \, (N - 1)$

Counting the Operations

Insertion Sort(A)

```
for j ← 2 to length(A)
    do
        key ← A[j]
        ▶Insert A[j] into the sorted sequence A[1, ..., j − 1]}
        i ← j − 1
        while i > 0 and A[i] > key
            do
                A[i + 1] ← A[i]
                i ← i − 1
        A[i + 1] ← key
```

Count Value

$\rightarrow \quad c_4 \sum_{j=2}^{N} j$

Counting the Operations

Insertion Sort(A)

for $j \leftarrow 2$ to length(A)
    do
        $key \leftarrow A[j]$
        ▶Insert $A[j]$ into the sorted sequence $A[1, ..., j-1]\}$
        $i \leftarrow j - 1$
        while $i > 0$ and $A[i] > key$
            do
                $A[i+1] \leftarrow A[i]$
                $i \leftarrow i - 1$
        $A[i+1] \leftarrow key$

Count Value

$\rightarrow \; c_5 \sum_{j=2}^{N} (j-1)$

Counting the Operations

Insertion Sort(A)

```
for j ← 2 to length(A)
    do
        key ← A[j]
        ▶Insert A[j] into the sorted sequence A[1, ..., j − 1]}
        i ← j − 1
        while i > 0 and A[i] > key
            do
                A[i + 1] ← A[i]
                i ← i − 1
        A[i + 1] ← key
```

Count Value

$\rightarrow \quad c_6 \sum_{j=2}^{N} (j-1)$

Counting the Operations

Insertion Sort(A)

for $j \leftarrow 2$ to length(A)
    do
        $key \leftarrow A[j]$
        ▶Insert $A[j]$ into the sorted sequence $A[1, ..., j-1]\}$
        $i \leftarrow j - 1$
        while $i > 0$ and $A[i] > key$
            do
                $A[i+1] \leftarrow A[i]$
                $i \leftarrow i - 1$
        $A[i+1] \leftarrow key$

Count Value

$\rightarrow \; c_7 \, (N-1)$

Outline

Cinvestav

The $T(N)$ function

The total number of operations

It is know as a function $T(N)$

$$T : \mathbb{N} \longmapsto \mathbb{N}$$

Something Notable

This generic name will be also be used for the recursive functions!!!

The $T(N)$ function

The total number of operations

It is know as a function $T(N)$

$$T : \mathbb{N} \longmapsto \mathbb{N}$$

Something Notable

This generic name will be also be used for the recursive functions!!!

Building a function for counting

Counting Equation

$$T(N) = c_1 N + c_2(N-1) + C_3(N-1) + c_4\left(\frac{N(N+1)}{2} - 1\right) + \ldots$$
$$c_5\left(\frac{N(N-1)}{2} - 1\right) + c_6\left(\frac{N(N-1)}{2} - 1\right) + c_7\left(N-1\right)$$

This can be reduced to something like...

$$T(N) = aN^2 + bN + c$$

Building a function for counting

Counting Equation

$$T(N) = c_1 N + c_2(N-1) + C_3(N-1) + c_4\left(\frac{N(N+1)}{2} - 1\right) + \ldots$$
$$c_5\left(\frac{N(N-1)}{2} - 1\right) + c_6\left(\frac{N(N-1)}{2} - 1\right) + c_7\left(N-1\right)$$

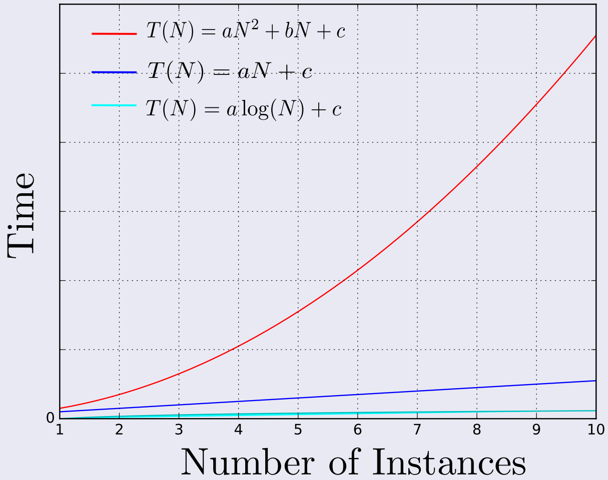
This can be reduced to something like...

$$T(N) = aN^2 + bN + c$$

Cinvestav

Example of Complexities

Something Notable



Outline

Cinvestav

The Worst and The Average Case Inputs

The Worst Case Input

- Upper bound on the running time of an algorithm.
- In case of **insertion sort**, it will be the permutation:

$$N, N-1, N-2, ..., 3, 2, 1 \tag{1}$$

The Worst and The Average Case Inputs

The Worst Case Input

- Upper bound on the running time of an algorithm.
- In case of **insertion sort**, it will be the permutation:

$$N, N-1, N-2, ..., 3, 2, 1 \quad (1)$$

The Average Case Input

- In the case of insertion sort, if half of the elements of $A[1, 2, ..., j-1]$ are less than $A[j]$ and half are greater.
- Then, **insertion sort** checks half of the elements i.e.:

$$\frac{j}{2} \quad (2)$$

The Worst and The Average Case Inputs

The Worst Case Input

- Upper bound on the running time of an algorithm.
- In case of **insertion sort**, it will be the permutation:

$$N, N-1, N-2, ..., 3, 2, 1 \tag{1}$$

The Average Case Input

- In the case of insertion sort, if half of the elements of $A[1, 2, ..., j-1]$ are less than $A[j]$ and half are greater.
- Then, **insertion sort** checks half of the elements i.e.:

$$\frac{j}{2} \tag{2}$$

The Worst and The Average Case Inputs

The Worst Case Input

- Upper bound on the running time of an algorithm.
- In case of **insertion sort**, it will be the permutation:

$$N, N-1, N-2, ..., 3, 2, 1 \tag{1}$$

The Average Case Input

- In the case of insertion sort, if half of the elements of $A[1, 2, ..., j-1]$ are less than $A[j]$ and half are greater.
- Then, **insertion sort** checks half of the elements i.e.:

$$\frac{j}{2} \tag{2}$$

The Worst and The Average Case Inputs

The Worst Case Input

- Upper bound on the running time of an algorithm.
- In case of **insertion sort**, it will be the permutation:

$$N, N-1, N-2, ..., 3, 2, 1 \tag{1}$$

The Average Case Input

- In the case of insertion sort, if half of the elements of $A[1, 2, ..., j-1]$ are less than $A[j]$ and half are greater.
- Then, **insertion sort** checks half of the elements i.e.:

$$\frac{j}{2} \tag{2}$$

The Worst and The Average Case Inputs

The Worst Case Input

- Upper bound on the running time of an algorithm.
- In case of **insertion sort**, it will be the permutation:

$$N, N-1, N-2, ..., 3, 2, 1 \tag{1}$$

The Average Case Input

- In the case of insertion sort, if half of the elements of $A[1, 2, ..., j-1]$ are less than $A[j]$ and half are greater.
- Then, **insertion sort** checks half of the elements i.e.:

$$\frac{j}{2} \tag{2}$$

Outline

Cinvestav

Why Do We Want Efficient Algorithms?

We have the following example

Assume, we have 10^6 numbers to sort!!!

Why Do We Want Efficient Algorithms?

We have the following example

Assume, we have 10^6 numbers to sort!!!

Now, we have the following algorithms

- Insertion Sort $\rightarrow$ $T(N) = c_1 N^2$
- Merge Sort $\rightarrow$ $T(N) = c_2 N \log_2 N$

- In our humble PC $c_2 = 50$ instructions per line

Why Do We Want Efficient Algorithms?

We have the following example

Assume, we have 10^6 numbers to sort!!!

Now, we have the following algorithms

- Insertion Sort $\rightarrow$ $T(N) = c_1 N^2$
- Merge Sort $\rightarrow$ $T(N) = c_2 N \log_2 N$

Under the following constraints for a PC and Supercomputer

- In a Supercomputer $c_1 = 2$ instructions per line
- In our humble PC $c_2 = 50$ instructions per line

Cinvestav

Why Do We Want Efficient Algorithms?

We have the following example

Assume, we have 10^6 numbers to sort!!!

Now, we have the following algorithms

- Insertion Sort $\rightarrow$ $T(N) = c_1 N^2$
- Merge Sort $\rightarrow$ $T(N) = c_2 N \log_2 N$

Under the following constraints for a PC and Supercomputer

- In a Supercomputer $c_1 = 2$ instructions per line
- In our humble PC $c_2 = 50$ instructions per line

Cinvestav

Why Do We Want Efficient Algorithms?

We have the following example

Assume, we have 10^6 numbers to sort!!!

Now, we have the following algorithms

- Insertion Sort $\rightarrow$ $T(N) = c_1 N^2$
- Merge Sort $\rightarrow$ $T(N) = c_2 N \log_2 N$

Under the following constraints for a PC and Supercomputer

- In a Supercomputer $c_1 = 2$ instructions per line
- In our humble PC $c_2 = 50$ instructions per line

Now

In addition

- The supercomputer can do 10^{10} instructions per second.
- The PC can do 10^{7} instructions per second.

Now

In addition

- The supercomputer can do 10^{10} instructions per second.
- The PC can do 10^7 instructions per second.

Final Result

- Time of Insertion Sort in the Supercomputer:

$$\frac{2(10^6)^2 \text{ ins}}{10^{10} \text{ ins/sec}} = 200 \text{ seconds} \tag{3}$$

- Time of Merge Sort in a humble PC:

$$\frac{2(10^6)\log(10^6) \text{ ins}}{10^7 \text{ ins/sec}} = 3.9 \text{ seconds} \tag{4}$$

Now

In addition

- The supercomputer can do 10^{10} instructions per second.
- The PC can do 10^{7} instructions per second.

Final Result

- Time of Insertion Sort in the Supercomputer:

$$\frac{2(10^6)^2 \text{ ins}}{10^{10} \text{ ins/sec}} = 200 \text{ seconds} \quad (3)$$

- Time of Merge Sort in a humble PC:

$$\frac{2(10^6)\log(10^6) \text{ ins}}{10^7 \text{ ins/sec}} = 3.9 \text{ seconds} \quad (4)$$

Now

In addition

- The supercomputer can do 10^{10} instructions per second.
- The PC can do 10^{7} instructions per second.

Final Result

- Time of Insertion Sort in the Supercomputer:

$$\frac{2(10^6)^2 \text{ ins}}{10^{10} \text{ ins/sec}} = 200 \text{ seconds} \tag{3}$$

- Time of Merge Sort in a humble PC:

$$\frac{2(10^6)\log(10^6) \text{ ins}}{10^7 \text{ ins/sec}} = 3.9 \text{ seconds} \tag{4}$$

Now

In addition

- The supercomputer can do 10^{10} instructions per second.
- The PC can do 10^{7} instructions per second.

Final Result

- Time of Insertion Sort in the Supercomputer:

$$\frac{2(10^6)^2 \text{ ins}}{10^{10} \text{ ins/sec}} = 200 \text{ seconds} \tag{3}$$

- Time of Merge Sort in a humble PC:

$$\frac{2(10^6)\log(10^6) \text{ ins}}{10^{7} \text{ ins/sec}} = 3.9 \text{ seconds} \tag{4}$$

Now

In addition

- The supercomputer can do 10^{10} instructions per second.
- The PC can do 10^{7} instructions per second.

Final Result

- Time of Insertion Sort in the Supercomputer:

$$\frac{2(10^6)^2 \text{ ins}}{10^{10} \text{ ins/sec}} = 200 \text{ seconds} \tag{3}$$

- Time of Merge Sort in a humble PC:

$$\frac{2(10^6)\log(10^6) \text{ ins}}{10^{7} \text{ ins/sec}} = 3.9 \text{ seconds} \tag{4}$$