

# Algorithms & Complexity - Introduction

Yanjun Ma

`yma@computing.dcu.ie`

CA313@Dublin City University (2009-2010)

2 November, 2009

# Outline

Notes on Complexity Hierarchy

Decision Problems

Time Complexity and Turing Machines

In-Class Test

# A hierarchy of common formal complexities: remarks

From a practical point of view

Polynomial algorithms are (usually) considered efficient. Everything slower than that is (usually) considered inefficient.

# A hierarchy of common formal complexities: remarks

From a practical point of view

Polynomial algorithms are (usually) considered efficient. Everything slower than that is (usually) considered inefficient.

However...

# A hierarchy of common formal complexities: remarks

From a practical point of view

Polynomial algorithms are (usually) considered efficient. Everything slower than that is (usually) considered inefficient.

However...

- Linear and quadratic algorithms  $\Rightarrow$  OK.

# A hierarchy of common formal complexities: remarks

## From a practical point of view

Polynomial algorithms are (usually) considered efficient. Everything slower than that is (usually) considered inefficient.

## However...

- Linear and quadratic algorithms  $\Rightarrow$  OK.
- Cubic algorithms  $\Rightarrow$  OK in some cases.

# A hierarchy of common formal complexities: remarks

## From a practical point of view

Polynomial algorithms are (usually) considered efficient. Everything slower than that is (usually) considered inefficient.

## However...

- Linear and quadratic algorithms  $\Rightarrow$  OK.
- Cubic algorithms  $\Rightarrow$  OK in some cases.
- Everything above  $\Rightarrow$  NOT OK.

# Notes on formal complexity

In practice:

# Notes on formal complexity

In practice:

- Complexity ignores constant factors. A problem whose complexity is  $10^{1000}n$  is polynomial, but is completely impractical in practice. A problem whose complexity is  $10^{-10000}2^n$  is exponential, but can be practical for not too big values of  $n$ .

# Notes on formal complexity

## In practice:

- Complexity ignores constant factors. A problem whose complexity is  $10^{1000}n$  is polynomial, but is completely impractical in practice. A problem whose complexity is  $10^{-10000}2^n$  is exponential, but can be practical for not too big values of  $n$ .
- Complexity ignores the size of the exponents in polynomials. A problem with time  $n^{1000}$  is polynomial, yet impractical ...

# Outline

Notes on Complexity Hierarchy

Decision Problems

Time Complexity and Turing Machines

In-Class Test

# Complexity of algorithms vs. complexity of problems

# Complexity of algorithms vs. complexity of problems

## Definition (Decision Problems)

A (binary) *decision problem* is a question in some formal system with a yes-or-no answer.

# Complexity of algorithms vs. complexity of problems

## Definition (Decision Problems)

A (binary) *decision problem* is a question in some formal system with a yes-or-no answer.

The problem itself is distinct from the methods used to solve it, called decision procedures or algorithms. A decision problem which can be solved by some algorithm is called decidable.

## Example

Given two (natural) numbers  $x$  and  $y$ , does  $x$  divide  $y$ ?

# Complexity of algorithms vs. complexity of problems

## Definition (Decision Problems)

A (binary) *decision problem* is a question in some formal system with a yes-or-no answer.

The problem itself is distinct from the methods used to solve it, called decision procedures or algorithms. A decision problem which can be solved by some algorithm is called decidable.

## Example

Given two (natural) numbers  $x$  and  $y$ , does  $x$  divide  $y$ ?

This is a yes-or-no question, and its answer depends on the values of  $x$  and  $y$ . An algorithm for this decision problem would tell how to determine whether  $x$  divides  $y$ , given  $x$  and  $y$ .

# Complexity of algorithms vs. complexity of problems

## Definition

Complexity of decision problems The complexity of a (decidable) decision problems is the one of the *most efficient algorithm* for that decision problem.

# Complexity of algorithms vs. complexity of problems

## Definition

Complexity of decision problems The complexity of a (decidable) decision problems is the one of the *most efficient algorithm* for that decision problem.

⇒ It is thus more difficult to compute the complexity of a problem ...

# Decision problems. Examples

## Example

- Is  $x$  prime?
- Is this list empty?
- Is this matrix symmetric?
- etc.

# Some definitions

## Definition (Algorithm)

An algorithm is a sequence of (elementary) instructions that makes possible to go from a well defined state to another well defined state.

# Some definitions

## Definition (Algorithm)

An algorithm is a sequence of (elementary) instructions that makes possible to go from a well defined state to another well defined state.

## Definition (Algorithm (formal))

An algorithm is a word (on some alphabet) which can be read and executed by a Turing Machine.

# Outline

Notes on Complexity Hierarchy

Decision Problems

Time Complexity and Turing Machines

In-Class Test

# Turing Machines

## Definition (Turing Machines)

A *Turing Machine*  $T$  reads an input word  $x$  and turns an initial (global) state into a (global) final state.

# Turing Machines

## Definition (Turing Machines)

A *Turing Machine*  $T$  reads an input word  $x$  and turns an initial (global) state into a (global) final state.

## Definition

A *deterministic computation* for machine  $T$  on input  $x$ , denoted  $T(x)$ , is the sequence of global states starting with the initial global state such that when reading each symbol in  $x$ , each global state yields the next one in the sequence.

# Turing Machines

## Definition (Turing Machines)

A *Turing Machine*  $T$  reads an input word  $x$  and turns an initial (global) state into a (global) final state.

## Definition

A *deterministic computation* for machine  $T$  on input  $x$ , denoted  $T(x)$ , is the sequence of global states starting with the initial global state such that when reading each symbol in  $x$ , each global state yields the next one in the sequence.

## Halting

Machine  $T$  is said to *halt* when  $T(x)$  is finite and the last state is a final global state  $F$ .

# Universal Turing Machines

We want a Turing Machine such that, for any Turing Machine  $T$  and any input data  $x$  we can get the output  $T(x)$ .

# Universal Turing Machines

We want a Turing Machine such that, for any Turing Machine  $T$  and any input data  $x$  we can get the output  $T(x)$ .

## Definition (Universal Turing Machines)

A Universal Turing Machine  $UT$  is such that for any Turing Machine  $T$  and any input data  $x$ ,  $UT(d(T), x) = T(x)$ , where  $d(T)$  is a (word) description of  $T$ .

# Universal Turing Machines

We want a Turing Machine such that, for any Turing Machine  $T$  and any input data  $x$  we can get the output  $T(x)$ .

## Definition (Universal Turing Machines)

A Universal Turing Machine  $UT$  is such that for any Turing Machine  $T$  and any input data  $x$ ,  $UT(d(T), x) = T(x)$ , where  $d(T)$  is a (word) description of  $T$ .

## Note

A computer is a Universal Turing Machine: it takes a description of a particular Turing Machine (this description is the program to execute) and some data input and simulates this Turing Machine.

# Nondeterministic Turing Machine

Deterministic Turing machines are characterised by the fact that a state  $q_i$  and an input symbol  $w_i$  can lead to only one state  $q_{i'}$ .

# Nondeterministic Turing Machine

Deterministic Turing machines are characterised by the fact that a state  $q_i$  and an input symbol  $w_i$  can lead to only one state  $q_{i'}$ .

This is not true for a nondeterministic Turing machine  $NT$ . We can explore all the paths at the same time!

# Nondeterministic Turing Machine

Deterministic Turing machines are characterised by the fact that a state  $q_i$  and an input symbol  $w_i$  can lead to only one state  $q_{i'}$ .

This is not true for a nondeterministic Turing machine  $NT$ . We can explore all the paths at the same time!

For a non-deterministic turing machine, to decide a language  $L$  means: for any input NOT in  $L$ , ALL computations must reject the input; for any input IN  $L$ , AT LEAST ONE computation must accept the input.

# Time complexity

## Definition (Time complexity)

The *(time) complexity* of a program (for a given input) is the number of elementary instructions that this program executes. This number is computed with respect to the size  $n$  of the input data.

## Definition (Time complexity)

$DTIME(T, x)$  denotes the number of steps of the (deterministic) computation  $T(x)$ . We will note  $DTIME(T) \in O(f(n))$  if  $DTIME(T, x) \in O(f(n))$  with  $n = |x|$  (length of  $x$ ).

# The class $P$

The class  $P$

$$P = \cup_{k \in \mathbb{N}} DTIME(n^k) \quad (1)$$

# The class $P$

## The class $P$

$$P = \cup_{k \in \mathbb{N}} DTIME(n^k) \quad (1)$$

$P$  is the (complexity) class of decision problems that can be solved using a deterministic Turing Machine and a polynomial amount of computation time ( $\Rightarrow$  polynomial time complexity).

# The class $P$

## The class $P$

$$P = \cup_{k \in \mathbb{N}} DTIME(n^k) \quad (1)$$

$P$  is the (complexity) class of decision problems that can be solved using a deterministic Turing Machine and a polynomial amount of computation time ( $\Rightarrow$  polynomial time complexity).

## Practically Feasible Algorithms

# Outline

Notes on Complexity Hierarchy

Decision Problems

Time Complexity and Turing Machines

In-Class Test

# In-class Test

## Definitions

- Time Complexity
- Space Complexity
- $O$
- Binary Decision Problem
- Class P

## Time Complexity

- Whether a function can be considered efficient
- Write the complexity (functions) using the  $O$  notation
- Given some pseudo code, compute the time complexity

## Time Complexity

- Whether a function can be considered efficient
- Write the complexity (functions) using the  $O$  notation
- Given some pseudo code, compute the time complexity

## Binary Search

- Construct a binary tree from a sorted array
- Analyse the worst-case complexity