جامعة طرابلس ـ كلية تقنية المعلومات

Design and Analysis Algorithms

تصميم وتحليل خوارزميات

ITGS301

المحاضرة الأولى: Lecture 1

Tripoli University · Faculty of Information Technology · Marwa Solla · m.solla@uot.edu.ly

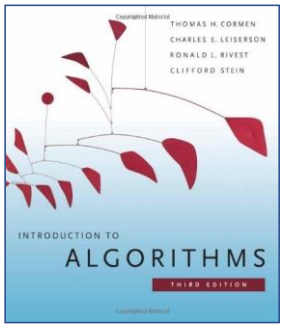
خريف 2022

Course Organization:

- Midterm Exam 40%
- ~~*Assignment (presentation) 10%*~~
- Final Exam 50%

References

Book: **Introduction to Algorithms third Edition by Cormen,**



Topics covered

Subjects
Introduction of algorithms
Algorithm specification
Performance analysis
Space & Time complexity
Asymptotic notations
Big O , Big Omega , & Big Theta
Complexity and Orders of Growth
Standard notations and common functions
Analysis of time Complexity
Sorting problem : Insertion sort
Time Complexity of Insertion Sort
RECURRENCE RELATIONS:
Factorial problem
Binary Search Trees.
SOLVING RECURRENCESteration method
The master method for solving recurrences

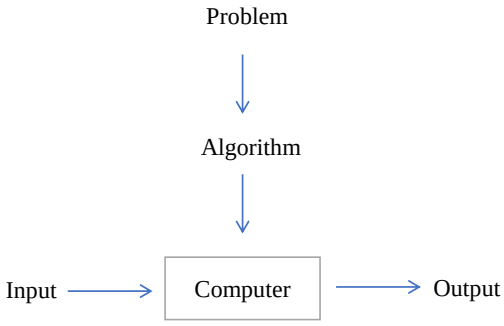
Subjects
The recursion-tree method for solving recurrences
Designing Algorithms / Divide-and-Conquer
Merge Sort
Quick Sort
Merge sort and Quick sort-Complexity
Dynamic Programming
Fibonacci Numbers problem
0/1 Knapsack Problem
Elementary Graph Algorithms
Representations of graphs
Breadth-first search
Greedy Algorithms
Minimum Spanning Tree:
Kruskal's Algorithm, & Prim's Algorithm
Single-Source Shortest Paths
Dijkstra's algorithm

Outcomes:

- Learn how to analyze algorithms and estimate their worst-case behavior
- Learn the concepts of designing algorithms
- The ability to classify algorithms and analyze their execution time.
- Studying some search algorithms.
- The ability to determine the efficiency of the algorithm, and compare it with others.

Notion of Algorithm

You have a problem ? You want to solve it
How you solve it on computer?
That is what is required.
What is a process?



What is an Algorithm?

- An algorithm is a sequence of unambiguous instructions for solving a problem.
- i.e., for obtaining a required output for any legitimate input in a finite amount of time.

- **Definiteness :** each instruction is clear and unambiguous.
- **Effectiveness :** each instruction must be very basic .
- **termination :** for a finite amount of time it never goes to infinite look
- **Correctness**

The algorithm must satisfy the following criteria:

- Each operation must be definite , meaning it must be perfectly clear.
- Each operation should be effective , meaning it can be done in a finite amount of time.
- An algorithm may have zero or more input.
- An algorithm produces one or more outputs.
- Terminates after a finite number of operations.

EXPRESSING ALGORITHMS "Algorithm specification"

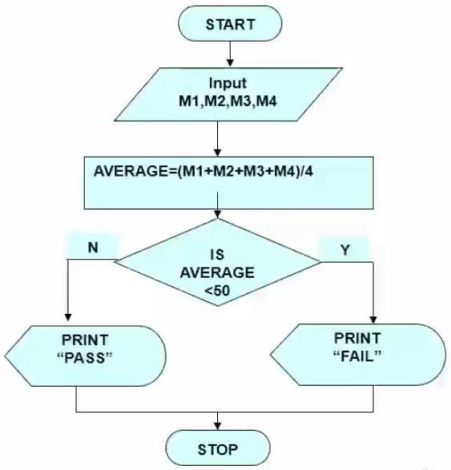
We express algorithms in whatever way the clearest:

* Natural Language: English

```
1. Input a set of 4 marks
2. Calculate their average by
   summing and dividing by 4
3. if average is below 50
4.       Print "FAIL"
5.  else
6.       Print "PASS"
```

EXPRESSING ALGORITHMS "Algorithm specification"

* Graphic representation : Flowchart



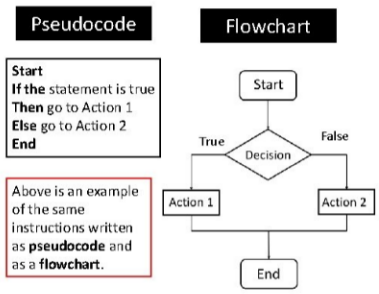
* Pseudo-Code

- It is a mixture of natural language and programming language.
- In this method , we describe algorithm as program

EXPRESSING ALGORITHMS "Algorithm specification"

- Pseudocode

we sometimes embed English statements into pseudocode. Therefore, unlike for real programming language, we can't create a complier that translates Pseudocode to machine code.



Pseudo-Code

- High-level description of an algorithm
- More structured than English prose
- Less detailed than a program
- Preferred notation for describing algorithms
- Hides program design issues

Pseudocode Details

- Control flow
  - **if** … **then** … [**else** …]
  - **while** … **do** …
  - **repeat** … **until** …
  - **for** … **do** …
  - Indentation replaces braces
- Method declaration
  - **Algorithm** *method* (*arg* [, *arg*…])
  - **Input** …
  - **Output** …
- Method call
  - *method* (*arg* [, *arg*…])
- Return value
  - **return** *expression*
- Expressions:
  - ← Assignment
  - = Equality testing
  - n^2 Superscripts and other mathematical formatting allowed

Pseudo-Code Conventions

1. Comments begin with // and continue until the end of line.
2. Blocks are indicated with matching braces { and }.
3. An identifier begins with a letter. The data types of variables are not explicitly declared.
4. Assignment of values to variables is done using the assignment statement.
   <Variable>:= <expression>;
5. There are two Boolean values TRUE and FALSE.
   - Logical Operators:AND, OR, NOT
   - Relational Operators:<, ≤,>,≥,=,≠,

Pseudo-Code Conventions

5. There is only one type of procedure: Algorithm, contains

-heading
-body

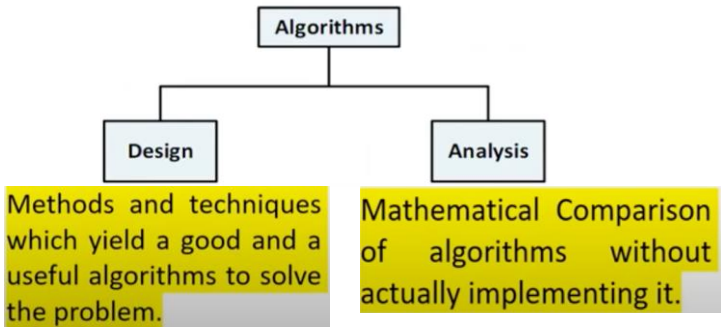
Algorithm Name (Parameters list) ⟶ heading

{
.....
..... Body
.....
}

Example :Algorithm

```
1. algorithm Max(A,n)
2. // A is an array of size n
3. {
4. Result := A[1];
5. for i:= 2 to n do
6. if A[i] > Result then
7. Result :=A[i];
8. return Result;
9. }
```

The study of algorithms includes many areas. In this course, we will focus on two areas which are how Design and Analysis of algorithms.



How to Analyze Algorithms

To execute an algorithms we use the computer's central processing unit (CPU) to perform operations. Also, we use the memory to hold the program and its data. Analysis of algorithms refers to the process of determining how much computing time and storage an algorithms requires.

How to Design Algorithms

we will study various design techniques such as Divide and conquer, and Greedy methods.

Analysis of Algorithms

Objective

Algorithm analysis aims to **measure the efficiency** of an algorithm using two metrics:

1. **Time efficiency (complexity)**: indicates how fast the algorithm solves the problem
2. **Space efficiency (complexity):** refers to the amount of memory units required by the algorithm without calculating the space needed for the input and output.

يهدف تحليل الخوارزمية إلى قياس كفاءة الخوارزمية باستخدام مقياسين:

(١) كفاءة الوقت: سرعة الخوارزمية في حل المشكلة

(٢) كفاءة المساحة: وحدات الذاكرة التي تتطلبها الخوارزمية دون حساب الذاكرة اللازمة للمدخلات والمخرجات

❖ Algorithm Efficiency

There is seldom algorithm for any any problem. when comparing two different algorithms that solve the same problem. we often find that one algorithm is more efficient than other using:

1. How much time it requires. (*Complexity of time*)
2. How much memory space it requires. (*space Complexity)*

Analysis of algorithms

Issues:

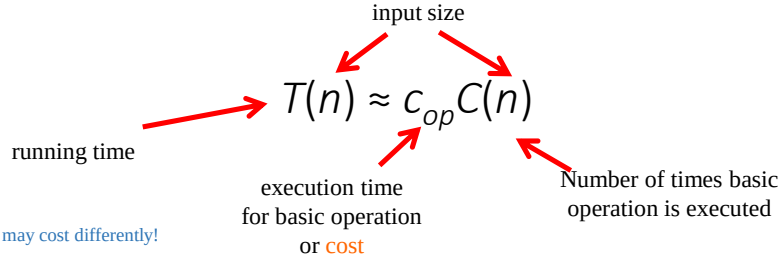
- Correctness
- time efficiency
- space efficiency
- optimality

Approaches:

- Theoretical analysis
- Empirical analysis

Theoretical analysis of Time efficiency

- Time efficiency is a function of input size.
- It is analyzed by determining the number of repetitions of the *basic operation* of the algorithm.
- *Basic operation*: the operation that contributes the most towards the running time of the algorithm



Note: Different basic operations may cost differently!

Measuring Running time

The approach: identify and count basic operations (steps) in the algorithm

Problem	Operation
Find x in an array	Comparison of x with an entry in the array
Multiplying two matrices with real entries	Multiplication of two real numbers
Sort an array of numbers	Comparison of two array entries plus moving elements in the array

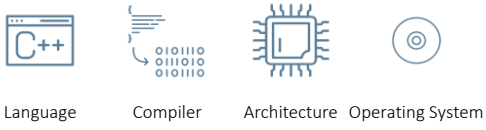
Problem	Size of input
Find x in an array	The number of the elements in the array
Multiply two matrices	The dimensions of the matrices
Sort an array	The number of elements in the array
Traverse a binary tree	The number of nodes
Solve a system of linear equations	The number of equations, or the number of the unknowns, or both

Empirical analysis of time efficiency

Many algorithms cannot be analyzed mathematically. The alternative way is the Empirical Analysis.

The steps are:

- Select a specific (typical) sample of inputs
- Decide on the efficiency metric (execution time or operation count)
- Implement the algorithm
- Run the code and record the observed data (time or number of operations)
- Analyze the data



Comparing Algorithms

Given 2 or more algorithms to solve the same problem, how do we select the best one?

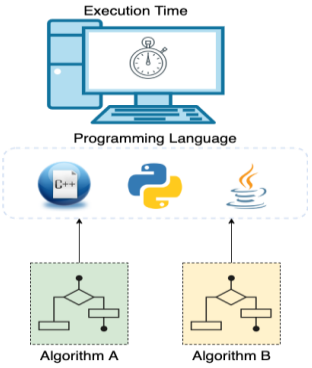
Some criteria for selecting an algorithm

1) Is it easy to implement, understand, modify?
2) How long does it take to run it to completion?
3) How much of computer memory does it use?

Comparing Algorithms

Compare two Algorithms A and B using Empirical analysis

- ✓ Implement both algorithms using a **programming language**
- ✓ Run both algorithms on the **same input**
- ✓ Report the **execution time** of each algorithm



Comparing Algorithms

Compare two Algorithms A and B using Theoretical analysis

- Uses a pseudo-code description of the algorithm instead of an implementation
- Characterizes running time as a function of the input size, ***n***
- Takes into account all possible inputs

Comparing Algorithms example

Problem: given a group of n numbers, determine the k^{th} largest

Algorithm 1

- Store numbers in an array
- Sort the array in descending order
- Return the number in position k.

Algorithm 2

- Store first k numbers in an array
- Sort the array in descending order
- For each remaining number, if the number is larger than the k^{th} number, insert the number in the correct position of the array
- Return the number in position k

Which Algorithm Is Better?

The algorithms are correct, but which is the best?

- Measure the running time (number of operations needed).
- Measure the amount of memory used.
- Note that the running time of the algorithms increase as the size of the input increases.

Analyzing Algorithms

- Predict the amount of resources required:
  Computational time: how fast the algorithm runs?
  Memory: how much space is needed?
- Fact: running time grows with the size of the input.
- Input size (number of elements in the input) Size of an array, # of elements in a matrix, vertices and edges in a graph

Analyzing Algorithms

Input Size

In analysis of algorithms, inputs size refers to the size of the problem instance to be solved.

Problem Type	Input Size
Sorting & Searching	Number of entries in the array
Graph	Number of vertices or edges or both
Computational Geometry	Number of points, vertices, edges, line segments, polygons, etc.
Matrix Operations	Dimensions of the matrices
Number Theory & Cryptography	Number of bits

يختلف قياس حجم المدخلات حسب نوع المشكلة المراد حلها.

In this course, we want to measure only the computation time (*Running Time*) of an algorithm.

Def: Running time = the number of Basic operations (steps) executed before termination

Basic Operation

An primitive operation is called a basic operation if it is of highest frequency among all other primitive operations.

Operation	Example
Comparison	>, <, >=, <=, ==, !=
Arithmetic	+, -, *, /, **, %, ++, --
Assignment	x = 3

العملية الرئيسية:

هي عملية أولية ذات تكرار أعلى من جميع العمليات الأولية الأخرى.

Asymptotic Notations

Asymptotic Notations

Asymptotic notations are mathematical tools to represent the time and space complexity of algorithms.

صيغ التحليل المقارب:

هي صيغ رياضية لتمثيل كفاءة الخوارزميات من منظور الوقت والمساحة (الذاكرة).

Asymptotic Notations

There are several kinds of mathematical notations which are very useful for this kind of analysis. we will present three asymptotic notations that are:

1) O-notation .
2) Ω- notation .
3) Θ- notation.

These notations are used to express the complexity of a given algorithm, or used to Compare two algorithms or more that are designed to solve the same problem

Asymptotic Notations

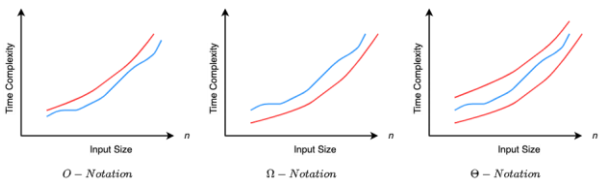
Asymptotic Notations

The following 3 asymptotic notations are used to represent the time/space complexity of algorithms:

(1) Big Oh Notation (O) : Upper bound

(2) Big Theta Notation (Θ) :Exact bound

(3) Big Omega Notation (Ω) : Lower bound

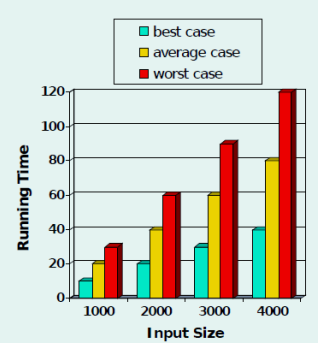


Analysis Cases

1. Worst case:
- T(N) = maximum time of algorithm on any input of size N.
- Provides an upper bound on running time.

2. Best case:
- T(N) = minimum time of algorithm on any input of size N.
- Provides a lower bound on running time.

3. Average case:
- T(N) = expected time of algorithm over all input of size N.
- Provides a prediction about the running time



The End .