Advanced Analysis of Algorithm

Department of Computer Science
Swat College of Science and Technology

CS Course : Advanced Analysis of Algorithm
Course Instructor : Muzammil Khan

Chapter 2

Introduction

History

- Procedures for solving geometric and arithmetic problems were formulated by ancient Greeks
- Some two thousands years ago, the procedure for finding greatest common divisor (GCD) was discovered by *Euclid*
- The word *algorithm* comes from the name of the 9th century Persian mathematician *Abu Abdullah Muhammad ibn Musa al-Khwarzimi*
  - His works introduced algebraic concepts
  - He worked in Baghdad at the time when it was the centre of scientific studies and trade
  - Al-Khwarzimi's name was translated into Latin, and eventually became *algorithm*

Algorithms Today

- The word originally referred only
  - To the rules of performing arithmetic
- The word evolved
  - To include all definite procedures for solving problems or performing tasks
- In the mid-twentieth century
  - *D.E. Knuth* undertook in depth study and analysis of algorithms
  - His work is embodied in his comprehensive book "The art of computer programming"
    - Which serves as a foundation for modern study of algorithms

Definition

- An algorithm is an orderly step-by-step procedure, which has the characteristics:
  1. It accepts one or more input value
  2. It returns at least one output value
  3. It terminates after finite steps
- An algorithm may also be viewed as a tool for solving a computational problem
  - Determine whether the number *x* is in the list *S* of *n* numbers. The answer is *Yes* if *x* is in *S* and *No* if it is not'
    - S = [5, 7, 11, 4, 9] n = 5 x = 9 is an instance of the problem
  - Solution to this instance is 'Yes'

Applications

- Algorithms have been developed to solve an enormous variety of problems in many application domains
- Some broad categories of algorithms are listed below
  - Sorting Algorithms
  - Searching Algorithms
  - String Processing
    - Pattern matching, Compression, Cryptography
  - Image Processing
    - Compression, Matching, Conversion
  - Mathematical Algorithms
    - Random number generator, matrix operations

Applications (Cont...)

- Some applications do not explicitly require algorithmic content at the application level
  - Even so, it may rely heavily upon algorithms
- Does the application require fast hardware?
  - Hardware design uses algorithms
- Does the application rely on networking?
  - Routing in networks relies heavily on algorithms
- Does it use a language other than machine code?
  - Compilers, Interpreters make extensive use of algorithms

Analysis of Algorithms

- Analyzing an algorithm has come to mean predicting the resources that it requires
- The purpose of algorithm analysis is to determine
  - Time efficiency (Complexity)
    - Performance in terms of running times for different input sizes
  - Space utilization
    - Requirement of storage to run the algorithm
  - Correctness (accuracy) of algorithm
    - Results are trustworthy, and algorithm is robust

Algorithm Efficiency

- Time efficiency remains an important consideration when developing algorithms
- Algorithms designed to solve the same problem may differ dramatically in efficiency
- These differences can be much more significant than differences due to hardware and software
- Example
  - Sequential search vs. Binary search
  - Next slide …

Algorithm Efficiency (Cont...)

- The number of comparisons done by
  - Sequential search and Binary search
  - When *x* (value being searched) is larger than all array items

Array Size	Number of comparisons - Sequential search	Number of comparisons - Binary search
128	128	8
1,024	1,024	11
1,048,576	1,048,576	21
4,294,967,296	4,294,967,296	33

Algorithm Efficiency (Cont...)

- Time taken by comparisons
  - In terms of algorithm execution time

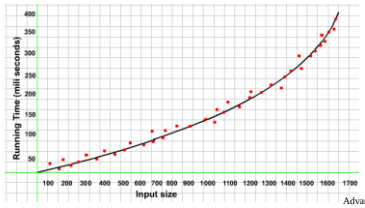
Execution time of **Algorithm 1**	Execution time of **Algorithm 2**
41 ns	1048 µs
61 ns	1s
81 ns	18 min
101 ns	13 days
121 ns	36 years
161 ns	$3.8 * 10^7$ years
201 ns	$4 * 10^{13}$ years

Analysis Approaches

- Basically three approaches can be adopted to analyze algorithm *running time* in terms of *input size*
  - **Empirical Approach**
    - Running time measured experimentally
  - **Analytical Approach**
    - Running time estimated using mathematical modeling
  - **Visualization**
    - Performance is studied through animation for different data sets

Empirical Approach

- The running time of algorithm is measured for different data sizes and time estimates are plotted against the input
- The graph shows the trend



Empirical Approach (Cont...)

- *Limitations* of Running time
- Running time critically depends on
  - Hardware resources used
    - CPU speed, IO throughput, RAM size
  - Software environment
    - Compiler, Programming Language
  - Program design approach
    - Structured programming, Object Oriented programming

Analytical Approach

- We need a measure that is
  - Independent of the computer, programming language, and complex details of the algorithm
- Usually this measure is in terms of
  - How many times a *basic operation* is carried out for each value of the *input size*
- Strategy
  - Running time is estimated by analyzing the *primitive operations* which make *significant contributions* to the overall algorithm time
  - These may broadly include
    - Comparing data items
    - Computing a value
    - Moving a data item
    - Calling a procedure

Analytical Approach (Cont...)

- Algorithm Specifications
- Algorithm is in
  - Plain text or natural language
    - High level description
  - Pseudo Code
    - Low level to facilitate analysis and implementation

Example

- Specification Using Natural Language
- Preorder Tree Traversal Algorithm

Step 1.	Push tree root to stack
Step 2.	Pop the stack. If stack is empty exit, else process the node
Step 3.	Travel down the tree following the left most path, and pushing each right child onto the stack
Step 4.	When leaf node is reached, go back to step 2.

Pseudo Code Convention

Declaration: Algorithm procedure name with parameters e.g MERGE(A,p,q)
Assignments : Using left arrows (←) e.g j← k ← p
Comparisons : Using symbols ≤ ≥ ≠ = > <
Logical : Using connectives **and** , **or**
Computations: Using arithmetic symbols
Exchanges: Using symbol ↔ e.g A[i] ↔ A[k] (swap)
Loops:

```
for ---- do ----,      for-----downto ---do---
while ---- do---
do -----  until---
```

Conditions: **if ---- then-----else-----**
Block structure: Using indentation

```
do---
    do----
        if--- else ----
```

Comments: Using symbol ▶

Example

```
1   for j ← 2 to n
2     do key ← A[j]
3       ▶Insert A[j] into sorted sequence A[1..j-1]
4         i ← j ← 1
5         while i > 0 and A[i] > key
6               do A[i+i] ← A[i]
7                  i ← i-1
8         A[i+1] ← key
```


Algorithm Design

- There are many approaches to design an algorithms
  - Divide-and-Conquer
  - Greedy
  - Dynamic Programming
  - Brute Force
  - Approximation
- Each has certain
  - Advantages &
  - Limitations

Course Outline

- Aims
  - To enable students to analyze the complexity of algorithms, and thus equip them with the skills to compare various algorithms for a problem and evaluate which one to use under given conditions
  - To familiarize them with algorithms for well known problems through a detailed discussion on these algorithms
  - To enable students to appreciate the role of algorithms in different application areas e.g. data mining, high performance computing
  - To introduce students to current research in selected application areas

Course Outline (Cont...)

Session	Lecture	Readings
1,2	**Introduction:** Introduction to algorithmic analysis, Mathematical preliminaries, asymptotic notation and analysis	Chapter 1-3
3-5	**Review of Sorting Algorithms:** Quick Sort, Merge Sort, Insertion Sort, Heap sort, Sorting in linear time	Chapter 6-8
6,7	**Review of Searching and Tree Structure Algorithms:** Linear and Binary search, Hashing, Red-Black trees	Chapter 11,12
8-10	**Graph Algorithms:** Graph representation, Bread-First and Depth-First search, Minimum Spanning Tree, Shortest Path	Chapter 22-25
11	Sessional 1	

Course Outline (Cont...)

12-14	**Dynamic Programming:** Assembly line scheduling, Matrix chain multiplication, Longest common subsequence, 0/1 Knapsack problem	Chapter 15
15	**String Matching Algorithms:** Naive string matching algorithm, String matching with finite automata	Chapter 32
16	**Greedy Algorithms:** Greedy Approach, Huffman codes, Activity selection	Chapter 16
17-19	**Advanced topics:** Introduction to NP-completeness, proofs and problems	Chapter 34
20	Sessional 2	
21-32	**Algorithms in various application areas**	Selected text and papers

End of Chapter

- You may have quiz next week