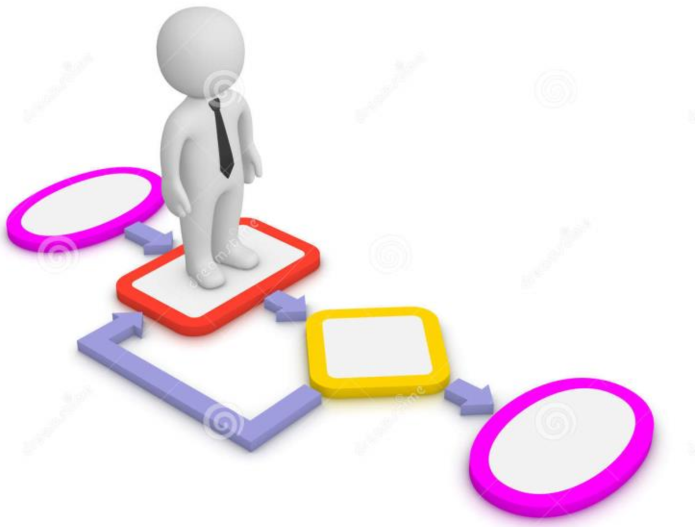
INTRODUCTION



Design & Analysis of Algorithms

Saman Riaz (PhD)
Associate Professor
RSCI, Lahore

Lecture # 01

Instructor

Saman Riaz (PhD)

- **Associate Professor: RIPHAH, RSCI, LAHORE**
  - **PhD** degree in *Computer Science and technology*
- **Email**: saman.riaz@riphah.edu.pk
- **Office hours**:
  - Mention on office door. Please send me an email for an appointment.

Course Objectives

Major objective of this course is:

- Design and analysis of modern algorithms
- Different variants
- Accuracy
- Efficiency
- Comparing efficiencies
- Motivation thinking new algorithms
- Advanced designing techniques
- Real world problems will be taken as examples
- To create feelings about usefulness of this course

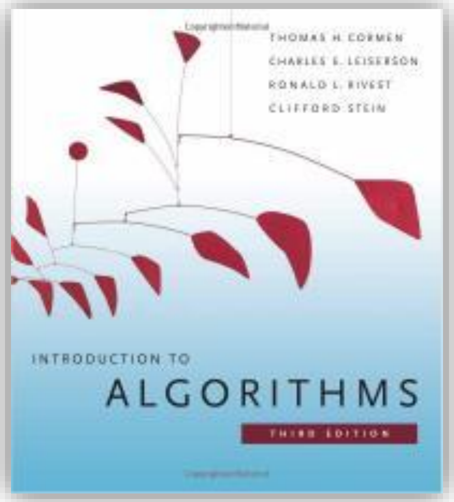
Learning Outcomes

On completion of this course, the student will have:

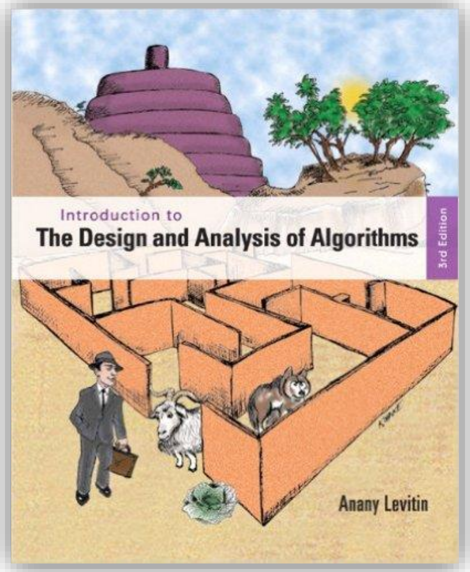
- A basic understanding of algorithm design and problem solving using fundamental techniques.
- Argue and prove correctness of algorithms
- Derive and solve mathematical models of problems
- The student will be able to demonstrate the associations between problem solving, algorithm design, and complexity analysis.
- Applied algorithm design, analysis, and implementation in various applications.
- Developed creativity and strategy skills for problem solving.

Recommended Text/ Reference Books

- *Introduction to Algorithms*, by Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein



- *Introduction to the Design and Analysis of Algorithms (3rd Edition)* by Anany Levitin.



Grading (Tentative)

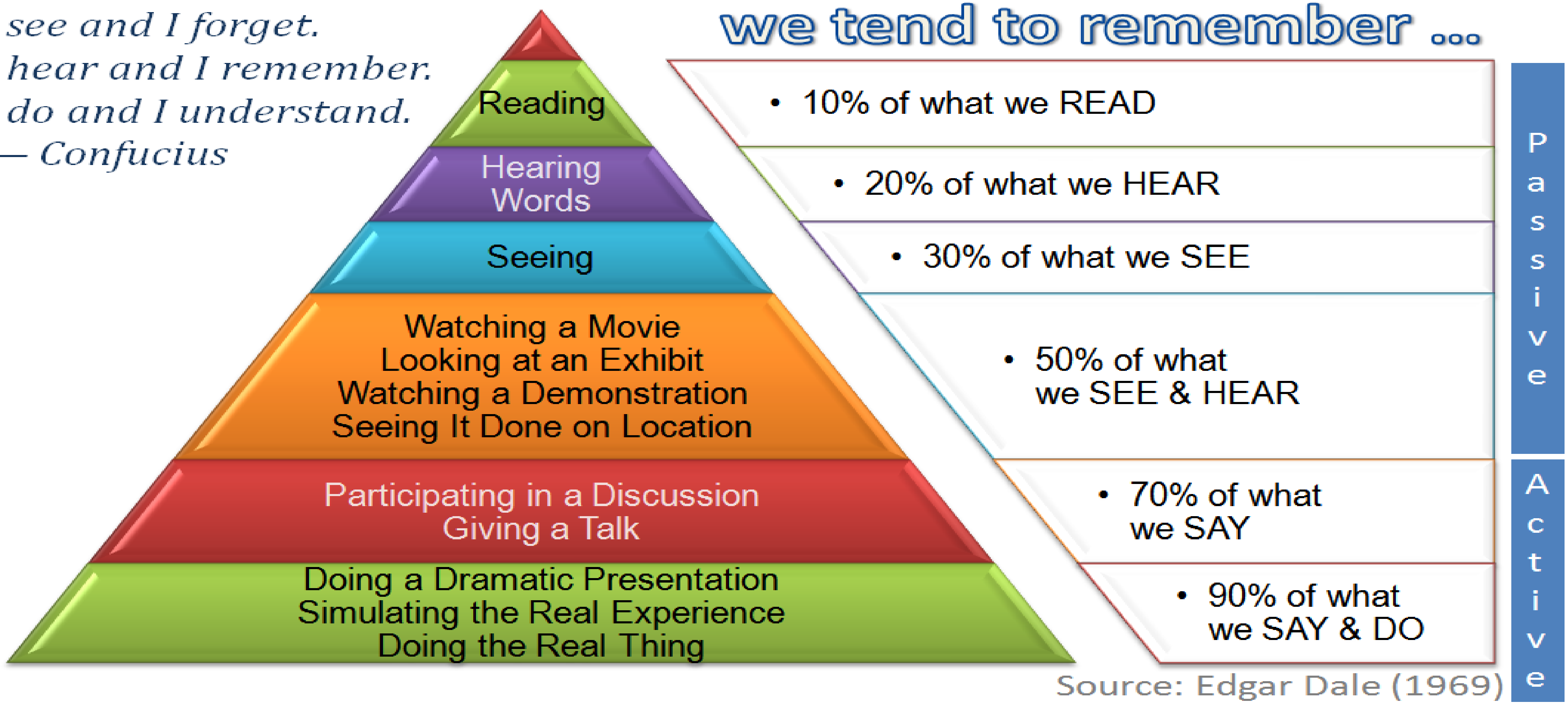
• Quizzes	10%
• Assignments	05%
• PBL	10%
• Mid-Term Exam	25%
• Final Exam	50%

The Cone of Learning

sparkinsight.com

I see and I forget.
I hear and I remember.
I do and I understand.
— Confucius

After 2 weeks, we tend to remember …



Source: Edgar Dale (1969)

Pre-Requisites

- Data Structures
- Programming

Course Contents

- **Introduction:** Introduction to algorithmic analysis, Mathematical preliminaries, asymptotic notation and analysis
- **Review of Sorting Algorithms:** Quick Sort, Merge Sort, Insertion Sort, Heap sort, Sorting in linear time
- **Review of Searching and Tree Structure Algorithms:** Linear and Binary search, Hashing, Red-Black trees
- **Graph Algorithms:** Graph representation, Bread-First and Depth-First search, Minimum Spanning Tree, Shortest Path
- **Dynamic Programming:** Assembly line scheduling, Matrix chain multiplication, Longest common subsequence, 0/1 Knapsack problem
- **String Matching Algorithms:** Naive string matching algorithm, String matching with finite automata
- **Greedy Algorithms:** Greedy Approach, Huffman codes, Activity selection
- **Number theory:** Greatest common divisor, Modular arithmetic, integer factorization
- **Advanced topics:** NP-completeness

ALGORITHM

- *A sequence of unambiguous instructions for solving a problem, i.e. for obtaining the required output for any legitimate input in a finite amount of time*

Fundamental data structures

Linear data structures

- Array
- Linked list
- Stack
- Queue

Operations: search, delete, insert

Implementation: static, dynamic

Fundamental data structures

Non-linear data structures

- Graphs
- Trees :
  - Rooted trees
  - Ordered trees
  - Binary trees

Graph representation: adjacency lists, adjacency matrix

Tree representation: as graphs; binary nodes

Fundamental data structures

Sets, Bags, Dictionaries

- **Set:** unordered collection of distinct elements

  Operations: membership, union, intersection

  Representation: bit string; linear structure

- **Bag**: unordered collection, elements may be repeated.

- **Dictionary**: a bag with operations search, add, delete.

Conclusion

- Algorithm classification
  - By types of problems
  - By design technique
- Design techniques
  - a general approach to solving problems

Definition

- **An algorithm is an orderly step-by-step procedure, which has the characteristics:**

Step 1: Obtain a description of the problem.

Step 2: Analyze the problem.

Step 3: Develop a high-level algorithm. Or Find an algorithm to solve it.

Step 4: Refine the algorithm by adding more detail.

Step 5: Review the algorithm.

Example:

’ ‘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’

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
    - Performance in terms of running times for different input sizes
  - Space utilization
    - Requirement of storage to run the algorithm
  - Correctness 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

Algorithm Efficiency (contd…)

- 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 (contd…)

- Another comparison in terms of algorithm execution time

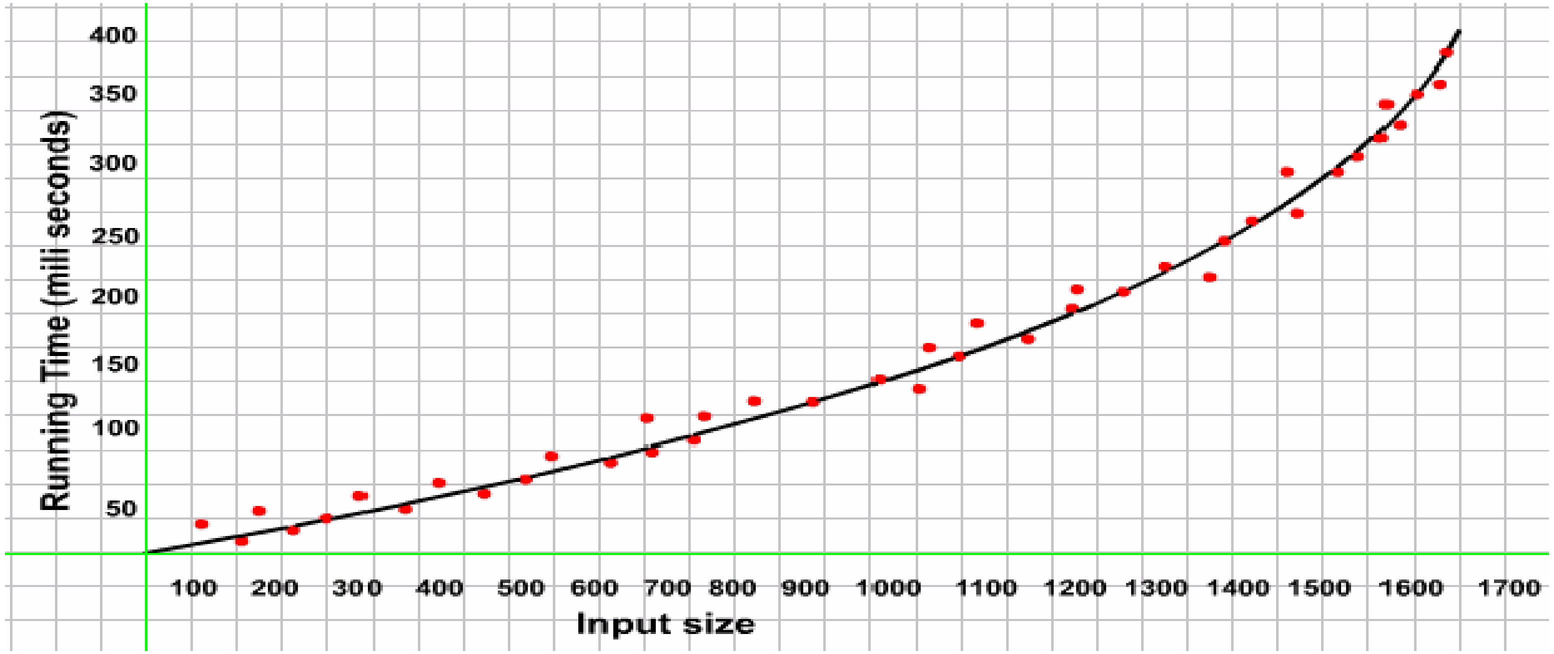
Execution time of Algorithm 1	Execution time of Algorithm 2
41 ns	1048 µs
61 ns	1s
81 ns	18 min
101 ns	13 days
121 ns	36 years
161 ns	$3.8 * 10^{7}$ years
201 ns	$4 * 10^{13}$ years

Approaches to Analysis

- Basically two 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

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.



Limitations

- Running time critically depends on:
  - Hardware resources used
    - (CPU speed, IO throughput, RAM size)
  - Software environment
    - (Compiler, Programming Language)
  - Program design approach
    - (Structured, Object Oriented)

Analytical Approach

- We want 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

Algorithm Specification

- Plain natural language
  - High level description

- Graphical representation called flowchart.

- Pseudo Code
  - Low level to facilitate analysis and implementation

Specification Using Natural Language

- Finding Max number in array
  - Step #1: Store first element of the array in variable ***max***
  - Step #2: Scan array to compare ***max*** with other elements
  - Step #3: Replace ***max*** with a larger element
  - Step #4: Return value held by ***max***

Pseudo code

- A mixture of natural language and high-level programming concepts that describes the main ideas behind a generic implementation of a data structure or algorithm
- It is more structured than usual prose but less formal than a programming language
- Expressions:

  Use standard mathematical symbols to describe numeric and Boolean expressions
- Method Declarations:

  Algorithm name (Find-Max)

Pseudocode (cont….)

- Programming Constructs:
  - decision structures: if ... then ... [else
  - while-loops. while ... do
  - repeat-loops: repeat ... until ...
  - for loop: for ... do
  - array indexing A[i], A[i.j]
- Methods:
  - calls: object method(args)
  - returns: return value

Primitive Operation

- Primitive Operation: Low-level operation Independent of programming language. Can be identified in pseudo-code for example:
  - ➢ Data movement (assign)
  - ➢ Control (branch, subroutine call, return)
  - ➢ Arithmetic an logical operations (e.g. addition, comparison)

- By inspecting the pseudo-code, we can count the number; primitive operations executed by an algorithm.

Pseudocode

- The function **FIND-MAX** finds the maximum element in an array. The array **A**, of size ***n,*** is passed as argument to the function.

```
FIND-MAX ( A[1..n] )

max ←A[1]                         ► Store first array element into variable max
for j←2 to n do                   ► Scan remaining elements
      if ( A[j] > max )           ► Compare an element  with max
            then  max ← A[j]      ► Replace max with a larger element
return max                        ► Return maximum element
```

Algorithm Design

- There are many approaches to designing algorithms:
  - Incremental
  - Divide-and-Conquer
  - Dynamic Programming
  - Approximation

- Each has certain advantages and limitations

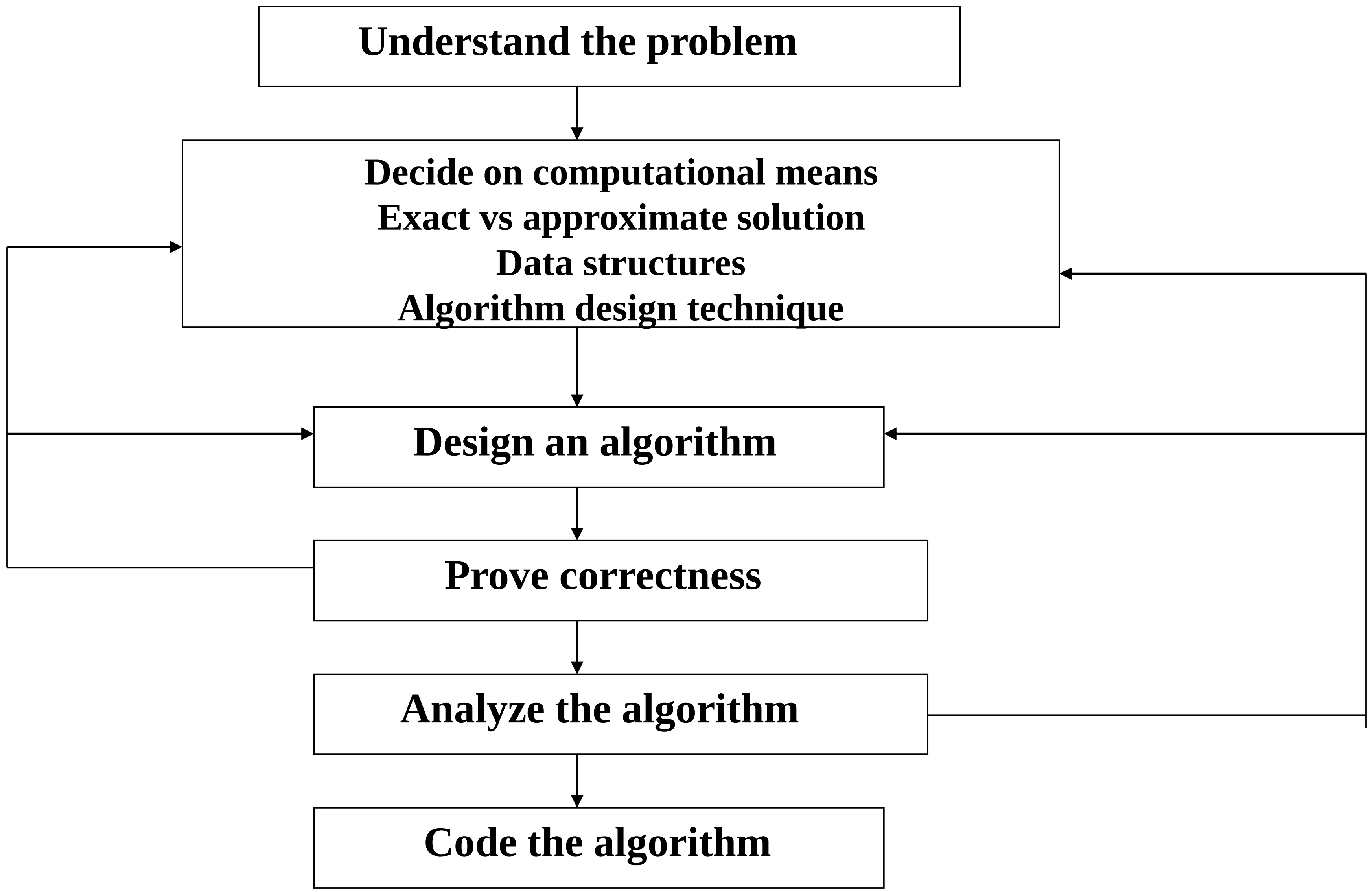
Examples to use algorithms

- An instructor needs to count number of students in a class quickly.
- A student needs to search allocated space for an entry test scheduled in a big hall where expected number of students are more than 10,000.
- A tourists wants to visit the entire attractive places in a city with minimum travel time and money.
- A mobile company wants to introduce a new calling package to maximize the number of users to switch to the new package.

" I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships. "

— Linus Torvalds (creator of Linux)

" Algorithms + Data Structures = Programs. " *— Niklaus Wirth*


Understand the problem
Decide on computational means
Exact vs approximate solution
Data structures
Algorithm design technique
Design an algorithm
Prove correctness
Analyze the algorithm
Code the algorithm

What does it mean to understand the problem?

- What are the problem objects?
- What are the operations applied to the objects?

Deciding on computational means

- How the objects would be represented?
- How the operations would be implemented?

Design an algorithm

- Build a computational model of the solving process

Prove correctness

- Correct output for every legitimate input in finite time
- Based on correct math formula
- By induction

Important problem types

- Sorting
- Searching
- String processing
- Graph problems
- Combinatorial problems
- Geometric problems
- Numerical problems

What's Analysis of Algorithms

- The theoretical study of algorithm's performance and resource usage.
- Other important features of an algorithm are
  - modularity
  - correctness
  - maintainability
  - functionality
  - robustness
  - user-friendliness
  - programmer time
  - Simplicity
  - extensibility
  - Reliability
- During this course our focus will be on **the performance and the storage requirements**.