DATA STRUCTURES

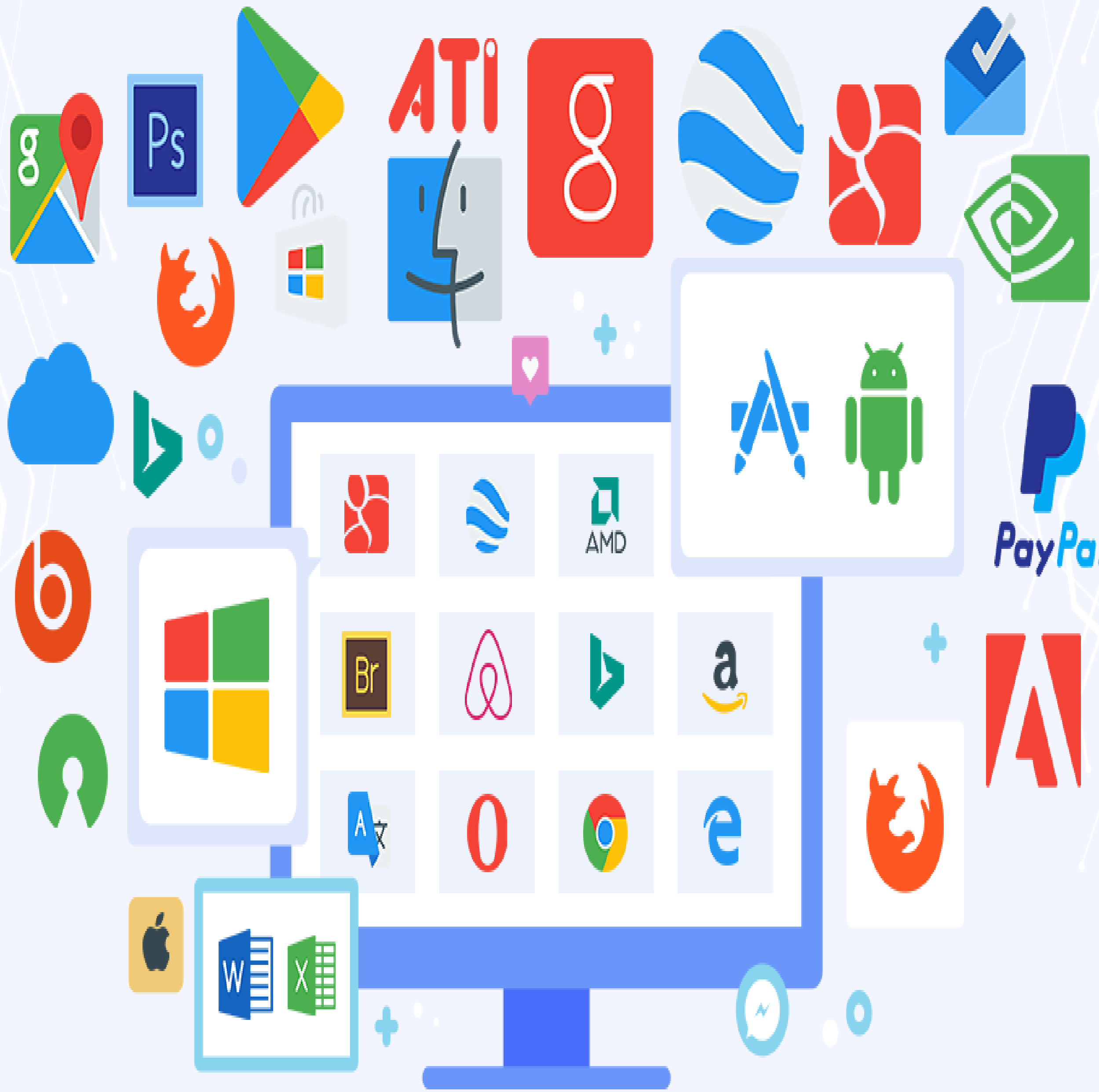
UNIT-1

BASIC CONCEPTS

Dr G.Kalyani

Topics

- **System Life Cycle**
- **Algorithm Specification**
- **Data Abstraction**
- **Abstract Data Type(ADT)**
- **Performance Analysis**


Ps
ATI
AMD
Br
PayPal

Introduction to System Life Cycle

- Systems are dynamic—they change over time.
- Projects are done for the purpose of developing systems—either to create new ones or to improve existing ones.
- The process used to develop software systems is called as System Life Cycle.
- Every system/project can be measured in three ways at any point in its life cycle:
  - time
  - cost
  - performance.

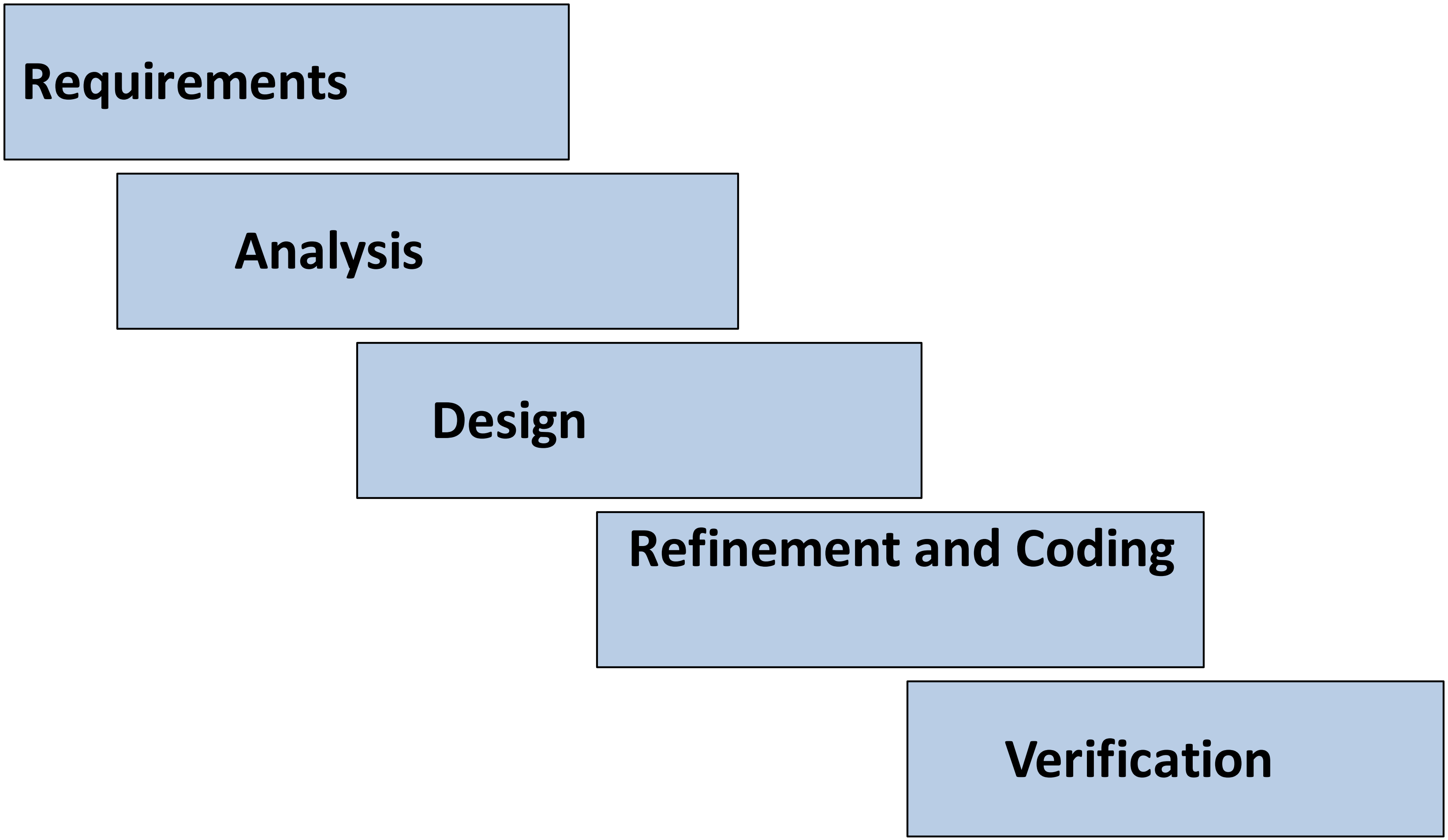
Introduction to System Life Cycle

- Time refers to the temporal progress of activities and extent to which schedules and deadlines are being met.

- Cost refers to the rate of resource expenditure as compared to budgeted resources.

- Performance refers to outputs of the project as compared to objectives, specifications, and requirements; meeting performance requirements is a measure of the quality of the project output.

Phases of System Life Cycle



SLC Phase-1: Requirements

- Begins with a set specifications that defines the purpose.

- Describes the information about what inputs are to be given and what is to be expected as result.

- Need to develop input and output descriptions for all the possible test cases.

SLC Phase-2: Analysis

- Divide the problem into sub problems.
- Two approaches for analysis:
  - Top-down Approach
  - Bottom-up Approach

SLC Phase-3: Design

- The Designers view the system in two perspectives:
  - the data objects that the program needs and
  - the operations performed on them.
- This perspective leads to the creation of Abstract Data Type(ADT).
- It requires specification of algorithm and algorithm design strategies.
- Ex: Design a system for university.
  - Data objects are students, courses and Faculty.
  - Operations are insertion, deletion and searching

SLC Phase-4: Refinement and Coding

- Choose the Representations of the data objects (Determining data structures) and Write algorithms for each operation on them.

- Efficiency of the algorithm depending on representation of the data objects.

- Refine the algorithms with cross-checking and develop the code.

SLC Phase-5: Verification

- **This phase consists of**
  - Developing the correctness proofs
  - Testing the program with a variety of input data
  - Removing the errors

- **Correctness Proofs:**
  - Proof with mathematical modelling
  - Time consuming and Difficult for larger systems
  - Select the techniques that have been proven correct.
  - Can be done before or during the coding

SLC Phase-5: Verification

- **Testing**
  - Requires a working code and sets of test data
  - Test data should be developed carefully so that it includes all the possible scenarios.
  - Ex: if your program contains 'switch' statement, test data should be chosen by covering all the cases.
  - Running time of the program is also important.
  - Initial tests focus on verifying that a program runs correctly, then reduce the running time

- **Error removal**
  - If any error occurs in testing correct the program to remove that error.
  - Debug the program

Topics

- **System Life Cycle**
- **Algorithm Specification**
- **Data Abstraction**
- **Abstract Data Type (ADT)**
- **Performance Analysis**

What is an Algorithm

An algorithm is a set of rules.

An algorithm is a step-by-step procedure

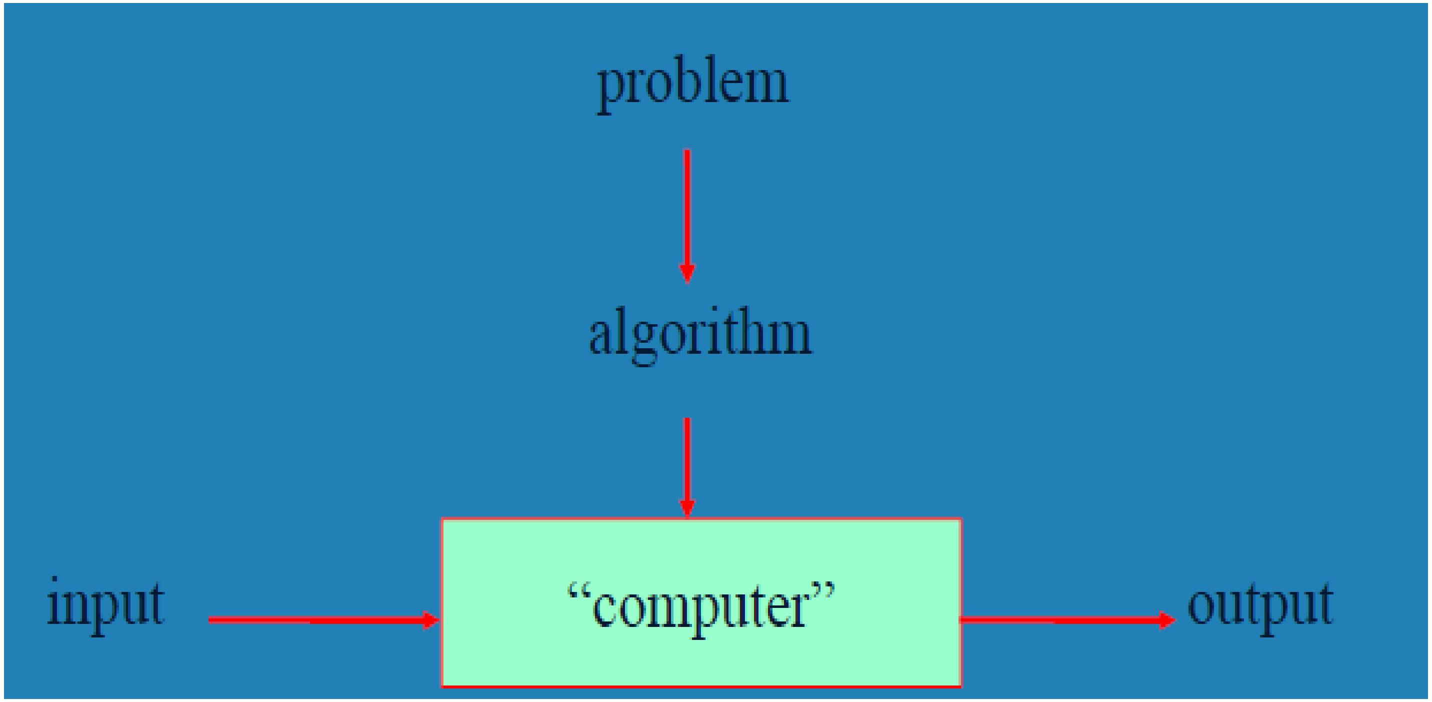
An algorithm is a sequence of computational steps

An algorithm is a sequence of operations performed data.

An algorithm is an abstraction of a program

What is an Algorithm

- Definition
  - An ***Algorithm*** is a finite set of instructions that, if followed, accomplishes a particular task.



Characteristics of an Algorithm

All algorithms must satisfy the following criteria:

(1)***Input***. There are zero or more quantities that are externally supplied.

(2)***Output***. At least one quantity is produced.

(3)***Definiteness***. Each instruction is clear and unambiguous.

(4)***Finiteness***. If we trace out the instructions of an algorithm, then for all cases, the algorithm terminates after a finite number of steps.

(5)***Effectiveness***. Every instruction must be basic enough to be carried out, in principle, by a person using only pencil and paper. It is not enough that each operation be definite and also must be feasible.

Describing Algorithms

- **Natural language**
  - English
    - Instructions must be definite and effectiveness
- **Graphic representation**
  - Flowchart
    - work well only if the algorithm is small and simple
- **Pseudo language**
  - Readable
  - Instructions must be definite and effectiveness

Topics

- **System Life Cycle**
- **Algorithm Specification**
- **Data Abstraction**
- **The Abstract Data Type**
- **Performance Analysis**

Data Abstraction

- Types of data
  - All programming language provide at least minimal set of predefined data types, plus user defined types

- Data types of C
  - Char, int, float, and double
    - may be modified by short, long, and unsigned
  - Array and Structure

Data Type

- Definition
  - A ***data type*** is a collection of ***objects*** and a set of ***operations*** that act on those objects
- Example of "int"
  - Objects: 0, +1, -1, ..., Int_Max, Int_Min
  - Operations: *arithmetic*(+, -, *, /, and %), *testing*(equality/inequality), *assigns*, *functions*
- Define operations
  - Its ***name***, possible ***arguments*** and ***results*** must be specified

- The design strategy for representation of objects is *Transparent* to the user

Topics

- **System Life Cycle**
- **Algorithm Specification**
- **Data Abstraction**
- **Abstract Data Type**
- **Performance Analysis**

Abstract Data Type

- **Definition**
  - An Abstract Data Type, or ADT, consists of
    - (a) a specification of the possible values of the data type
    - (b) a specification of the operations that can be performed on those values

- The abstract datatype is special kind of datatype, whose behavior is defined by a set of values and set of operations.
- The keyword "Abstract" indicates we can use these datatypes, and we can perform different operations. But how those operations are working that is totally hidden from the user.
- The ADT is made of with primitive datatypes, but operation logics are hidden.
- Why Abstract Data Type ?
  - implementation-independent

Classifying the Functions of an ADT

- **Creator/constructor:**
  - Create a new instance of the designated type
- **Transformers**
  - Modify the instance by using one or more operations
- **Observers/reporters**
  - Provide information about an instance of the type, but they do not change the instance

*Note:

- An ADT definition will include at least one function from each of these three categories

An Example of the ADT

```
ADT Nat_No
    objects: an ordered subrange of the integers starting at zero
             and ending at the maximum integer (INT_MAX) on
             the computer
    functions: for all x, y is Nat_No, TRUE, FALSE in
               Boolean and +, -, <, and == are the integer operations

        Boolean Is_Zero(x)        ::= if (x==0) return FALSE
        Nat_No Add(x, y)          ::= if ((x+y)<= INT_MAX) return x+ y
                                      else return INT_MAX
        Boolean Equal(x, y)       ::= if (x== y) return TRUE
                                      else return FALSE
        Nat_No Successor(x)       ::= if (x== INT_MAX) return x
                                      else return x+ 1
        Nat_No Subtract(x, y)     ::= if (x< y) return 0
                                      else return x-y
end Nat_No
```

Task on Example of the ADT

- **Add the following operations to the Nat_No ADT:**
  - Predecessor,
  - Multiply
  - Is_Greater
  - Is_Lesser
  - Division

- **Create an ADT Boolean with the operations:**
  - And
  - Or
  - Not
  - Xor

Topics

- System Life Cycle
- Algorithm Specification
- Data Abstraction
- Abstract Data Type
- Performance Analysis

Performance Evaluation

- What are criteria to judge a program
  - Does the program ***meet*** the given ***specifications*** of the task?
  - Does it ***work correctly***?
  - Does the program contain ***documentation*** that show ***how to use it*** and ***how it works***?
  - Does the program ***effectively use functions*** to create logical units?
  - Is the program's code ***readable***?
  - Does the program ***efficiently use the storage***?
  - Is the program ***running time is acceptable*** for the task?

Performance Evaluation(Cont.)

- Evaluate a program
  - **M**eet specifications, **W**ork correctly, **G**ood user-interface, **W**ell-documentation, **R**eadable, **E**ffectively use functions, **Running Time Acceptable, Efficiently use Space**
- How to achieve them?
  - Good programming style, experience, and practice

Performance Evaluation

- **Performance Evaluation**
  - Performance **Analysis**
  - Performance **Measurement**

- **Performance Analysis** - prior
  - an important branch of CS, ***complexity theory***
  - estimate ***time*** – ***Time Complexity***
  - estimate ***space*** – ***Space Complexity***
  - machine independent

- **Performance Measurement** -posterior
  - The actual ***time*** and ***space*** requirements
  - machine dependent

Space Complexity

- Definition
  - The ***space complexity*** of a program is the amount of memory that it needs to run to completion
- The space needed is the sum of
  - ***Fixed*** space and ***Variable*** space

- Fixed space
  - Includes the instructions, variables, and constants
  - Independent of the number and size of Input and Output
- Variable space
  - Depends on an instance '*I*' of the problem
  - Includes dynamic allocation, functions' recursion
- Total space of any program
  - $S(P)= c+ \boldsymbol{S_p(Instance)}$

Time Complexity

- **Definition**

  The ***time complexity, T(p),*** taken by a program P is the sum of the compile time and the run time

  T(P)= compile time + **run** (or **execution**) **time**

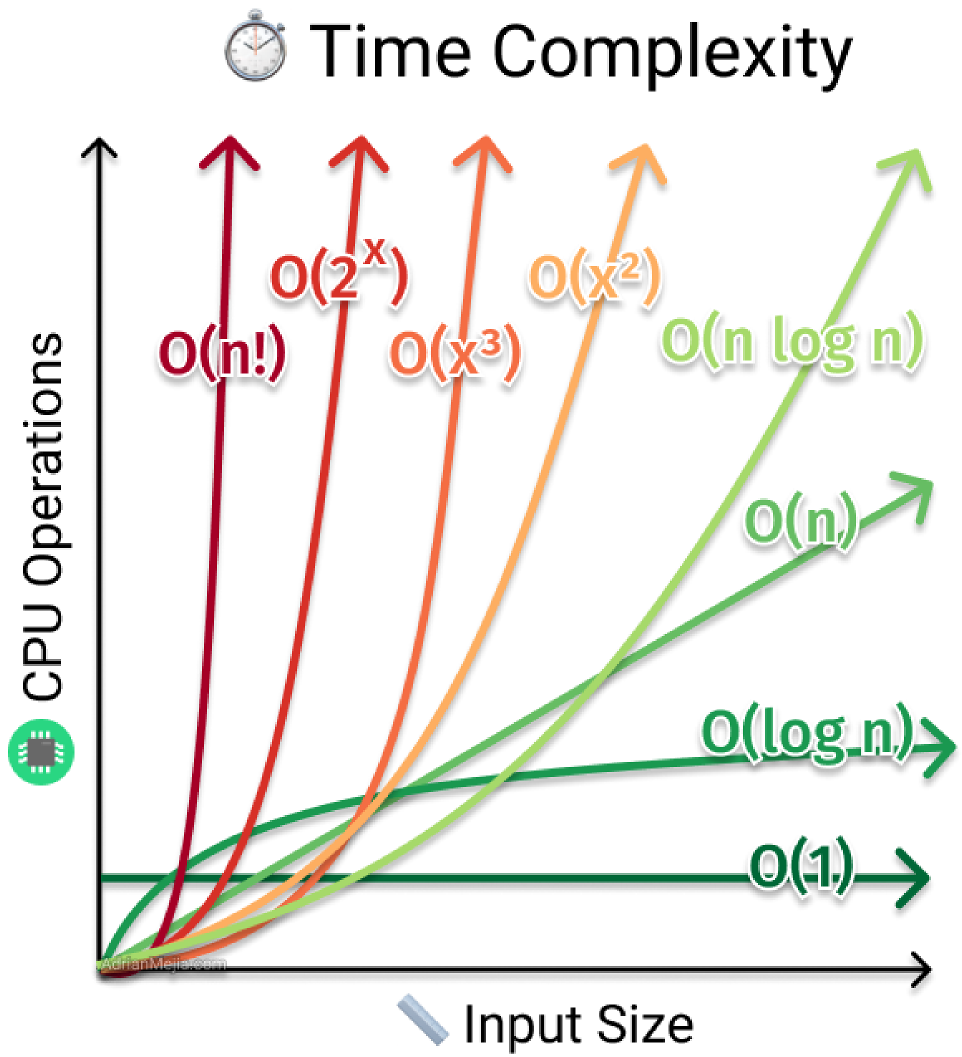
  $= \boldsymbol{c} + \boldsymbol{t}_p(n)$

  *Compile time does not depend on the instance characteristics

- **How to evaluate?**
  - Use the system clock (machine dependent)
  - Number of ***steps*** performed (machine-independent)

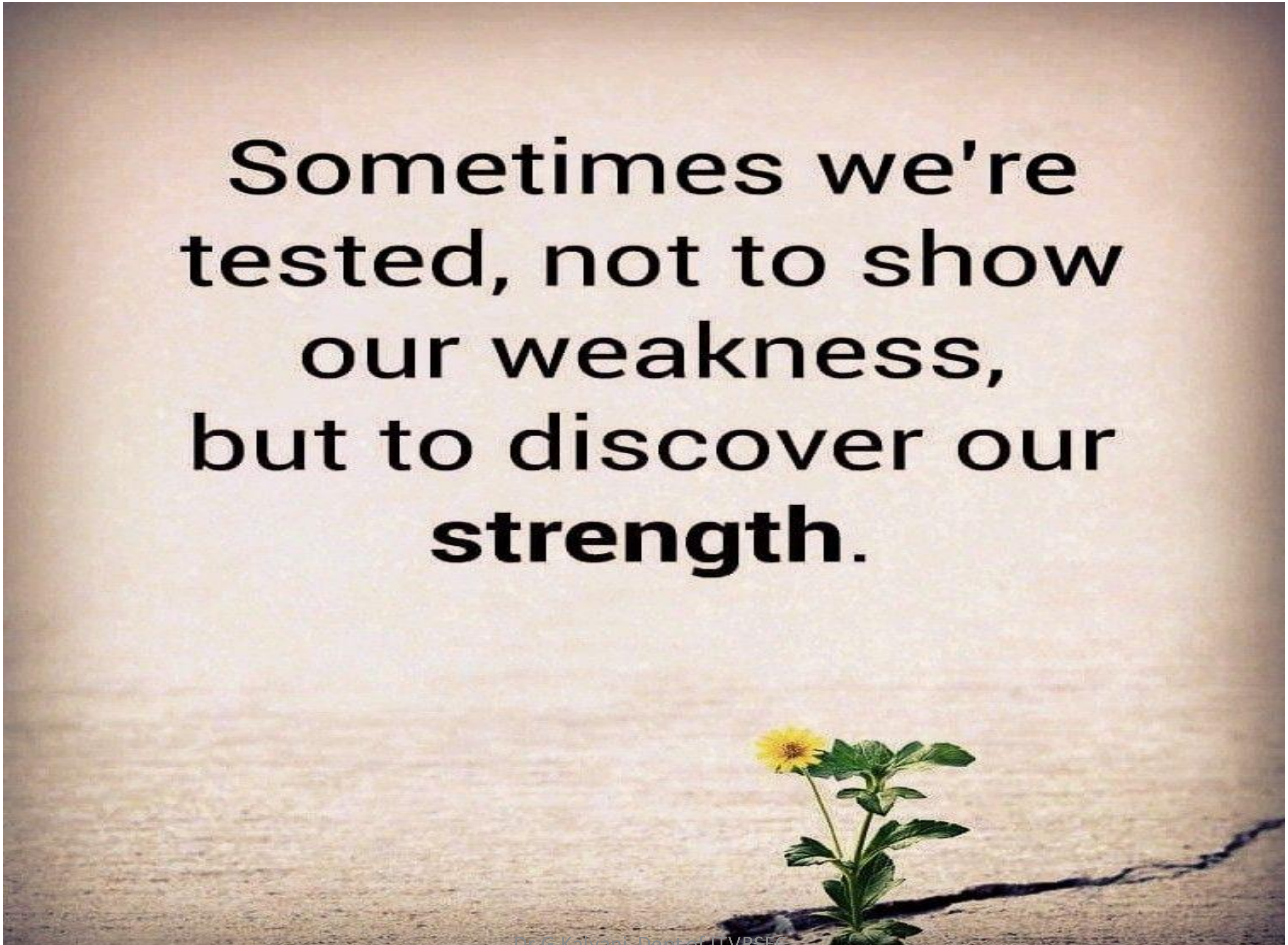
- **Definition of a program step**
  - A ***program step*** is a syntactically or semantically meaningful instruction whose execution time is independent of the instance characteristics.

Time Complexity



Topics

- **System Life Cycle**
- **Algorithm Specification**
- **Data Abstraction**
- **Abstract Data Type**
- **Performance Analysis**


Sometimes we're
tested, not to show
our weakness,
but to discover our
strength.