INTRODUCTION TO DATA STRUCTURE AND ALGORITHM

ROMI FADILLAH RAHMAT

DATA STRUCTURE

Data Structure is a systematic way to organize data in order to use it efficiently. Following terms are the foundation terms of a data structure.

- **Interface** – Each data structure has an interface. Interface represents the set of operations that a data structure supports. An interface only provides the list of supported operations, type of parameters they can accept and return type of these operations.
- **Implementation** – Implementation provides the internal representation of a data structure. Implementation also provides the definition of the algorithms used in the operations of the data structure.

DATA STRUCTURE CHARACTERISTIC

- **Correctness –** Data structure implementation should implement its interface correctly.
- **Time Complexity** – Running time or the execution time of operations of data structure must be as small as possible.
- **Space Complexity** – Memory usage of a data structure operation should be as little as possible.

WHY WE NEED DATA STRUCTURE?

- As applications are getting complex and data rich, there are three common problems that
- applications face now-a-days.

❑**Data Search** – Consider an inventory of 1 million(106) items of a store. If the application is to search an item, it has to search an item in 1 million(106) items every time slowing down the search. As data grows, search will become slower.

❑**Processor Speed** – Processor speed although being very high, falls limited if the data grows to billion records.

❑**Multiple Requests** – As thousands of users can search data simultaneously on a web server, even the fast server fails while searching the data.

- To solve the above-mentioned problems, data structures come to rescue. Data can be organized in a data structure in such a way that all items may not be required to be searched, and the required data can be searched almost instantly.

VARIABLES

- Take a look at this equation

$$x^2 + 2y - 2 = 1$$

- The equation has some names (x and y) which hold values (data) → it means x and y is the place holders for representing data.
- In order to hold or store data in computer science we use variable as a term

DATA TYPES

- Data type in programming language → is a set of data with values having predefined characteristics.
- Examples : Integer, floating point unit number, char, double, etc
- Computer memory is filled with 0 and 1, it is very hard to do the coding in 0 and 1, to solve this programming languages and compilers are providing the facility of data types.
- Example : integer → 2 bytes , float → 4 bytes. It means in memory we are combining 2 bytes(16bits) and calling it as integer.
- There are two type of data types :
  - System defined data types (Primitive data types)
  - User defined data types.

EXECUTION TIME CASES

- There are three cases which are usually used to compare various data structure's execution time in a relative manner.

❑**Worst Case** – This is the scenario where a particular data structure operation takes maximum time it can take. If an operation's worst case time is $f(n)$ then this operation will not take more than $f(n)$ time, where $f(n)$ represents function of n.

❑**Average Case** – This is the scenario depicting the average execution time of an operation of a data structure. If an operation takes $f(n)$ time in execution, then m operations will take $mf(n)$ time.

❑**Best Case** – This is the scenario depicting the least possible execution time of an operation of a data structure. If an operation takes $f(n)$ time in execution, then the actual operation may take time as the random number which would be maximum as $f(n)$.

BASIC TERMINOLOGY

- **Data** – Data are values or set of values.
- **Data Item** – Data item refers to single unit of values.
- **Group Items** – Data items that are divided into sub items are called as Group Items.
- **Elementary Items** – Data items that cannot be divided are called as Elementary Items.
- **Attribute and Entity** – An entity is that which contains certain attributes or properties, which may be assigned values.
- **Entity Set** – Entities of similar attributes form an entity set.
- **Field** – Field is a single elementary unit of information representing an attribute of an entity.
- **Record** – Record is a collection of field values of a given entity.
- **File** – File is a collection of records of the entities in a given entity set.

ALGORITHM

- Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.
- From the data structure point of view, following are some important categories of algorithms –

❑**Search** – Algorithm to search an item in a data structure.

❑**Sort** – Algorithm to sort items in a certain order.

❑**Insert** – Algorithm to insert item in a data structure.

❑**Update** – Algorithm to update an existing item in a data structure.

❑**Delete** – Algorithm to delete an existing item from a data structure.

CHARACTERISTICS OF AN ALGORITHM

Not all procedures can be called an algorithm. An algorithm should have the following characteristics

- **Unambiguous** – Algorithm should be clear and unambiguous. Each of its steps (or phases), and their inputs/outputs should be clear and must lead to only one meaning.
- **Input** – An algorithm should have 0 or more well-defined inputs.
- **Output** – An algorithm should have 1 or more well-defined outputs, and should match the desired output.
- **Finiteness** – Algorithms must terminate after a finite number of steps.
- **Feasibility** – Should be feasible with the available resources.
- **Independent** – An algorithm should have step-by-step directions, which should be independent of any programming code.

WHY WE NEED TO ANALYSIS THE ALGORITHMS?

- To solve one problem we can have various solutions. Similarly, in computer science there can be multiple algorithms exists to solve one problems (for example : sorting problem has many algorithm like insertion sort, selection sort, quick sort, and many more).
- The algorithm analysis helps us determining which of them is efficient in terms of time and space consumed.

GOAL OF ANALYSIS OF ALGORITHMS

- The goal of analyisis of algorithms is to compare algorithms (or solutions) mainly in terms of running time but also in terms of other factors (memory, developers effort, etc).

WHAT IS RUNNING TIME ANALYSIS

- It is the process of determining how processing time increases as the size of the problem increases. Input size is number of elements in the input and depending on the problem type the input may be different type
- In general, we encounter the following types of inputs;
  - Size of an Array
  - Polynomial Degree
  - Number of elements in matrix
  - Number of bits in binary representation of the input
  - Vertices and edges in graph.

HOW TO COMPARE ALGORITHMS?

- We should defines objective measures :
  - Execution time? → it is depend on the computer
  - Number of statements executed? → number of statement will varies depend on programming language used as well as the style of the programmer.
  - Ideal Solution? →Let us assume that we expressed running time of given algorithm as a function of the input size *n* (i.e *f(n)*) and compare these different functions corresponding to running time. This kind of comparison is independent from machine time, programming style, etc.

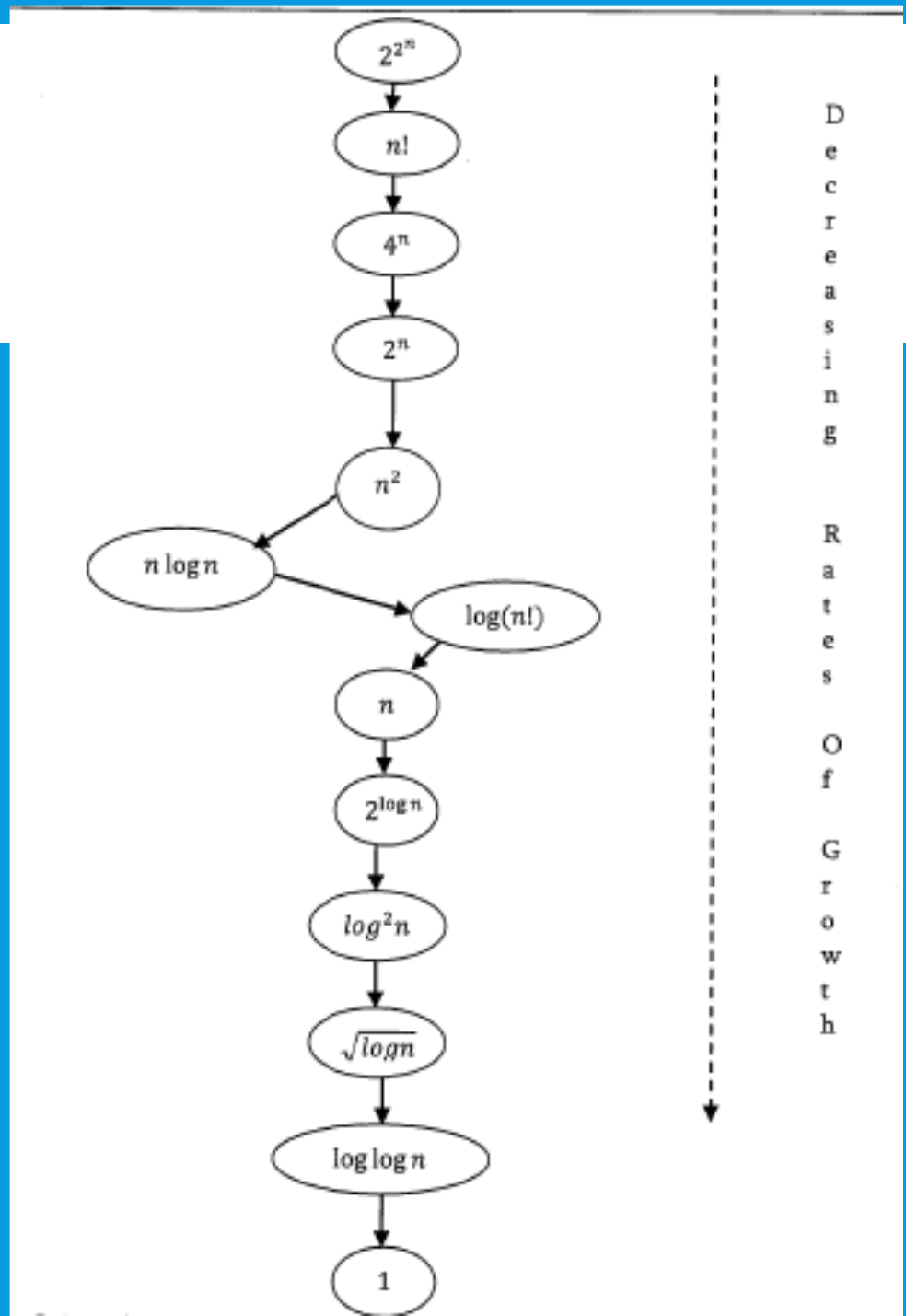
WHAT IS RATE GROWTH?

- Rate growth → The rate at which the running time increases as a function of input
- Example : $n^4 + 2n^2 + 100n + 500 \approx n^4$

COMMONLY USED RATE OF GROWTH

Time complexity	Name	Example
1	Constant	Adding an element to the front of a linked list
$logn$	Logarithmic	Finding an element in a sorted array
n	Linear	Finding an element in an unsorted array
$nlogn$	Linear Logarithmic	Sorting n items by 'divide-and-conquer'-Mergesort
n^2	Quadratic	Shortest path between two nodes in a graph
n^3	Cubic	Matrix Multiplication
2^n	Exponential	The Towers of Hanoi problem

RELATIONSHIP BETWEEN RATES OF GROWTH



HOW TO WRITE AN ALGORITHM

- There are no well-defined standards for writing algorithms. Rather, it is problem and resource dependent. Algorithms are never written to support a particular programming code.
- As we know that all programming languages share basic code constructs like loops (do, for, while), flow-control (if-else), etc. These common constructs can be used to write an algorithm.
- We write algorithms in a step-by-step manner, but it is not always the case. Algorithm writing is a process and is executed after the problem domain is well-defined. That is, we should know the problem domain, for which we are designing a solution.

EXAMPLE

Problem – Design an algorithm to add two numbers and display the result.

step 1 – START

step 2 – declare three integers a, b & c

step 3 – define values of a & b

step 4 – add values of a & b

step 5 – store output of step 4 to c

step 6 – print c

step 7 – STOP

EXAMPLE

Algorithms tell the programmers how to code the program. Alternatively, the algorithm

can be written as –

step 1 – START ADD

step 2 – get values of a & b

step 3 – c ← a + b

step 4 – display c

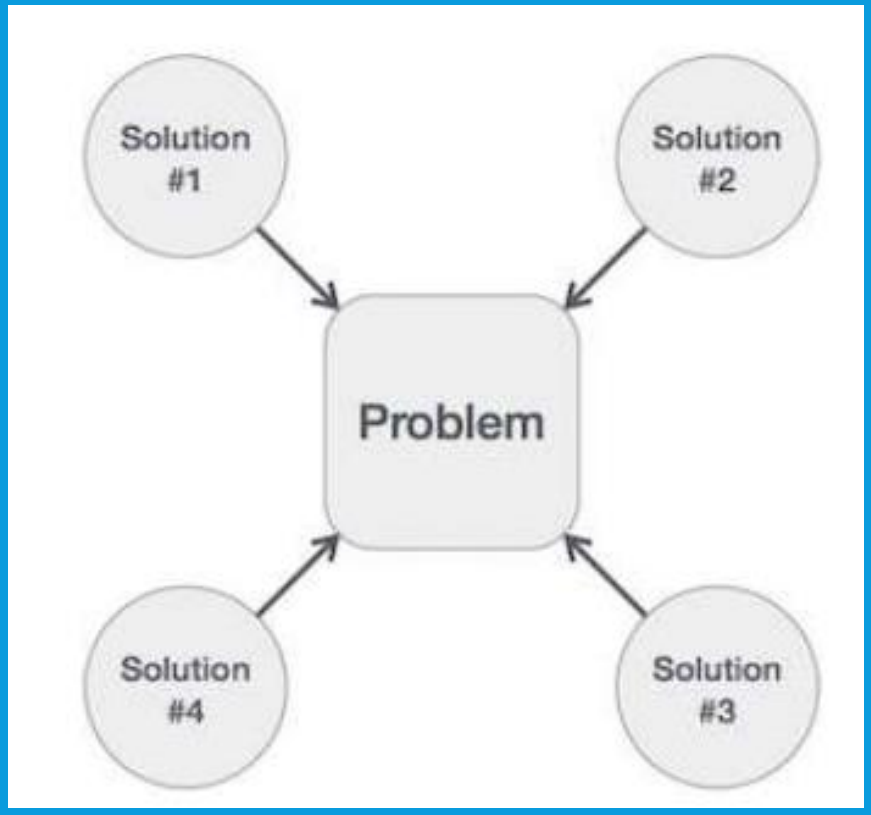
step 5 – STOP

In design and analysis of algorithms, usually the second method is used to describe an algorithm. It makes it easy for the analyst to analyze the algorithm ignoring all unwanted definitions. He can observe what operations are being used and how the process is flowing.

Writing step numbers, is optional.

We design an algorithm to get a solution of a given problem. A problem can be solved in more than one ways.

Many solution algorithms can be derived for a given problem. The next step is to analyze those proposed solution algorithms and implement the best suitable solution.



ALGORITHM ANALYSIS

Efficiency of an algorithm can be analyzed at two different stages, before implementation and after implementation.

A Priori Analysis – This is a theoretical analysis of an algorithm. Efficiency of an algorithm is measured by assuming that all other factors.

A Posteriori Analysis – This is an empirical analysis of an algorithm. The selected algorithm is implemented using programming language.

ALGORITHM COMPLEXITY

Time Factor – Time is measured by counting the number of key operations such as comparisons in the sorting algorithm.

Space Factor – Space is measured by counting the maximum memory space required by the algorithm.

The complexity of an algorithm f(n) gives the running time and/or the storage space required by the algorithm in terms of n as the size of input data.

SPACE COMPLEXITY

The space required by an algorithm is equal to the sum of the following two components –

- A fixed part that is a space required to store certain data and variables, that are independent of the size of the problem. For example, simple variables and constants used, program size, etc.
- A variable part is a space required by variables, whose size depends on the size of the problem. For example, dynamic memory allocation, recursion stack space, etc.

EXAMPLE

- Algorithm: SUM(A, B)
- Step 1 - START
- Step 2 - C ← A + B + 10
- Step 3 – Stop

Here we have three variables A, B, and C and one constant. Hence S(P) = 1+3. Now, space depends on data types of given variables and constant types and it will be multiplied accordingly.

TIME COMPLEXITY

Time complexity of an algorithm represents the amount of time required by the algorithm to run to completion. Time requirements can be defined as a numerical function T(n),where T(n) can be measured as the number of steps, provided each step consumes constant time.

For example, addition of two n-bit integers takes n steps. Consequently, the total computational time is T(n) = c*n, where c is the time taken for the addition of two bits. Here, we observe that T(n) grows linearly as the input size increases..