Part 1

DSA [20 Hrs.]

DSA Introduction

Data Structures

Tree Based DSA

Recursion

Hashing

Part 2

Graph Based DSA

Sorting & Searching Algorithms

Greedy Algorithms

Dynamic Programming

Competitive Programming Algorithms

Introduction

- ❑ A computer program is a collection of instructions to perform a specific task. For this, a computer program may need to store data, retrieve data, and perform computations on the data.
- ❑ A data structure is a named location that can be used to store and organize data.

Good Computer Program

- ❑ Run efficiently and correctly.
- ❑ User friendly interface.
- ❑ Easy to read and understand.
- ❑ Easy to debug.
- ❑ Easy to modify.
- ❑ Easy to maintain.

Good Computer Program

Program consists of 2 things:

a) Algorithms

b) Data Structures

*Good program is combination of both.

DSA Introduction

- An **algorithm** is a step by step process to solve particular problem statement.
- A **data structure** represents the logical relationship that exists between individuals elements of data to carry out certain tasks.

Algorithm??

- An **algorithm** is a precise procedure for solving a problem in finite number of steps.
- An **algorithm** states the actions to be executed and the order in which these actions are to be executed.

Algorithmic Properties

- ❑ It must be correct.
- ❑ Composed of series of concrete steps.
- ❑ No ambiguity.
- ❑ Composed of finite number of steps.
- ❑ Must terminate.
- ❑ It takes zero or more inputs.
- ❑ It should use less memory space as much as possible.
- ❑ It results in one or more outputs.

How to develop algorithm?

- ❑ Defining the algorithm.
- ❑ Validating the algorithm.
- ❑ Expressing the algorithm.

Efficiency of an algorithm.

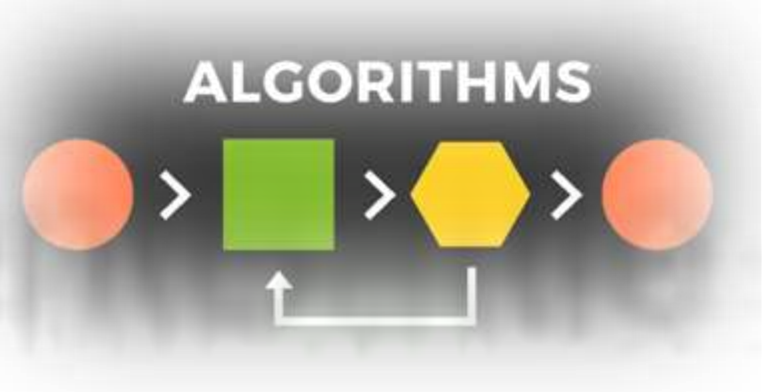
- ❑ Algorithms are programs in general form.
- ❑ It is an idea upon which program is designed.
- ❑ It should be independent of the programming language.
- ❑ Every programmer having enough knowledge and experience should understand it.
- ❑ It should be applicable to inputs of all sizes.

Efficiency of an algorithm.

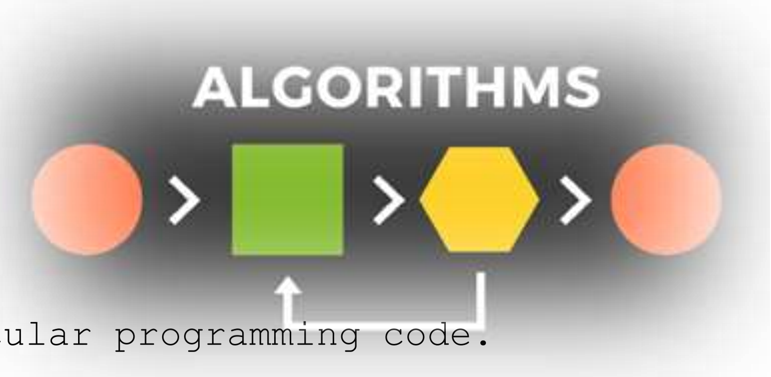
- ❑ Efficiency of an algorithm denotes the rate at which an algorithm solves a problem of size n.
- ❑ Time and Space
- ❑ Time = no. of steps algorithm executes.
- ❑ Space = no. of unit memory storage it requires.
- ❑ Algorithm complexity calculation: Time taken & Space required.

Characteristics of an algorithm.

- ❑ Unambiguous
- ❑ Input
- ❑ Output
- ❑ Finiteness
- ❑ Feasibility
- ❑ Independent



How to write an Algorithm?



- ❑ It is problem and resource dependent.
- ❑ Algorithm are never written to support a particular programming code.
- ❑ Problem domain should be well defined.
- ❑ Example: Design an algorithm to add 2 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
```

Time Complexity of an algorithm.

- The time taken for an algorithm is comprised of:

    Compilation Time

    Run Time

- Compile time is the time taken to compile an algorithm.

    It checks syntax and semantic errors.

- Run time is the time to execute the compiled program.

Is runtime calculated only for executable statements or declaration statements?

Time Complexity of an algorithm.

- ❑ Time complexity of an algorithm is generally classified as three types:

  Worst Case

  Average Case

  Best Case

Time Complexity of an algorithm.

- ❑ **Worst Case**: Longest time that an algorithm will use to produce desired results.
- ❑ **Average Case** : Average time that the algorithm will use. It depends upon probability distribution.
- ❑ **Best Case**: Shortest time that the algorithm will use to produce desired results.

Space Complexity of an algorithm.

- ❑ Space complexity of the program is the amount of memory consumed.
- ❑ Fixed amount of memory.
- ❑ Variable amount of memory.
- ❑ Memory taken by instruction not in control of programmer.
- ❑ Memory taken by variable is in control of programmer.

```
Datatype Array[size];
```

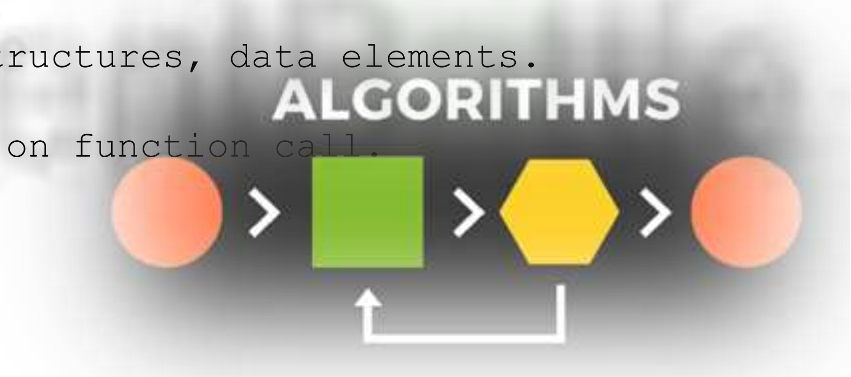
Space Complexity of an algorithm.

❑ There are 3 different spaces considered for determining the amount of memory used by the algo.

1. **Instruction Space:** occupied by compile version of program.

2. **Data Space:** used to hold variables, data structures, data elements.

3. **Environment Space:** used for run time stack on function call.



Problem Solving??

- ❑ A Programmer should identify all the requirements to solve the problem:
- ✓ Type of programming language
- ✓ Narration of the program describing tasks to be performed
- ✓ Frequency of processing
- ✓ Input & Output of the program
- ✓ Limitations & restrictions for the program
- ✓ Detailed specifications

Algorithm Analysis

- ❑ **Method 1:** Code each algorithm and run them to see how long they take.
    - ❑ **Problem:** How will you know there is better program?
- ❑ **Method 2:** Develop a model of the way computers work and compare how the algorithm behaves in the model.
- ❑ Able to distinguish between good and bad algorithms.

How to measure algorithm performance?

- ❑ Length of the program (LOC)
- ❑ Ease of programming (bugs, maintenance)
- ❑ Memory required
- ❑ Running time

Importance of studying Algorithm?

- ❑ Interview Preparations : provides competitive advantage during job search process.

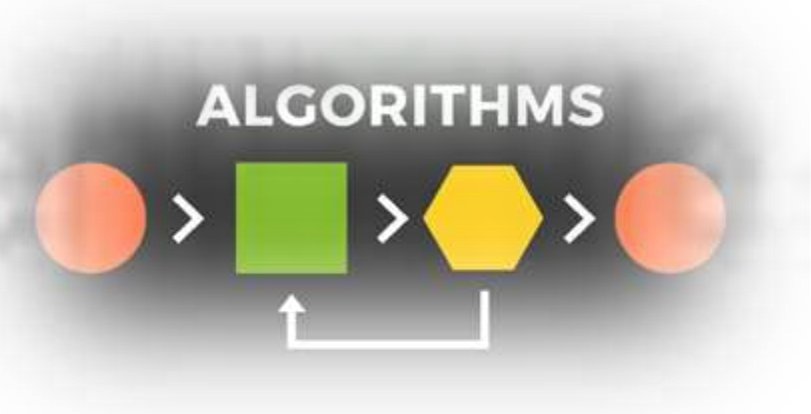
- ❑ Keeping your programming skills sharp. "Practice makes Perfect"

- ❑ Long term career goals.

"Practicing algorithms will increase your skill and your visibility at work".

Summary

- ❑ Introduction
- ❑ Good Computer Program
- ❑ DSA Introduction
- ❑ Algorithm
- ❑ Algo. Properties
- ❑ How to develop algo.
- ❑ Efficiency of algo.
- ❑ Characteristics of algo.
- ❑ How to write an algo.
- ❑ Time complexity
- ❑ Space complexity
- ❑ Problem solving
- ❑ Algorithm Analysis
- ❑ How to measure algo. Performance
- ❑ Importance of studying algorithm



TCS Exam Example

There is a JAR full of candies for sale at a mall counter. JAR has the capacity N, that is JAR can contain maximum N candies when JAR is full. At any point of time. JAR can have M number of Candies where M<=N. Candies are served to the customers. JAR is never remain empty as when last k candies are left. JAR if refilled with new candies in such a way that JAR get full.
Write a code to implement above scenario. Display JAR at counter with available number of candies. Input should be the number of candies one customer can order at point of time. Update the JAR after each purchase and display JAR at Counter.
Output should give number of Candies sold and updated number of Candies in JAR.
If Input is more than candies in JAR, return: "INVALID INPUT"
Given,
N=10, where N is NUMBER OF CANDIES AVAILABLE
K =< 5, where k is number of minimum candies that must be inside JAR ever.

Example 1:(N = 10, k =< 5)
Input Value
3
Output Value
```
NUMBER OF CANDIES SOLD : 3
NUMBER OF CANDIES AVAILABLE : 7
```

Example : (N=10, k<=5)
Input Value
0
Output Value
```
INVALID INPUT
NUMBER OF CANDIES LEFT : 10
```

Algorithmic Solution

```
Step1:  START
2: DECLARE AND INITIALIZE VARIABLES…N=10 AND K<=5
3. ACCEPT INPUT FROM THE USER..
4. CHECK CONDITION..
5. IF TRUE: DISPLAY THE POSITIVE / ACCEPTED RESULTS
6. IF FASLE: DISPLAY THE NEGATIVE/ REJECTED RESULTS..”INVALID INPUT.
7. STOP
```