1. Introduction

2. What is Data Structure and Algorithm

Data Structure: - In computer science, a data structure is a data organization, management, and storage format that enables efficient access and modification. More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data.

Algorithms:- It is a combination of sequence of finite steps to solve a particular problem.

For Example:- Multiplication of two numbers

```
def Mul():

  1. Take two numbers(a,b)
  2. Multiply two numbers a and b and store the value of result in c
  3. return c
```

Properties of Algorthims:-

- It should terminate after finite time
- It should produce at least one output
- It is independent of any sort of programming language.
- It should be unambiguous (Deterministic)

  Deterministic:- For the same output will come always.

  Not Deterministic:- For the same input different output will come. And this is not at all preferable whenever we write any sort of algorithms.

3. Why should we study this?

- If you want to work for any Product Based Company like Google, Face book etc. The very first step of selection criteria of any candidate is to check their coding skills (First Round of Interview). The more you confident in these subjects, more are the chances to get selected.

Question: Based on the above mentioned properties, can you determine whether the below program is an algorithm or not?

```
While 1:
      print('Hello world')
```

Answer:- Infinite times.... Not an algorithm...

Steps Required to construct an algorithm:

- Problem Definition -> what is the problem they are asking.
- Design algorithm -> out of existing algorithms which algorithm is more suitable to this problem statement.
- Existing algorithm -> Divide & Conquer, Greedy Technique, Dynamic Programming, Backtracking and so on.
- Draw flow chart.
- Testing -> for every input correct output is coming or not.
- Implementation -> coding part.
- Analysis

Note: Design algorithm and analysis are two major steps.

Analysis: If any problem contains more than one solution, the best one will be decided by the analysis based on mainly two factors:

1. **Time Complexity**- CPU time

2. **Space Complexity**- Main Memory Space\

Note: Time Complexity is more powerful than space Complexity because processor cost is more costly.

Time complexity: T(P)=C(P)+R(P)

C(P): Compile-time is the time at which the source code is converted into an executable code.

R(P): Run time is the time at which the executable code is started running.

Types of analysis:

1. Apostiary Analysis (Relative Analysis):

- Dependent on language of compiler and the type of hardware
- Exact answer
- Different answer
- Program run fast because of the type of hardware used.

2. Apriori Analysis (Absolute Analysis):

- Independent on language of compiler and the type of hardware.
- Approximate answers.
- Same answer
- Program run fast because of nice logic.

Apriori Analysis: It is a determination of order of magnitude of a statement.

O(magnitude)

Examples for better understanding of the concepts:

Problem 1:

```
x=y+z                       O(1) – Constant time
```

Problem 2:

```
x=y+z                       O(1) – Constant time
for i in range(1,n+1):
    x=y+z                   O(n) -time
```

Overall time complexity = O(1)+ O(n) = O(n)

Problem 3:

```
x=y+z                       O(1) – Constant time
for i in range(1,n+1):
    x=y+z                   O(n) –time
for i in range(1,n+1):
    for j in range(1,j+1):
        x=y+z               $O(n^2)$ –time
```

Overall time complexity- O(1)+O(n)+ $O(n^2)$ = $O(n^2)$

Problem 4:

```
i=n                         O(1) – Constant time
while i>1:
    i=i-1                   O(n-1) = O(n) – time
```

Overall time complexity=O(n)

Problem 5:

```
i=n                         O(1) – Constant time
while i>=1:
    i=i-2                   O(n/2)=O(n) –time
```

Overall time complexity=O(n)

Problem 6:

```
i=n                         O(1) – Constant time
while i>=1:
    i=i-30                  O(n/35)=O(n) – time
    i=i-5
```

Overall time complexity=O(n)

Problem 7:

```
i=1                         O(1) – Constant time
while i<n:
    i=2*i                   O(log₂n) – time
```

Overall time complexity=O($\log_2 n$)

Conclusion:

- Time complexity is loop only
- Not only loop but larger loop
- And if in a program there is no loop at all- O(1)