Introduction to Algorithms

In [ ]:
```
--> 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.
```

Example of an Algorithms

In [ ]:
```
Suppose that we are having a python code for Getting sum of Two Numbers:

Code:
    a=10
    b=20
    c=a+b
    print(c)

For the Above Code The algorithms will look like:

    1.Start
    2.Initialize a variable a=10
    3.Intialize b variable b=20
    4.Create a variable c that will store the sum of a and b
    5.Print(c)
    6.End
```

Introduction to Pseudocodes

In [ ]:
```
--> Pesudocode is a informal way of writing a program for better understanding
--> Pseduodcode doesnot compiler or interpreated
```

Example of PseudoCode

In [ ]:
```
Code:
    a=10
    b=20
    c=a+b
    print(c)

For the Above Code The algorithms will look like:

    1.Start
    2.Initialize a variable a=10
    3.Intialize b variable b=20
    4.Create a variable c that will store the sum of a and b
    5.Print(c)
    6.End

For the Above code The pseudoCode will look like:

    Begin
      Input A=10
      Input B=20
      Compute C=A+B
    Display C
    End
```

Components of an Algorithms

In [ ]:
```
#Components of Algorithms
1.Input Step --> Each and every algorithm is having atleast one input
2.Assignment Step --> each and every algorithm is having atleast one assignment operator
3.Decision Step --> if and else(Optional)
4.Repititive step --> for and while(Optional)
5.Output Step --> Eaxh and every algorithm is having atleast one output step.
```

Properties of an Algorithms

In [ ]:
```
1. Finiteness --> Algorithm will terminate after a finite number of step
2. Definiteness --> Algorithm must be Unambigous(clear)
3. Genarlity --> that algorithm will work for any number of input
4. Effectiveness --> algorithm is effective if it is taking least time and space complexity then your alogrithm is effective
5. Input and output

If any algorithm is following all these properties then that algorithm is a Perfect Algorithm.
```

Approches of Designing an Algorithms

In [ ]:
```
Each and Every Algorithm is implemented based on These Two Factors:
    1.Space Complexity --> Your algorithm how much space in the memory
    2.Time Complexity --> Running time of an algorithm is known as Time Complexity
```

Analysis of an Algorithm

In [ ]:
```
--> Suppose that M is an algorithm and n is the number of input so the algorthim m is directly propotional
    to the number of input
--> M is directly propotional to n(number of input)
```

Asymptotic Notations

In [ ]:
```
Asymptotic Notation --> gives an idea how our algorithm is working as compare to our algorithm.

There are Three Types of Asymptotic Notation:
    1.Big O Notation --> worst case time complexity--> bad time complexity
    2.Big Omega Notation -->Best case time complexity
    3.Big theta Notation -->Average Case time complexity
```

General time Complexities

In [ ]:
```
--> Ascending order of General Time Complexities --> O(1) , O(Log n), O(n),O(nlog n),O(n^2),O(n^3),O(2^n)
--> Time Complexity is totally based on Number of Iterations
```

Different Cases for Understanding Different Time Complexities

Case 1

In [ ]:
```
Case 1:
    a=10
    b=20
    if a>b:         --> O(1)
        print(a)
    else:
        print(b)
```

Case 2

In [ ]:
```
Case 2:
    x=3
    x+=2  --> O(1)
    print(x)
```

Case 3

In [ ]:
```
Case 3:
    n=300
    for i in range(n): --> O(n)
        print(i)
```

Case 4

In [ ]:
```
Case 4:
    n=10
    for i in range(n//2):    --> O(n//2)
        print(i)
```

Case 5

In [ ]:
```
Case 5:
    for i in range(n):   --> O(n)
        print(i)
    for j in range(n): --> O(n)
            print(j)


Time Complexity --> O(2n)  --> constant part always be ignored in case of time complexity.
```

Case 6

In [ ]:
```
Case 6:
    n=5
    for i in range(n):  --> O(n)
        for j in range(n): --> O(n)
            print(i)

Time Complexity --> O(n*n)--> O(n^2)
```

Case 7

In [ ]:
```
Case 7:
    n=4
    for i in range(n):  --> O(n)
        for j in range(n): --> O(n)
            for k in range(n):  -->O(n)
                print(i,j,k)

Time Complexity --> O(n*n*n)--> O(n^3)
```

Case 8

In [ ]:
```
Case 8:
    n=4
    for i in range(n):  --> O(n)
        for j in range(n): --> O(n)
            for k in range(n):  -->O(n)
                for a in range(n):  --> O(n)
                    print(i,j,k,a)

Time Complexity --> O(n*n*n*n)--> O(n^4)
```