In [2]:
```
"""Introduction of Algorithm
Components and Properties of an Algorithms
Time and Space Complexity
Asymptotic Notations - Big O, Omega and Theta Notation
Examples of Time Complexity
Best , Worst and Average Cases"""
```

Out[2]:
```
'Introduction of Algorithm\nComponents and Properties of an Algorithms\nTime and Space Complexity\nAsymptotic Notations - Big O, Omega and Theta Notation\nExamples of Time Complexity\nBest , Worst and Average Cases'
```

In [ ]:
```
Introduction of Algorithms-->Step by step representation of a problem is nothing but an algorithms
#Code:
a=10
b=20
c=a+b
print(c)
#Algorithms
1.Start
2.Initialize a variable a=10
3.Initialize b variable b=20
4.Create a variable c that will store the sum of a and b
5.Print(c)
6.end
```

In [4]:
```
PseudoCode-->Pseudocode is a informal way of writing a program for better understanding
             Pseudocode doesnot compiler or interpreated

Pseudocode:
    Begin
        Input A
        Input B
        Compute C=A+B
        print(C)
    End
```

Traceback:
```
  Input In [4]
    PseudoCode-->Pseudocode is a informal way of writing a program for better understanding
                 ^
SyntaxError: invalid syntax
```

In [ ]:
```
#Components of an Algorithm
1.Input Step --> &cah and Every algorithm have atleast one input
2.Assignment Step
3.Decision Step --> for if and else (Optional)
4.Repeatative step --> for and while(optional)
5.Output Step

#Properties of an algorithm:
1.Finitiness -->algorithm will terminate after afinite steps of iteration
2.Definiteness --> algorithm will not be unambiguous(It must be clear)
3.Generality --> our algorithm will work for any number of input
4.Effectiveness --->Algorithm is effective when it doesnot take much time and space
5.Output and Input
```

In [11]:
```
prime number --->  int   (General --> 0 to infinite)
even and odd number -->int (general 0 --infinite)
palindrome --->str and int ()
```

Traceback:
```
  Input In [11]
    prime number ---> int   (General --> 0 to infinite)
                  ^
SyntaxError: invalid syntax
```

In [ ]:
```
#Approches of Designing an algorithms
#1.Space Complexity --> Your algorithm how much space in the memory
2.Time Complexity --> Running time of an algorithm is known as Time Complexity
```

In [ ]:
```
#Analysis of an algorithm
#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)
```

In [13]:
```
fact=1
num=500000
for i in range(1,num+1):
    fact=fact*i
print(fact)
```

Traceback:
```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [13], in <cell line: 3>()
      2 num=500000
      3 for i in range(1,num+1):
----> 4     fact=fact*i
      5 print(fact)

KeyboardInterrupt:
```

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

In [ ]:
```
#If you are using if and else in your program then time complexity will be O(1)
a=10
b=29
if a>b:
    print(a)
else:
    print(b)
```

In [ ]:
```
n=300   #o(n)
for i in range(n):
    print(i)
```

In [ ]:
```
#General Time Complexities --> o(1),o(log n),o(n),o(nlogn),o(n^2),o(n^3),o(2^n)
```

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

In [ ]:
```
#Code Segement
x=10
y=20
if x<y:
    print("y")     --> o(1)
else:
    print("x")
```

In [ ]:
```
loops --> Time Complexity
```

In [ ]:
```
#Code Segment
n=1000
for i in range(n):     --> O(n)
    print(i)
```

In [14]:
```
#Code Segement
n=1000
for i in range(n):    ---> O(n)
    print(i)
for j in range(n):  ---> O(n)
    print(i)
```

Traceback:
```
  Input In [14]
    for i in range
                  ^
SyntaxError: invalid syntax
```

In [16]:
```
#Code Segement
n=4
for i in range(n):    #---> o(n)
    for j in range(n): #---> o(n)
        print(i,j)  #--> o(n^2)-->9
```

```
0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
```

In [ ]:
```
#Code Segement
n=4
for i in range(n):   #---> o(n)
    for j in range(n): #---> o(n)
        for k in range(n):  #---> o(n^3)
            print(i,j,k)  -->o(n^3)
```

In [ ]:
```
#Code Segement
n=4
for i in range(n):   #---> o(n)
    for j in range(n): #---> o(n)
        for k in range(n):  #-->o(n)
            for l in range(n):  #-->o(n)
                print(i,j,k)  -->o(n^4)
```

In [ ]:
```
n=int(input())
i=0
while i<n:
    while i<n:
        print(i)
        i=i+1
    i=i+1
```

In [18]:
```
#sum of n numbers
n=int(input())
sum=0
for i in range(0,n+1):     #-->o(n)
    sum=sum+i              #o(1)
print(sum)   #o(n+1) o(n)
```

```
10
55
```

In [19]:
```
n=int(input())
sum=(n*(n+1))//2    #o(1)
print(sum)
```

```
10
55
```

In [ ]:
```
def linear_search(array,key):
    for i in range(len(array)):
        if array[i]==key:
            print("Element found at ,",i)
            break
    else:
        return "element not found"
array[10,20,30,40,50]
key=200
Big Omega --> if the key value is 10 then we will get time complexity of o(1) and it is the best case
of above algorithm
Big Theta --> Average case --> if the key value is 30 then the time complexity is o(n//2) and it is average case
Big o Notation ---> Worst Case -->if the key value is preeset at the last index or the key value is not present
then time complexity will be o(n).
```

In [20]:
```
#Time Complexity of elements in linkedlist and array
access an element of an array/linkedlist -->o(n)
Traversing on array/linkedlist -->o(n)
insertion at first position of linkedlist -->o(1)
Deletion of element from first pos of linkedlist --> o(1)
Insertion at given position -->o(n)
deletion at given position --> o(n)
Insertion at last --> o(n)
Deletion at last -->o(n)
```

Traceback:
```
  Input In [20]
    access an element of an array/linkedlist -->o(n)
           ^
SyntaxError: invalid syntax
```

In [ ]:
```
#Summery
Array -->It is collection of similar element and each and every element is stored at contigous
memeroy location.
Complexities:
    Insertion at first pos of array -->o(1)
    Insertion at given pos of array -->o(n)
    Insertion at last pos of array --> o(n)
    Deletion from first pos of array --> o(1)
    Deletion at given pos of array --> o(n)
    Deletion from last pos of array --> o(n)
    Traversal of an array -->o(n)
    Access any element of an array -->o(n)


Operations on an array:
    Insertion --> at first position , At given position , at last position
    Deletion --> at first position , At given position , at last position
    Traversal --> Visiting each and every element of an array

List --> It a collection of disimilar datatypes and it is an object in python.

Linkedlist --> It is a collection of Nodes. Nodes is a combination of data and next
(address of next Node). We can change the size and element of a linkedlist at runtime(dynamic)

Complexities:
    Traversing on array/linkedlist -->o(n)
    Insertion at first position of linkedlist -->o(1)
    Deletion of element from first pos of linkedlist --> o(1)
    Insertion at given position -->o(n)
    deletion at given position --> o(n)
    Insertion at last --> o(n)
    Deletion at last -->o(n)

Operations on an array:
    Insertion --> at first position , At given position , at last position
    Deletion --> at first position , At given position , at last position
    Traversal --> Visiting each and every element of an array

Stack: it is Linear Data Structure that will follow the principle of LIFO(Last in First out)
Insertion (push) and Deletion(Pop) botha are done from the same end(Top)

Terminologies/Operations of Stack:
    1.Push --> Insert any element at the Top of the STACK.
    2.Pop --> Deletion of element from the top of the Stack
    3.Peek --> Return the Top Value of the Stack

Time Complexity:
    Push --> O(1)
    Pop --> O(1)
    Peek --> O(1)


Queue: It is linear data structure that will follow the princple of FIFO(First in first out)
Insertion will be done from Rear(Last)
Deletion will be done from Front(Start)

Terminologies and Operations of Queue:
    1.Enqueue --> Insert any element in the Queue(Rear)
    2.Dequeue --> Deletion of element in Queue(Front)
    3.Front/Peek --> return Front value

Time Complexities -->
1.Enqueue --> O(1)
2.Dequeue --> O(1)
3.Peek(Front) --> O(1)

Time Complexity --> Runtime of an algorithm
1.BIG O --> Worst Case Time Complexity
2.BIG OMEGA --> Best Case time Complexity
3.BIG THETA  ---> Average Case Time Complexity

Data structure and Algorithms --> It is way to storing the data so that we can use that data
in an efficient manner.
Types of Data Structure:
    1.Primitive
    2.Non Primitive --> Based on Storage Criteria -->Linear(Array,linkedlist, queue abd stack)
                                                  -->Non Linear (Trees and Graph)
                    --> Physical Data Structure and Logical/ADT

Variations of Linkedlist
1.Singly linkedlist
2.Doubly Linkedlist
3.Circular linkedlist
4.Doubly Circular linkedlist

Variation of Queue:
    1. Single Queue
    2.Cirucular Queue
    3.Priority Queue
    4.Double Ended Queue(Dequeue)

Time Complexity of Linear Search --> O(n)
```