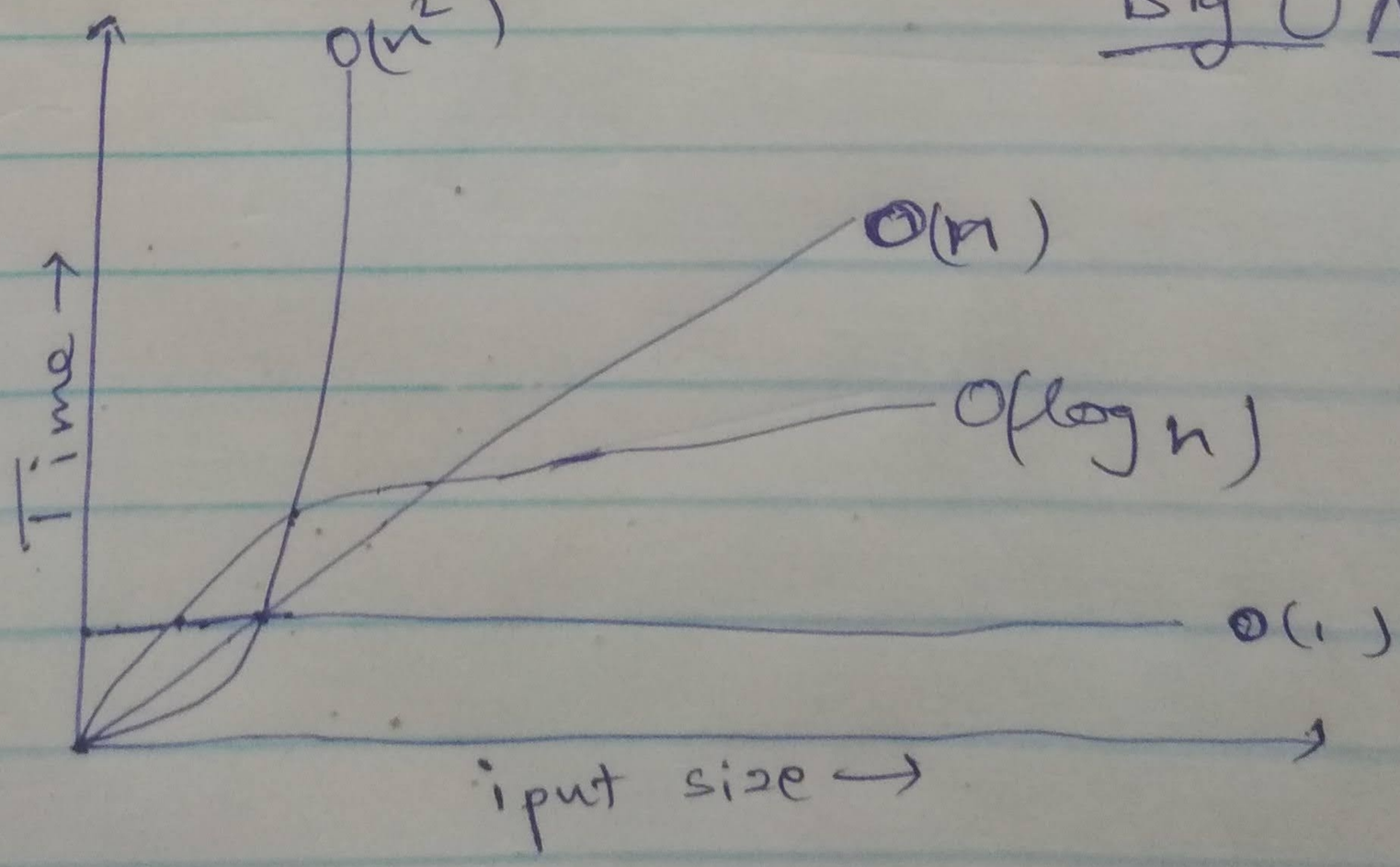
Day-01

Time Complexity

Time complexity of an algorithm signifies the total time required by the program to run till its compilation. The time complexity of algorithm is most commonly expressed using the big O notation. it's an asymptotic notation to represent the time complexity. There are some others to such as theta and Omn.

Note: We calculate this by using the number of times the statements get executed, Not the absolute time the system takes; 'coz, it varies for different machines.

Big O Notation



Constant Complexity O(1) :-

→ Run time won't change.

→ No matter what the input value is.

Sample code :-

```
print ("Hello Dude..")
```

Linear Complexity : O(n) :-

→ Run vary depending on input value.

→ For example printing the items in an array (or list).

→ It proportionally becomes longer to complete as the input grows.

#python3 code:

```
def printList(l):
    for e in l:
        print(e)
```

Quadratic Complexity : $O(N^2)$

→ takes number of steps equal to the square of it's input value.

```
for i in lst:
    for j in lst:
        print(i,j)
```

Polynomial Time $O(n^c)$:-

- represented as $O(n^c)$, when $c > 1$.
- we want to stay away from polynomial running times (quadratic, cubic, n^c, etc).

→ depending on the nested loops.

Logarithmic Time O(log n) :-

- used to divide problems in half every time.
- this algorithm splits the problem in half on each iteration.
- for example, to find a number in a sorted array, called "Binary Search"

to find 'n' => low = 0, high = len(arr) - 1, mid = 0

-> inside while loop

mid = (high + low) // 2

check if n is present at mid

if n is greater, ignore left half

if n is lower, ingnore right half

if wet reach the ~~end~~ of loop, and find 'n', return it.

Linearithmic - O(n log n) :-

- slightly slower than linear.
- still better than quadratic algo.

For example :- for each value in the data1 [O(n)] use the binary search [O(log n)] to search the same value in data2.

code :-

```
for value in data1:
    result.append(binary_s(data2, value))
```

Exponential time :- $O(2^n)$

- The calculations performed by an algorithm double every time as the input grows.

Example:-

```
def fibonacci(n):
    if n <=1:
        return n
    return fibonacci(n-1)+fibonacci(n-2)
```

Factorial Time- $O(n!)$

- multiplication of all positive integers less than itself.

→ Example:- permutations of string.