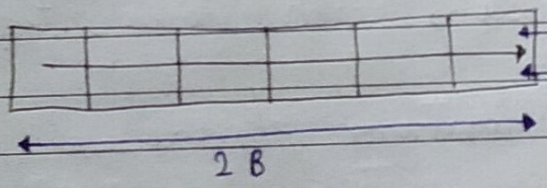
DATA STRUCTURES

→ The logical and physical representation of collection of data is termed as data structures.

Ex = int A;
- → Requires 2 bytes of memory
- → Constrain = Store only Integer values.



Stores Integer type
Physical Representation

* Physical representation deals with the storage and memory allocations.
* Logical representation is the constraints and operations applied on the variable.

Ex = int A; = It can only store Integer type values.

Types of Data Structures

Linear Data-Structures

→ In which data is arranged in sequential order.

Non-Linear Data Structures

→ Data is not arranged in sequential

We can perform various operations such as :-

INSERTION = Adding certain values.
DELETION = Removing certain values.
SORTING = Arranging
SEARCHING = Checking each value till the desired condition satisfies.
TRAVERSING = Accessing each value till the last.
MODIFICATIONS = Performing changes.

*** Difference between Searching and Traversing.

Searching is accessing and checking each data till the condition satisfies.
As soon as we got the condition, can stop.

Traversing is accessing each data till last.

* ALGORITHMS :-

→ The step - by - step description of the program, it is termed as Algorithm.
→ Kind of blueprint of the code.
→ Pseudo Code.
→ It is helpful so that code can be designed smoothly without any confusion.

TIME COMPLEXITIES =

→ The time taking in the Execution of the program according to the size of Input.

Best case = Minimum time taken in the Execution of program

Worst Case = Maximum time taken in the Execution of program.

Average Case = The Execution time between best and worst case.

Eg = (1) 2 3 4 5 6 7 ---n

Best Case = Only 1 checking
∴ 1 trial

$\Omega(1)$ Best Case ∝ Constant

Worst Case = Checking all conditions till n.
Maximum time taken.

$O(n)$ Worst Case ∝ n

Average Case = Checking all conditions till it get satisfied.

~~$O(n)$~~ $T = \frac{n(n+1)}{2} \Rightarrow T = \frac{(n+1)}{2}$

Average Case ∝ n

* Conclusion = Best Case & Constant = $\Omega(1)$
Worst Case & $n = O(n)$
Average Case & $n = O\left(\frac{n+1}{2}\right) = O(n)$

* Rule = (To find Time Complexity)
⇒ Point the most growing term.
⇒ Eliminate the coefficient

Ex = $\left(2n^2 + \frac{n}{2} + 1\right)$

→ In the above Equation, if we put n = 1
The highest growing term is $2n^2$

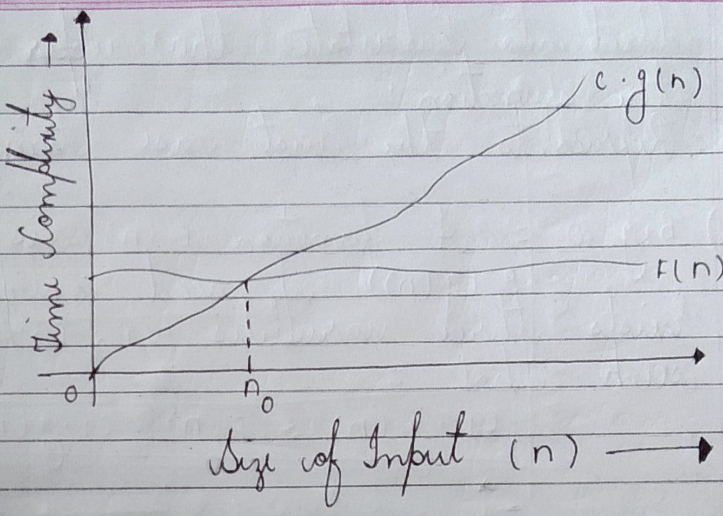
→ From $2n^2$ if we eliminate the coefficient then we get n^2.

Hence, time complexity of the given Equation becomes $O(n^2)$.

ASYMPTOTIC NOTATIONS

1) Big O = A function F(n) is said to be $O(g(n))$ if and only if there exists two variables c and n_0, such that =

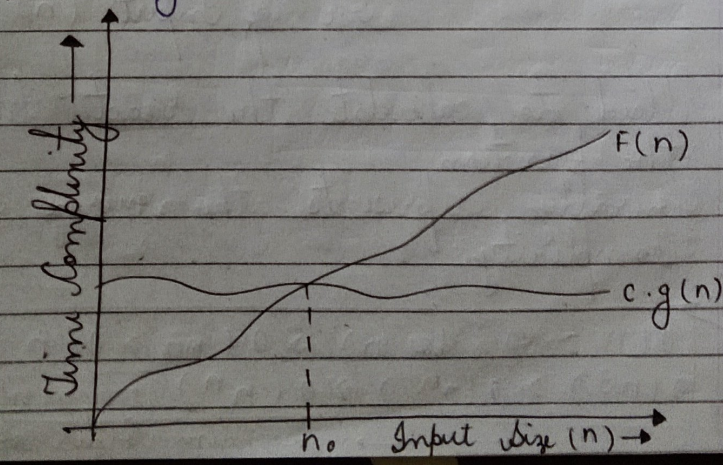
$$0 \leq F(n) \leq c \cdot g(n) \quad \forall\, n \geq n_0$$



→ Describes the lower bound of the fu
→ Used to calculate the worst case.

ii) Big Ω = A function F(n) is said
be Ω(g(n)) if and onl
there exists two constants c and n
such that =

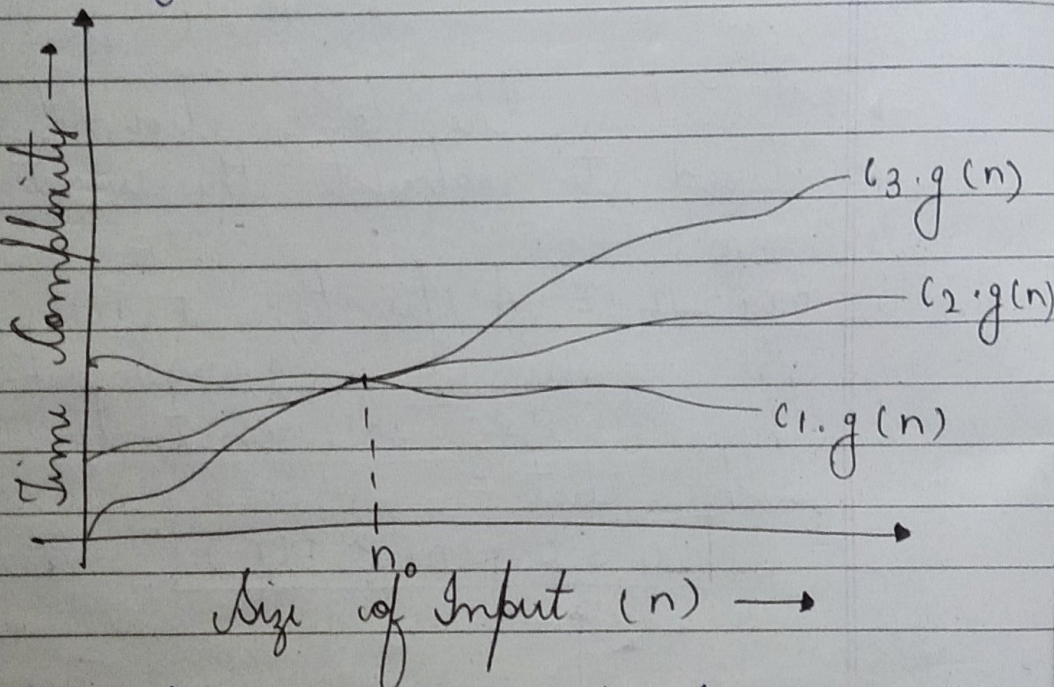
$$\boxed{0 \leq c \cdot g(n) \leq F(n)} \quad \forall\, n \geq n_0$$



→ Used to calculate the upper bound of the function.
→ Represents the best case time complexity.

iii) Big O = A function F(n) is said to be O(g(n)) if and only if there exists three variables c_1, c_2 and n_0 such that =

$$c_1 \cdot g(n) \leq F(n) \leq c_2 \cdot g(n) \quad \forall\ n \geq n_0$$

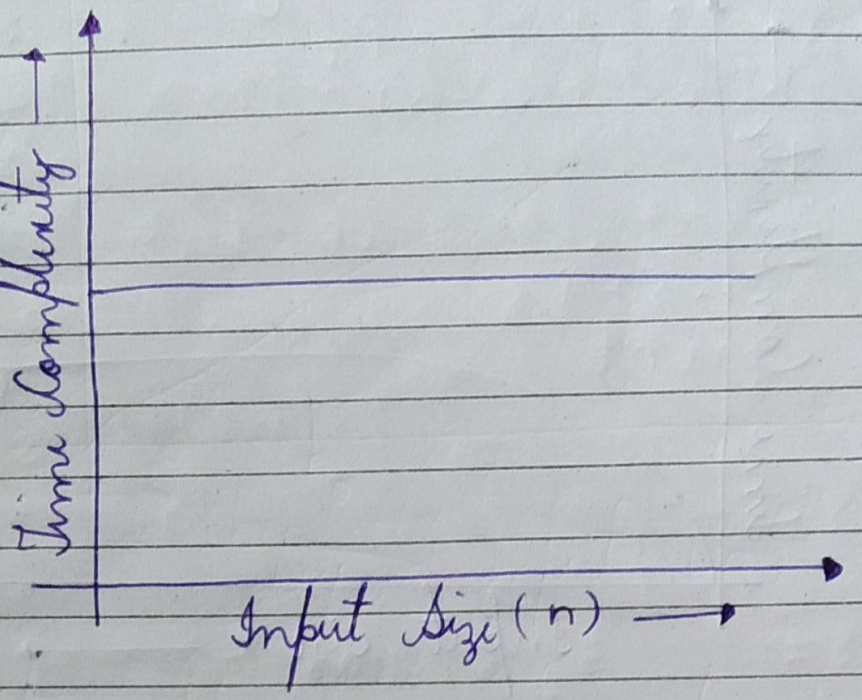


→ Used to calculate the exact bound of the function.
→ Generally represents the Average Case time complexity.

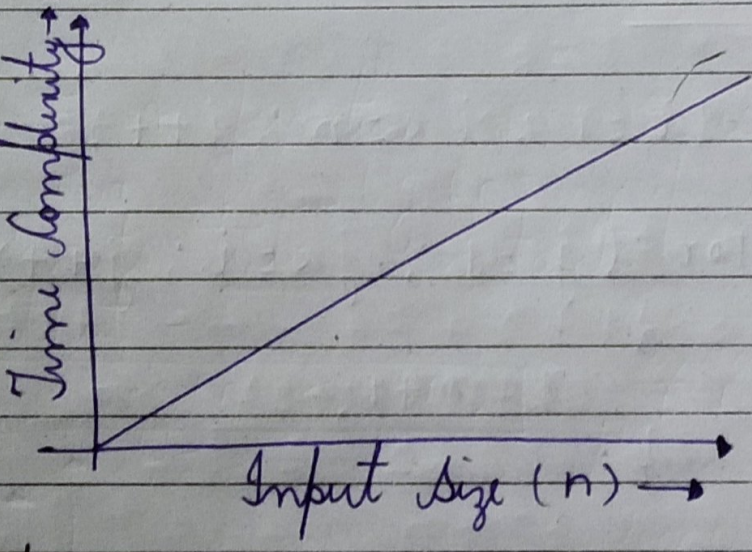
$O(1) > O(\log n) > O(n) > O(n \log n) > O(n^2) > O(n^3) > O(2^n) > O(n^n)$

→ Ascending Order of Time Complexity

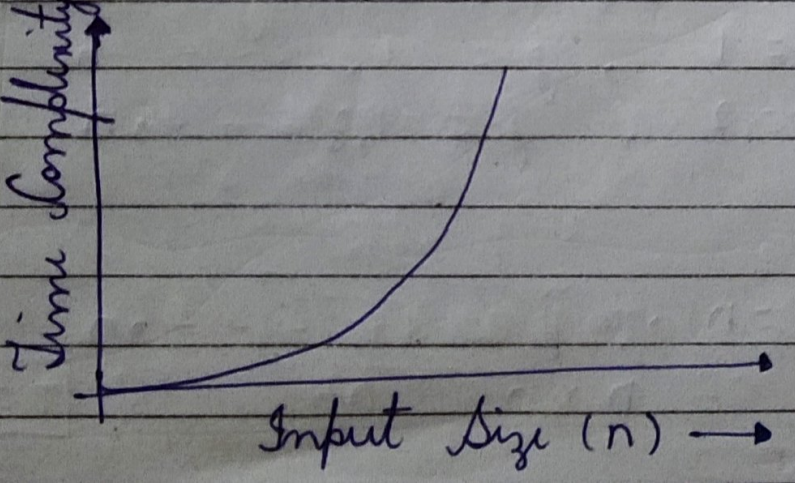
Constant Time = O(1)



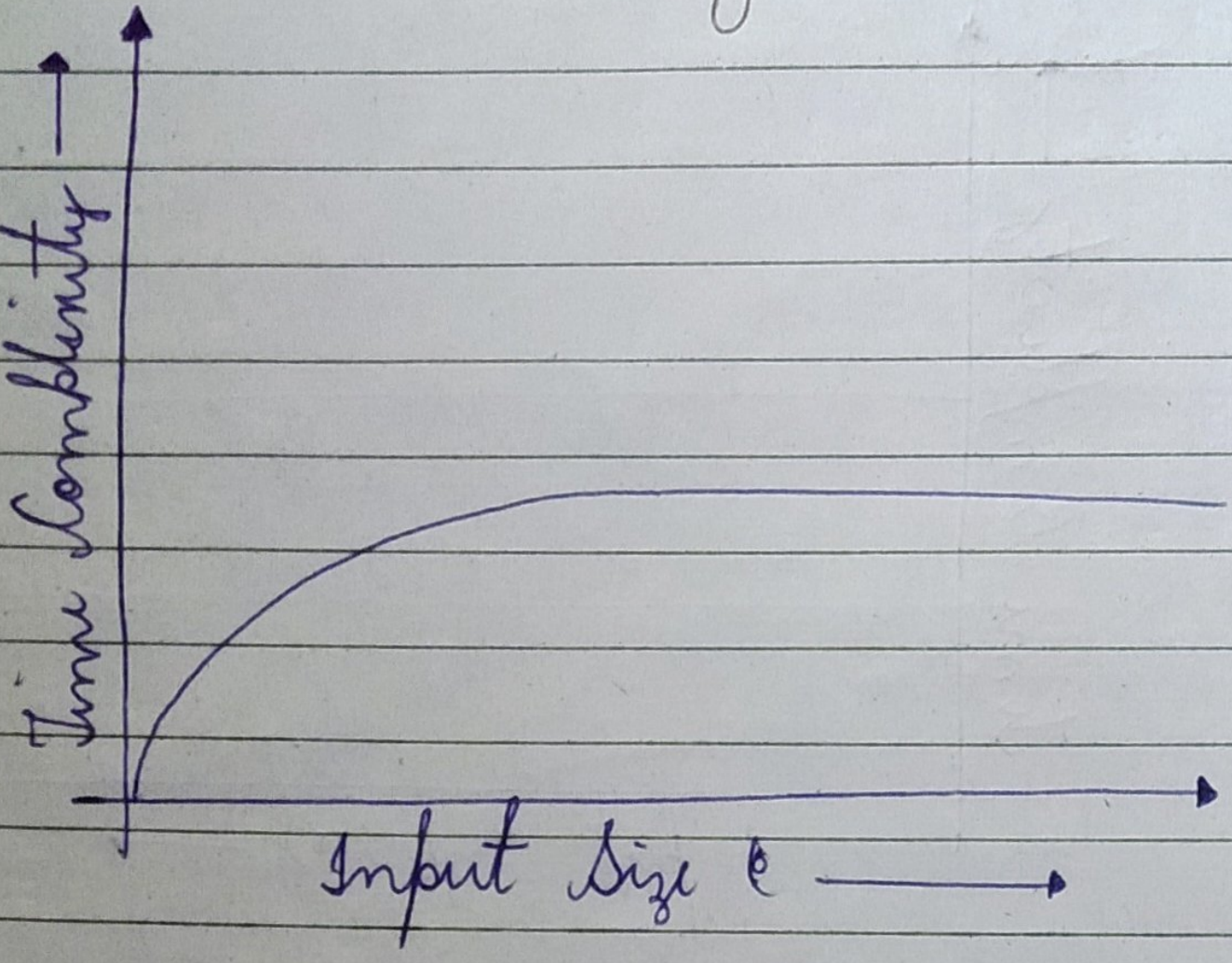
Linear Time = O(n)



Quadratic Time = O(n^2)



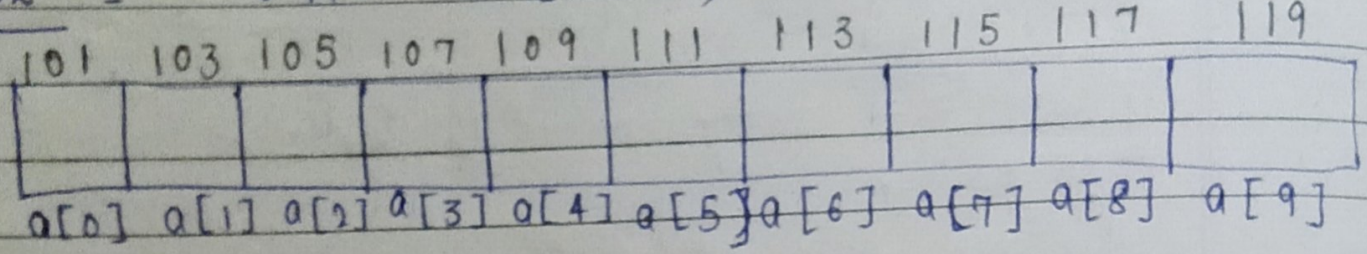
Logarithmic Time = O(log n)



ARRAYS

Linear data structure.
Collection of Homogenious (same type) data Items.

Ex = int A[10];



OPERATIONS ON ARRAY =

- Traversing
- Insertion
- Deletion
- Searching
- Sorting

ALGORITHM =

- Pseudo Code
- Step by Step Instruction to perform a task

Important steps for writing Algorithms =

1. Each algorithm starts with keyword ALGORITHM followed by name of the algorithm and

list of paramiters attatched Inside round brackets.

Ex = ALGORITHM Traverse_Array (A, N);

2. The body of each Algorithm must start with keyword BEGIN and Ends with keyword END.

3. Every valid line must End with keyword semi-colon (;)

4. The value assigned to the variable is done with a keyword (:=).

Ex = A := 20;

* TRAVERSING AN ARRAY =
→ Visiting Each and Every Element of the Array Exactly Once.

Ex = ALGORITHM Traverse_Array (A, N)
// A is the Array with N Elements
This Elements traverse the Array and prints Elements One by One.

BEGIN
FOR I := LB to UB // LB and UB is the lower and Upper bounds of Array.

Date: / /

Visit A[I] and Print : A[I]

```
I := I+1;
END;
```

☞ Note := Compiler assigns resources to a program only when, there is assignment statement and return statement.

Frequency is the number of times Instruction

Statement No.	Statement	Complexity (0/1)	Frequency	Statement Total
1.	Algorithm Traverse_Array (A,N)	0	1	0
2.	BEGIN	0	1	0
3.	For I:= LB to UB	1	(N+1)	N+1
4.	print Arr[I]	0	1	1
5.	I := I+1;	1	N	N
6.	END;	0	1	1

∴ T = O(N)

$\sum$ S.T = (2N+1)