DESIGN AND ANALYSIS OF ALGORITHMS [BSCA 34]

UNIT - I

Algorithm:

Algorithm is a set of step to complete a task.

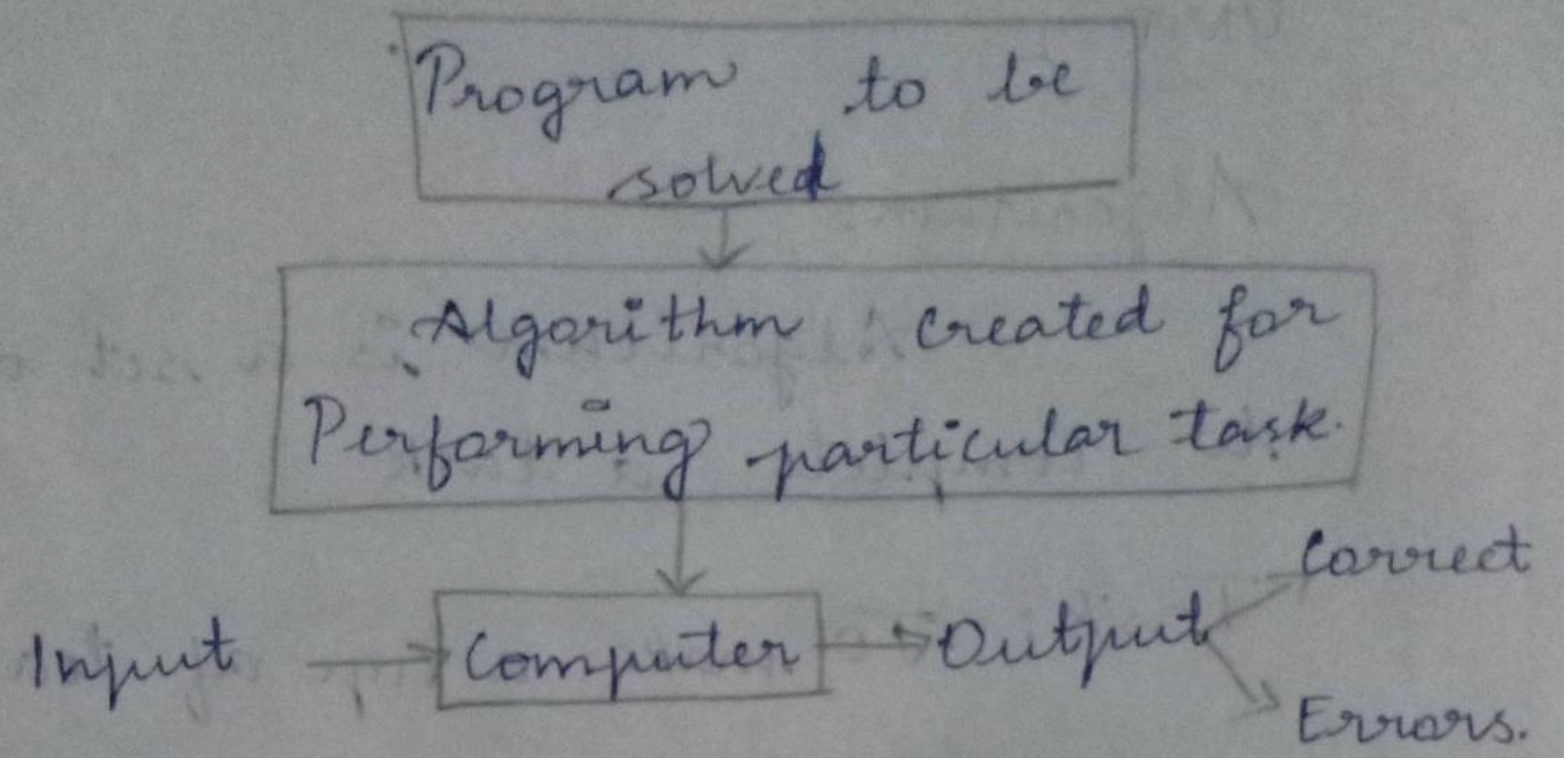
Task: To make a cup of tea.

Algorithm:

* Add water, milk to the bowl
* Boil it, add tea leaves
* Add sugar, and then serve in a cup.

Definition:

* Algorithm is a finite set of Instruction that can be used to perform some Calculations
* It is a best way to represent the solutions of a particul ar program in a very simple and efficient way.
* Algorithm can be impitemen ted in any programming languages.



How to write an algorithm:

* Sequence of instruction written in simple English.
* It is divided into two step

Rules for Writting:

1. Heading consists of keywords algorithm and Name of the algorithm with parameters of list

Eg; Algorithm name(P_1, P_2 ... P_n).

↓ Keyword ↓ name of algorithm ↓ Parameter list.

2. Heading Contains

// Problem description:

// Input:

// Output:

3. Body of Algorithm written in any program in construct, for, while or assignment statement.

4. Compound Statement should be written in {and}.

5. Single line Commants or written return using // as beginning

6. Identifier:

* They should be written in first alphabet not by digit

* Use Alpha Numeric

* Underscore is parameter.

7. Assignment Operator (←).

Eg

Variable ← Expression;

8. i) Boolean Operator (T/F).

ii) Logical Operator (AND, OR, NOT)

iii) Relational Operator (≤, ≥, ≠).

9. Array are stored [ ].

10. i) To get input: read()

eg) read(val)

ii) To Print Output: write()

(eg) write(" Wellome").

11. Conditional Statement

(if - then - else)

if (Conditional) then

Statement 1

else

Statement 2

12. While (conditional) do

{

statment 1

statement 2

⋮

Statement n

}

13. False Statement.

for Variable ← value 1 to value n
do

{

Statement 1

⋮

Statementn

}

(eg) for i ← 1 to n

{

write [i]

}.

14. Repeat until statement.

repeat _ until statement

repeat

stat 1

⋮

stat n

until (condition)

15. i) Break used to exit from inner loops

ii) Return used to return control from one point to another.

Characterics of Algorithm:

* Must take an include input.

* Must give some Output (yes, no, any values, etc).

Definiteness: Each instruction is clear and unambiguous.

↓ Confuse ill a word

Finiteness: Algorithm terminates after a finite number of steps.

Effectiveness: Each instructions must be very simple

Algorithm:

Eg-1 Sum of n numbers:

```
Algorithm sum(0,1).
// Problem description: To find
                sum of a number
// Input : n
// Output : sum of n number.
Result ← 0
for i ← 1 to n do
result ← result + i
return result.
```

Example-2.

Write an algorithm to check whether the given number is odd or even

```
Algorithm even_odd(a).
// Problem description: Find given
                no of odd or even.
// Input : a
// Output : Print the msg odd or
                                even
```

a = 5 1 - Quot.
5

if (a % 2 == 0) then

write ("Even Number").

else

write ("Odd Number").

Eg- 3.

Write an algorithm for sorting the elements

Algorithm ascending (a[], n)

Problem Description: Sort the elements

Input : An array which contains elements and n is the total elements.

Output : Display elements in ascending order.

for (i ← 1 to n) do

for (j ← i+1 to n) do

if (a[i] > a[j])

{

temp ← a[i]

a[i] ← a[j]

a[j] ← temp

}

write ("Sorted list is :");

pf (n = 3)

① | 5 | 3 | 2 |

② | 3 | 5 | 2 |

③ | 3 | 2 | 5 |

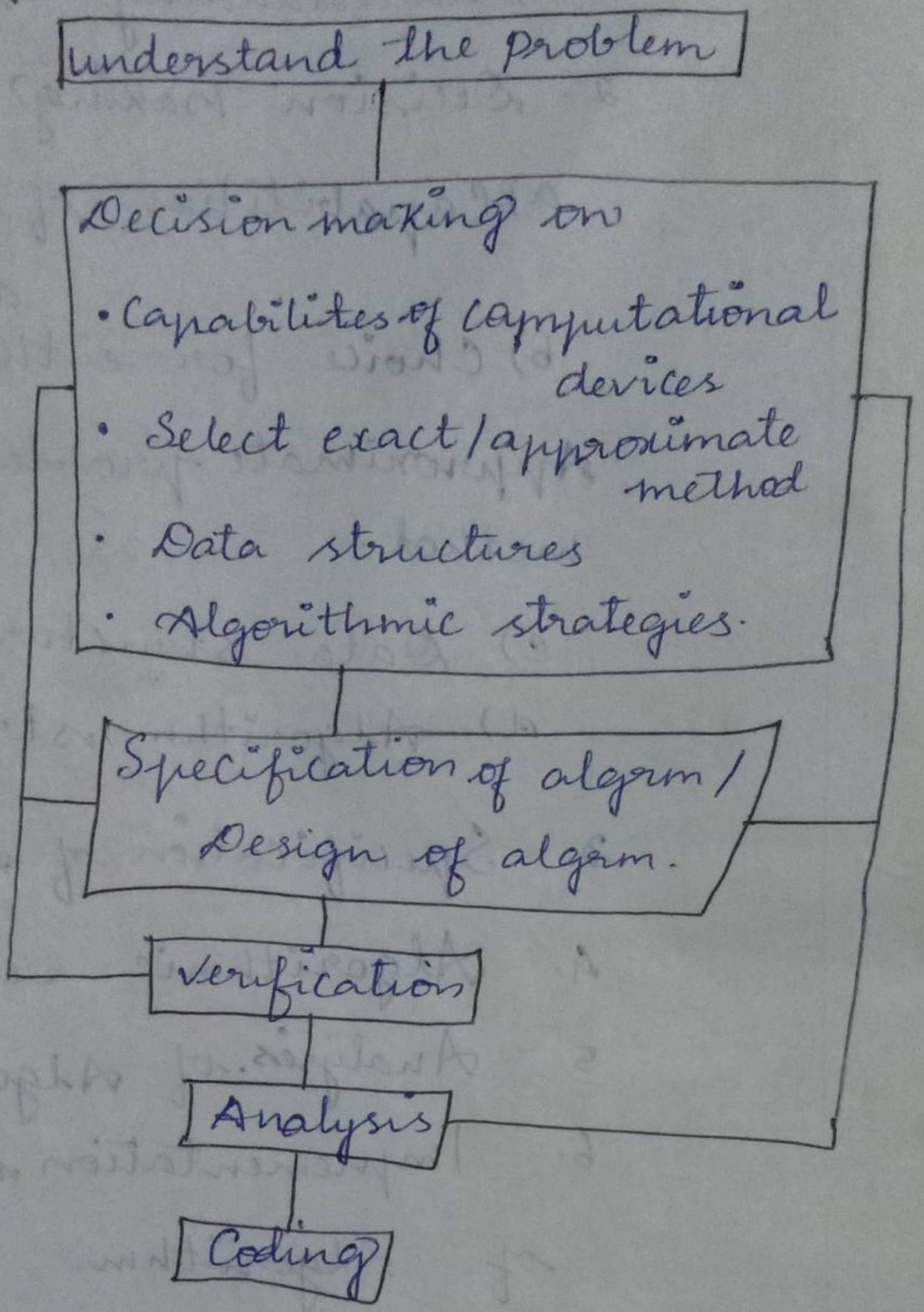
↓

| 2 | 3 | 5 |

Algorithm Design:

1. Understanding the Problem
2. Decision making
   a) Capabilities of computional ~~devices~~ devices
   b) Choice for either exact / Approximate problem solving method.
   c) Data structure
   d) Algorithm strategies.
3. Specification of Algorithm.
4. Algorithmic verification
5. Analyses of Algorithm.
6. Implementation or Coding of Algorithm.

Algorithm design Steps.



1. Understanding the problem:

* First need to understand the Problem statement Completely.

* Read the problem descri-Ption Carefully and ask question for

clarifying the doubts)

* Some types of problems that are commonly occurs.

* To solve such problems there are typical algorithms which are already available.

* After applying such an existing algorithm it is necessary to find its strength and weakness (eg. efficiency, memory, utilization).

* (Sometimes we have to design an algorithm on your own

* The input to the algorithm is called instance of the problem)

* Then decide the range of Input so that the boundary values of algorithm get fixed. The algorithm should work correctly for all valid inputs.

2. Decision Making:

* After analyzing the input

-and need to decide the following certain issues)
Such as capabilities of computa
-tional devices, whether to use
~~exact or a~~

* Exact or approximate prob
-lem solving

*. which data structures has
to be used, and to find the
algorithmic technique for solving
the given problem.

a) Capabilities of Computational devices

* We can classify an algorithm
from execution point of view as
Sequential algorithm and Parallel
algorithm.

* Instructions are executed one
after another.

* Instructions are executed
in parallel.

* i.e space and time is
essential.

b) Choice for either exact or approxi
-mate problem solving method:

(* If Problem needs to be solved correctly then we need exact algorithm & otherwise if the problem is so complex that we won't get the exact solution, then in that situation we need to choose approximation algorithm)

c) Data structures: (for organizing & storing data)

Proper data structure is required before for designing the actual algorithm.

d) Algorithmic Strategies:

Algorithmic techniques or algorithmic paradigm are

i) Brute force

ii) Divide-and-conquer

iii) Dynamic programming

iv) Greedy technique

v) Back tracking.

3. Specification of algorithm:

Algorithm → Using natural language
→ Pseudo code
→ Flow chart.

i) Natural language

* It is very simple to specify an algorithm using natural language.

* But many times specification of algorithm by using natural language is not clear. But, each and every step is definite.

for example, Write an algorithm to Perform addition of two numbers:

Step1 : Read the first number say a

Step2 - Read the second number say b

Step3 - Add the two numbers and store the result in a variable c

Step4 - Display the result

ii) Psendo code:

* Psendo code is nothing but a combination of natural

language and programming language constructs.

* A pseudo code is usually more precise than a natural language.

eg. Write an algorithm for performing addition of two numbers.

Algorithm sum(a,b)

// Problem description : This algori-thm performs addition of two numbers integers

// Input : Two integers a and b

// Output : addition of two integers

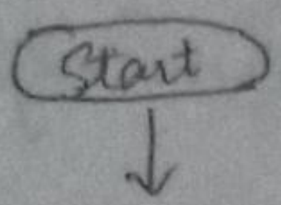
c ← a+b

write(c).

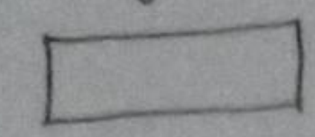
iii) Flow Chart:

* It representing the algorithm is by flow chart. Flow Chart is a graphical representation of an algorithm.

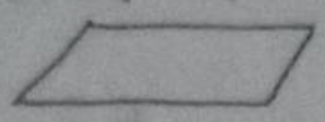
* It is useful when algorithm is small & simple Computation



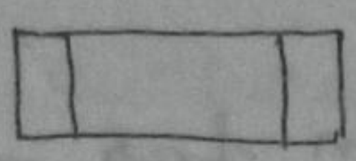
Start state.



Transition

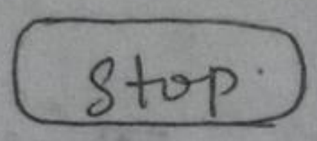


Processing or assignment statements



Input- Output statements

conditional statement



Stop state.

For. eg:

4. Algorithm Verification:

* Algorithmic Verification means checking correctness

* Check whether the algorithm gives correct output in finite amount of time for a valid set of input

* A common method of proving the correctness of an algorithm is by using mathematical induction is used.

5. Analysis of algorithm:

* Time efficiency of an algorithm
* Space efficiency of an algorithm
* Simplicity of an algorithm
* Generality of an algorithm
* Range of input.

i) Time efficiency:

It means the amount of time taking in the algorithm to by an algorithm to run

ii) Space efficiency.

It means amount of space (memory) taken by an algorithm.

iii) Simplicity:

It means generating sequence of instructions which are easy to understand.

iv) Generality:

It is easy to design an algorithm in more general way, than designing for a particular set of input.

v). Range of Input:

It Comes in picture when we execute an algorithm

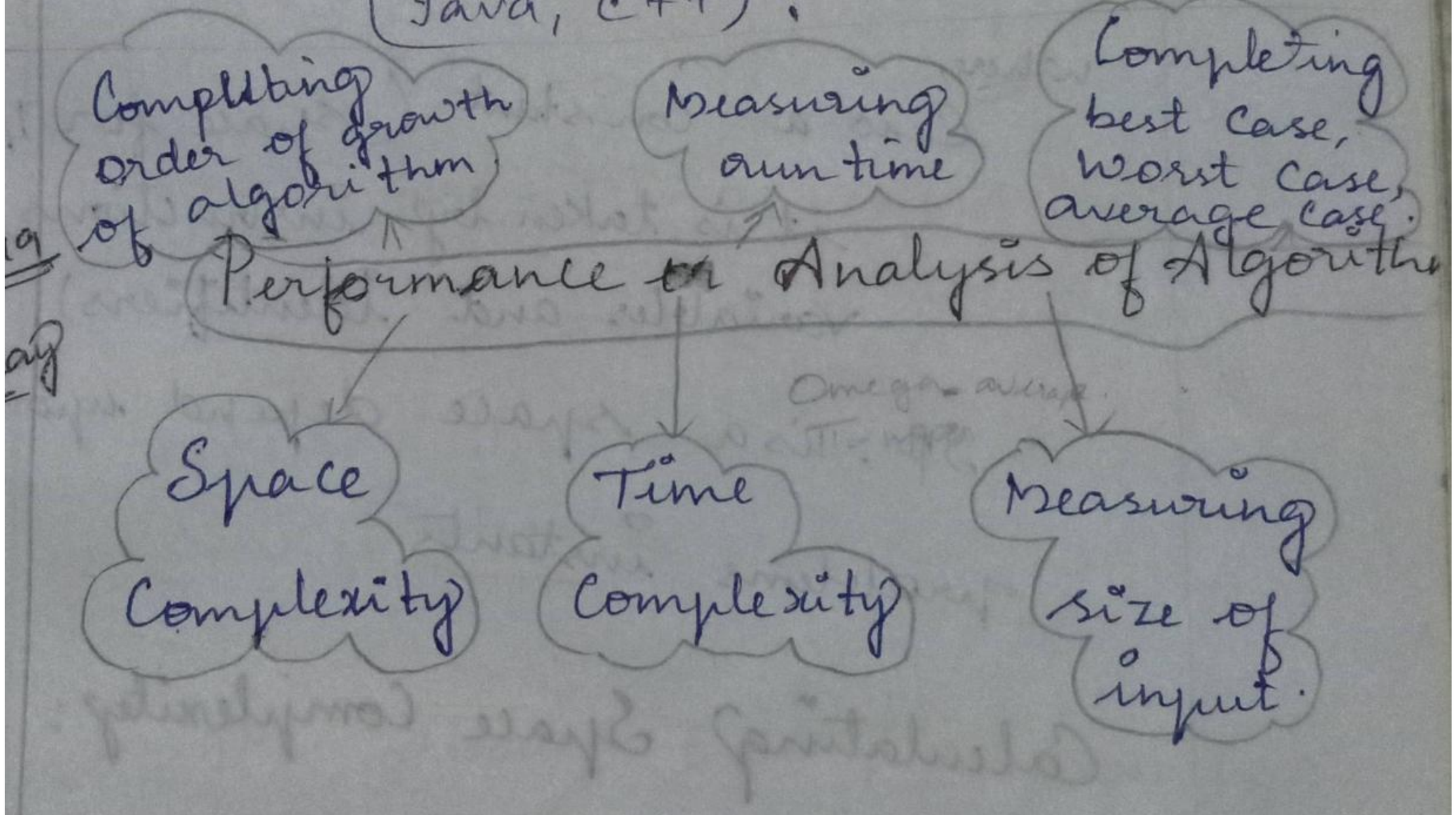
6. Implementation of Algorithm.

* The Implementation of algorithm is don by suitable Programming languages.

Eg,

* if algorithm consists of objects, then we choose object orien -ted programming languages.

(Java, C++).



i) Space Complexity:

It means amount of space (memory) taken by an algorithm.

For any algorithm, memory may be used for the following

i) Variable (constant variable, temporary variable).

ii) Program instruction

iii) Execution

Space requirements (S(P)) is calculated.

$S(P) = C + SP$.

where,

C is a Constant (space for I/P — it is taken by instructions, variables and identifiers).

SP = is a space depend upon on problems instants (eg, sample)

Calculating Space Complexity:

Eg,

```
int z= a+b+c
return(z).
```

In this example, variables a, b, C, Z are integers, its takes two bytes

each therefore total memory.

∴ Total memory = 8+2 = 10.

↳ Additional two bytes used for return values this type of space Complexity is called constant Complexity.

Eg-2;

```
int sum (int a[ ], int n).
  {
    int x=0
    for( int i= 0; i<n; i++).
    {
      x= x+a[i].
    }
  return x
  }.
```

In the above Example 2xn bytes of space is required for the a[ ] elements.

Two bytes each for x, i, n, return values

∴ Total memory = 2n + 8, which is

increasing Linearly in the inputs value n.

Hence it is called Linear Space Complexity.

ii) Time Complexity:

* Amount of time taken by an algorithm to run.

* It is difficult to completed compute the time Complexity in terms of
→ Physically clocked time.

* For example, in multi_user System execution time depend on

⇒ System Load.

⇒ Number of other programs running

⇒ Instruction set

⇒ Speed of hardware

∴ ~~T(n)~~

Time Complexity is therefore given in terms of frequency

Count.

Frequency count is the Count denoting number of times of execution of statements.

Problems

To find the square of the Number

Solution 1

```
S ← 0
for i + 1 to n do
S ← S+n
return S.
```

n = 3

$n = 3^2$

n = 9.

Solution 2.

```
return n*n.
```

⇒ In solution 1, loop will run n numbers of times so time Complexity is n times. If n increases time also increases.

⇒ In solution 2, the time Complexity is Constant because time will never

depend on n values.

=> Hence Algorithm Performance may vary with different types of input values.

∴ We use worst-Case time Complexity.

Calculating time complexity:

Example-1:

```
Statement;
```

Time Complexity will be Constant. This type of algorithm is called time complexity.

Eg-2;

```
for(i=0; i<n; i++)
{
 Statement;
}
```

The running time of loop is directly proportional to N.

This time is called linear time complexity.

Eg-3:

```
for(i=0; i<n; i++)
{
 for(j=0; j<n; j++).
 {
   statement;
 }
}
```

The running time of two loop is proportional to square of n.

This time Complexity is called Quadratic time complexity.

Example-4:

```
While (low<= high)
{
 mid = (low + high) ;
          2.
 if (target < list[mid]
 high = mid-1;
 else if (target > list[mid]
  low = mid +1;
 else break;
```

Running time of algorithm is proportional to the Numbers of time n can be n/2. This type is called logarithmic time complexity.

Asymptatic Notation:

* The Performance can be measured by computing time complexity of each algorithm.

* Asymptatic Notation is a shorthand way to represent the time complexity.

* Using Asymptatic notation we use time complexity as "fastest possible", "Slowest Possibe", "average possible".

* Asymptotic Notation is represent to Mathematical tool

* The Notations are,

i) Big oh (O) - fastest

ii) Big Omega (Ω) - slowest

iii) Big theta (Θ). - average

i) Big oh (O) - fastest

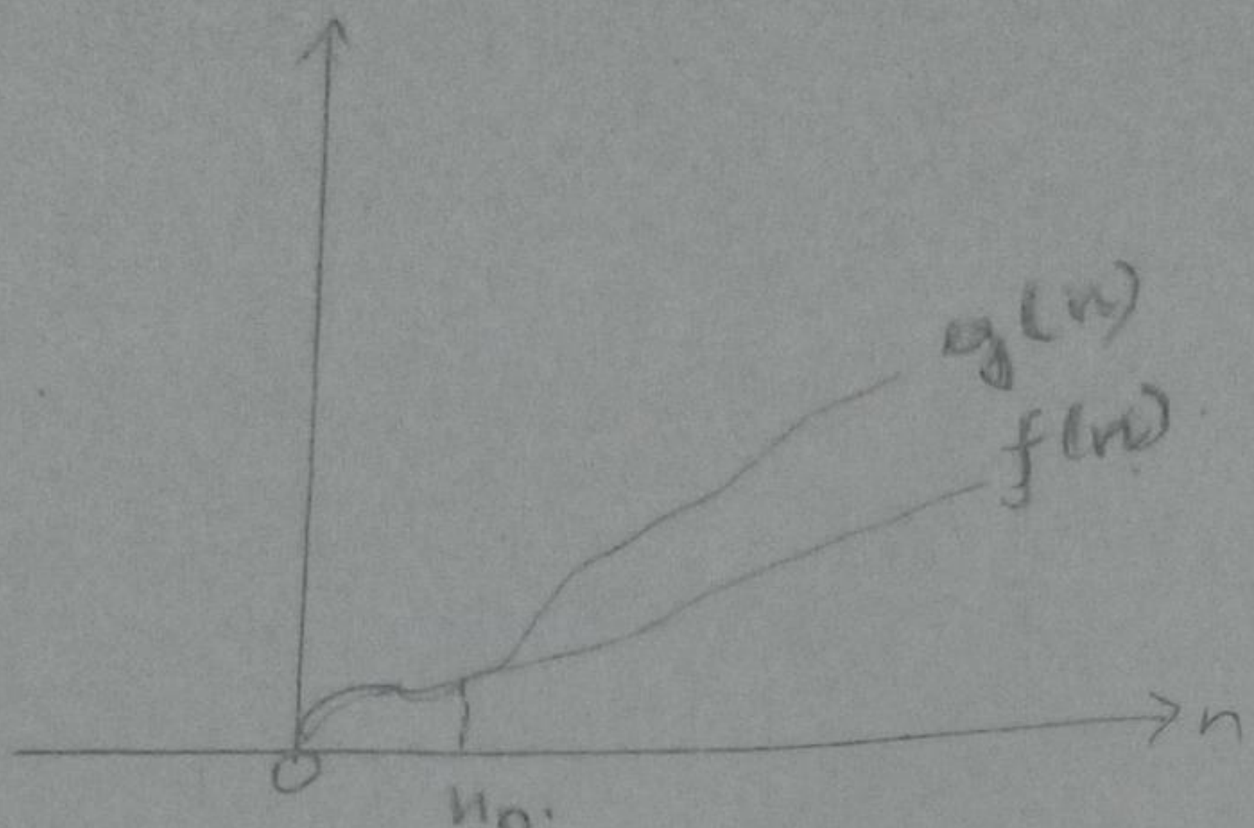
* It is denoted by O. This methods is used for representing the "upper bound" of algorithms runnings time.

* That is we give longest amount of time taken by the algorithm Completely. Let $f(n)$ and $g(n)$ be the two non-negative functions

$f(n) = Og(n)$ is true (i.e $f(n) \in O(g(n))$

iff $f(n) \leq C * g(n)$,

for some values (> 0 $n \geq n_0 \geq 1$).



Eg.

$f(n) = 3n+2$

$g(n) = n.$

if $\rightarrow c = 4.$

Then $f(n) \leq c * g(n).$

$3n+2 \leq c*n$

$3n+2 \leq 4n$

$2 \leq 4n-3n$

$2 \leq n$

$\therefore n \geq 2$

Hence we conclude that for n greater than or equal to 2, we get

$f(n) \leq g(n).$

Thus we proved $f(n) = O(g(n))$ is true (or) $f(n) \in O(g(n))$

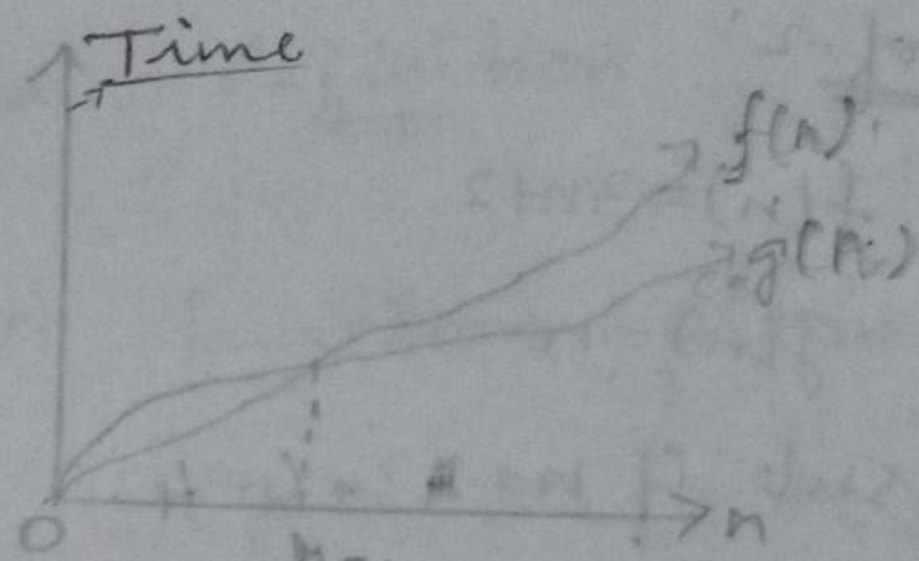
ii) Big Omega (Ω) - Lowest Bound

* Omega notations is denoted by "Ω".

*. This Notation is used to represent "Lower Bound" of algorithm running time.

* Omega notation is used for shortest amount of time taken by the algorithm to complete.

Graphical Representation:



Formula definition:

Let $f(n)$ and $g(n)$ be the two non-negative functions

$f(n) = \Omega(g(n))$ is true (i.e $f(n) \in (\Omega g(n))$

iff $f(n) \geq C * g(n)$

for some values

$n, \geq n_0 \geq 1$

$C > 0$

Example:

To prove: $f(n) \geq c * g(n)$.

Consider $f(n) = 3n+2$

$g(n) = n$.

if $c = 4$, then

Substitude in $f(n) \geq c * g(n)$

$3n+2 \geq 4*n$

$2 \geq 4n-3n$

$2 \geq n$

$\therefore n \leq 2$

Proof-2:

$f(n) = 3n+2$

$g(n) = n$

sub if $n = 1$, $c = 4$.

$3(1)+2 \geq 4*1$

$3+2 \geq 4$

$5 \geq 4$.

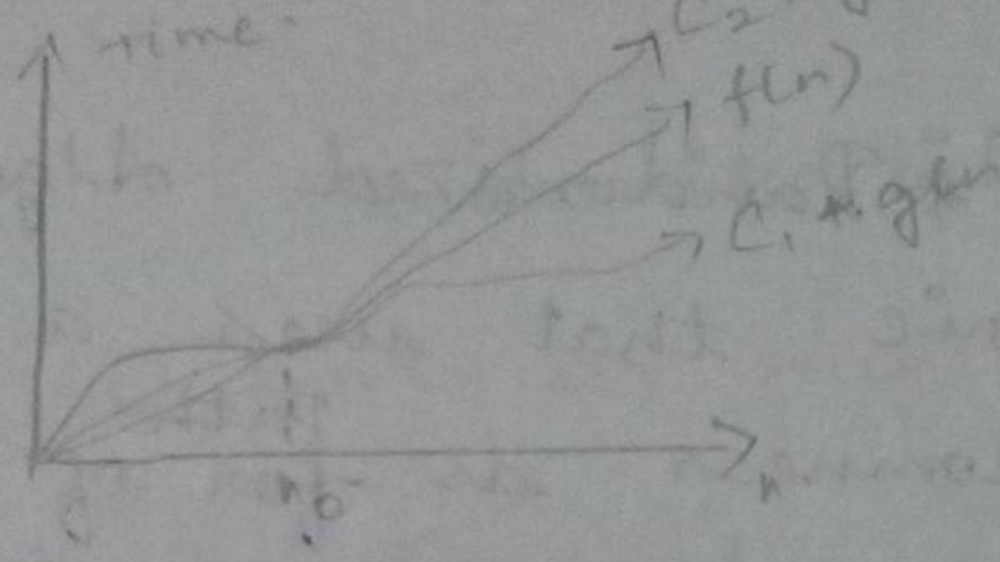
Thus we proved $f(n) \in (\Omega g(n))$.

iii) Big theta (Θ).

* Theta(Θ) is notations is denoted by "Θ"

* This method is used to represent the running time is between upper bound and Lower bound (Average).

Graphical Representation:



Formula definition:

A function $f(n)$ is said to be in $f(n) = \Theta(g(n))$ or $f(n) \in \Theta(g(n))$ is true, iff $f(n) \in \Theta(g(n))$.

iff $C_1 * g(n) \leq f(n) \leq C_2 * g(n)$.

To prove : for upper bound.

if $C = 4$, $n = 4$.

$f(n) = 3n + 2$

$g(n) = n$

$14 \leq 16$ is true

For Lower bound:

$c = 2, n = 4$ $f(n) = 3n + 2$

$1 + 4 \geq 8$ is true: $g(n) = n$

Th

Randomized Algorithm

* Randomized Algorithm is a technic that uses a source of randomness as path for of its logic

* It is used to reduces either time complexity or Space complexity

*. The Algorithm works by generating a random number r, with in a specified range of numbers and make decision based on r's value.

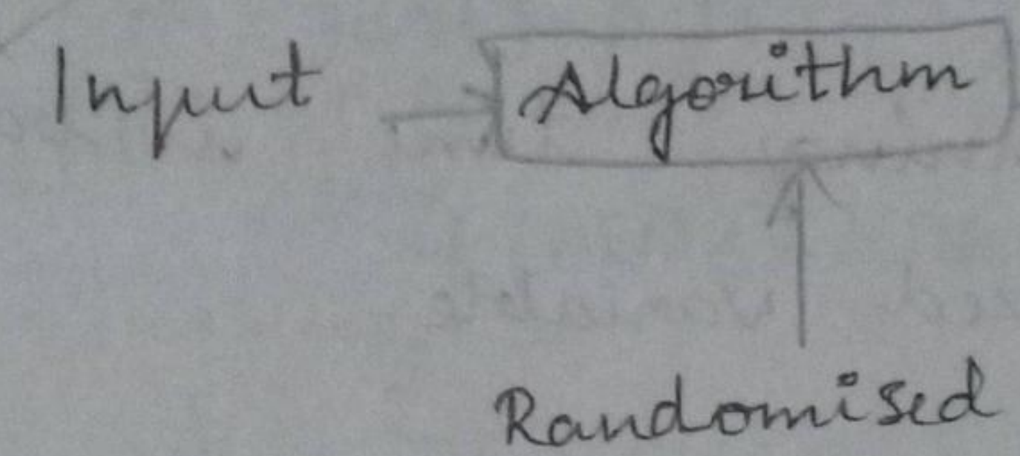
Deterministic Algorithm:

Diagram:

Input → [Algorithm] → Output

Deterministic
it take more
to every input
checking to
is not

* The Output or running time are the functions of input variable only.



* The Output or running times are the functions of input and Randomised bits.

Advantages:

* It works simpler and efficient than Deterministic Algorithm.

Types of Randomised Algorithm:

1. Las Vegas algorithm,
2. Monte Carlo algorithm.

i) Las Vegas Algorithm:

This type of algorithm always produces a correct result or simply

doesn't find anything

* It works on the upper bound Scenario (worst case).

* Running time is based on randomized variable.

Eg

Randomized quick sort.

ii) Monte Carlo Algorithm:

* It is less efficient than Las Vegas algorithm.

* The Output may be incorrect in some probability.

* Running is time is deterministic

Eg

Algorithm for find it approximate median.

Example for Randomized quick Sort:

```
Partition(A, P, r)
{
    x = A[r]
    i = P-1
    for(j = p to r-1)
    {
        if(A[j] ≤ x)
        {
            i = i+1
            exchange A[i] with A[j]
        }
    }
    exchange A[i+1] with A[r]
    return i+1
}
```

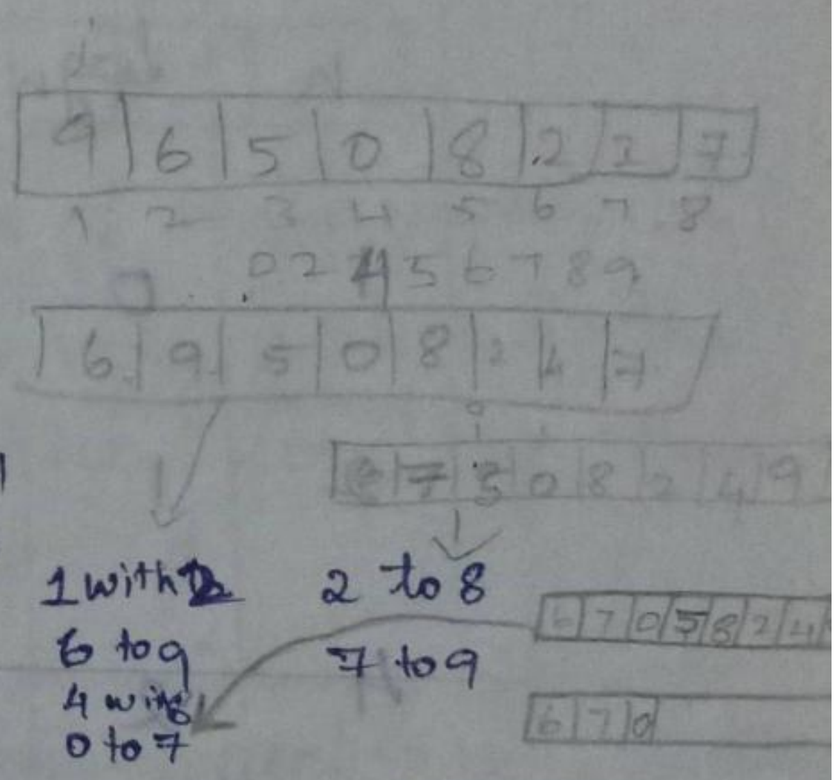
Step-1 : $x = A[8], x = 7$

$i = 0$

for $j = p$ to $r-1$ (1 to 7)

if $A[j] \le x$

A[8]	i = 0	1 to 7	$A[1] \le 7 \Rightarrow 9 \le 7$	$i = i+1$ (0+1) $i = 1$	1 with 2	2 to 8
=7					6 to 9	7 to 9
		At	$A[4] \le 8 \Rightarrow 0 \le 7$	$i = 2$	4 with 1	
					0 to 7	



Quick Sort is a divide and conquer algorithm. It is picks an elements as pivot and partion the iven array around the picked pivot.

Some of quick sort algorithm picks.

i) First element as pivot.
ii) Second element as Pivot.
iii) Median element as Pivot.
iv) Random element as pivot.

Quick Sort Algorithm:

Partition (A, P, r) → pivot (last element)
↳ array → start Position

{
  x = A[r]
  i = P - 1
  for (j = P to r - 1)
  { if (A[j] ≤ x)
    {
      i = i + 1
      exchange A[i] with A[j]
    }
  }
  exchange A[i+1] with A[r]
  return i+1
}

A[3] =

1 (P)	2	3 (r)
5	3	2

A[1] A[2] A[3]

Step 1:

x = A[r] = A[3]

∴ x = 2

i = P - 1 (1-1)

∴ i = 0

for (j = 1(P) to r-1 (3-1)

(j = 1 to 2)

if (A[1] ≤ x

5 ≤ x [false]

{ no execution }

exchang - - - - X

retu - - - X

}}

Step 2:

→ j will be Incremented.

→ x = 2, i = 0.

for (j = 2 to 2)

if (A[2] ≤ x)

3 ≤ 2 [false]

{ no execution }

exchange - - - X

return - - - X

}}

Step 3:

→ j will be incremented.

→ x = 2, i = 0.

for (j = 3 to 2) false.

if (X)

{ no execution. }

exchange [A[i+1] with A[r]

A[1] with A[3]

A[1]		A[3]
2	3	5

Sorted

நட்பில் அகரமாவோம்...
To be Prime in Friendship...

நலப்பணியில் சிகரமாவோம்...
To be Ultimate in Service...

Order of growth:

Measuring the performance of an algorithm in relation with the input size n is called order of growth

for, eg. The order of growth for varying input size of n is as given below.

n	$\log n$	$n \log n$	n^2	2^n
1	0	0	1	2
2	1	2	4	4
4	2	8	16	16
8	3	24	64	256
16	4	64	256	65536
32	5	160	1024	A294, 967, 296.