Unit-1

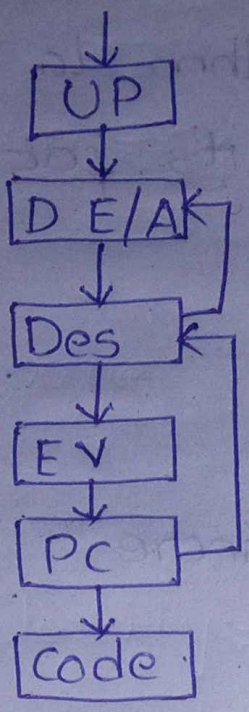
(1) Write an algorithm to read a number and find it's factors.

```
Alg factor(n,i)
{
  Read n value
  for i=1 to n step incre
  {
    if(n%i==0)
      print i value
  }
}
```

Design of an algorithm:

Designing of an algorithm depends on sequential steps where every step has to receive information from the previous step.

* Understanding the problem statement
* Deciding whether it is exactly or approximately solvable.
* Design
* Evaluation metrics
* Proving the correctness
* Coding



(1) Write an algorithm to print first n natural numbers.

```
Alg naturalnumbers(n,i)  →it will be scanned 1 time
{                        ——————→ 1 time
  Read n value           ——→ 1 time
  for i=1 to n step incre →n+1 times
  {                      ——————→ 1 time
    print i value        ——→ n times
  }                      ——————→ n times
}                        ——————→ 1 time
                                 4n+5 times
```

Order of $4n+5 = 1$

$\therefore$ Time complexity $= O(n)$

* If the polynomial is $5n^2+2n+3$ then the Time Complexity $= O(n^2)$.

Note;

* for(initialization; cond; incre/decre){....}
* while(condition){....}
* do{...}while();
* We use semicolon(;) in dowhile for terminating the statement.

(2) Write a sample code to calculate summation of 2D array elements.

(3) Write a sample code to calculate no. of zeroes in a 1D array.

(2)

~~Alg 2D array sum(arr1, arr2, i, j)~~

Alg summation(i, j, sum, r, c) ⟶ 1

{ ⟶ 1

Read r value ⟶ 1

Read c value ⟶ 1

Declare arr[r][c] ⟶ 1

for i=0 to r step Incre ⟶ r+1 (n+1)

{

for j=0 to c step Inc ⟶ r×c (n^2)

{

Read array element a_{ij} ⟶ r×c (n^2)

}

}

Initialize sum=0 ⟶ 1

for i=0 to r step Inc ⟶ r+1 (n+1)

{

for j=0 to c step Inc ⟶ n^2

```
    {
      sum = sum + a_ij  ------> n^2
    }
  }
  print sum value  ------> 1
}
                          4n^2+2n+9
```

∴ Time Complexity = $O(n^2)$

(3)

```
Alg NumOfZeroes(arr,n,i,count)  ------> 1
{                               ------> 1
  Read n value                  ------> 1
  Declare arr[n]                ------> 1
  Initialize count=0            ------> 1
  for i=0 to n step Inc         ------> n+1
  {                             ------> n
    Read arr[i]                 ------> n
  }                             ------> n
  for i=0 to n step Inc         ------> n+1
  {                             ------> n
    if arr[i]==0                ------> n
    {                           ------> n
      count+=1
    }                           ------> n
  }                             ------> n
  print count value             ------> 1
}                               ------> 1
                                10n+9
```

∴ Time Complexity = $O(n)$

Specification of Algorithms:

An algorithm can be specified by using -

- natural language
- flowcharts
- pseudo code

Pseudo Code(False code):

Pseudo code is intermediate form of code of an algorithm. While writing pseudo code we need to follow certain steps -

(1) comment lines - //

(2) Block of code - {
...
}

(3) Termination - ;

(4) for loop - for i:=1 to n step Inc/dec

(5) while loop - while(cond)
{
...
}

(6) until Rep - Rep
{
...
}
until(cond);

(7) Assignment - := (or) ←

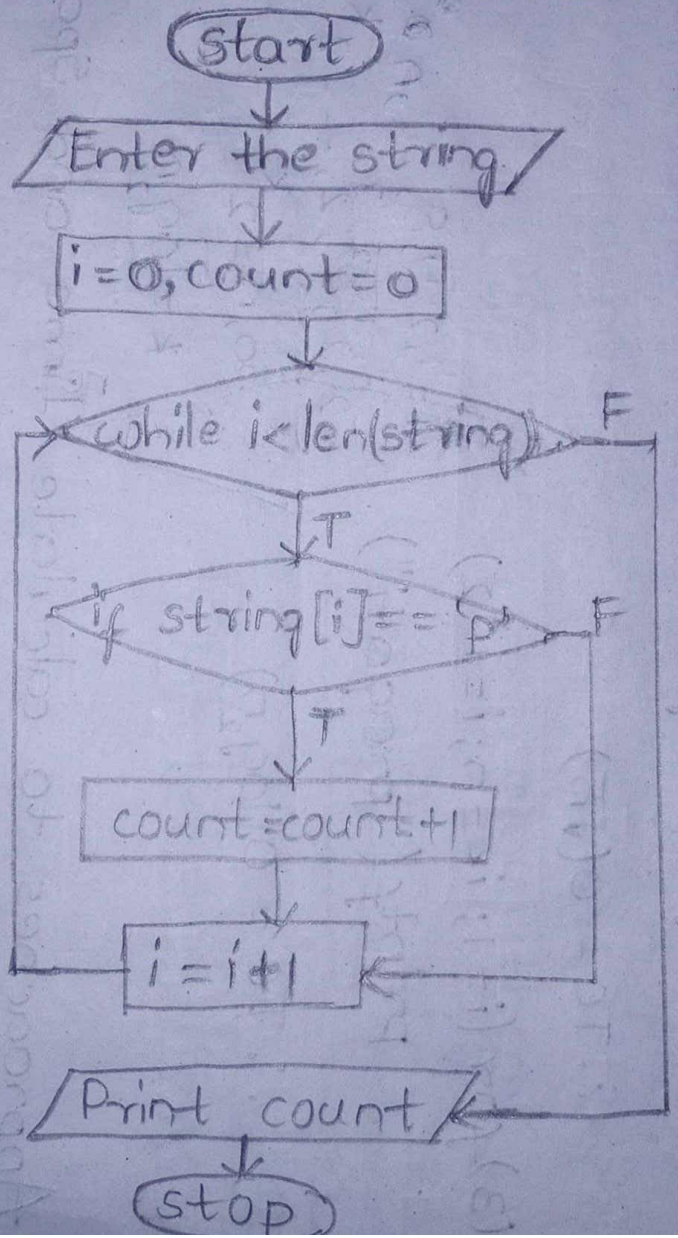
(8) Operators (Unary, Binary, Ternary)

Q Write an algorithm, flowchart and pseudo code to identify no. of p's in a given string.

Algorithm

```
Alg search(str,count, i, len,
           chr)
{
 Read str //using gets
 Read chr
 Initialize count=0
 len = strlen(str)
 for i=0 to len step Inc
 {
  if str[i] == chr
  {
   count+=1;
  }
 }
 print count value
}
```

Flow Chart



Pseudo Code

```
//string search
Alg search(str,count,i,len)
{
 gets(str);
 len:=strlen(str);
 count:=0
 for i:=0 to len-1 step Inc
 {
  if str[i]=='p'
  {
   count+=1;
  }
 }print count
}
```

Identify time complexity for below code sequences:

(1) for (i=1; i<n; i++) ⟶ n+1
{ printf("pragati"); ⟶ n

2n+1

∴ TC = O(n)

(2) for (i=1; i²<=n; i++) ⟶ i<=√n
printf("pragati")

∴ TC = O(√n)

(3) for (i=1; i<=n; i=i*2)
printf("pragati")

i = 1, 2, 4, 8, ---, n
$2^0, 2^1, 2^2, 2^3, \dots 2^k$
So, $n = 2^k$
$k = \log_2 n$

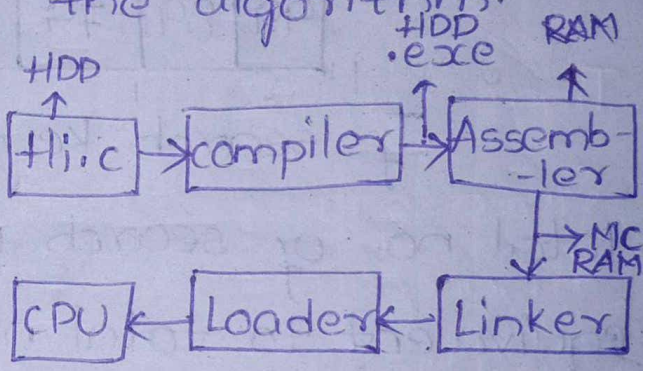
∴ TC = $O(\log_2 n)$

Approaches to calculate Time and Space complexity:

Analysis of time and space complexity will be done in two ways i.e. before implementation of algorithm or code (Priorier Analysis) and after implement-ation or execution of the code (Posterior Analysis).

Differences b/w prior and posterior analysis:

Prior	Posterior
* It is an inaccurate po procedure.	* It is an accurate procedure.
* It is cheaper than posterior analysis.	* It is costlier (due to H/w, s/w, N/w support).
* Maintenance phase is not required to tune the algorithm.	* Maintenance phase is required to tune the algorithm.



Measures of complexity analysis:

Algorithm analysis can be done in three different ways-

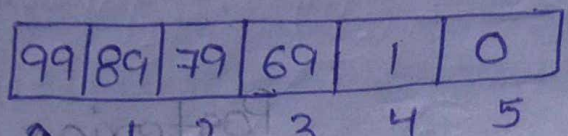
- Best case analysis
- Average case analysis
- Worst case analysis.

Eg: linear, binary searches.

Linear Search:

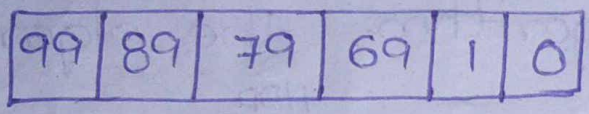
Linear search Time complexity analysis can be done in three ways-

* Best case Time complexity:



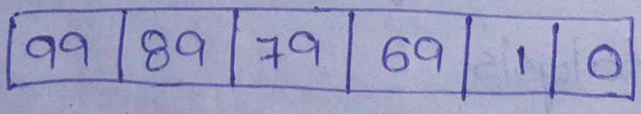
If key or search element is 99 then total no. of search procedures required is 1. So, time complexity can be given interms of constant time i.e. O(1).

*Worst case Time complexity:



If search key or element is 0 then total no. of search procedures are equivalent to total no. of elements in the input array i.e. n. So, the TC can be given as O(n).

* Average case Time complexity:



Average case analysis always lies in b/w best & worst case TC values. If search key is 79 then ~~the~~ total no. of search procedures will be $\frac{n}{2}$. So, the TC can be given as $O(\frac{n}{2})$.

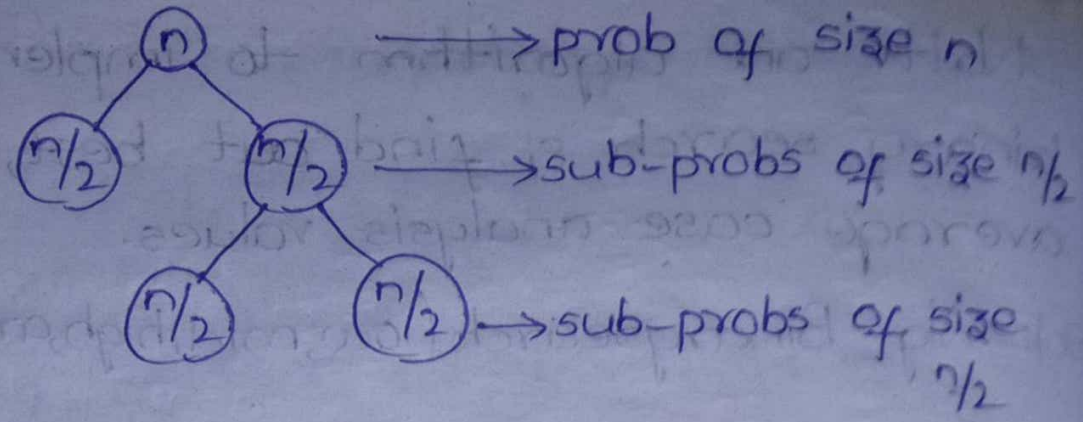
Write an algorithm to implement binary search & find out best, worst & average case analysis values.

Sol: Alg BinarySearch(low, mid, high, arr, i, n, key).

```
{
  arr ← sorted array
  n ← size of array
  key ← value to be found
  low = 1
  high = n
  mid = (low + high)/2
  while(low <= high)
  {
    mid = (low + high)/2
    if (arr[mid] == key)
      print mid
      break
    else if (arr[mid] < key)
      low = mid + 1
    else
      high = mid - 1
  }
}
```

* Best case time complexity is at mid point.
  $= O(1)$.

* Average and worst cases:



After k iterations, length of array = 1

$$\frac{n}{2^k} = 1$$

$$n = 2^k$$

$$k = \log_2 n$$

∴ Time complexity = $O(\log_2 n)$.

Time Complexity Analysis of Binary Search:

Given array elements are 8,6,9,14,1,0.

Arrange the elements in ascending order.

(sort):

L ↑					H ↑
0	1	6	8	9	14
0	1	2	3	4	5

Steps:

Case-1: key = 6

mid = (L+H)/2 = 0+5/2

mid = 2

a[mid]	key
6	6

6 == 6

Case-2: key = 14

mid = (L+H)/2 = (0+5)/2 = 2

a[mid]	key
a[2]	14

$6 < 14$

Then, we have to do $L = mid + 1$

$= 2 + 1 = 3$

Therefore,

0	1	6	8	9	14
0	1	2	3	4	5
			↓		↓
			L		H

$mid = (L+H)/2 = (3+5)/2 = 4$

a[mid]	key
a[4]	14

$9 < 14$

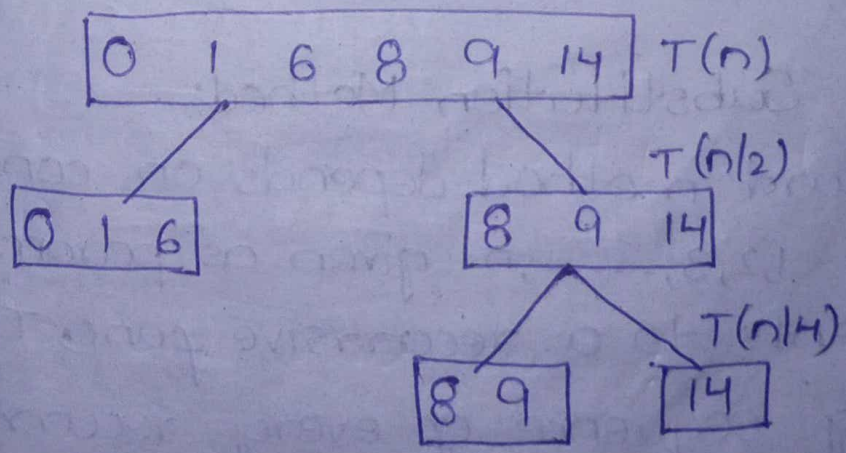
Then, $L = mid + 1 = 4 + 1 = 5$

0	1	6	8	9	14
0	1	2	3	4	5
					↓
					L, H

$mid = (L+H)/2 = (5+5)/2 = 5$

a[mid]	key
a[5]	14

$14 == 14$



$T(n) = T(n/2) + 1$.

$T(n/2)$ is time complexity or total time taken to implement such operations on sublist.

(i) Indicates the unit of time required to divide list into sublists.

Recurrence Relation:

Recurrence relation is an eqn which is used to solve given task with the help of recurrsive functions.

General format of recurrence relation can be given as:

$$T(n) = T(n-1) + n.$$

with an initial condition $T(1)$ or $T(0) = 0$. (or) 1.

Solution of RR:

We can solve RR with the help of two methods

(i) Forward or Backward method

(ii) Master's method.

* Forward Substitution Method:

Forward method depends on constant values like $1, 2, 3, \ldots, n$ given as parameters or arguments to a recurrsive func, $T(n)$ by identifying sequence of every recurrsive func. outputs. We need to obtain or findout a formula F.

Q. Find Time complexity of RR $T(n)=T(n-1)+n$ with an initial condition $T(0)=0$.

Ans: Given RR $T(n) = T(n-1)+n$

$T(0) = 0$

$T(1) = T(n-1)+n$

$= T(0)+1$

$= 0+1 = 1$

$T(2) = T(1)+2 = 1+2=3$

$T(3) = T(2)+3 = 3+3=6$

∴ The sequence is 1,3,6- - -

Finding formula for output sequence in every case is difficult. So forward substitutio method is not suggestable to find TC of any RR.

Formula: $\frac{n(n+1)}{2} = \frac{n^2}{2}+\frac{n}{2} = \frac{1}{2}n^2 = T(n^2)$

*Backward Substitution Method:

$T(n) = T(n-1)+n$

$T(0) = 0$

$T(n-1) \Rightarrow n = n-1$

$T(n-1) = T(n-1-1)+n-1$

$T(n-1) = T(n-2)+n-1 \longrightarrow ①$

$T(n-2) = T(n-2-1)+n-2$

$= T(n-3)+n-2 \longrightarrow ②$

From ① & ②

$$T(n) = (T(n-2)+n-1)+n$$

$$= T(n-2)+2n-1$$

$$= (T(n-3)+n-2)+2n-1$$

$$= T(n-3)+3n-3$$

If $k=3$ then,

$$T(n) = T(n-k)+kn-k$$

If $k=n$ then,

$$T(n) = T(n-n)+n*n-n$$

$$= T(0)+n^2-n$$

$$= 0+n^2-n$$

$$= O(n^2)$$

Q. $T(n) = T(\frac{n}{2})+1$.

Sol: $T(n) = T(\frac{n}{2})+1$

$T(1)=1$ [consider initially]

If $n=\frac{n}{2}$

$T(\frac{n}{2}) = T(\frac{n}{4})+1 \longrightarrow ①$

If $n=\frac{n}{4}$, $T(\frac{n}{4}) = T(\frac{n}{8})+1 \longrightarrow ②$

If $n=\frac{n}{8}$, $T(\frac{n}{8}) = T(\frac{n}{16})+1 \longrightarrow ③$

From the given relation, & by considering eq^n^s ①, ②, ③

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

$= \left(T\left(\frac{n}{4}\right)+1\right)+1$

$= T\left(\frac{n}{4}\right)+2$

$= \left(T\left(\frac{n}{8}\right)+1\right)+2$

$= T\left(\frac{n}{8}\right)+3$

$T(n) = T\left(\frac{n}{2^3}\right)+3$

If $k=3 \Rightarrow T\left(\frac{n}{2^k}\right)+k$

If $n=2^k \Rightarrow T\left(\frac{n}{n}\right)+k$

$T(n) = T(1)+\log_2 n$

$T(n) = 1+\log_2 n$

$\therefore T(n) = O(\log_2 n)$

Q. Identify TC of given RR.

(i) $T(n)=T(n-1)+5$ (ii) $T(n)=2T\left(\frac{n}{2}\right)+n$

Sol: (i) $T(n)=T(n-1)+5$

If $n=n-1 \Rightarrow T(n-1)=T(n-2)+5$

If $n=n-2 \Rightarrow T(n-2)=T(n-3)+5$

If $n=n-3 \Rightarrow T(n-3)=T(n-4)+5$

Now, $T(n)=T(n-1)+5$

$= (T(n-2)+5)+5$

$= T(n-2)+10$

$= (T(n-3)+5)+10$

$= T(n-3)+15$

$= (T(n-4)+5)+15$

$T(n) = T(n-4)+20$

If $k=4 \Rightarrow T(n) = T(n-k)+5k$

If $k=n$ then,

$T(n) = T(n-n)+5n$

$= T(0)+5n$

$= 5n$

$\therefore T(n) = O(n)$

(ii) $T(n) = 2T(\frac{n}{2})+n$

If $n=\frac{n}{2} \Rightarrow T(\frac{n}{2}) = 2T(\frac{n}{4})+\frac{n}{2}$

If $n=\frac{n}{4} \Rightarrow T(\frac{n}{4}) = 2T(\frac{n}{8})+\frac{n}{4}$

If $n=\frac{n}{8} \Rightarrow T(\frac{n}{8}) = 2T(\frac{n}{16})+\frac{n}{8}$

Now, $T(n) = 2T(\frac{n}{2})+n$

$= 2(2T(\frac{n}{4})+\frac{n}{2})+n$

$= 4T(\frac{n}{4})+2\frac{n}{2}+n \quad = 4T(\frac{n}{4})+2n$

~~$= 4T(2T(\frac{n}{8}))+n$~~ $= 4(2T(\frac{n}{8})+\frac{n}{4})+2n$

$= 8T(\frac{n}{8})+3n$

$= 8(2T(\frac{n}{16})+\frac{n}{8})+3n$

$= 16T(\frac{n}{16})+4n$

If $k=4 \Rightarrow T(n) = 2^k T(\frac{n}{2^k})+kn$

Let $n=2^k \Rightarrow nT(\frac{n}{n})+n\log_2 n$

$T(n) = nT(1)+n\log_2 n$

$\therefore T(n) = O(n\log_2 n)$

Q. Write code snippet to find out factorial of a given number using sequential and recursive approach.

Sol: Recursive approach:

```
int fact(int n)
{
  if n==0
    fact=1;
  else
    for(i=n;i>=1;i--)
      fact = n*fact(n-1);
  return fact;
}
```

Iterative approach:

Sol

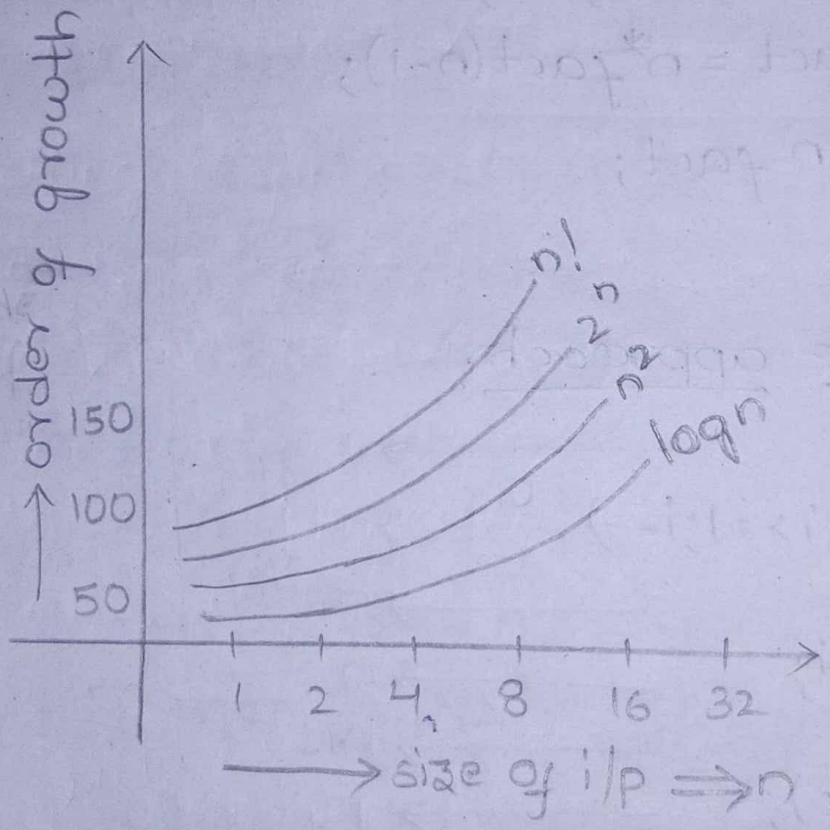
```
f=1;                          ----> 1
for(i=n;i>=1;i--)             ----> n+1
{                             ----> n
  f=f*i;                      ----> n
}                             ----> n
printf(f);                    ----> 1
```

4n+3

∴ Time Complexity = O(n)

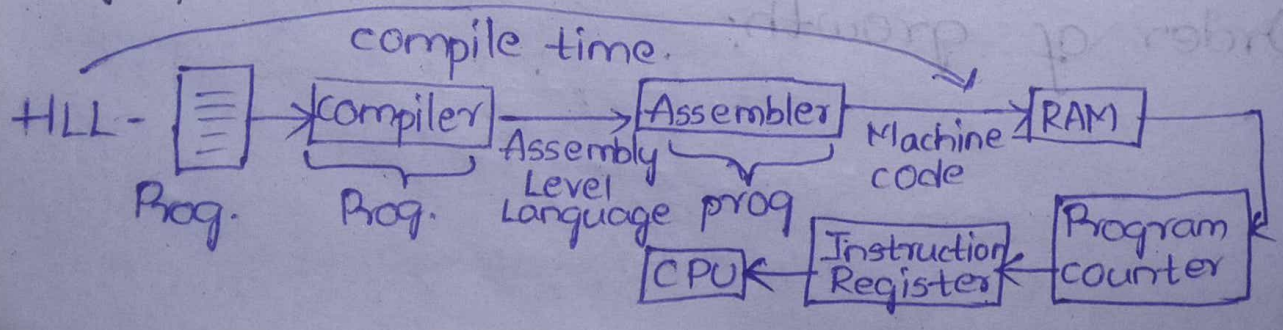
Order of n:

n	$\log n$	n^2	2^n	$n!$
1	0	1	2	1
2	0.3010	4	4	2
4	0.6020	16	16	24
8	0.9030	64	256	40,320
16	1.2041	256	65,536	$20922789 89 \times 10^4$
32	1.5051	1024	4,294,967,296	$263130836 9 \times 10^{26}$

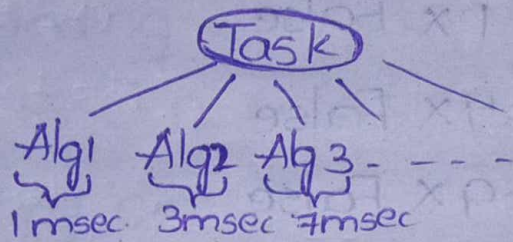


Based on i/p size n, the functions TC varies from 0 to n.

Asymptotic Notations:



- Asymptotic notations are used to identify runtime of an algorithm.
- We have 3 basic notations -
  1. Big-O notation (↑ upper bound)
  2. Big-Ω notation (↓ lower bound)
  3. Big-Θ notation (← → avg.).



Asymptotic notations are used to find the runtime of each & every algorithm & solution as metioned in the above diagram.

Big-O notation:

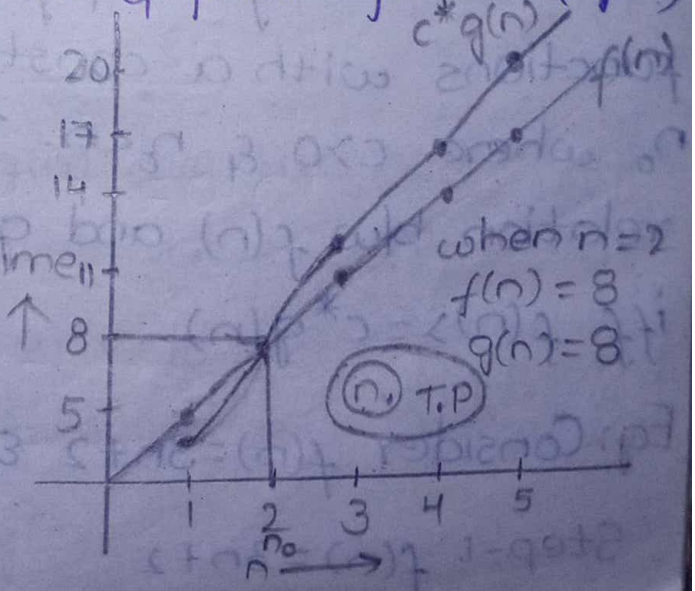
Consider $f(n)$ & $g(n)$ are two non-neg. functions, with a constant values c & n_0 where $c > 0$ & $n_0 < n$. Then we can give a relation b/w $f(n)$ & $g(n)$ as: $f(n) = O(g(n))$.

iff $f(n) <= c^* g(n)$.

Q. Consider $f(n) = 3n+2$ & prove $f(n) = O(g(n))$

Step-1: $f(n) = 3n+2$

If $n=1$, $f(n) = 5$
If $n=2$, $f(n) = 8$
If $n=3$, $f(n) = 11$
If $n=4$, $f(n) = 14$
If $n=5$, $f(n) = 17$



Step-2: Finding c (constant):

As per the Big-O notation, use below condition to identify c.

$f(n) <= c * g(n)$

$3n+2 <= c * n$

If n=1, c=1, $5 <= 1$ × False

If n=2, c=2, $8 <= 4$ × False

If n=3, c=3, $11 <= 9$ × False

If n=4, c=4, $14 <= 16$ ✓ True

Calculating $c * g(n)$:

If n=1, $c * g(n) = 4 \times 1 = 4$

If n=2, $c * g(n) = 4 \times 2 = 8$

If n=3, $c * g(n) = 4 \times 3 = 12$

If n=4, $c * g(n) = 4 \times 4 = 16$

If n=5, $c * g(n) = 4 \times 5 = 20$

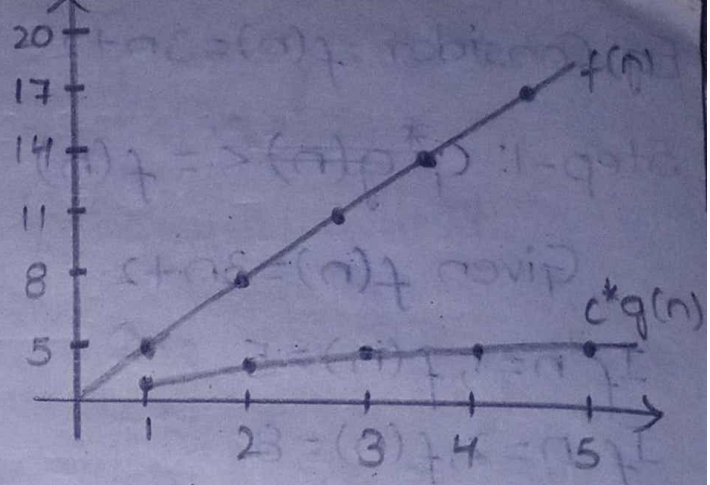
Big-Ω notation:

Consider $f(n)$ & $g(n)$ are two non-negative functions with a constant value c and n_0 where $c > 0$ & $n_0 < n$. Then we give a relation b/w $f(n)$ and $g(n)$ as: $f(n) = \Omega(g(n))$ iff $f(n) >= c * g(n)$.

Eg: Consider $f(n) = 3n+2$ & prove $f(n) = \Omega(g(n))$

Step-1: $f(n) = 3n+2$

If n=1, f(n) = 5
If n=2, f(n) = 8
If n=3, f(n) = 11
If n=4, f(n) = 14
If n=5, f(n) = 17



Step-2: Finding c (constant):

As per big- Ω notation, use below condition to identify c-

$$f(n) >= c * g(n)$$

$$3n+2 >= c * n.$$

If n=1, c=1, $5 >= 1$ ✓ True

So, c=1 [c ranges from 1 to 3].

Calculating c*g(n):

If n=1, c*g(n) = 1X1 = 1
If n=2, c*g(n) = 1X2 = 2
If n=3, c*g(n) = 1X3 = 3
If n=4, c*g(n) = 1X4 = 4
If n=5, c*g(n) = 1X5 = 5

Big-Θ notation:

Consider f(n) & g(n) are two non-negative functions with constant values c_1, c_2 & n_0 where $c_1, c_2 > 0$ & $n_0 < n$. Then we give a relation b/w f(n) & g(n) as: $f(n) = \Theta(g(n))$ iff $c_1 * g(n) <= f(n) <= c_2 * g(n)$.

Eg: Consider $f(n)=3n+2$ & prove $f(n)=\Theta(g(n))$

Step-1: $c_1 * g(n) <= f(n)$

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

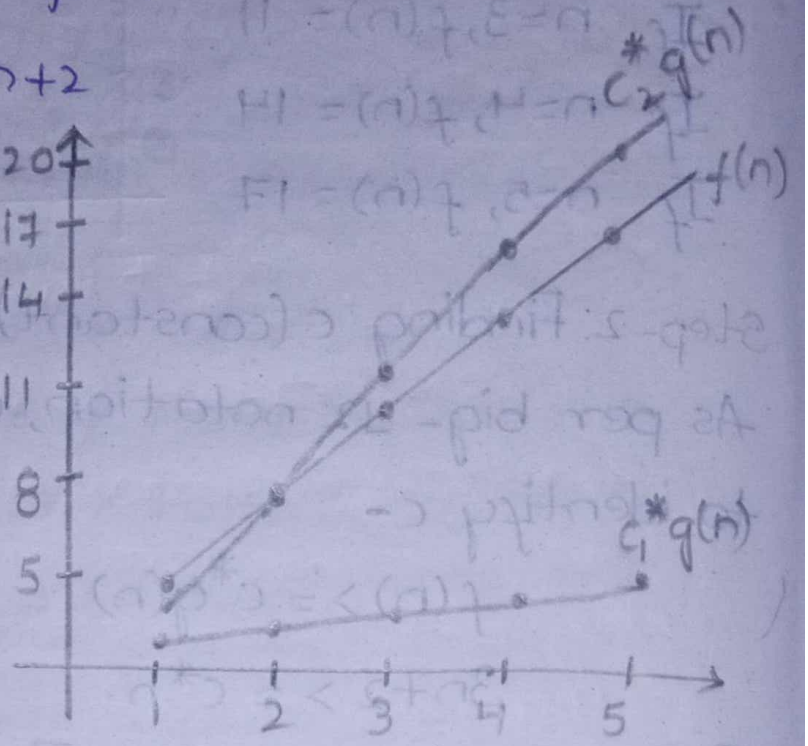
If $n=1, f(n)=5$

If $n=2, f(n)=8$

If $n=3, f(n)=11$

If $n=4, f(n)=14$

If $n=5, f(n)=17$



Find c_1:

$f(n) >= c_1 * g(n)$

If $n=1, c_1=1$, $5 >= 1$ ✓ True

If $n=2, c_1=2$, $8 >= 4$ ✓ True

If $n=3, c_1=3$, $11 >= 9$ ✓ True

If $n=4, c_1=4$, $14 >= 16$ ✗ False

So, c_1 ranges from 1 to 3.

& Let $c_1=1$

If $n=1, c_1 * g(n) = 1 \times 1 = 1$

If $n=2$, $=2$

If $n=3$, $=3$

If $n=4$, $=4$

If $n=5$, $=5$

Similarly $c_2=4$ from Big-O notation.