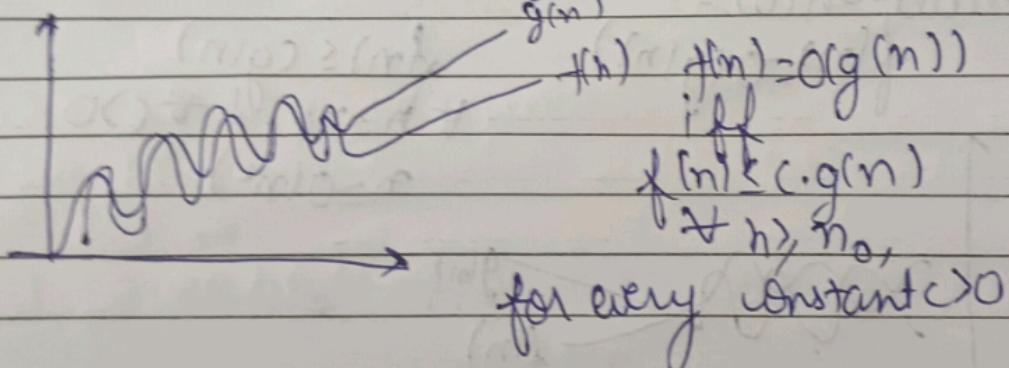


ASSIGNMENT NO:- 01

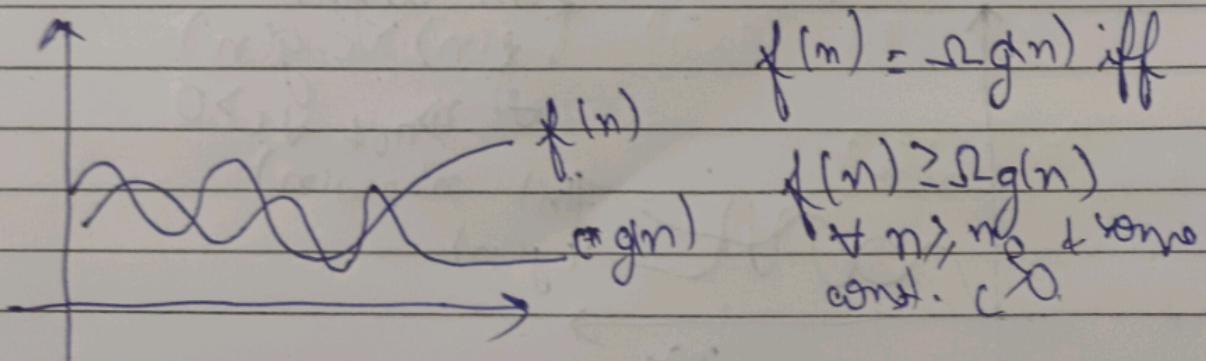
Q1.) Asymptotic Notation

- These notations are used to tell the complexity of an algorithm when the input is very large.
- It describes the algorithm efficiency / performance in a meaningful way. It describes the behaviour of time or space complexity for large instance characteristics (5 types)

(i) Big Oh Notation (O) :- The function $f(n) = O(g(n))$, iff there exists a $\text{tve constant } c \in \mathbb{R}$ & K such that $f(n) \leq c \cdot g(n)$ for all $n, n \geq K$.

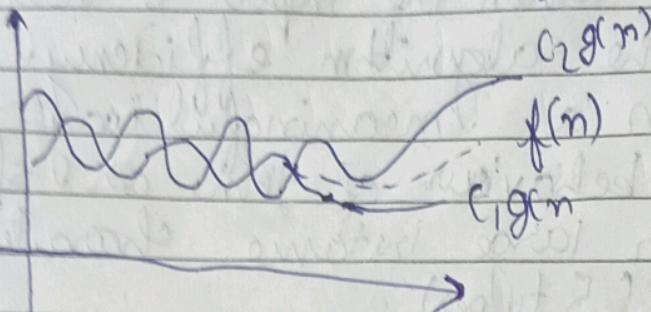


(ii) Big Omega notation (Ω) :- The function $f(n) = \Omega(g(n))$ iff there exists a $\text{tve constant } c \in \mathbb{R}$ & K such that $f(n) \geq c \cdot g(n)$ for all $n, n \geq K$



(iii) Big Theta notation (Θ): The function $f(n) = \Theta(g(n))$, iff there exists a +ve constant $c_1, c_2 \in \mathbb{R}$ such that

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n,$$

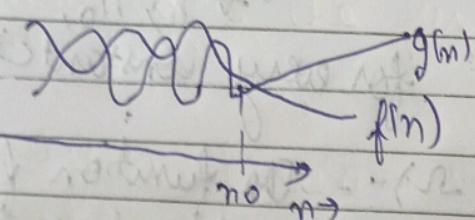


$$f(n) = \Theta(g(n)) \text{ iff } c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall n \in \mathbb{N}, \text{ where } c_1, c_2 > 0$$

(iv) Small-Oh (o):-

$$f(n) = o(g(n)) \quad f(n) \leq c g(n) \quad \forall n \geq n_0 \quad \forall c > 0$$

$$n \geq 0 \quad n^2$$

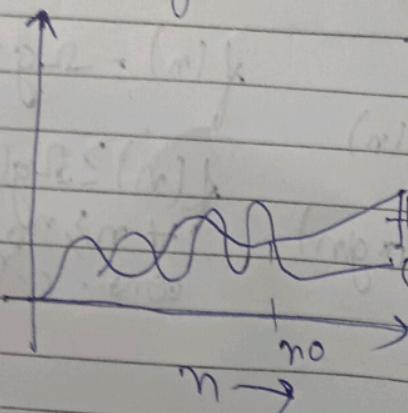


(v) Small-Omega (ω):

$$f(n) = \omega(g(n)) \quad \text{lower bound}$$

$$f(n) \geq c g(n) \quad \forall n \geq n_0 \quad \forall c > 0$$

$$n^2 = \omega(n)$$



2. Time complexity of a loop means no. of times it has run.

i	1	2	4	8	16	32	\dots	$[2^k]$
value	2	2^2	2^3	2^4	2^5	2^6		

$i = 1, 2, 4, 8, 16, 32 \dots i = 2^k$ this means k times
i.e. $2^k = n$

$$k \log_2 2 = \log_2 n \quad [\log_2 2 = 1]$$

$$T.C = O(\log n)$$

$$3. T(n) = \{3T(n-1), n > 0\}$$

By forward substitution

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

$$T(0) = 3T(-1) = 0$$

$$T(1) = 3T(1-1) = 3T(0) = 3$$

$$T(2) = 3T(2-1) = 3T(1) = 3 \times 3 = 3^2$$

$$T(3) = 3T(3-1) = 3T(2) = 3 \times 3^2 = 3^3$$

$$T(n) = 3^n$$

$$\therefore T.C = O(3^n)$$

$$4. T(n) = \begin{cases} 2T(n-1) + 1, & n > 0 \\ \dots & \dots \end{cases}$$

By forward substitution,

$$T(0) = 1$$

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

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

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

$$2^n - (2^{n-1} - 2^{n-2} - 2^{n-3} - \dots - 2^1 - 2^0)$$

$$= 2^n - (2^n - 1)$$

$$= 1$$

$$T.C = O(1)$$

5. The value of 'i' increases by one for each iteration. The value contained in 'i's' at the i^{th} iteration is the sum of the first ' i ' +ve integers. If K is the total no. of iterations

taken by an program then while loop terminate if:

$$1 + 2 + 3 + \dots + K \rightarrow [K(K+1)/2] > n$$

$$\text{So, } K = O(\sqrt{n})$$

$$T.C = O(\sqrt{n})$$

$$6. O(n) = T.C$$

7. void function(int n)

```

int i, j, k, Count=0;
for(i=0/2; i<=n; i++) O(n)
  for(j=i; j<=n; j=j+2) O(log n)
    for(k=i; k<=n; k=k+2) O(log n)
      Count++;
    }
  }
}

```

$$T.C = O(n \log^2 n)$$

8. function(int n)

```

if (n==1)
  return;
for(i=1 to n) O(n)

```

```

  for(j=1 to n) O(n)
    printf("*");
  }
}

```

```

  function(n-1);
}

```

$$[T.C = O(n^2)]$$

9. void function(int n)

```

  for(i=1 to n) O(n)

```

```

    for(j=1; j<=n; j=j+1) O(n)
      printf("*");
    }
  }
}

```

```

}

```

$$[T.C = O(n^2)]$$

10. For the function $n^k c n^m$ what is the asymptotic relationship b/w these functions:-

Assume that $k > 1$ & $c > 1$ are constants.
Find out the value of c & no. for which

relationship

n^k is $O(c^n)$

As c^n is the upper bound