Name:- Suroju Akshay Kumar

Sub:- Design and Analysis of Algorithm

Sec:- AI/ML

Roll no:- 61

TCS 409

Q1. What do you understand by Asympatic notations Define different Asympatic notation with examples

A. Asympatic notations are the mathematical tools which are used to tell the complexity of an algorithm when the input is very large.

$O(n^2)$ = no. of instructions

where n is number of input

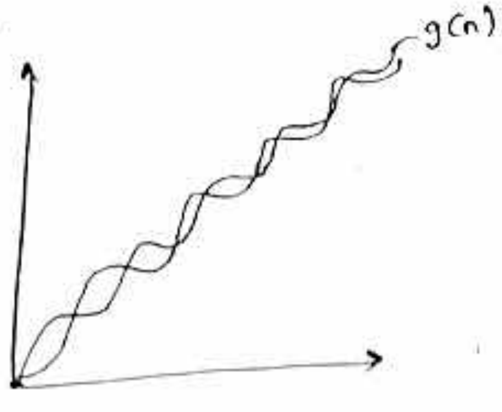
Different Asympatic notations are:-

① <u>Big O notation (O)</u>:-

It describe the upper bound of an algorithm's time complexity in the nearst - case

$f(n) = O(g(n))$

eg:-



$g(n)$ is tight upper bound of $f(n)$

$f(n) = O\,g(n)$

if $f(n) \leq c\,g(n)$

$\forall\ n \geq n_0$

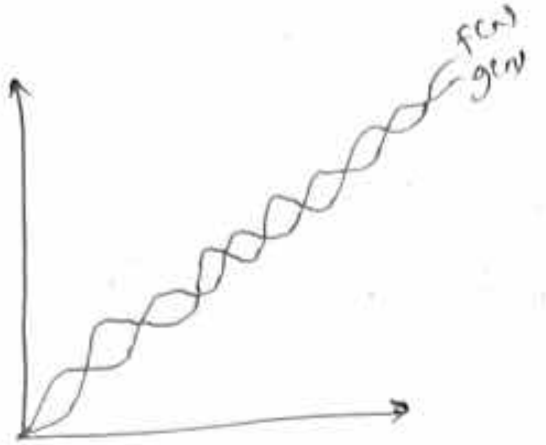
and for some constants $c > 0$

- If the algo has time complexity of $O(n^2)$, it means algo's runtime grows quadratically with the size of input

② Omega notation(Ω):-

Omega notation describe the cover bound of an algorithm's time complexity in the best-case scenario.

$$f(n) = \Omega\, g(n)$$



g(n) is tight upper bound of f(n)

$$f(n) = O\, g(n)$$

$$\text{if } f(n) \geq c\, g(n)$$

$$\forall\ n \geq n_0$$

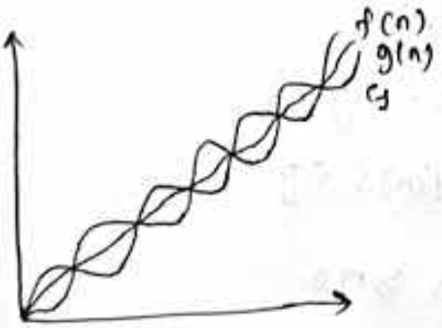
and for some constants $c > 0$

• If the algo has time complexity of $O(n^2)$, it means algo's runtime grows quadractically with the size of input.

③ Theta notation (Θ):-

Theta notation describes both the upper and lower bounds of an algorithm's time complexity, providing a tight bound

$$f(n) = \Theta\, g(n)$$



$f(n) = O\, g(n)$ and

$f(n) = \Omega\, g(n)$

$$c_1\, g(n) \leq f(n) \leq c_2\, g(n)$$

$\forall\ n > \max(n_1, n_2)$ and some constants

$c_1 > 0$ & $c_2 > 0$

4. Small O notation (o):-

It decribes an upper bound on a function that is not tight

$f(n) < c\,g(n)$

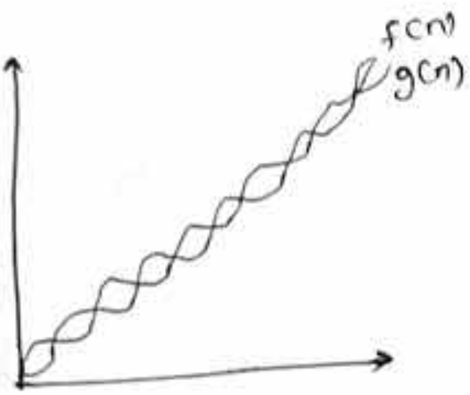
$\forall\ n > n_0$ & for some constant

$c > 0$

g(n) is upper bound of f(n)

5. Small Ω notation (ω):-

It describe lower bound on a function which is not tight



$f(n) > g(n)$

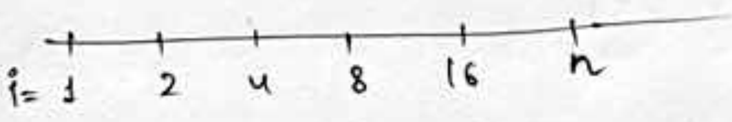
$\forall\ n > n_0$ & for some constant $c > 0$.

g(n) is the lower bound of f(n)

Q2. What should be time complexity of for (i = 1 to n) {i = i * 2}

A.

```
Sum = 0
for (i = 1; i < n; i * = 2)
{
   sum += i
}
```



This is forming a G.P

$n = ar^{k-1}$

where $a = 1$

$r = \frac{4}{2} = 2$

So, $n = 1 \times 2^{k-1}$

$n = 2^{k-1}$

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

$2n = 2^k$

taking log on both sides

$\log_2(2n) = k\log_2 2$

$k = \log_2(2n)$

$\boxed{\log_2(2)} \Rightarrow$ constant

$k = \log_2(2n)$

$k = \log_2 n + 1$

$k = \log_2 n$ (1 is constant)

time complexity = $O(\log_2 n)$

Q3) $T(n) = \{ 3(T(n-1)$ if $n > 0$, otherwise $1)$

$T(0) = 1$

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

for $T(1)$

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

$= 3 \times 1$

A for $T(2)$

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

$= 3T(1)$

$= 3T(0)$

$= 3 * 3 * 1$

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

$= 3T(2)$

$= 3 * 3 * 3 * 1$

for $T(n)$

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

$= 3 * 3 * 3 * \text{---}$ n time $= 3^n$

$$\boxed{T(n) = O(3^n)}$$

Q4) $T(n) = \{ 2T(n-1)$ if $n > 0$, else $1\}$

$T(0) = 1$

A. for $T = 1$.

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

$= 2T(0) - 1 = 2 - 1 = 1$

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

$= 2T(1) - 1$

$= 2(1) - 1$

$= 2 - 1$

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

$= 2T(2) - 1$

$= 2(2-1) - 1$

$= 4 - 2 - 1$

$\vdots$

for $T(n)$

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

$= 2^n - (2n-2) - \cdots - 4 - 2 - 1$

$\boxed{T(n) = O(1)}$

Q5.
```
int i = 1; s = 1;
while (s<n) {
    i++;
    s = s+i;
    print ("#");
}
```

$S_i = S_{i-1} + i$

when $i = 1$, $S_1 = S_0 + i \Rightarrow S = 1.$

When $i = 2$, $S_2 = S_1 + 1 = 1 + 2 \Rightarrow S = 3$

When $i = 3$, $S_3 = S_2 + 3 = 3 + 3 = 6$

$\rightarrow 1 + 3 + 6 + 10 + \cdots\cdots\cdots k = n$

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

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

$$O(k^2) = n$$

$$\boxed{k = \sqrt{2}}$$

Q6. 'void function (int n) {

```
    int i, count = 0;
  for (i=1; i*i <n; i++)
    count ++
  }
```

check $i*i <= n$

( ) i*i should be less than or equal to n

A. when $i=1$, $1*1 \le n \Rightarrow 1 \le n$

$i=2$, $2\times2 = 4 \le n \Rightarrow 4 \le n$

⋮

$i=n$, $\sqrt{n}\times\sqrt{n} = n \le n = ?$

so, the loop will be

$1, 2, 3, \ldots \sqrt{n}$

no. of iteration k is bound by $\sqrt{n}$

so, time complexity is $O(\sqrt{n})$

Q7.
```
void function (int n) {
  Int i, j, k , count=0;
  for (i=n/2; i<=n; i++)
    for (j=1; j<=n; j=j*2)
      for (k=1; k<=n; k=k*2)

        count++
```

A. 1. i iterates from n/2 to n. Its time complexity is O(n)

2. j iterates from 1 to n with a double increment (j=j*2)

Its, time complexity is O(log n)

3. k iterates from 1 to n with a double increment (k=k*2)

its, ~~temp~~ time complexity is O(log n)

$O(n) \times O(\log n) \times O(\log n) = O(n \log^2 n)$

Q8.
```
function (int n) {
        if (n== 1) return;
        for(i = 1 to n) { (n times) } O(n^2)
        for( j= 1 to n) { (n times)
        print ("*");
    }
  }
  function (n-3); T(n-3) times
  }
```

A. The time complexity of both the inner loops is $O(n^2)$

$T(n) = T(n-3) + O(n^2)$

as $T(1) = O(1)$

Thus, $T.C = O(n^2)$

Q9. Time Complexity of

```
Void function (int n) {
    for (i=1 to n) {
        for (j=1; j<=n ; j=j+1)
        printf("*");
    }
}
```

A. for j: $n/1 + n/2 + n/3 + \cdots n/n$

$n = 1 \cdot 2^{k-1} \Rightarrow n = 2^k/2 \Rightarrow 2n = 2^k$

taking log both side

$\log 2n = \log 2^k$

$T.C = O(\log_2 n)$

Thus, the T.C is $O(n \log_2 n)$

Q10. For the functions, n^k and c^n, what is the asymptotic relationship between these function? Assume that $k >= 1$ and $c > 1$ are constant, Find the value of c and no for which relation holds

A. n^k grows polynomially with n

c^n grows Exponentially with n

thus $c^n = n^k$

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

Find the value of C and n

$\log n^k = \log C (c^n)$

$\Rightarrow C \geq e$ and $n_0 = k$.