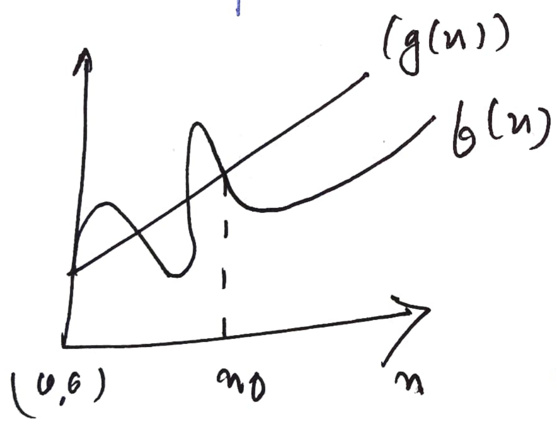
NAME . AKSHAY PATWAL
ROLL NO - 09
SEC . DS

Q1. Asymptomatic Notation : These are language to to express the required time of space by an algorithm to solve a given problem.

(a) Big O Notation : It is notation for the worst case analysis of an algorithm (upper bound). According to it for a two function $f(n)$ & $g(n)$ $f(n) = O(g(n))$ if and only if there exist n_0 & c such that.



$0 \leq f(n) \leq c \times g(n)$ for all $n \geq n_0$

de $n + n^2 = O(n^2)$

Here $f(n) = n + n^2$, $g(n) = n^2$

$n + n^2 \leq n^2 + n^2$ ($\because n < n^2$, $n^2 = n^2$)

$n + n^2 \leq 2n^2$ (here $c = 2$) for $n_0 = 1$

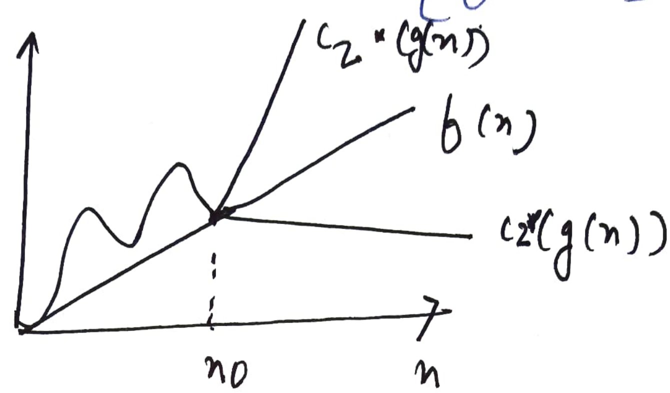
So $f(n) = O(g(n))$

or $n + n^2 = O(n^2)$

(b) Big Theta (Θ) : for any case time complexity (tightly bound)

for any two func $f(n)$ & $g(n)$

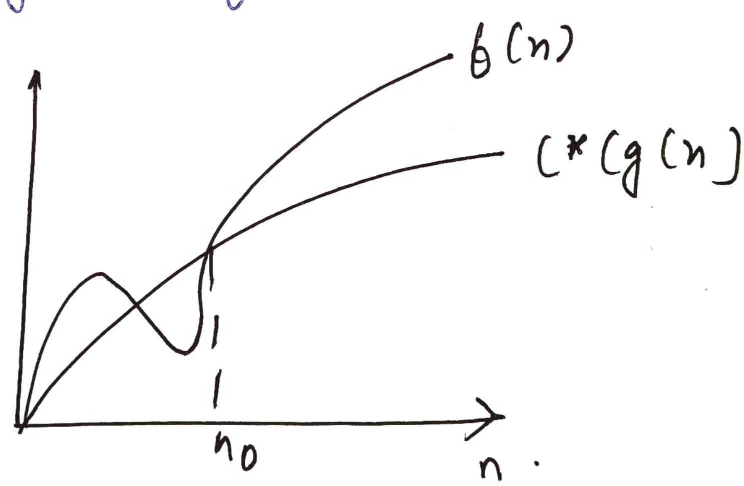
$f(n) = \theta(g(n))$ if and only if there exists n_0, c_1, c_2 such that $0 \le c_2 \cdot g(n) \le f(n) \le c_1 \cdot (g(n))$ [for $n \ge n_0$]



(c) **Big Omega (Ω)** : For best case complexity (lower bound)

$f(n) = \Omega(g(n))$ if $\exists$ n_0 c_1

$\ni$ $0 \le c_1 * g(n) \le f(n)$ $\forall$ $n >= n_0$



Q2. TC of for ($i = 1$ to n) {$i = i * 2$}

Series $\rightarrow$ $1, 2, 4, 8, 16 \; - - - \; n$ (GP)

$a = 1 \qquad r = 2$

$t_K = ar^{K-1} \Rightarrow n = a * 2^{K-1}$

$\Rightarrow n = 2^{K-1}$

$\Rightarrow 2^K = 2n$

$\Rightarrow K = 2 \log_2 n$.

so TC $\Rightarrow O(\log_2 n)$

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

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

let $n = n-1$ $\quad T(n-1) = 3T(n-2)$

$Tn = 3^2 T(n-2)$

or $T(n) = 3^3 \; T(n-3)$

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

$T(n) = 3^n T(0) = 3^n$

so TC $\Rightarrow$ $O(3^n)$

~~Q4. $2^k T(n-k) - 2^{k-1} - 2^{k-2} \ldots 2^1 - 2^0$~~

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

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

let $n=n-1$ $\quad T(n-1) = 2T(n-2)-1$

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

$= 2^2 \; T(n-2) - 2 - 1$

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

so $T(n) = 2^2 (2T(n-3)-1) - 2 - 1$

$= 2^3 \; T(n-3) - 2^2 - 2 - 1$

or

$T(n) = 2^k T(n-K) - 2^{k-1} - 2^{K-2} \;---\; 2^1 . 2^0$

$T(0) = 1$ let $n-k = 0$ so $K = n$

$T(n) = 2^n T(n-n) - 2^{n-1} - 2^{n-2} \;---\; 2^1 - 2^0$

$= 2^n - 2^{n-1} - 2^{n-2} \ldots 2^1 - 2^0$

$= 2^n - (2^{n-1} + 2^{n-2} + \ldots + 2^1 + 2^0)$ GP.

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

~~$T(n) = 2^n - \frac{1(2^n-1)}{2-1} = 2^n -$~~

So TC $\Rightarrow O(1)$

Q5. Int i = 1, s = 1; while (s < n){

i++; s = s+i;

printf("#")

}

Series $\rightarrow$ 1, 3, 6, 10, 15, 21, 28 ...... n

1st iteration $\Rightarrow s = s + 1$

2nd iteration $\Rightarrow s = 0 + 1 + 2$

Till $\Rightarrow 1 + 2 + 3 + \ldots\ldots + + <= n$.

$\frac{K * (K+1)}{2} <= n$

or $O(K^2) <= n$

or $K = O(\sqrt{n})$

so TC $= O(\sqrt{n})$

Q6. for (i = 1; i*i <= n; i++)

count ++

let loop run till K i = K

$K^2 <= n$

$K <= \sqrt{n}$

so TC $\Rightarrow O(\sqrt{n})$

Q7. for $(i = n/2;\ i <= n;\ i++)$ $O(n)$

for $(j = 1;\ j <= n;\ j = j+2)$ $O(\log n)$

for $(k = 1;\ k <= n;\ k\ k+2)$ $O(\log n)$

so TC $\Rightarrow O(n \log^2 n)$

Q8. Function (int n) {

if (n = 1) return;

for (i = 1 to n) {

for (j = 1 to n) {

(print(*))

}

}

function (n-3);

}

Recurrence Relation $\Rightarrow T(n) = T(n-3) + n^2$

or $T(n) = T(n-6) + 2n^2$

$T(n) = T(n-9) + 3n^2$

or $T(n) = T(n-3k) + kn^2$

$T(1) = 0$, $n - 3k = 1 \Rightarrow k = \frac{n-1}{3}$

so $T(n) = T(1) + \frac{(n-1)}{3} n^2$

so TC $\Rightarrow O(n^3)$

Q9. for (i = 1 to n) {

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

i	j	times
1	$1 \to n$	n
2	$1 \to n$	$\frac{n}{3}$
3	$1 \to n$	$n/3$
⋮	⋮	⋮
n	$1 \to n$	1
		$n \log n$

Q10. Find asymptotic relation btw n^K & a^n $K >= 1$ & $a > 1$ are constants. Find c & n_0 for which relation holds.

Sol

$$n^K = O(a^n)$$

$$n^K \le a^n, \quad c \;\forall\; c > 0 \;\&\; n \ge n_0$$

let $n = n_0$

$$n_0^K \le c \cdot a^{n_0}$$

[so let $K = a = 3$]

$[n_0^3 \le c \cdot 3^{n_0}$ so $c \ge 1$ & $n_0 \ge 1]$

Q11.

```
void fun (int n) {
    int i = 0;   j = 0
    while (i < n) {
        i = i + j ;
        j++
    }}
```

Series $\Rightarrow$ 0, 1, 3, 6, 10, 15 - - - -

let at least iterations

$$n = 0 + 1 + 2 + 3 + 4 + 5 + \dots + k$$

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

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

$$n \simeq k^2$$

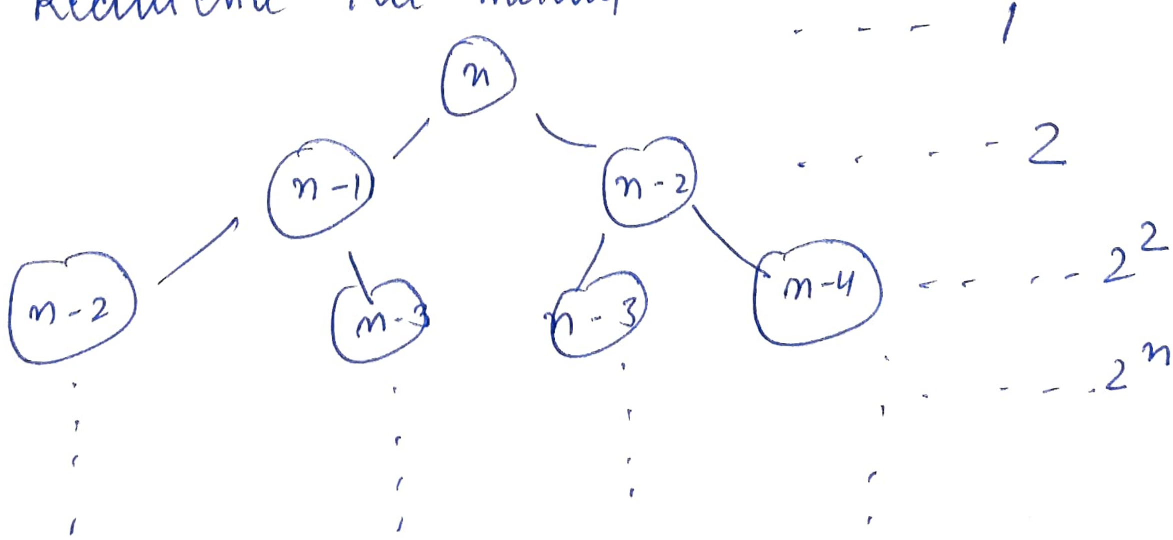
$$k \simeq \sqrt{n}$$

so TC $\Rightarrow$ $O(\sqrt{n})$

Q12. Recurrence relation for fibonacis series

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

using Recurrence Tree method.



$TC = 1 + 2 + 4 + \dots + 2^n = \frac{1(2^{n+1} - 1)}{2-1} = 2^{n+1} - n$

so $TC = O(2^n)$

<u>Space complexity</u> : Space complexity of fibonacci series using recursion is proportional to height of recurrence tree

so $SC \Rightarrow O(n)$

Q13. Write code for complexity

(i) $n \log n$

```
for (i to n)
{
    for (j=1, j<n, j*=2)
        O(1) statements
}
```

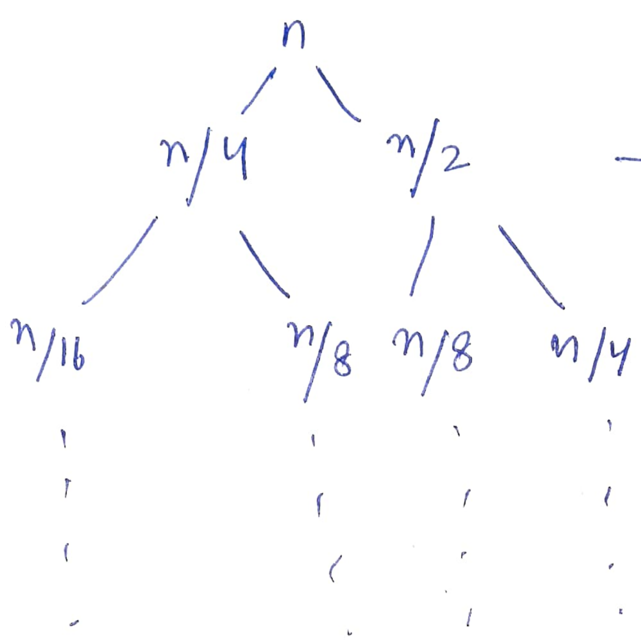
(ii) n^3

```
for (i to n)
    for (j to n)
        for (k to n)
            O(1) statements
```

(iii) $\log(\log n)$

```
int i = n
while (i>0)
{ - - - - - - -
  - - - - - }
i = √i ;
}
```

Q14 $T(n) = T(n/4) + T(n/2) + cn^2$



$\rightarrow cn^2$

$\rightarrow \frac{cn^2}{16} + \frac{cn^2}{4} = \frac{5cn^2}{16}$

$\rightarrow \frac{cn^2}{256} + \frac{cn^2}{64} + \frac{cn^2}{64} + \frac{cn^2}{16}$

$= \frac{25cn^2}{256}$

so $T(n) = c\left(n^2 + \frac{5n^2}{16} + \frac{25n^2}{256} + \cdots\right)$

Here $r = \frac{5}{16}$ so $S_n = \frac{1}{1-r}$

$T(n) = cn^2 \left(1 + \frac{5}{16} + \frac{25}{256} + \cdots\right)$

$= cn \left(\frac{1}{1-\frac{5}{16}}\right) = cn^2 \times \frac{16}{11}$

so TC $\Rightarrow \Theta(n^2)$

Q15. int fun (int n)

{

for (i to n)

for (j = 1 ; j < n ; j += 1) {

O(1) for k

} }

i	j	times
1	1 → n	n = 1
2	1 → n	(n - 1) / 2
3	i → n	(n - 1)/3
⋮	⋮	⋮
n	i → n	(n - 1/n

[TC ⇒ O(n log n)]

Q16. For (int i=2; i<=n; i = pow (i,K))
{
O(1);
}

$i = 2, 2^K, 2^{K^2}, 2^{K^3} \ldots\ldots 2^{K^x}$

$n = 2^{K^x}$

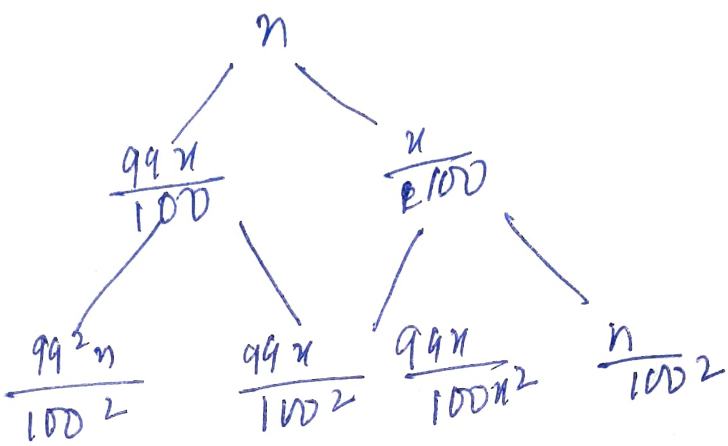
$\log n = K^x \log 2$

$\frac{\log \log n}{\log 2} = x \log K$

$x = \frac{\log \log n}{\log 2 + \log K}$

so TC ⇒ $O(\log \log x)$

Q 17 $\quad T(n) = T\left(\frac{99n}{100}\right) + T\left(\frac{n}{100}\right)$



If we take longer branch i.e $\frac{99n}{100}$

$TC \Rightarrow \log_{\frac{100}{99}} n \cong \log n$

$n\left(\frac{99}{100}\right)^k \qquad k = \log_{\frac{100}{99}} n$

$T(n) = n\left(\frac{\log \frac{100}{99}}{100}\right)^n = (O\, n \log_{99} n)$

Q 18. Increasing of growth

(a) $100 < \log\log n < \log n < \sqrt{n} < n < \log n < n^2 < 2^n$
$< 2^{2n} < 4^n < n!$

(b) $1 < \log\log n < \sqrt{\log(n)} < \log n$
$< \log 2n < 2\log n < n < 2n < 4n < n^2 \log n < n^2 < \log(n!)$
$< 2^{2n} < n!$

(c) $96 < \log_8 n < \log_2 n < 5n < n\log_8(n) < n\log_2 n$
$< 8n^2 < 7n^2 < \log n! < 8^{\wedge}2n < n!$

Q 19. Linear Search

```
for (i=0 to K-1)
    {   if (or [i] = key)
            {   return i;
            }
        return -1;
    }
```

Q 20. <u>Iterative Insertion Sort</u> :

```
void insertion - sort (int arr [ ], int n)
    {   int i, temp, j;
        for i ← 1 to n
        {
            Temp ← arr [i];
            j ← i - 1;
            while (j >= 0 AND arr [j] > temp)
            {
                arr [j+1] ← arr [j]
                j ← j - 1;
            }
        arr [j+1] ← temp
        }
}
```

Recursive Insertion sort ⇒

```
void recursive_insertion_sort (int arr [], int n]
{ if (n <=1)
        return
    recursive - insertion - sort (arr, n-1)
    val = arr [n - 1]
    pos = n - 2
    while (pos >= 0 && arr [pos] > val ) {
        arr [pos + 1] = arr [pos]
        pos = pos - 1
    }
    arr [pos + 1] = val
}
```

It is called online sorting because it provided one sorted element at a time & produces a partial solution without considering future elements.

Q 21.

Algorithm	Time Complexity: Best case	Average Case	Worst Cas
① Bubble sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
② selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
③ merge sort	$O(n \log n)$	$O(n\log n)$	$O(n \log n)$
④ insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$
⑤ Quick sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
⑥ sleep sort	$O(n \log n)$	$O(n \log n)$	$O(n\log n)$

Q22.

SNo	Algorithm	Inplace	Stable	Online sorting
1.	Bubble sort	✓	✓	✗
2.	Selection sort	✓	✗	✗
3.	Merge sort	✗	✓	✗
4.	Insertion sort	✓	✓	✓
5.	Quick sort	✗	✗	✗
6.	Heap sort	✓	✗	✗

Q23. Recursive Binary search

```
int b-search (int arr[ ], int l , int r ,int x )
{   if (l > r)
        return -1;
    int  m = (l + r)/2
    if (arr[m] = x)
            return m;
    else if (arr [m]<x )
            b-search (arr , m+1 r, x );
    else
        b - search (arr, l, m-1 , x
    }
```

Iterative Binary search

```
int binary search (int arr [ ], int l n, int x)
{   l = 0 , r = n - 1
    while (l<r)
```

```
{ ~~l = 0, r = n - 1;~~
{
    m = (l + r) / 2
    if (arr [m] = x) return m;
    else if (arr [m] < x) l = m - 1;
    else r = n - 1;
}
return -1;
}
```

Time & space complexity of Iterative Binary search ⇒ O(log n) & O(1)

Time & space complexity of Recursive Binary search ⇒ O(log n), O(log n)

Q24. Recurrence Relation for Binary search ⇒

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