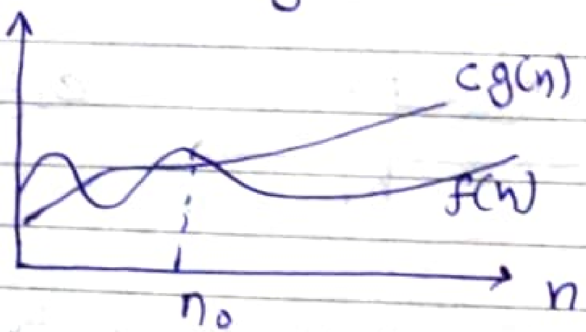
Assignment - 1

Ans 1- Asymptotic notation are used to represent the complexities of algorithms for asymptotic analysis.

These notation are used for very large input.

1- Big-oh (O) -

It gives upper bound for a function $f(n)$ to within a constant factor.

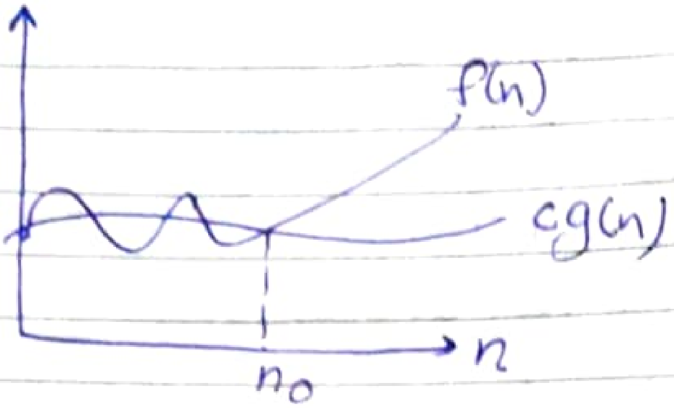


$$f(n) \le c \cdot g(n) \quad \forall \; n \ge n_0 , \; c > 0$$

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

2- Big omega Notation (Ω)

Big-Omega (Ω) notation gives a lower bound for a $f(n)$ to within a constant factor

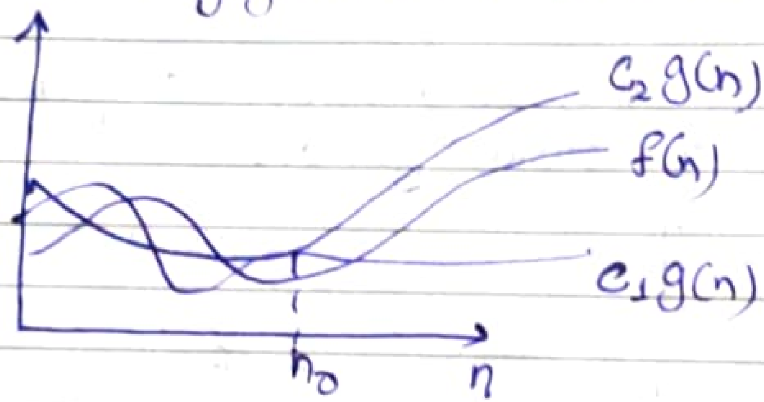


$\Omega(g(n)) = \{f(n)$: There exist (+)ve constant c & n_0 such that $0 \leq cg(n) \leq f(n) \quad \forall\, n \geq n_0\}$

eg - $\Omega(n \log n)$

3- <u>Big theta Notation (Θ)</u>

it gives bound of function within a constant factor.



$\Theta(g(n)) = \{f(n)$: There exist (+)ve constant c_1, c_2 and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \ \forall\, n \geq n_0\}$

eg - $\Theta(n^2)$

Ans 2- for (i=1 to n)
{
(i = i * 2);
}

1, 2, 4, 8, ... n

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

Ans 3- $T(n) = \begin{cases} 3T(n-1) & n>0 \\ 1 & n=0 \end{cases}$

$$T(n) = 3T(n-1) \quad -1$$
$$T(n-1) = 3T(n-2)$$
$$T(n) = 9T(n-2) \quad -2$$
$$T(n) = 3^3 T(n-3) \quad -3$$
$$T(k) = 3^k T(n-k) \quad -4$$
for $T(n-k) = T(0)$
$$n-k=0$$
$$n=k$$

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

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

$$T(n) = 2T(n-1) - 1 \quad -1$$
$$T(n-1) = 2T(n-2) - 1$$
$$T(n) = 4T(n-2) - 1 - 2 \quad -2$$
$$T(n) = 8T(n-3) - (1+2+4) \quad -3$$

$$T(n) = 2^k T(n-k) - (\underbrace{1+2+4+\cdots 2^{k-1}}_{K \text{ terms.}})$$

$$T(n-k) = T(0)$$

$$n = k$$

$$T(n) = 2^n T(0) - (\underbrace{1+2+4+\cdots\cdots}_{K \text{ terms}})$$

Its a G.P

$a = 1$

$r = 2$

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

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

$$T(n) = 1$$

$$T(n) = \Theta(1)$$

Ans 5-

```
int i=1, s=1;
while (s<=n) {
        i++;
        s=s+i;
        printf("#");
}
```

$$\underbrace{1, 3, 6, 10, 15, \ldots\ldots n}_{K \text{ terms}}$$

Its k^{th} term is $\frac{k(k+1)}{2} = n$

$$K = \sqrt{n}$$

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

Ans 6-

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

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

Ans 7- $T(n) = O(n * \log_2 n * \log_2 n)$

$$T(n) = O(n * (\log_2 n)^2)$$

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

Ans 8-

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

$$T(n) = T(n-3) + n^2 \quad \text{— (1)}$$

$$T(n-1) = T(n-4) + (n-1)^2$$

$$T(n) = T(n-4) + n^2 + (n-1)^2$$

$$T(n) = T(n-5) + n^2 + (n-1)^2 + (n-2)^2$$

$$T(n) = T(n-k) + (n^2 + (n-1)^2 + (n-2)^2 \dots (k-2) + \dots)$$

for $T(n-k) = 1$
$k = n-1$

$$T(n) = T(1) + (n^2 + (n-1)^2 + (n-2)^2 + \dots\dots)$$
$(n-3)$ terms

$$T(n) = T(1) + (4^2 + 5^2 + \dots n^2)$$

$$T(n) = T(1) + \left(\frac{(n-3)(n-2)(2n-5)}{6}\right)$$

$$T(n) = 1 + \left(\frac{2n^3 + \dots}{6}\right)$$

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

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

Ans 9- void function(int n) {

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

~~Outer loop - n times~~ $i=1$ n times
~~Inner loop - 1,~~ $i=2$ · 1, 3, 5, ... n $\quad n$
$i=3$ 1, 4, 7 ... n $\quad n/2$
⋮ $\quad n/3$
$i=n$ 0

$$T(n) = \left(\underbrace{n + \frac{n}{2} + \frac{n}{3} - -}_{n \text{ times}} \right)$$

~~$T(n) = O(\log_2 n)$~~ $T(n) = O(n \log n)$

Ans10- for the functions n^k and a^n, what is the relation

$k \geq 1$ & $a > 1$

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

Ans 11-

```
void fun(int n)
{
    int j = 1, i = 0;
    while (i<n)
    {
        i = i+j;
        j++;
    }
}
```

$$\underbrace{0, 3, 6, 10, 15, \ldots\ldots, n}_{K\text{ terms.}}$$

So for this series is

K^{th} term is $\frac{K(K+1)}{2}$

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

$$K \simeq \sqrt{n}$$

$$T = \Theta(\sqrt{n})$$

Ans 12- Recurrence relation of fabonacci series is

$$T(n) = \left\{ \begin{array}{l} T(n-1) + T(n-2) + 1 \\ \end{array} \right\}$$

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

$$T(n) = 4T(n-4) + 3$$

$$T(n) = 8T(n-6) + 7$$
$$T(n) = 16T(n-8) + 15$$

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

for $T(n-2k) = T(0)$
$$n = 2k$$
$$k = \frac{n}{2}$$

$$T(n) = 2^{n/2} T(0) + (2^{n/2} - 1)$$

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

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

hence space complexity of fabinacci series is O(n) as it depends on height of recursive tree & it is equal to n in fabinacci series.

Ans 13. → n(log n)

```
void fun(){for(int j=0 ; j<n ; i++ {
        for( int i=0 ; i<n ; i=i*2)
        {
            print("*");
        }
    }
}
void main()
{
    fun();
}
```

→ n^3

```
#include <stdio.h>
void main()
  {
      int n;
      cin>>n;
      for(int i=0; i<n; i++) {
           for(int j=0; j<n; j++) {
              for(int k=0; k<n; k++) {
                   x++;
              }
           }
      }
  }
```

→ log (log n)

```
#include<bits/stdc++.h>
void fun(int n)
    {
        if(n==2)
           return 1;
        else
          fun(sqrt(n));
    }
 void main()
    {
        fun(100);

    }
```

Ans 14 - $T(n) = T(n/4) + T(n/2) + cn^2$

$T(1) = c$

$T(0) = 0$

$cn^2 \longrightarrow cn^2$

$T(n/4) \quad T(n/2)$

$c\left(\frac{n^2}{16}\right) \quad c\left(\frac{n^2}{4}\right) \longrightarrow \frac{5n^2}{16}c$

$T(n/16) \quad T(n/8) \quad T(n/8) \quad T\left(\frac{n}{4}\right)$

$\vdots \qquad \rightarrow \frac{25n^2}{256}c$

$T(n) =$ Cost of each level

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

it is a G.P

with $a = n^2$

$r = \frac{5}{16}$

So sum of s.p.

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

$T(n) = \Theta(n^2)$

Ans 15 - for (int i to n)
{
for (int j = 1 ; j < n ; j += i)
{
// O(1)
}
}

$$\underbrace{n, \frac{n}{2}, \frac{n}{3}, \frac{n}{4}, \frac{n}{5}, \ldots\ldots 1}_{k \text{ times}}$$

$$k = \log_2 n$$

$$n\left(1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots\ldots \frac{1}{n}\right)$$

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

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

Ans 16 - for (int i = 2 ; i <= n ; i = pow (i, k))
{
// O(1)
}

$$2,\ 2^k,\ 2^{(k)^2},\ 2^{k^3}, \ldots\ldots n$$

If G.P $a = 2$

$r = 2^k$

k^{th} term $= ar^{k-1}$

$$n = 2(2^k)^{k-1}$$

$$\cancel{\log n = (k-1)\log 2^k}$$

let $k^{(k-1)} = x$

$$k \log_k k = \log x$$

$$k = \log x \quad \text{—— (1)}$$

$$n = 2^x$$

$$\log_2 n = x \log_2 2$$

$$x = \log_2 n$$

$$\log x = \log_2(\log n)$$

from (1)

$$k = \log(\log(n))$$

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

Ans 17 -

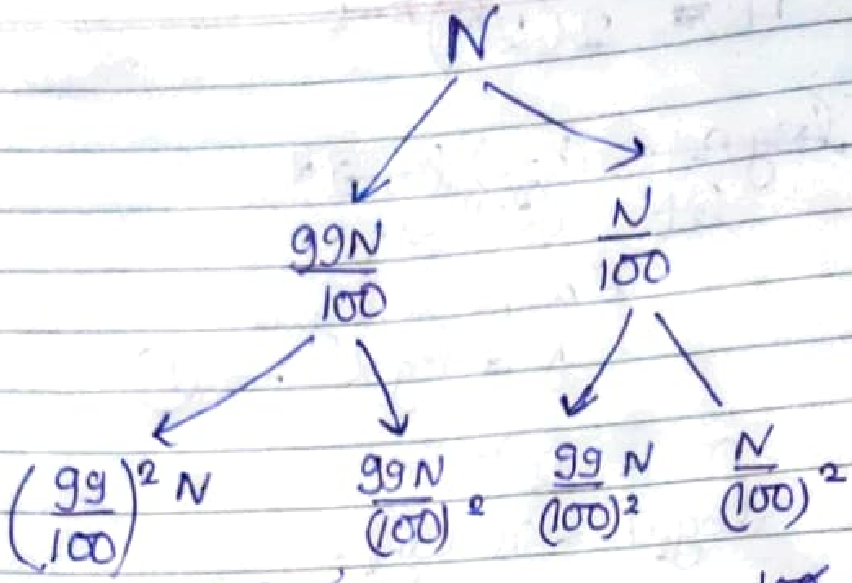
here pivot is divided in 99% & 1%

so

$$T(n) = T\left(\frac{99}{100}N\right) + T\left(\frac{N}{100}\right) + N$$

Now as here we can use 2 extreams of a tree

where starting point is N



N

N

$N\left(\frac{(99)(99)}{100\times100} + \frac{99(1)}{100\times100}\right) + \frac{100}{100\times100}N = \frac{99}{100}N + \frac{N}{100}$

$= N$

$\vdots$

$= N$

So cost of each level is N only.

Total cost = height * Cost of each level.

So for 1st stream - $N, \frac{99N}{100}, \left(\frac{99}{100}\right)^2 N, \ldots$

$$\left(\frac{99}{100}\right)^{h-1} N = 1$$

$$\left(\frac{99}{100}\right)^{h-1} = \frac{1}{N}$$

$$N = \left(\frac{100}{99}\right)^{h-1}$$

$$\log N = h \log (1)$$

$$h = \log N \text{ or}$$

$$h = \frac{\log N}{\log(100/99)} + 1$$

height of 2nd stream

$$N, \frac{N}{100}, \frac{N}{(100)^2}, \frac{N}{(100)^3}, \cdots 1$$

$$N\left(\frac{1}{100}\right)^{h-1} = 1$$

$$N = (100)^{h-1}$$

$$(h-1)\log 100 = \log N$$

$$h = \frac{\log N}{\log 100} + 1 \quad \& \quad h = \log N \quad \text{(approx)}$$

$$T(n) = O(N \log N)$$

So time complexity is $O(N \log N)$

height of both extre is $\frac{\log N}{\log 100} + 1$ of $\left(\frac{1}{100}\right)$

and $\frac{\log N}{\log\left(\frac{100}{99}\right)} + 1$ of $\left(\frac{99}{100}\right)$

So we can conclude that if division is done more then height of tree will be more &

and when division ratio is less then height is less.

Answer 18 -

a) n, $n!$, $\log n$, $\log\log n$, $root(n)$, $n\log n$

2^n, 2^{2n}, 4^n, n^2, 100

Ans -

$O(100) < O(\log\log n) < O(\log n) < O(\sqrt{n}) < O(n) <$

$O(n\log n) < O(n^2) < O(2^n) < O(2^{2n}) < O(4^n)$.

b) $2(2^n)$, $4n$, $2n$, 1, $\log(n)$, $\log(\log(n))$, $\sqrt{\log(n)}$, $\log 2n$, $2\log n$, ~~...~~ n, $\log(n!)$, $n!$, n^2, $n\log(n)$

$O(1) < O(\log(\log(n))) < O(\log(n)) < O(\log 2n) < O(2\log n)$

$< O(n) < O(n\log(n)) < O(\log(n!)) < O(2n)$

$< O(4n) < O(n^2) < O(n!) < O(2(2^n))$.

c). $8^{\wedge 2n}$, $\log_2 n$, $n\log_6(n)$, $n\log_2(n)$, $\log(n!)$,

$\log_8(n)$, 96, $8n^2$, $7n^3$, $5n$.

Ans - $O(96) < O(\log_8(n)) < O(\log_2 n) < O(\log(n!)) <$

$O(n\log_6(n)) < O(n\log_2(n)) < O(5n) < O(8n^3)$

$< O(7n^3) < O(n!) < O(8^{\wedge 2n})$.

Ans 19-

```
void LinearSearch ( int arr[], int n, int key)
{
    for (i=0 to i=n )
        if arr[i] == key
            cout<< found";
        else
            continue

}
```

Ans 20- Iterative Insertion Sort

```
void Insertion sort( arr , n ) {

    int i, temp, j
    for i-1 to n
    {
        temp = arr[i]
        j = i-1
        while j>=0 && arr[j] >temp
        {
            arr [j+1] = arr[j]
            j--
        }
        arr[j+1] = temp

    }.

}
```

Recursive Insertion sort

```
insertion sort (arr, n)
{
    if n<=1
        return;

    insertionsort (arr, n-1);
    last = arr[n-1];
     j = n-2

    while (j >=0 and arr[j] > last)
    {
        arr[j+1] = arr[j]
          j--
    }
    arr[j+1] = last;

}
```

Insertion sort is called online sorting because it don't know the whole input, it might make decision that later turn out to be not optimal.

Other algorithm are off-line algorithms that are discussed in lectures.

Ans 21 -

	Time complexity			Space
	Best	Avg	worst	
Bubble sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection sort	$O(n^2)$	$O(n^2$	$O(n^2)$	$O(1)$
Insertion sort	(n)	$O(n^2)$	$O(n^2)$	$O(1)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$ {due to recursion}
Quick sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(n)$
Heap sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$

Ans 22 -

	inplace	stable	Online sorting
Bubble Sort	Yes	Yes	No
Selection Sort	Yes	No	No
Insertion Sort	Yes	Yes	Yes
Merge Sort	No	Yes	No
Quick Sort	Yes	No	No
Heap Sort	Yes.	No.	No.

Ans 23-

```
BinarySearch (arr, int n, key)
{
    beg = 0
    end = n-1
    while (beg <= end)
    {
        mid = (beg + end)/2

        if [arr[mid] == key]
            found
        else if  arr[mid] < key
                beg = mid + 1

        else
            ~~beg = end~~ end = mid - 1

    }

}
```

Time complexity of Linear search - $O(n)$
Space complexity of Linear search - $O(1)$

Time complexity of Binary search = $O(\log n)$

Space complexity of Binary search = $O(n)$

Ans 24 - $T(n) = T\left(\frac{n}{2}\right) + 1$ //