Name: Megha Sharma
Course: BTech CSE
Sem: 5 Roll No: 39
URN: 1961104
Section: C

Design And Analysis Of Algorithms

1) **Asymptotic Notations** :- These are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.

There are mainly 3 Asymptotic Notations :-

i) Big - O notations
ii) Omega (Ω) notations
iii) theta (θ) notations.

→ **Big - O notations** represent the upper bound of the running time of an algorithm.

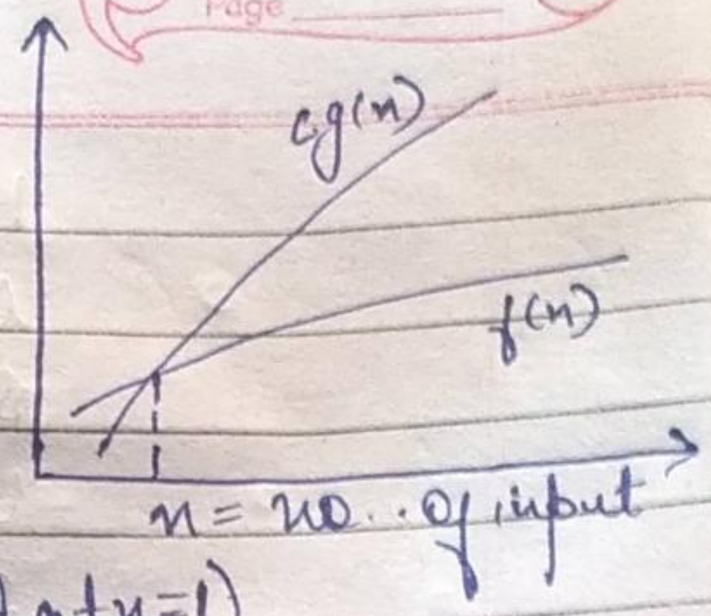
$O(g(n)) = \{ f(n)$: there exist positive constant c and n_0 such that $0 \leq f(n) \leq g(n)$ for all $n = n_0\}$

where $c > 0$ and $n \geq n_0$.

This notation is known as upper bound of the algorithm, or a worst case of an algorithm.

Ex :- $f(n) = 3\log n + 100$

$g(n) = \log n$



$3\log n + 100 <= c * \log(n)$

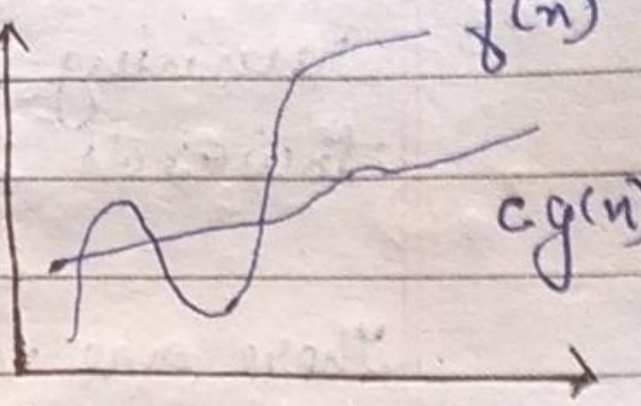
$\boxed{c = 150 \text{ f } n \geq 2}$ (undefined at $n=1$)

→ Omega (Ω) Notation represents the lower bound of the running time of an algorithm.

$\Omega(g(n)) = \{f(n)$: there exist positive constant c & n_0 such that

$0 \leq c\, g(n) \leq f(n)$

for all $n, n \geq n_0$.



This notation is known as the lower bound of algorithm, or a best case of an algorithm.

Example -: $f(n) = 3n+2$

$c \cdot g(n) \leq f(n)$ c = constant $g(n) = n$

$c \cdot n \leq 3n+2$

$cn - 3n \leq 2$

$n(c-3) \leq 2$

$n \leq \frac{2}{c-3}$

if we assume $c = 4$, then $n_0 = 2$

$\boxed{c = 4,\ n_0 = 2}$

→ Theta (Θ) notation encloses the function from above and below. Since it represent the upper & the lower bound of the running time of an algorithm.

$\Theta(g(n)) = f(n)$: there exist positive constant c_1, c_2 & n_0 such that

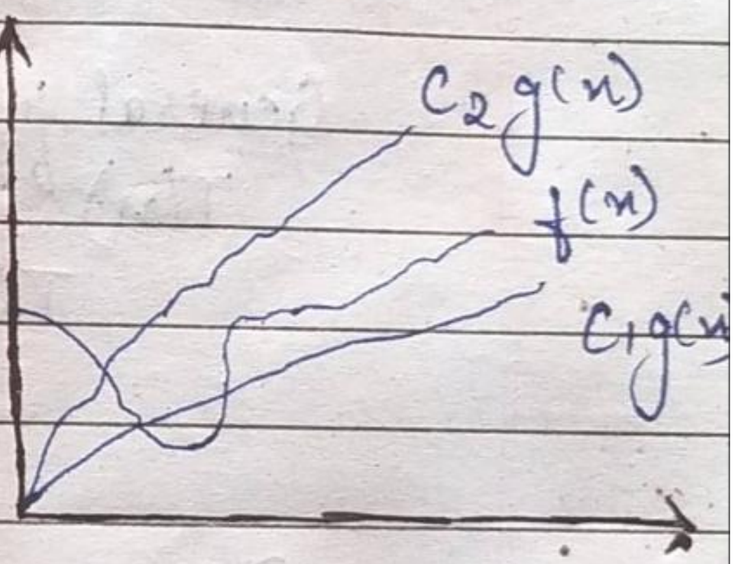
$$0 \leq c_1 * g(n) \leq f(n) \leq c_2 * g(n)$$

for all $n > n_0$.

This is known as tight Bound of an algorithm or a average case of algorithm.

Example

$$f(n) = 5n^3 + 16n^2 + 3n + 8$$
$$= 5n^2 \leq 5n^3 + 16n^2 + 3n + 18 \leq$$
$$(5+16+3+8)n^3$$
$$= 5n^3 \leq f(n) \leq 32n^3$$



$$\boxed{c_1 = 5, c_2 = 32, n_0 = 1}$$

$f(n) \longleftrightarrow \Theta(n^3)$

Ans 2 :- $i = 1, 2, 4, 8, 16, \ldots\ldots K^{th}$ term $\ldots n$

$$G_n = ar^{n-1}$$
$$G_n = 1(2)^{K-1}$$
$$n = 2^{K-1}$$
$$\log_2 n = (K-1)\log_2 2$$
$$\boxed{K = \log_2 n + 1}$$

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

Ans3 :- $T(n) = 3\,T(n-1)$

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

$T(n) = 3 \times 3T(n-2)$

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

$T(n) = 3 \times 3 \times 3T(n-3)$

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

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

$T(n) = 3^3 \times 3T(n-4)$

$T(n) = 3^4 \times T(n-4)$

⋮

General form

$T(n) = 3^i T(n-i)$ - (1) $\quad T(0) = 1$

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

$n - i = 0$

$\boxed{n = i}$

Put $n = i$ in eq (1)

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

$T(n) = 3^n T(0) \quad (T(0) = 1$ (Given))

$T(n) = 3^n$

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

Ans4 :- $T(n) = \begin{cases} 2T(n-1) - 1 & \text{if } n > 0 \\ \text{; otherwise} \end{cases}$

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

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

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

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

⋮

$T(n) = 2^i (T(n-i) - (2^{i-1} + 2^{i-2} + 2^{i-3} + \dots +$

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

$n - i = 0$

$\boxed{n = i}$

Time complexity = $O(1)$

Ans 5 :-

No. of step (K)	i	j
0	0	1
1	1	2
2	3	3
3	6	4
4	10	5
5	15	6
6	21	7
⋮	⋮	⋮
K step	n	⋮

$T(n) = O(K)$

$i = 0, 1, 3, 6, 10, \ldots n$

$Sn = 1 + 3 + 6 + 10 + 15 + \ldots + n$

$Sn = 1 + 3 + 6 + 10 + \ldots (n-1) + n$

$- (+) \quad (-)$

$0 = 1 + 2 + 3 + 4 + 5 + \ldots + n$

$n = 1 + 2 + 3 + 4 + \ldots$ K-step

$n = \frac{K}{2}[2(1) + (K-1)1]$

$2n = K[2 + K - 1]$

$2n = K^2 + K \Rightarrow 2n = (K + 1/2)^2 - (1/2)^2 \Rightarrow 2n + (1/2)^2 = (K + 1/2)^2$

$K + 1/2 = \sqrt{2n + (1/2)^2}$

$K = \sqrt{2n + (1/2)^2} - 1/2$

$$T(n) = T(K)$$
$$T(n) = T(\sqrt{2n + (1/2)^2} - 1/2)$$
$$T(n) = O(\sqrt{n})$$

Ans 6 :-

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

Since i is moving from 1 to $\sqrt{n}$ with linear growth so,

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

Ans 7 :-

```
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++;
    }
```

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

Ans8 :-

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

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

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

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

⋮

General Term

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

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

$n = i + 1 \Rightarrow \boxed{n-1 = i}$

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

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

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

$T(n) = \frac{n(n-1)(2n+1)}{6}$

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

Ans9-: Time complexity of

```
void function (int n)
{   for (i=1; i<n; i++)
    {   for (j=1; j<=n; j=j+i)
            printf ("#");
    }
}
```

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

Ans10-:

If $c > 1$ then exponential c^n for outgrows any term

So, that one is

$$n^k \text{ is } O(c^n)$$

Ans12-: $T(n) = T(n-1) + T(n-2) + c$

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

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

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

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

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

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

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

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

⋮

General Term

$$T(n) = 2^i [(n-i) + (2^0 + 2^1 + 2^2 + \cdots + 2^{i-1})] c$$

$$n - i = 0$$

$$\boxed{n = i}$$

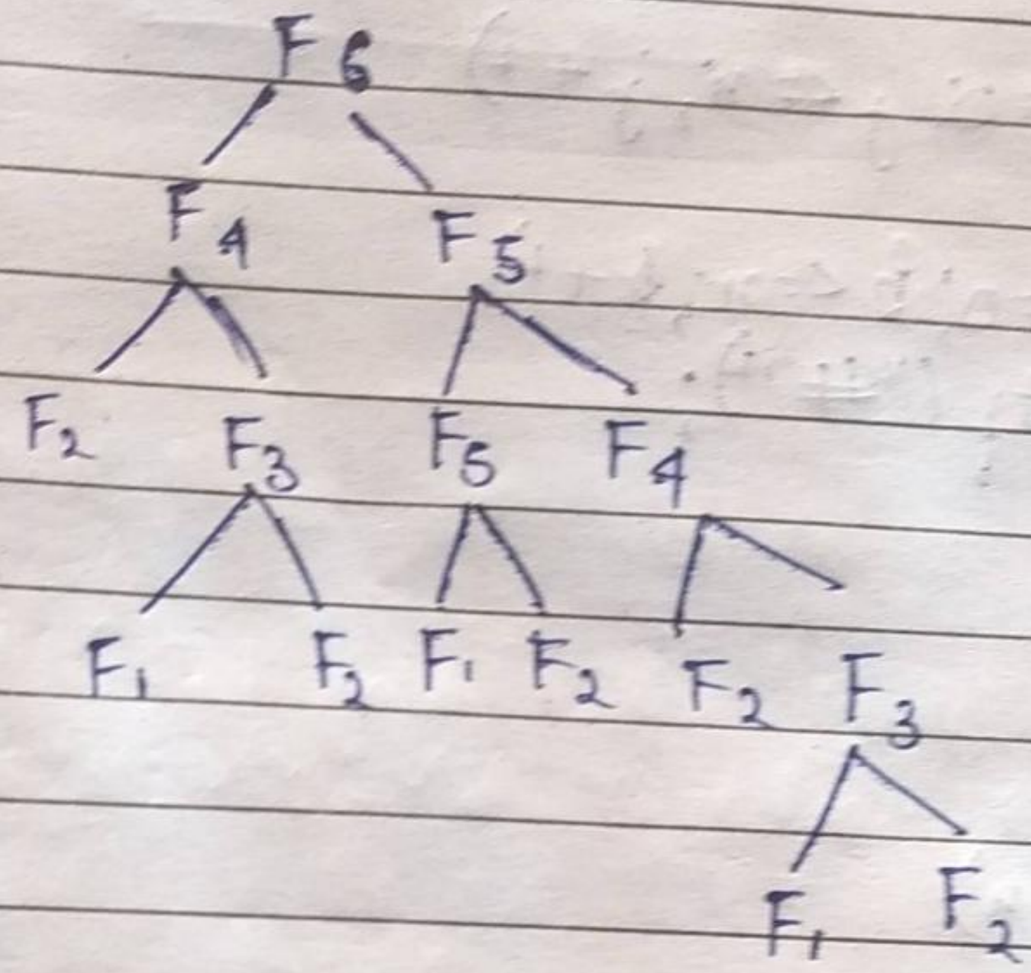
$$T(n) = 2^n T(0) + (2^0 + 2^1 + 2^2 + \cdots + 2^{n-1})C$$

$$T(n) = 2^n (1) + \frac{2^0 (2^{n-1} - 1) C}{2 - 1}$$

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

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

fib(6)



The maximum depth is proportional to N, hence the space complexity of fibonacci series is O(n)

Ans 3 i) $n \log n$

```
void fun ()
{
  int i, j;
  for (i=1; i<=n; i++)
  {
    for (j=0; j<=n; j=j*2)
      printf ("#");
    printf ("\n");
  }
}
```

ii) n^3

```
void fun(int n)
{
int i,j,k;
for(i=0; i<=n;i++)
{
for(j=0; j<=n; j++) :
{
for(k=0; k<=n; k++)
  printf("#");
}
}
}
```

iii) $\log(\log n)$

```
void Seve of Eratosthenes (int n)
{
    bool prime[n+1];
memset (prime, true, sizeof(prime));
for(int p=2; p*p<=n; p++)
{
if (prime[p]==true)
{
for(int i= p* p; i<=n; i+=p)
    prime[i]= false;
}
}
for (int p=2; p<=n; p++)
  if(prime[p])
```

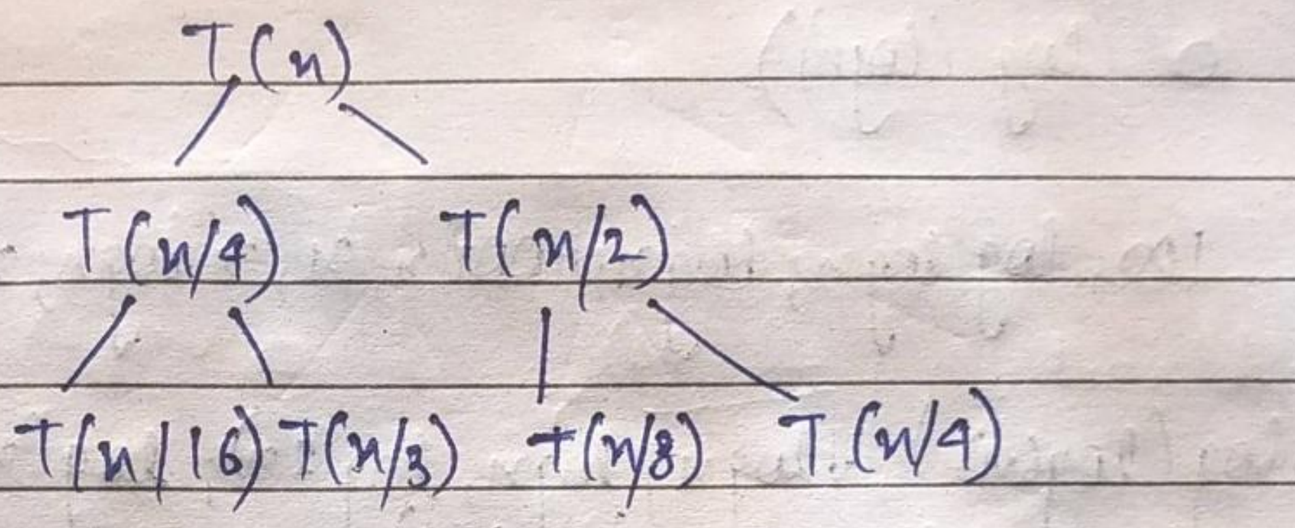
```
    count << p << endl;
}
```

Ans14) $T(n) = T(n/4) + T(n/2) + cn^2 \quad T(1) = c$

$n = n/2$

$T(n/2) = T(n/3) + T(n/4) + c(n^2/4)$

$T(n) = T(n/4) + 2T(n/16) + c(n^2/16 + n^2/4 + n^2)$



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

$$T(n) = n^2 c\left[1 + \frac{5}{16} + \frac{5^2}{16^2} + \cdots\right]$$

$$\boxed{T(n) = O(n^2)}$$

Ans15→ for $i=1$, the inner loop is executed n times.
for $i=2$, the inner loop is executed $n/2$ times.
for $i=3$, the inner loop is executed $n/3$ times
for $i=4$, the inner loop is executed $n/4$ times
:
for $(i=n)$, the inner loop is executed n/n times.

Total time = $n + n/2 + n/3 + \cdots + n/n$.

$= n(1 + 1/2 + 1/3 + \cdots + 1/n)$

$= n \log n$

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

Ans16 :- $O(\log(\log n))$

Ans18 :- a) $100, \log\log n, \log n, \text{root } n, n, n\log n, n^2, 2^n, 2^{2n}, 4^n, n!$

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

c) $96, \log 8^n, \log n, \log(n!), 5n, n\log_6 n, n\log_2 n, 8n^2, 7n^3, 8^{2n}, n!$

Ans19 :-

```
LINEAR_SEARCH(A, key)
    comp ← 0, f ← 0
for i = 1 to A.length
    comp ← comp + 1
if A[i] == key
print "Element founded"
    f = 1
    if f == 0
        print "Element not found"
print comp
```

Ans20 :- INSERCTION - SORT (A)

for j = 2 to A.length

Key = A[j]

i = j - 1

while i > 0 and A[i] > key

A[i+1] = A[i]

i = i - 1

A[i+1] = Ky

Recursive Method of insertion sort

~~INSERTION-SORT (A, n-1)~~

INSERTION-SORT (A, n)

if n ≤ 1

return

INSERTION-SORT (A, n-1)

key = [n-1];

j = n-2;

while j ≥ 0 & A[j] > key

A[j+1] = A[j]

j = j-1

A[j+1] = key.

→ Insertion sort considers one input element per iteration & produce a partial solution without considering future elements thats why it is called online sorting.

Some other Sorting algorithms discussed :-

- Bubble Sorting
- Selection Sorting
- Merge Sort
- Heap Sord
- Quick Sort
- Counting Sort

Ans 21

	Best Case	Average Case	Worst Case.
BubbleSort	$\Omega(N)$	$\Theta(N^2)$	$O(N^2)$
Selection Sort	$\Omega(N^2)$	$\Theta(N^2)$	$O(N^2)$
Insertion Sort	$\Omega(N)$	$\Theta(N^2)$	$O(N^2)$
Merge Sort	$\Omega(N \log N)$	$\Theta(N \log N)$	$O(N \log N)$
Heap Sort	$\Omega(N \log N)$	$\Theta(N \log N)$	$O(N \log N)$
Quick Sort	$\Omega(N \log N)$	$\Theta(N \log N)$	ON^2
CountSort	$\Omega(N+K)$	$\Theta(N+K)$	$O(N+K)$

Ans 22 :-

	In place	Stable	Online
BubbleSort	Yes	Yes	Yes
InsertionSort	Yes	Yes	Yes
Selection Sort	Yes	No	Yes
Heap Sort	Yes	No	Yes
Merge Sort	No	Yes	Yes
QuickSort	Yes	No	Yes
Count Sort	No	Yes	Yes.

Aus 23 :- Linear Search

```
LINEAR_SEARCH (A, key)
    found ← 0
for i = 1 to N
    if A[i] == key
        found ← 1
print "Element found"
break
if found == 0
    print "Element Not found"
```

Time complexity — O(n)
Space complexity — O(1)

Binary Search (Iterative)

```
BINARY_SEARCH (A, beg, end, key)
  while beg ≤ end
      mid = beg + (end - beg)/2
  if mid == key
          return mid
  if A[mid] < key
          beg = mid + 1
  if A[mid] > key
      end = mid - 1
  return -1
```

Time complexity - O(log₂n)
Space complexity - O(1)

Binary search (Recursion)

BINARY_SEARCH (A, beg, end, key)

if end ≥ beg
    mid = (beg + end)/2

if A[mid] == item
        return mid+1
else if A[mid] < item
return BINARY_SEARCH (A, mid+1, end, key)

else
    return BINARY_SEARCH (A, beg, mid-1, end)
return -1

Time complexity - O(log n)
Space complexity - O(1)

Ans T(n) = T(n/2) + C