Name - Sapna Nusuni
Course - B.Tech (CSE)
ERN/Sec - G1/C.

Date
Page

Assignment - 1

1. What do you understand by asymptotic notation. Define different Asymptotic notation with example.

→ Asymptotic notations are the mathematical notations used to describe the running time of an algo when the i/p tends towards a particular value or a limiting value.

→ There are mainly 3 Asymptotic notations :-

(i) Big-o notation

- It represents the upper bound of running time of an algo.
- This notation is called as upper bound of the algo., or a worst case of an algo
- $O(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$, where $c > 0$ and $n \geq n_0$.
- e.g.,

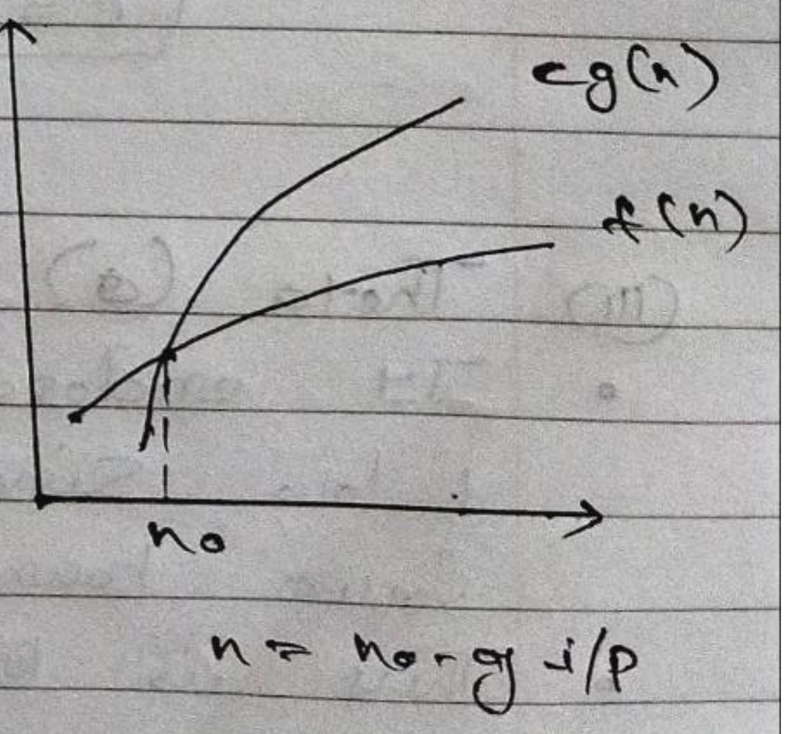
$f(n) = 3 \log n + 100$

$g(n) = \log n$

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

$c = 150$ and $n > 2$

(undefined at $n = 1$)



(ii) Big Omega (Ω) Notation

- It represents the lower bound of the running time of an algo.
- This notation is known as lower bound of an algo, or best Case of an algo.

- $\Omega(g(n)) = \{f(n)$: there exist positive constant c and n_0 such that $0 \leq cg(n) \leq f(n)$ $\forall n$, $n > n_0$.

- e.g.,

$f(n) = 3n+2$

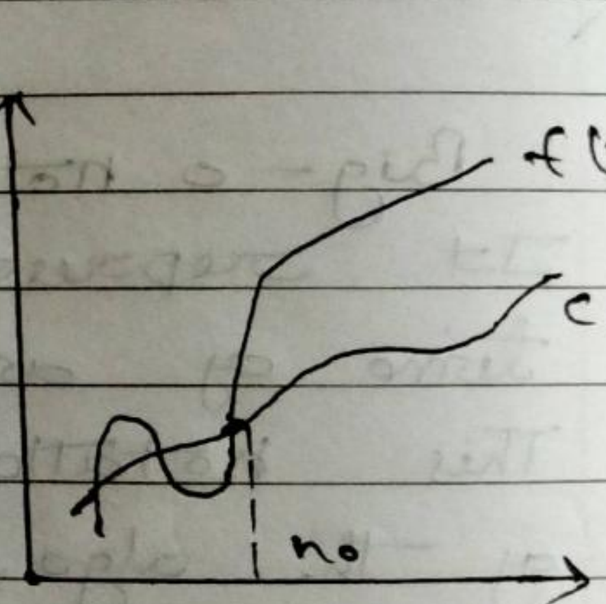
$c \cdot g(n) \leq f(n)$

[c = Constant, $g(n) = n$]

$cn \leq 3n+2$

$cn - 3n \leq 2$

$n(c-3) \leq 2 \Rightarrow n \leq \frac{2}{c-3}$



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

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

(iii) Theta (Θ) notation.

- It enclose the function from above and below. Since, it represents the upper and lower bound of running time of an algo.
- This is known as tight bounds of an algo, or an average Case of algo.

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

$0 \leq c_1 * g(n) \leq f(n) \leq c_2 * g(n)$ $\forall$ $n > n_0$.

- eg,

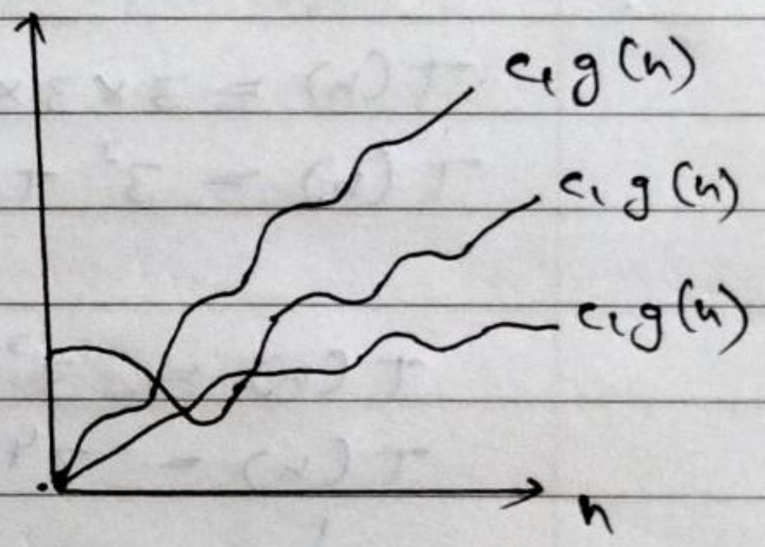
$f(n) = 5n^3 + 16n^2 + 3n + 3$

$5n^3 \leq 5n^3 + 16n^2 + 3n + 8 \leq (5+16+3+8)n^3$.

$5n^3 \leq f(n) \leq 32n^3$

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

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



2. what should be the time complexity:

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

i = 2, 4, 8, 16, ------ k^{th} term --- n

$G_n = ar^{n-1}$

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

$n = 2^{k-1}$

$\log_2 n = (k-1)\log_2 2$

$\boxed{k = \log_2 n + 1}$

$\Theta(n) = \log n$

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

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

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

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

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

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

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

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

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

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

General form :-

$T(n) = 3T(n-i)$ ——— (1) $[T(0) = 1]$

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

$n - i = 0$

$\boxed{n = i}$

Putting $n = i$ in eq (1);

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

$T(n) = 3^n T(0)$ $[T(0) = 1$, given$]$

$T(n) = 3^n$

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

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

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

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

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

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

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

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

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

$\hookleftarrow T(n-3) = 2T(n-4) - 1$

$T(n) = 2^3 (2T(n-4) - 1) - 2^2 - 2 - 1$

$T(n) = 2^4 T(n-4) - 2^3 - 2^2 - 2 - 1$

⋮

General form:-

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

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

$n - i = 0$

$\boxed{n = i}$

$T(n) = 2^n T(0) - (1 + 2 + 2^2 + 2^3 + \cdots 2^{n-1})$

$[T(0) = 1]$

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

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

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

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

5. What should be the T.C. of :—

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

Ans

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

$T(n) = O(k)$

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

$S_n = 1+3+6+0+15+\cdots\cdots+h$

$S_n = 1+3+6+10+\cdots\cdots+(n-1)+h$

$0 = 1+2+3+4+5+\cdots\cdots\cdots+h$

$h = 1+2+3+4+\cdots$ K Stcp

$h = \frac{k}{2}[2(1)+(k-1)1]$

$2h = k[2+k-1]$

$2h = k^2+k \Rightarrow 2h = \left(k+\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2$

$2h + \left(\frac{1}{2}\right)^2 = \left(k+\frac{1}{2}\right)^2$

$k+\frac{1}{2} = \sqrt{2h+(1/2)^2}$

$k = \sqrt{2h+(1/2)^2} - 1/2$

$T(h) = T(k)$

$T(h) = T\left(\sqrt{2h+(1/2)^2} - 1/2\right)$

$\boxed{T(h) = O\sqrt{h}}$

6. T.C. g :—

```
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

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

7. Time Complexity of

```
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 ++;
}
```

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

$$\boxed{O(n(\log n)^2}$$

8. Time Complexity of

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

$T(n) = T(n-1) + n^2 \quad [T(n-1) = T(n-2) + (n-1)]$

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

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

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

|

General Term :-

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

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

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

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

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

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

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

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

9. T.C. of:

```
void function (int n)
{
  for (i=0 to n) {
    for (j=1; j<=n; j=j+i)
      printf ("#");
  }
}
```

O(n√n)

10. For the function n^k and a^n, what is the asymptotic relation b/w these function? Assume that $k >= 1$ and $a > 1$ are Constants.
Find out the value of c and n_0.
for what relation holds.

If $c > 1$ then the exponential c^n for eventually grows any term, So that answer is:
n^k is $O(c^n)$.

12. Write the recurrence relation for recursive function that prints Fibonacci series. Solve recurrence relation to get T.C. of program what will be the Space Complexity of the program and why?

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

$T(n-2) \approx T(n-1)$

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

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

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

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

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

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

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

⋮

General Term :-

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

$n - j = 0$

$n = j$

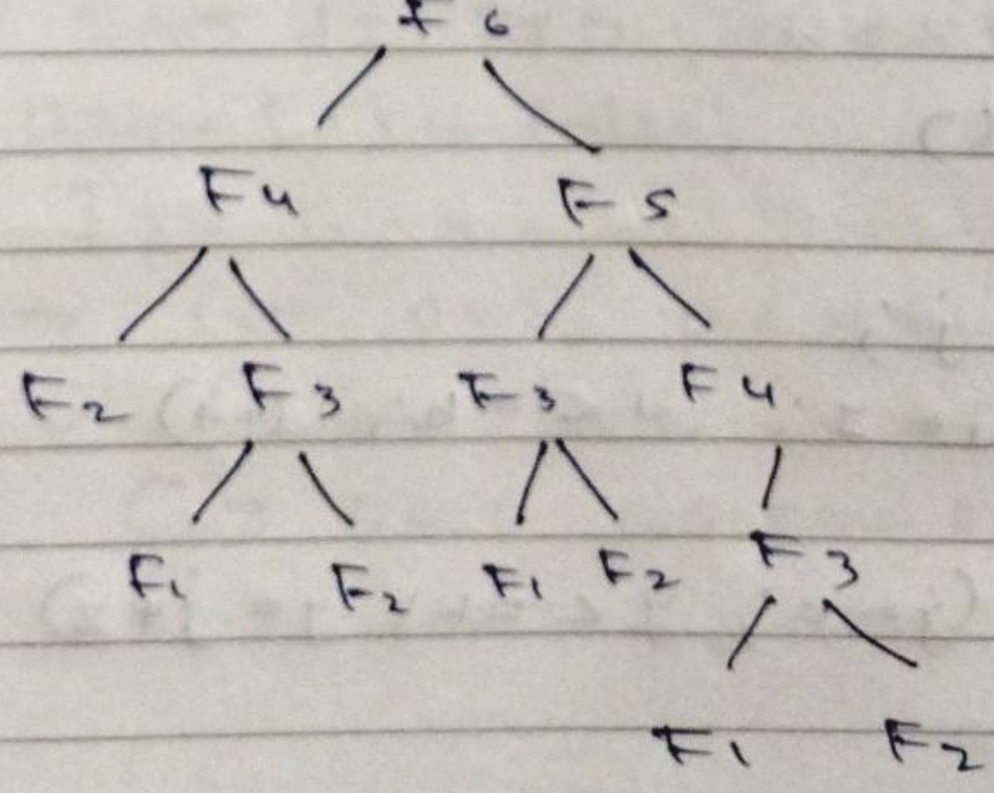
$T(n) = 2^n T(0) + (2^0 + 2^1 + 2^2 + \cdots 2^{n-1}) c$

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

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

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

fib. (6)



The max. depth is proportional to N, hence the Space Com. of Fibonacci recursive is $O(N)$.

11. What is the T.C of Code and why?

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

i = 0, 1, 3, 6, 10, 15, ------

j = 1, 2, 3, 4, 5, 6 ------

So, i will go n till n and general formula for K^{th} term is $n = \frac{K(K+1)}{2}$

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

13. Write programs which have T.C. :-

(b) 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 (logn)

→
```
void Sieve 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])
      Cout << p << endl;
    }
```

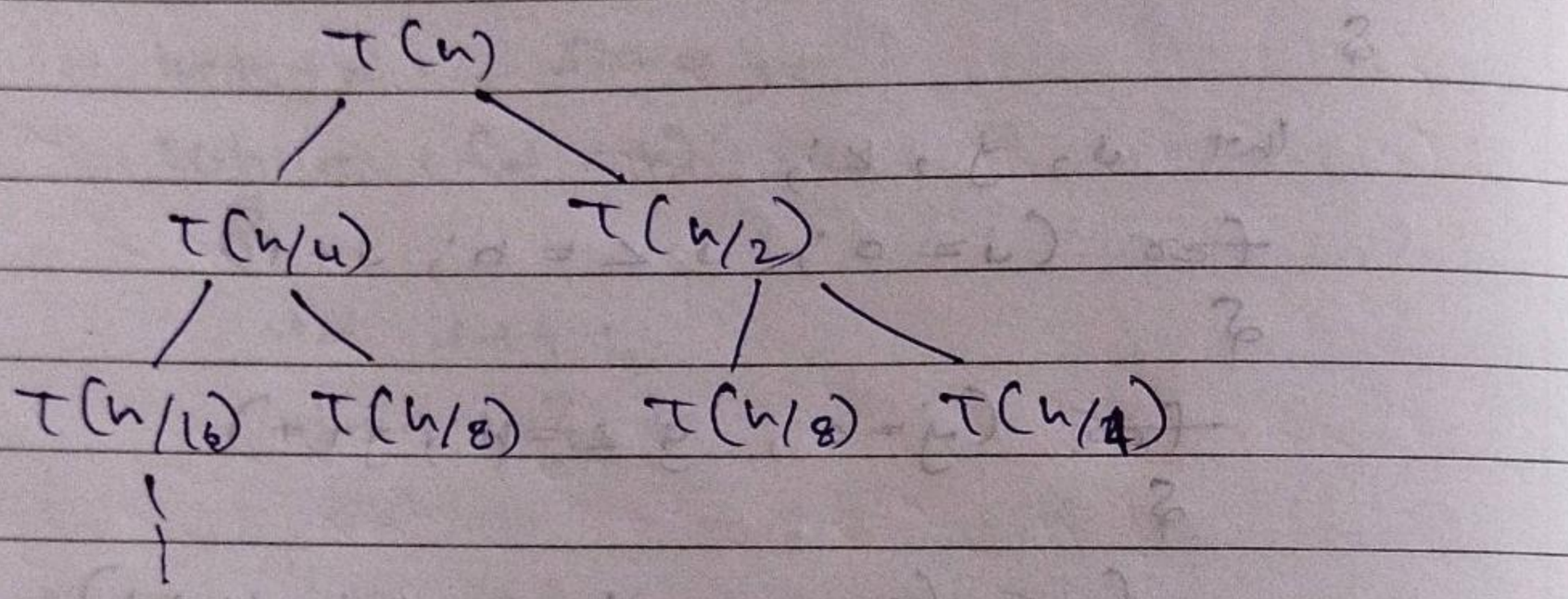
14. $T(n) = T(n/4) + T(n/2) + cn^2$

$T(1) = c$

$n = n/2$

$T(n/2) = T(n/8) + T(n/4) + c(n^2/4)$

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



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

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

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

15. T.C. of :-

```
int fun (int n)
{
  for (int i=1; i<=n; i++) ---- n
  {
    for (int j=1; j<n; j=j*i)
    {
      // Some O(1) task
    }
  }
```

For $i = 1$, inner loop is executed n times.
For $i = 2$, inner loop is executed $n/2$ times.
for $i = 3$, inner loop is executed $n/3$ times.

{

for $i = n$, inner loop is executed n/n times.

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

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

$= n \log n$.

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

16. T.C. of :-

```
for (int i=2; i<=n; i = pow(i, k))
{
    // some O(1) expressions.
}
```

where, k is a Constant.

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

18. Arrange in inc. order of rate of growth

(a) 100, log log n, log n, 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, $\log_8 n$, $\log_2 n$, log(n!), 5n, $n \log_6 n$, $n \log_2 n$, $8n^2$, $7n^3$, 8^{2n}, n!

19. Write linear Search pseudo code :-

```
Linear Search (A, Key)
    Comp ← 0, f ← 0
    for i = 1 to A length
    Comp ← Comp + 1
    if A[i] == Key
    print "Element found"
    f = 1
    if f == 0
        print "Element not found"
    Print Comp.
```

20. Write pseudacode for - - - - - - - - -

Ans Iterative Method of Insertion Sort →

Insertion 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] = Key

Recursive Method →

Insertion Sort (A, n)

if n ≤ 1

return

Insertion Sort (A, n-1)

Key = [n-1];

j = n-2;

while j ≥ 0 and A[j] > Key

A[j+1] = A[j]

j = j - 1

A[j+1] = Key

→ Insertion Sort Considers one i/p element per iteration and produces a partial Solution without Considering future elements That's why id is called online Sorting.

Other Sorting algos that have been discussed in lecture are :-

- Bubble Sort
- Selection Sort
- Quick Sort
- Merge Sort
- Heap Sort
- Counting Sort.

21. Complexity of all Sorting ---------

	Best Case	Average Case	Worst Case
Bubble Sort	$\Omega(N)$	$\Theta(N^2)$	(N^2)
Selection Sort	$\Omega(N^2)$	$\Theta(N^2)$	(N^2)
Insertion Sort	$\Omega(N)$	$\Theta(N^2)$	(N^2)
Merge Sort	$\Omega(N \log N)$	$\Theta(N \log N)$	$(N \log N)$
Heap Sort	$\Omega(N \log N)$	$\Theta(N \log N)$	$(N \log N)$
Quick Sort	$\Omega(N \log N)$	$\Theta(N \log N)$	(N^2)
Counting Sort	$\Omega(N+k)$	$\Theta(N+k)$	$O(N+k)$

22. Divide all Sorting algo into ------

	In Place	Stable	online.
Bubble Sort	Yes	Yes	Yes
Insertion Sort	Yes	Yes	Yes
Selection Sort	Yes	No	Yes
Merge Sort	No	Yes	Yes
Quick Sort	Yes	No	Yes
Heap Sort	Yes	No	Yes
Count Sort.	No	Yes	Yes

23. Write recursive (iterative -------

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 — $O(\log_2 n)$
Space Complexity — $O(1)$

Binary Search (Recursive) →

```
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 - $O(1)$

24. Write recurrence relation for binary recursive Search.

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