

Name :- Anuk Prasad

See - A

University Roll no. (96102)

Subject - DAD

Course - B.Tech (CS.B.)

Assignment - 1

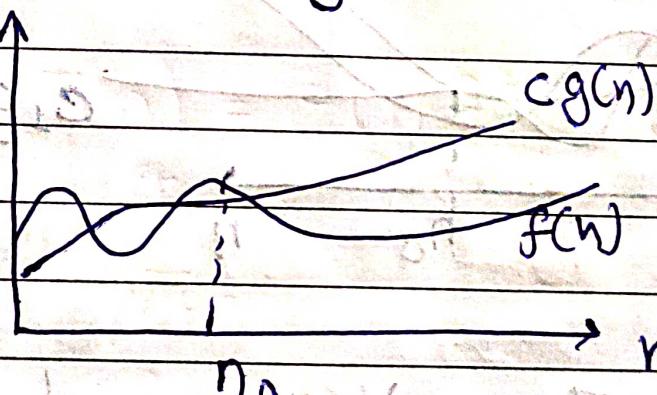
Assignment - 1

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

These notations are used for very large inputs.

1- Big-oh (O) -

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

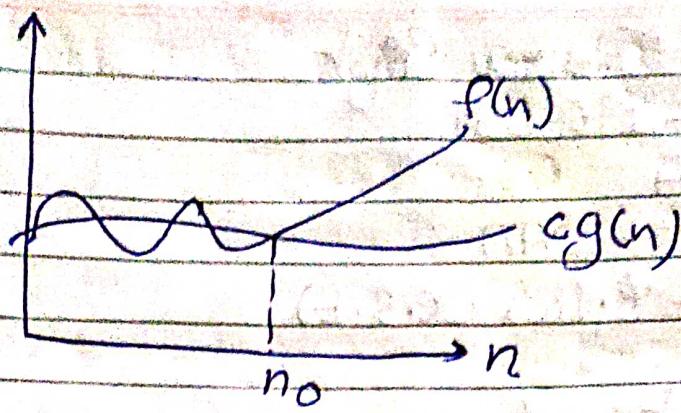


$$f(n) \leq c \cdot g(n) \quad \forall n \geq n_0, c > 0$$

$$\text{eg} - O(n^2 + 3n) = O(n^2)$$

2- Big Omega Notation (Ω)

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

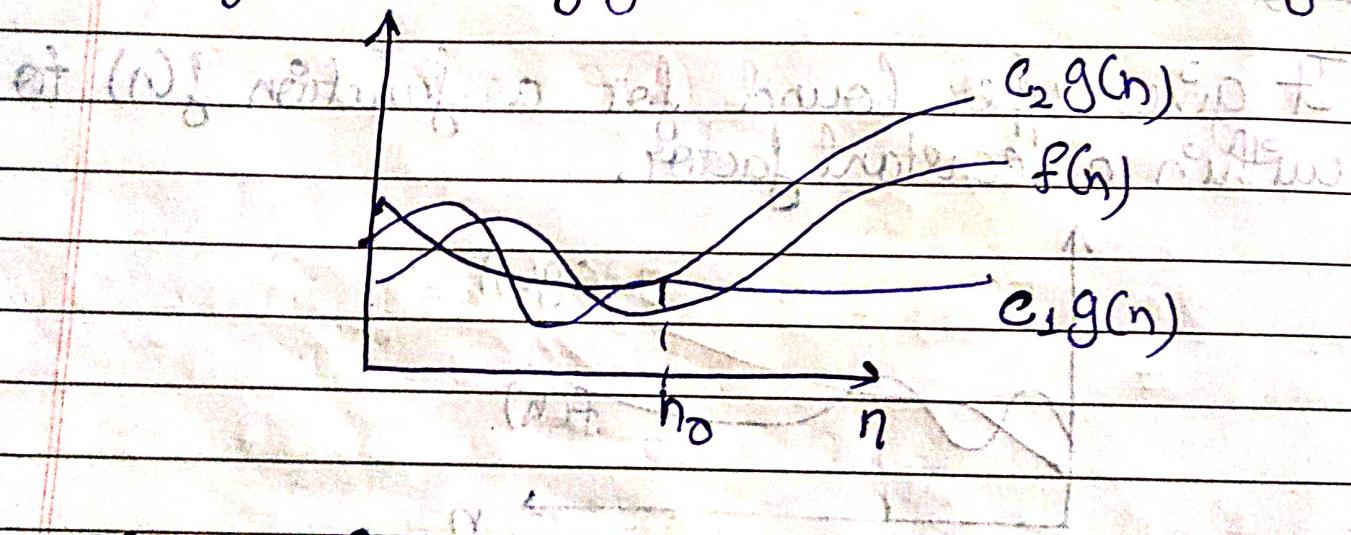


$\Omega(g(n)) = \{f(n) : \text{There exist } (+)\text{ve constant } c \text{ & } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \forall n \geq n_0\}$

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

3- Big theta Notation (Θ)

It gives bound of function within a constant factor.



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

eg - $\Theta(n^2)$

Ans 2 - for $i^2 \leq n$

$$(i^2 = i * i); \\ \{$$

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

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

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

$$T(n) = 3T(n-1) \quad \text{--- ①}$$

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

$$T(n) = 9T(n-2) \quad \text{--- ②}$$

$$T(n) = 3^3 T(n-3) \quad \text{--- ③}$$

$$T(k) = 3^k T(n-k) \quad \text{--- ④}$$

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

$$n-k=0$$

$$n=k$$

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

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

$$T(n) = \Theta(3^n),$$

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

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

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

$$T(n) = 4T(n-2) - 1 - 2 \quad \text{--- ②}$$

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

$$T(k) = 2^k T(n-k) - (1+2+4+\dots+2^{k-1})$$

$$T(n-k) = T(0) \quad k \text{ terms}$$

$$n = k$$

$$T(k) = 2^n T(0) - (1+2+4+\dots) \quad k \text{ terms}$$

It's a G.P

$$a = 1$$

$$r = 2$$

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

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

$$T(n) = 1$$

$$T(n) = O(1)$$

Ans 5- int i=1, s=1;
while (s <= n) {

i++;

s = s + i;

printf ("%#");

}

1, 3, 6, 10, 15, - - - - - n

← K terms →

Its kth term is $-k(k+1) / 2 + n$

$$K = \sqrt{n}$$

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

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) {

 if (n == 1) return;

 for (i = 1 to n) {

 for (j = 1 to n) {

 printf("*");

}

$$T(n)$$

$$n^2$$

}

$$T(n) = T(n-3) + n^2 - ①$$

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

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

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

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

(K-2) terms

for $T(n-k) = 1$

$$k = n-1$$

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

(n-3) terms

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

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

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

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

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

Ans 9- void function (int n) {

for ($i=1$ to n) {

 for ($j=1$; $j \leq n$; $j=j+1$)
 printf("%*s");

}

~~Outer loop - n times i=1 n times~~

~~Inner loop - 1, i=2, 1, 3, 5, ..., n~~

~~i=3 1, 4, 7, ..., n~~

~~i=n 0~~

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

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

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

$$k \geq 1 \text{ & } a > 1$$

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

Ans 11 - void fun (int n)

{
int $i^o = 1$, $x^o = 0$;
while ($x^o < n$)
{

$i^o = x^o + i^o$;
} $y^o = i^o + r$;

}

0, 3, 6, 10, 15, ..., n

so for this series is K. terms.

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

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

$k \approx \sqrt{n}$

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

Ans 12 - Recurrence relation of fibonacci series

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

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

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

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

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

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

for $T(n-2k) = T(0)$

$$n = 2k$$

$$R = n$$

$\frac{n}{2}$

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

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

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

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

n13. $\rightarrow n(\log n)$

if fun for (int $i=0$; $j < n$; $i++$ {
 for (int $i=0$; $i < n$; $i=i*2$)
 {
 cout << "*";
 }
 }
}
}

void main()

{

? fun();

$\rightarrow n^3$

```
#include <iostream.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++) {
```

```
                n++;
```

```
}
```

```
}
```

```
}
```

```
}
```

$\rightarrow \log(\log n)$

```
#include <bits/stdc++.h>
```

```
void fun(int n)
{
```

```
    if (n == 2)
```

```
        return 1;
```

```
    else
```

```
        fun(sqrt(n));
```

```
}
```

```
void main()
```

```
{
```

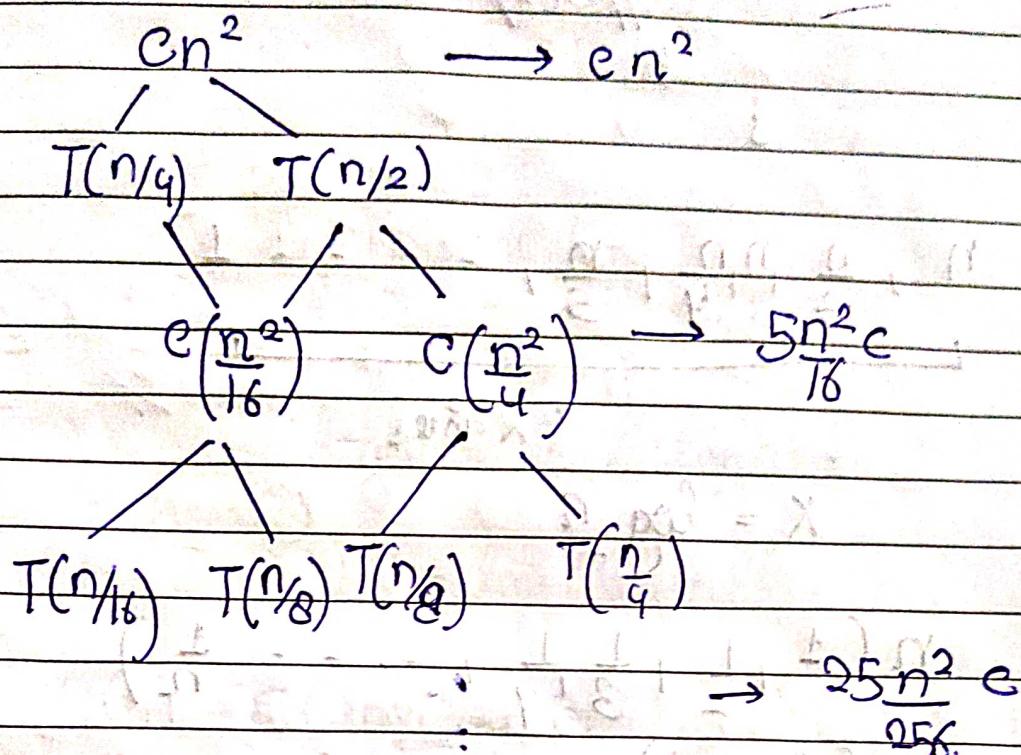
```
    fun(100);
```

```
}
```

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

$$T(1) = 0$$

$$T(0) = 0$$



$T(n)$ = Cost of each level

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

it is a G.P

$$\text{with } a = n^2$$

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

∴ So sum of S.P.

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

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

Ans 15 - $\text{for } (\text{int } i \text{ to } n)$

$\text{for } (\text{int } j = 1; j < n; j++)$

// O(1)

{ }

$$n, \frac{n}{2}, \frac{n}{3}, \frac{n}{4}, \frac{n}{5}, \dots, 1$$

k times

$$k = \log_2 n$$

$$\Rightarrow n \left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n} \right)$$

$$(n \log n)$$

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

Ans 16 - $\text{for } (\text{int } i = 2; i \leq n; i = \text{pow}(i, k))$

// O(1)

{ }

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

$$\text{If G.P } a = 2$$

$$\gamma = 2^k$$

$$k^{\text{th}} \text{ term} = a \gamma^{k-1}$$

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

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

$$\text{let } k^{(k-1)} = n$$

$$k \log k = \log n$$

$$k = \log n \rightarrow ①$$

$$n = 2^x$$

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

$$x = \log_2 n$$

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

from ①

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

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

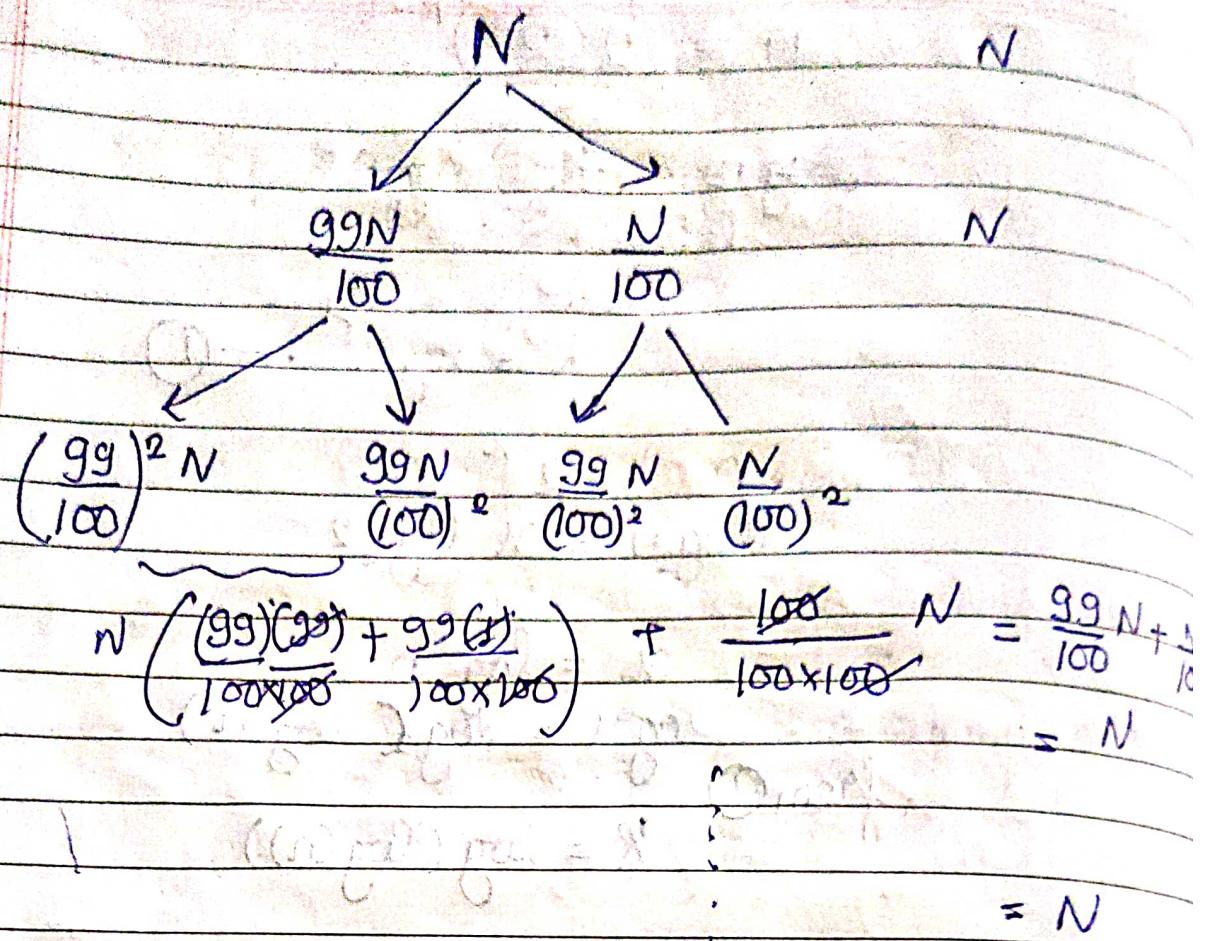
Ans 17-

hence 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 extremes of a tree
where starting point is N



So cost of each level is N only.

* Total cost = height * Cost of each level.

So for 1st stem - $N, \frac{99N}{100}, \frac{(99/100)^2 N}{100}, \dots$

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

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

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

$$\log N = h \log \left(\frac{100}{99}\right)$$

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

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

height of 2nd stream

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

$$\frac{N}{100^{h-1}} = 1$$

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

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

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

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

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

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

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

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

and when division ratio is less then height is less.

a) n , $n!$, $\log n$, $\log \log n$, $2^{\log(n)}$, $n \log n$
 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) $2(2n)$, $4n$, $2n$, 1 , $\log(n)$, $\log(\log(n))$, $\log(n!)$, $n \log n$,
 $\log 2n$, $2 \log n$, ~~$\log n!$~~ , $\log(n!)$, $n!$, n^2 ,
 $n \log 6$)

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

c) 8^{12n} , $\log n$, $n \log_8(n)$, $n \log_2(n)$, \log
 $\log_8(n)$, 96 , $8n^2$, $7n^3$, $5n$

Ans $O(96) < O(\log_8(n)) < O(\log n) < O(\log(n!))$
 $O(n \log_8(n)) < O(n \log_2(n)) < O(5n) < O(8)$
 $< O(7n^3) < O(n!) < O(8^{12n})$.

Ans 19-

Void Linear Search (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 $n \leq 1$

return,

insertionsort (arr, n-1),

last = arr[n-1],

j° = n - 2

while ($j^{\circ} \geq 0$ and $arr[j^{\circ}] > last$)

{

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	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
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	No	No	No
Heap Sort	Yes.	No.	No.

$$L + (nT) = (nT) - 15.84$$

Ans 21 -

Time complexitySpace

	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	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
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 -

inplacestableOnline sorting

Bubble Sort	Yes	Yes	No
Selection Sort	Yes	No	No
Insertion Sort	Yes	Yes	Yes
Merge Sort	No	Yes	No
Quick Sort	No	No	No
Heap Sort	Yes.	No.	No.

An8 23-

Binary Search (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 = $O(n)$

28 24 - $T(n) = T(n/2) + 1$

2 //