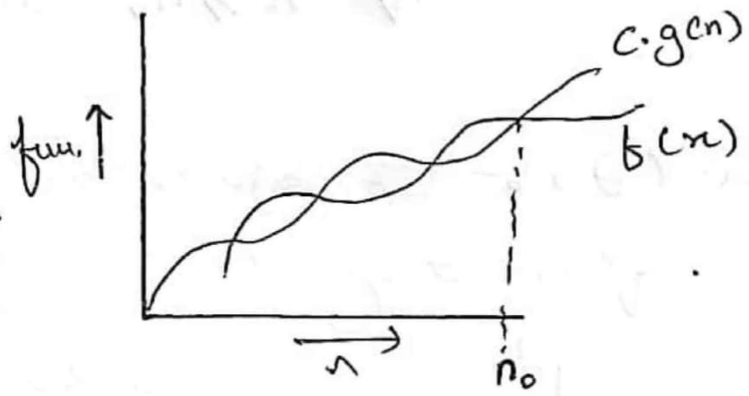
ASSIGNMENT - 1

Q1 The notations that are used to tel the complexity of an algorithm when input is very large is known as asymtotic notation.
The various types of asymtotic notatio are :-

(i) Big - Oh (O) :-

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

g(n) is tight upper bound f



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

iff

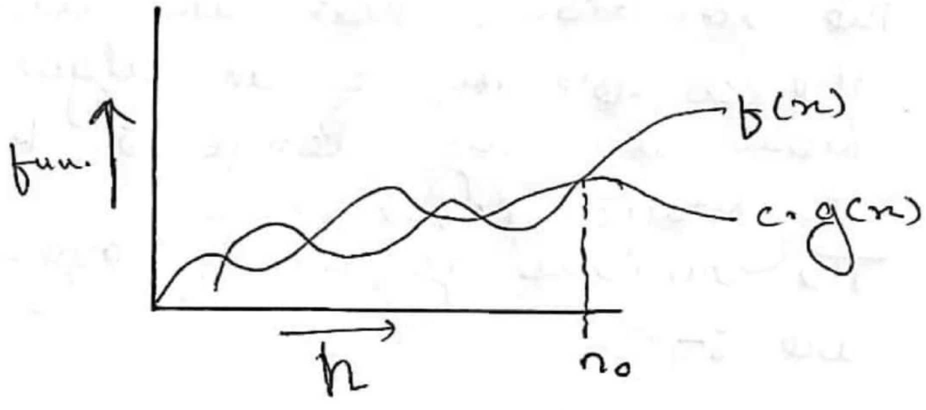
$f(n) <= c \cdot g(n)$

$\forall\ n >= n_0$ and $c > 0$.

(ii) Big - Omega (Ω) :-

$f(n) = \Omega(g(n))$

g(n) is 'tight' lower bound of f



$f(x) = \Omega(g(x))$

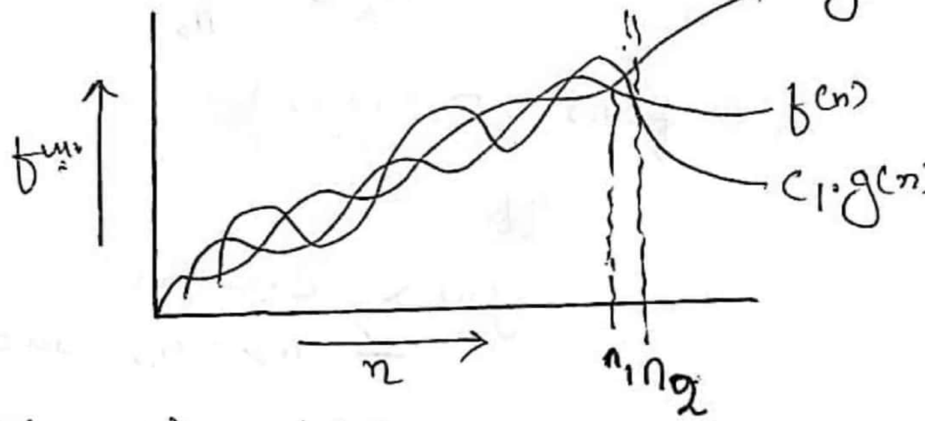
iff.

$f(x) > c \cdot g(x)$

$\forall\ x \geq n_0$ and $c > 0$.

(iii) Theta (θ) :- It gives both upper & lower

$f(n) = \theta(g(n))$



$\theta(g(n)) = f(n)$

iff.

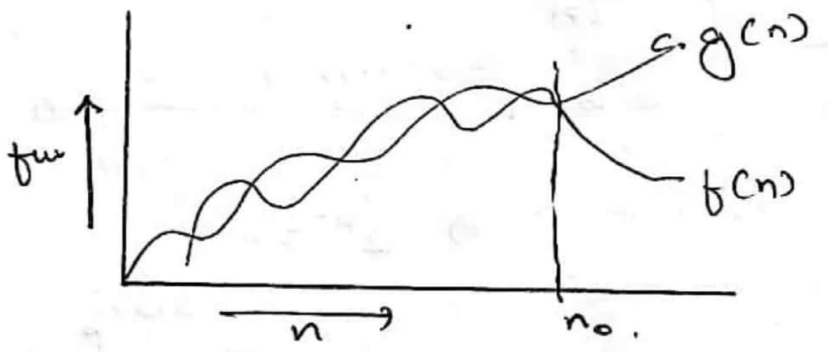
$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

$\forall$ ~~[illegible]~~ ~~[illegible]~~ $c_1, c_2 > 0$. an

$\max(n_1, n_2) \geq n$

(iv) small - oh (o) :-

It gives the upper bound of the fu



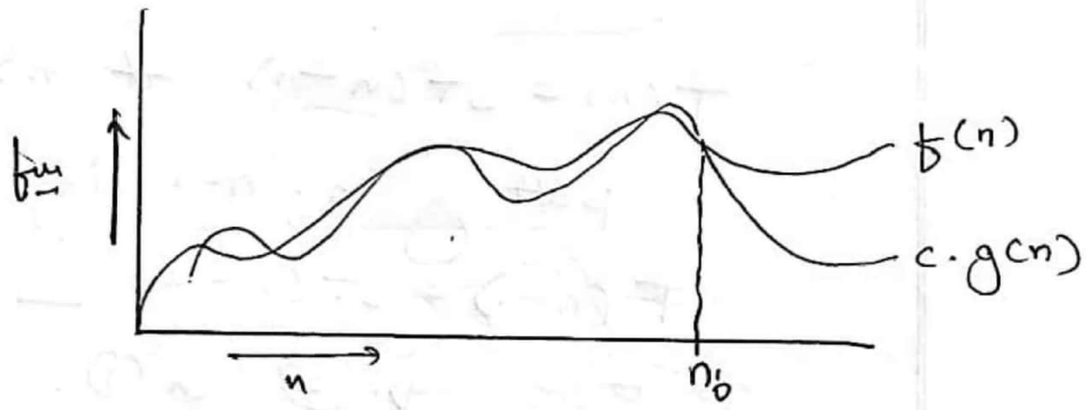
$f(n) = o(g(n))$

iff.

$f(n) < c.g(n) \quad \forall \quad n > n_0$ and $c > 0$.

(v) small omega (ω) aka rho :-

It gives the lower bound of the function (or algorithm).



$f(n) = \omega(g(n))$

iff

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

$\forall \; n > n_0$ and $c > 0$.

Q2 for (i=1 to n)
i = i*2

$\sum_{i=1}^{n} \text{step}^{*2}_{i}$

$\Rightarrow$ 1, 2, 4 ---- , n (k terms).

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

taking log

$k-1 = \log(n).$

$k = \log(n) + 1.$

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

Q3

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

$T(n) = 3T(n-1) \quad \forall \; n>0.$ — (1)

putting $n = n-1$ in eq. (1).

$T(n-1) = 3T(n-2)$ — (2)

putting eq. (2) in (1).

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

$= 3^2 \cdot T(n-2)$ — (3)

putting $n = n-2$ in eq. (1).

$T(n-2) = 3T(n-3)$ — (4)

putting eq (4) in (3).

$T(n) = 3^3 T(n-3).$ — (5)

$\Rightarrow T(n) = 3^k T(n-k)$ — (6)

Base case
$T(0) = 1$

$\Rightarrow n - k = 0$
$n = k$

putting ~~[illegible]~~ ~~[illegible]~~ $k=n$ in eq. (6).

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

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

Q4

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

$T(n) = 2T(n-1) + 1 \quad \forall\ n > 0.$ — (1).

putting $n = n-1$ in eq (1).
$T(n-1) = 2T(n-2) - 1$ — (2).
putting eq. (2) in (1).
$T(n) = 2(2T(n-2) + 1) - 1$
$= 2^2 T(n-2) + 1 + 2$ — (3).

putting $n = n-2$ in eq (1).
$T(n-2) = 2T(n-3) + 1$ — (4).
putting eq (4) in eq. (3).
$T(n) = 2^2 (2T(n-3) + 1) + 1 + 2$
$= 2^3 T(n-3) ~~+~~ (1 + 2 + 2^2)$

$\Rightarrow T(n) = 2^k T(n-k) - (1+2+2^2 --- +2^k)$. — (5)

~~[illegible]~~

Base case $\Rightarrow T(0) = 1$

$n - k = 0$

$k = n$

putting $k=n$ in eqn. (5)

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

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

$= 2^n - \frac{2^n - 1}{2-1}$

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

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

$\Rightarrow T(n) = O(1)$.

5.)
```
int i=1, s=1;
 while (s<=n)
 {
     i++; s=s+i;
     printf("#");
 }
```

i	s
1	1
2	3
3	6
4	10

i $\boxed{1}$ 2 ~~3~~ 4 s $\boxed{1}$ ~~3~~ ~~6~~ 10

1, 3, 6, 10 --- n terms

$\Rightarrow S = 1 + 3 + 6 + 10 + --- + k$.

$S = 1 + 3 + 6 + 10 --- + k_{n-1} + k_n$

$\ominus S = \ominus 1 \ominus 3 \ominus 6 \ominus 10 --- \ominus k_{n-1} \ominus k_n$

$0 = 1 + 2 + 3 + 4 ---- - k$

~~[illegible]~~ $k(k-1)$ ~~[illegible]~~

$\Rightarrow$ ~~k~~ $n \approx k^2$

$k = \sqrt{n}$

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

Q6

```
~~f~~ void fuction (int n) {
    int i, count = 0;
        for(i = 1; i*i <= n; i++)
            count ++;

}
```

$1, 2, 3, \text{——} \sqrt{n}$

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

Q7

```
void fuction (int n) {
    int i, j, k, count = 0;
    for (int i = n/2 ; i<=n; ++i)
        for(j=1; j<=n; j=j*2)
            for (k=1; k<=n; k=k*2)
                count ++;

}.
```

$$\sum_{i=n/2}^{n} \sum_{j=1}^{n\,(j*2)\text{ steps}} \sum_{\substack{k=1 \text{ ~~steps~~}\\ \text{step}=k*2}}^{n} 1.$$

$$\sum_{i=n/2}^{n} \sum_{\substack{j=1\\ \text{step } j*2}}^{n} \log(n).$$

$$\sum_{i=n/2}^{n} (\log(n))^2$$

$(n/2+1)(\log(n))^2 \Rightarrow T(n) = O(n(\log$

Q8

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

$$\sum_{i=1}^{n} \sum_{j=1}^{n} 1 = \sum_{i=1}^{n} n = n^2$$

$T(n) = T(n-3) + n^2$ — (1)

putting $n = n-3$ in eq (1)

$T(n-3) = T(n-6) + (n-3)^2$ — (2)

putting (2) in eq (1).

$T(n) = T(n-6) + (n)^2 + (n-3)^2$ — (3)

putting $n = n-6$ in eq (1).

$T(n-6) = T(n-9) + (n-6)^2$ — (4)

putting eq (4) in (3).

$T(n) = T(n-9) + n^2 + (n-3)^2 + (n-6)^2$

$\Rightarrow T(n) = T(n-3k) + n^2 + (n-3)^2 + \cdots$ ~~[illegible]~~ $+ (n + 3(k-1))^2$

$T(1) = 0.$

$n - 3k = 1$

$k = \frac{n-1}{3}.$

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

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

Q9

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

$$\sum_{i=1}^{n} \sum_{\substack{j=1 \\ \text{Step}=i}}^{n} 1$$

$n = 1+(k-1)i$

$\frac{n-1}{i}+1 = k$

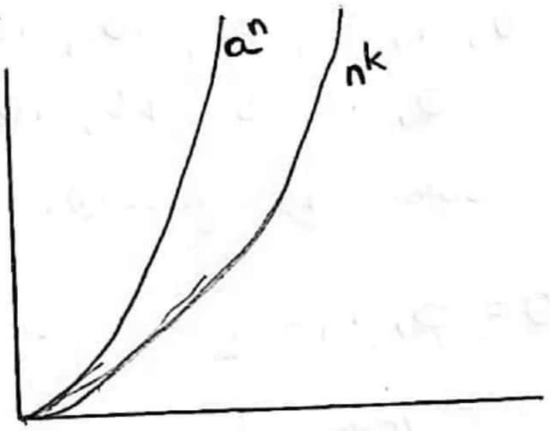
$$\sum_{i=1}^{n} \left(\frac{n-1}{i}+1\right)$$

$$(n-1)\sum_{i=1}^{n} 1/i + \sum_{i=1}^{n} 1.$$

$$(n-1).\log n + n.$$

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

Q10



$\Rightarrow n^k = O(a^n)$

$\therefore n^k \leqslant a^n.c \quad \forall c>0$ and $n \geqslant n_0$

let $n=n_0$.

$\Rightarrow n_0^k \leqslant c.a^{n_0}$

$n_0^3 \leqslant c.3^{n_0}$. $k=a=3$ (say)

$\Rightarrow c \geqslant 1$ & $n_0 \geqslant 1$.

Q11

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

$n = \boxed{20.}$

j	i
1	0
2	1
3	3
4	6
5	10
6	15
7	21

$S = 0, 1, 3, 6, 10, 15 \dots k$ — ①

$S = \quad 0, 1, 3, 6, 10 \dots\dots + k$ — ②.

Sub. ② from ①.

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

$n = 0 + 1 + 2 \dots k.$

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

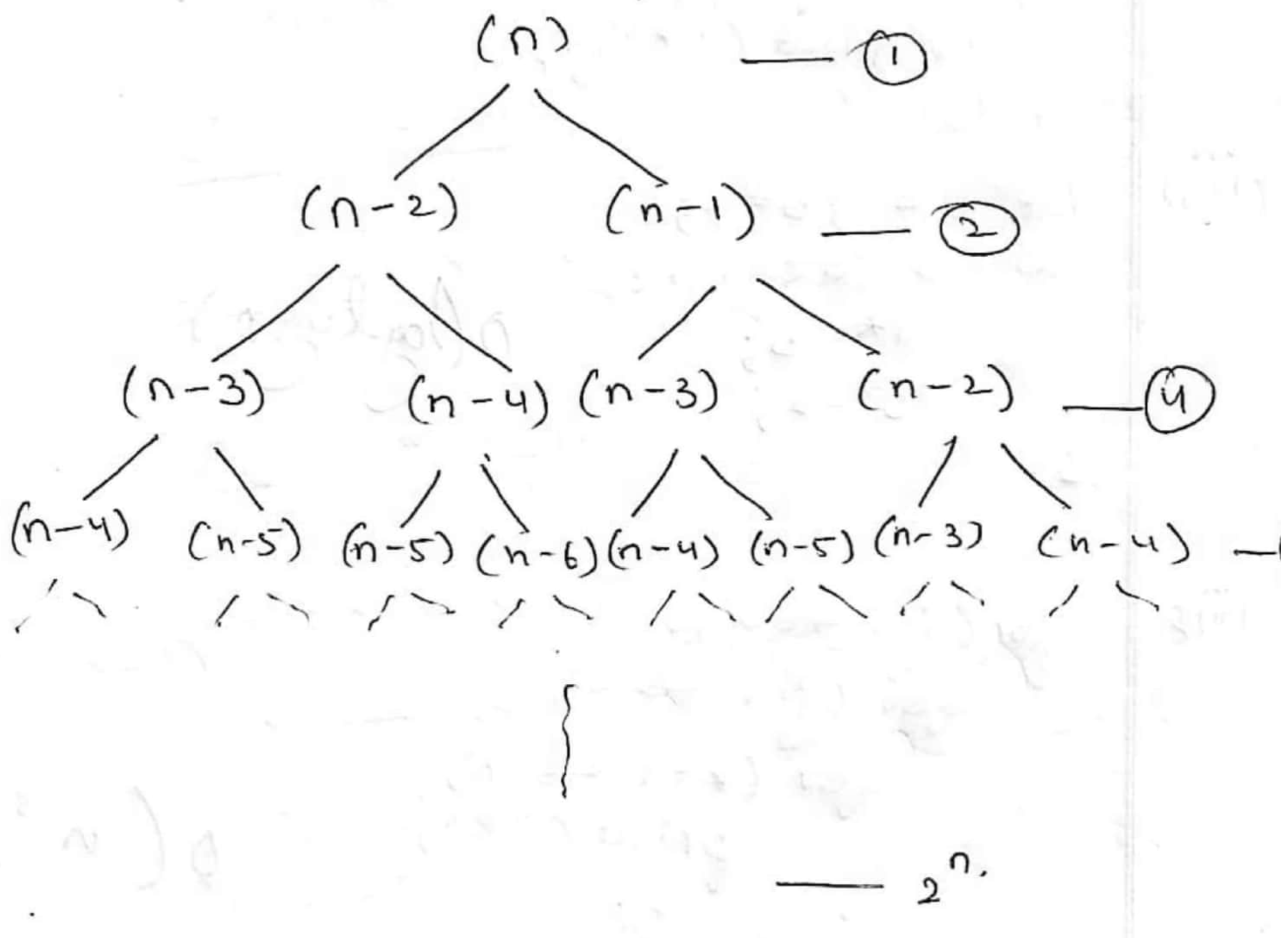
$\Rightarrow n \simeq k^2$

$k = \sqrt{n}$

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

Q12 $0, 1, 1, 2, 3, \text{——}, T_n.$

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



(n) — ①
(n-2) (n-1) — ②
(n-3) (n-4) (n-3) (n-2) — ④
(n-4) (n-5) (n-5) (n-6) (n-4) (n-5) (n-3) (n-4)
— 2^n.

$$T = 1+2+4+ \text{——} + 2^n$$

$a = 1$
$r = 2.$

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

$$= 2^{n+1} - 1$$

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

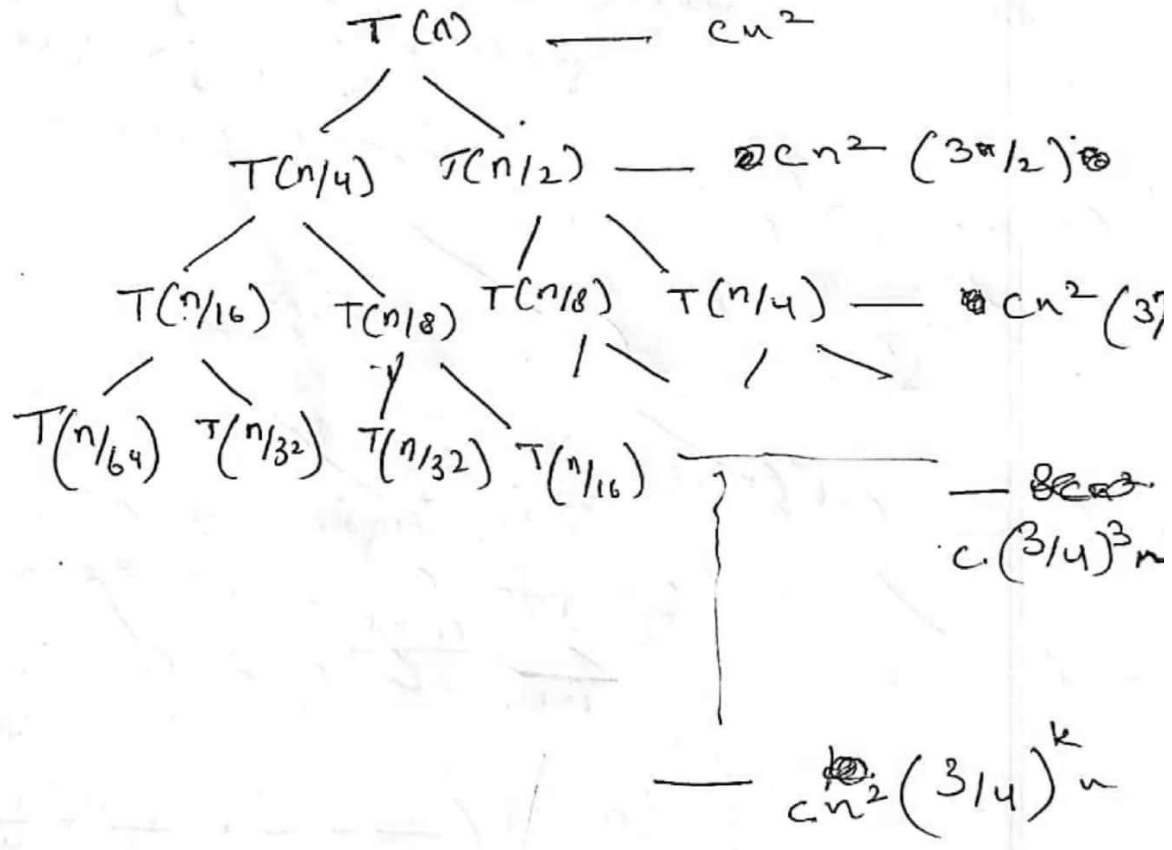
Q13

(i)
```
for (int i=0; i<=n, ++i)
    for (int j=1; j<=n; j*=2)
        print ("*");
```
O(n log n)

(ii)
```
int a=1, b=2;
while (a<=n) {
        a*=b;
        b*=2;
}.
```
O(log log n)

(iii)
```
for (i=1 to n) {
    for (j=1 to n) {
        for (k=1 to n) {
            print ("*");
        }
    }
}
```
O(n³)

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


T(n) — cn²
T(n/4) T(n/2) — cn² (3/2)
T(n/16) T(n/8) T(n/8) T(n/4) — cn² (3
T(n/64) T(n/32) T(n/32) T(n/16)
c.(3/4)³n
— cn²(3/4)ᵏ n

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

$n = 2^k$

$k = \log n$

$T(n) = cn^2 \left[1 + \left(\frac{3}{4}\right) + (3/4)^2 + \dots \right.$

$(3/4)^{\log n}$

$= cn^2 \cdot (1)$

$= n^2$

$\Rightarrow T(n) = O(n^2).$

15

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

$$T(n) = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ \text{step } i}}^{n-1} (1)$$

$$= \sum_{i=1}^{n} \frac{n-1}{i}$$

$$= (n-1)\left(\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} - - - n\right)$$

$$= (n-1)\log n$$

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

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

Q16

```
for(int i=2; i<=n; i=pow(i,k)) {
        Some O(1) expression
}
```

$$i = 2 \quad 2^{k} \quad 2^{k^2} \quad 2^{k^3} \quad \text{------}$$

$$\Rightarrow ~~T(n)~~ \; 2^{k^x} = n$$

$$\log(n) = k^x \log 2$$

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

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

$$\Rightarrow x = \log(\log(n$$

$\therefore$ k is constant

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

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

$T(1) = 0$ — ①

putting $n = \frac{99}{100}n$ in eq ①.

$T(99/100\, n) = T\left(\left(\frac{99}{100}\right)^2 n\right) + n/100$ —

putting eq ② in ①.

$T(n) = T\left(\left(\frac{99}{100}\right)^2 n\right) + 2n/100.$ — ③

|

$T(n) = T\left(\left(\frac{99}{100}\right)^k n\right) + kn/100$ — ④

$\left(\frac{99}{100}\right)^k n = 1.$

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

$k = \log_{\frac{100}{99}} n.$ — ⑤

putting $k = \log_{\frac{100}{99}} n$ in eq ④.

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

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

18

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

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

(c). $96 < \log_2(n) = \log_8(n) < n\log_6(n) = n\log$
$= \log(n!) < 5n < 8n^2 < 7n$
$< 8^{2n}.$

Q19

```
INPUT ARR[N], KEY;
for (i = 0 to n-1) {
    if (ARR[i] = key) {
        return i;
    }
}
return -1;
```

Q20 (a) Iterative Insertion Sort

```
void InsertionSort (int arr[], int n)
{
    int i, temp, j;
    for (i = 1 to n-1) {
        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;
    }
}.
```

(b) Recursive ~~Sort~~ Insertion Sort

```
void InsertionSort (int arr[], int n)
{
    if (n<2)
        return.

    InsertionSort (arr, n-1);
    last = arr[n-1], j=n-2;
    while ( j>=0 AND arr[j] > temp {
            arr [j+1] = arr [j];
            j = j-1;
    }
    arr [j+1] = last;
}
```

Insertion sort is online algorithm because it process the element one-by-one in a serial fashion without considering the future elem Whereas bubble sort, selection sort and merge sort are offline as they require all inputs on which they can process the ~~data~~ for correct output i.e these algorithms want all the input beforehan

Q21

	Algorithm	Best Case	Avg. 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	Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
4	Merge Sort	$O(n\log n)$	$O(n\log n)$	$O(n\log$
5	Quick Sort	$O(n\log n)$	$O(n\log n)$	$O(n^2)$
6	Heap Sort	$O(n\log n)$	$O(n\log n)$	$O(n\log$

Q22

	Algorithm	In-place	Stable	Online
1	Bubble Sort	✓	✓	✗
2	Selection Sort	✓	✗	✗
3	Insertion Sort	✓	✓	✓
4	Merge Sort	✗	✓	✗
5	Quick Sort	✗	✗	✗
6	Heap Sort	✓	✗	✗

Q23 Iterative Binary search

```
int BinarySearch (int arr[], int l, int r, int
{
    while (l<=r){
        int m = (l+r)/2.;
        if (arr[m]= x)
            return m;
        else if (arr[m] < x)
            l= m+1;

        else
            r = m-1;
    }
    return -1;
}
```

Recursive Binary Search

```
int BinarySearch(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)
    return BinarySearch(arr, m+1, r, x);
    else
        return BinarySearch (arr, l, m-1,
}
```

→ Iterative Binary Search

(c) Time Complexity

Best Case = $O(1)$

Average Case = $O(\log n)$

Worst Case = $O(\log n)$.

(d) Space Complexity = $O(1)$
(All cases)

→ Recursive Binary Search

(c) Time Complexity

Best Case = $O(1)$

Average Case = $O(\log n)$

Worst Case = $O(\log n)$

(d) Space complexity :

Best Case = $O(1)$

Average Case = $O(\log n)$

Worst Case = $O(\log n)$

Q4 $T(n) = T(n/2) + 1$

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