Assignment - 01

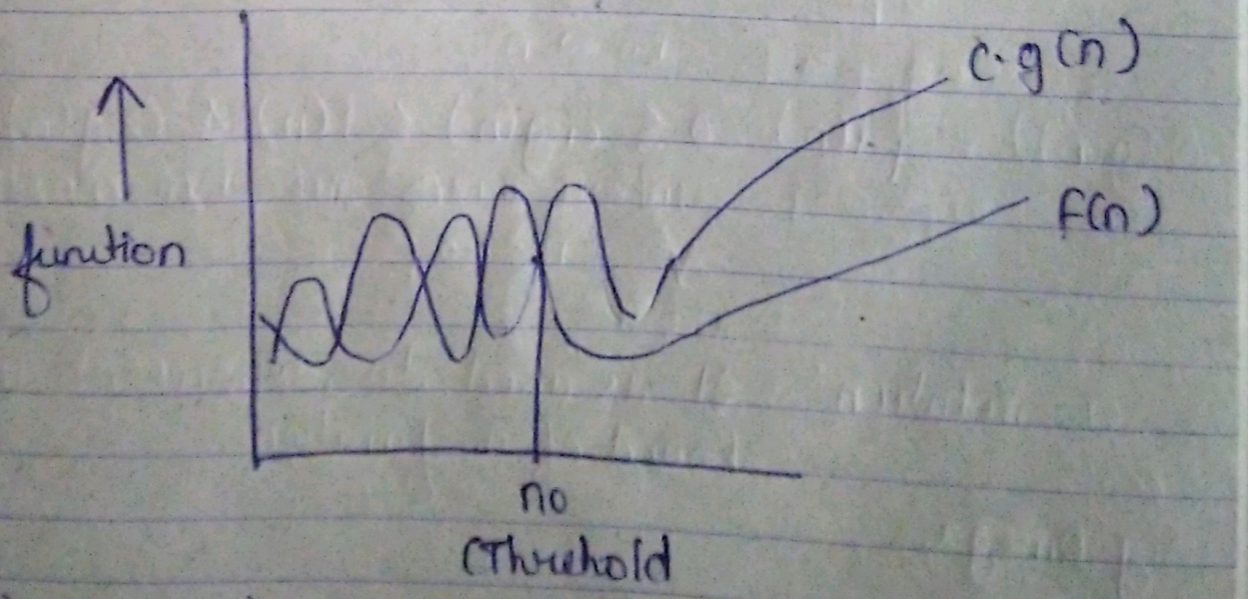
Name :- Ashish Sah
Course :- Btech (C.S.E)
Section :- B
Roll No :- 1961025

Q1 Asymptotic notation :- These notations are used to tell the complexity of an algorithm when the input is very large or we can say towards infinity.

Types :-

1) Big - O

Big - O commonly wrote as O, is an asymptotic notation for the worst case or the ceiling of growth for a given function, that means the function's complexity will not cross the growth of the Asymptotic notation in any case



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

$g(n)$ is the "tight" upper bound of $f(n)$

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

iff

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

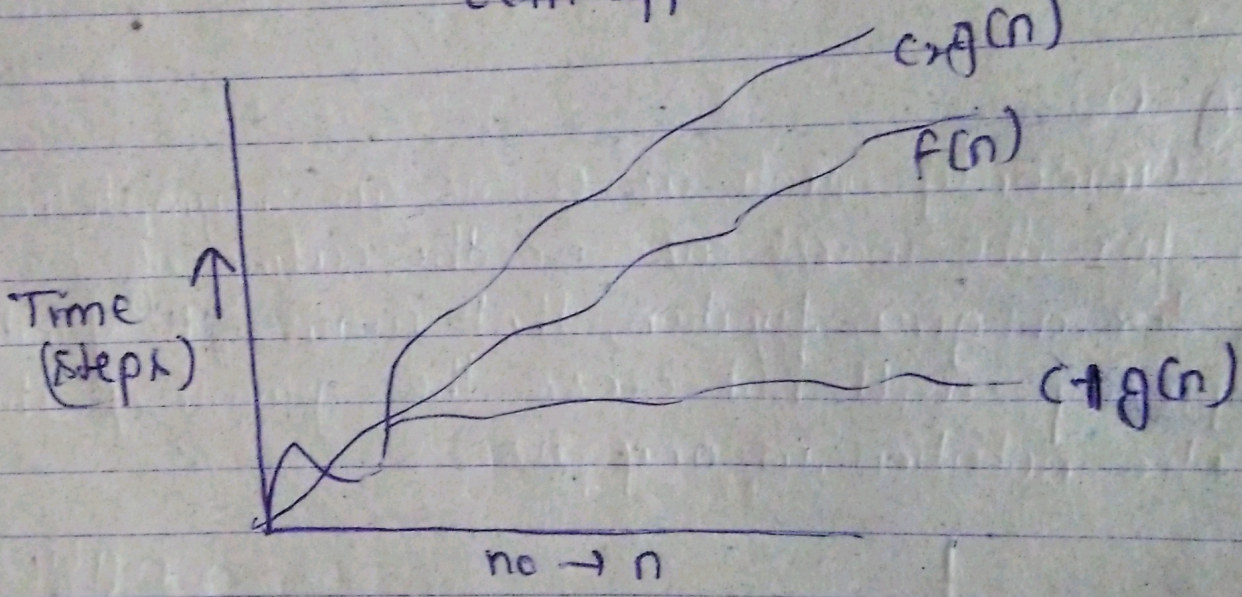
$\forall\ n \geq n_0$, some constant $c > 0$.

Ex:

Algo 1 $\longrightarrow$ $O(n^2+3n+4) \longrightarrow O(n^2+n)$

Algo 2 $\longrightarrow$ $O(2n^2+4) \longrightarrow O(n^2)$

② Θ-notation :- Theta notation used to represent both upper and lower bounds

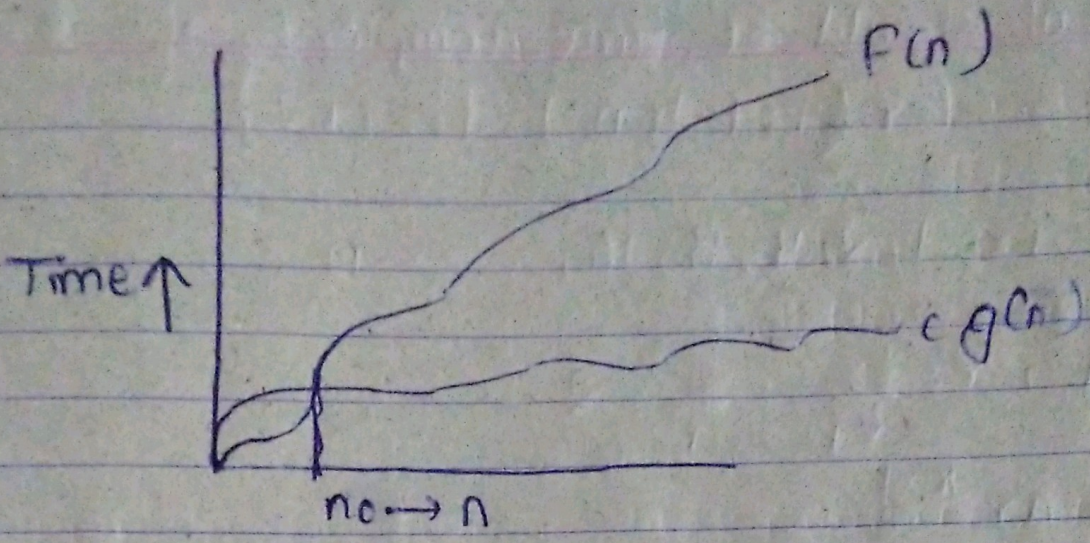


$\Theta(g(n)) = \{ f(n) : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$
where c_1, c_2 are +ve constants
$\forall\ n \geq n_0 \}$

③ Ω-notation :- It is used to represent lower Bound of a function

↙ Big omega

* This notation is used for the best case for a given function.



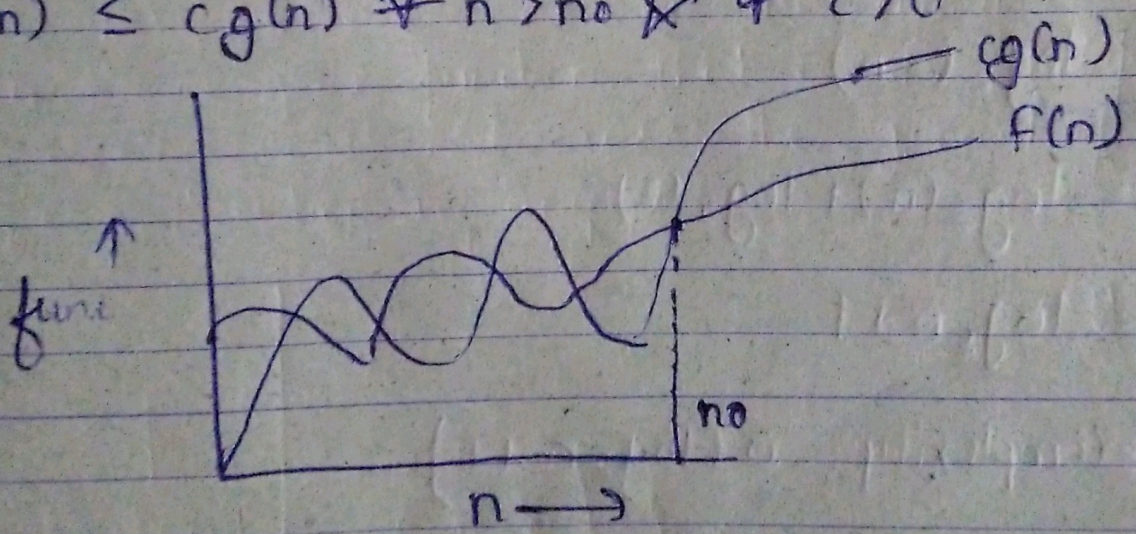
$\Omega(g(n)) = \{f(n) : 0 \leq cg(n) \leq f(n)$ where c is a +ve constants & $\forall\ n \geq no\}$.

④ Small-oh (o) :-

o gives us upper bound

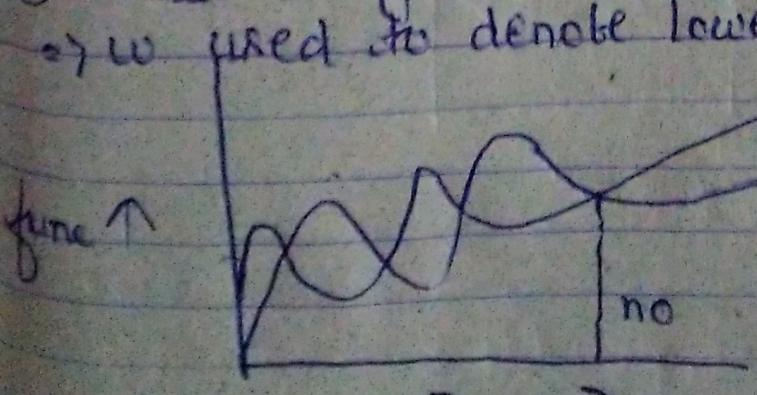
$f(n) = o(g(n))$

$f(n) \leq cg(n)\ \forall\ n > no$ & $\forall\ c > 0$



⑤ Small omega (ω)

⇒ ω used to denote lower bound



Lower bound

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

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

$\forall\ n > no$ & $\forall\ c > 0$

Q2 What should be time complexity of following code.

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

Ans $i = 1, 2, 4, 8, 16, \ldots n$

$a = 1, \quad r = \frac{t_2}{t_1} = \frac{2}{1} = 2$

$t_k = ar^{k-1}$

$n = (1)(2)^{k-1}$ $\quad \left[a^{b-c} = \frac{a^b}{a^c} \right]$

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

$2^k = 2n$

$k = \log_2(2n)$

$= \log_2(n) + \log_2(2)$

$k = \log_2 n + 1$

Time complexity $= O(\log_2 n + 1)$

$= O(\log_2 n)$

Q3 $T(n) = \{3T(n-1)$ if $n>0$, otherwise $1\}$

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

$T(n) = 3T(n-1)$ — ①

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

$T(n) = 9T(n-2)$ — ②

$T(n) = 3^3 T(n-3)$ — ③

$T(k) = 3^k T(n-k)$ — ④

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

$n-k=0$

$n=k$

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

$T(n) = 3^n$

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

Q4 $T(n) = \{2T(n-1)-1$, $n>0$ $\quad 1$, $n=0\}$

$T(n) = 2T(n-1)-1$ — ①

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

$T(n) = 4T(n-2)-1-2$ — ②

$T(n) = 8T(n-3)-(1+2+4)$ — ③

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

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

$n = k$

$T(k) = 2^n T(0) - (1 + 2 + 4 + \cdots)$

k terms

It's a G.P

$a = 1$

$r = 2$

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

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

$T(n) = 1$

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

Qs

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

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

← k terms →

It's k th term is $\frac{k(k+1)}{2} = n$

$k = \sqrt{n}$

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

Q6 Void function (int n)
{ int i, count = 0;
for (int i = 1; i * i < n; i++)
count ++;
}
$T(n) = O(\sqrt{n})$

Q7 $T(n) = O(n * \log_2 n * \log_2 n)$
$T(n) = O(n * (\log_2 n)^2)$
$T(n) = O(n (\log n)^2)$

Q8
```
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)
}
```

$T(n) = T(n-3) + n^2$ ——— (1)
$T(n-1) = T(n-4) + (n-1)^2$
$T(n) = T(n-4) + n^2 + (n-1)^2$
$T(n) = T(n-5) + n^2 + (n-1)^2 + (n-2)^2$

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

for $T(n-k) = 1$

$k = n-1$

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

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

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

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

$T(n) = n^3$

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

Q9

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

i = 1 n times

i = 2 1, 3, 5, ---n n

i = 3 1, 4, 7, ---n n/2

i = n 0

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

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

Q10 For the functions n^k and a^n, what is the relation

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

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

Ans

Q11
```
void fun(int n)
{ int j=1, i=0;
  while (i<n)
  {   i = i+j;
      j++;
  }
}
```

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

k terms.

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

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

$$k \simeq \sqrt{n}$$

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

Q12 Recurrance relation of fabonacci series is

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

$$T(n) = 2T(n-2) + 1$$
$$T(n) = 4T(n-4) + 3$$
$$T(n) = 8T(n-6) + 7$$
$$T(n) = 16T(n-8) + 15$$

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

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

$n = 2k$

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

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

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

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

Space complexity of fabonacci series is $O(n)$ as it depends on height of recursive tree and it is equal to n in fabonacci series.

Q13 $n(\log n)$

```
void fun()
{ for(int j=0; j<n; j++)
  {
      for(int i=0; i<n; i=i*a)
      {
        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++)
            {
                n++;
            }
        }
    }
}
```

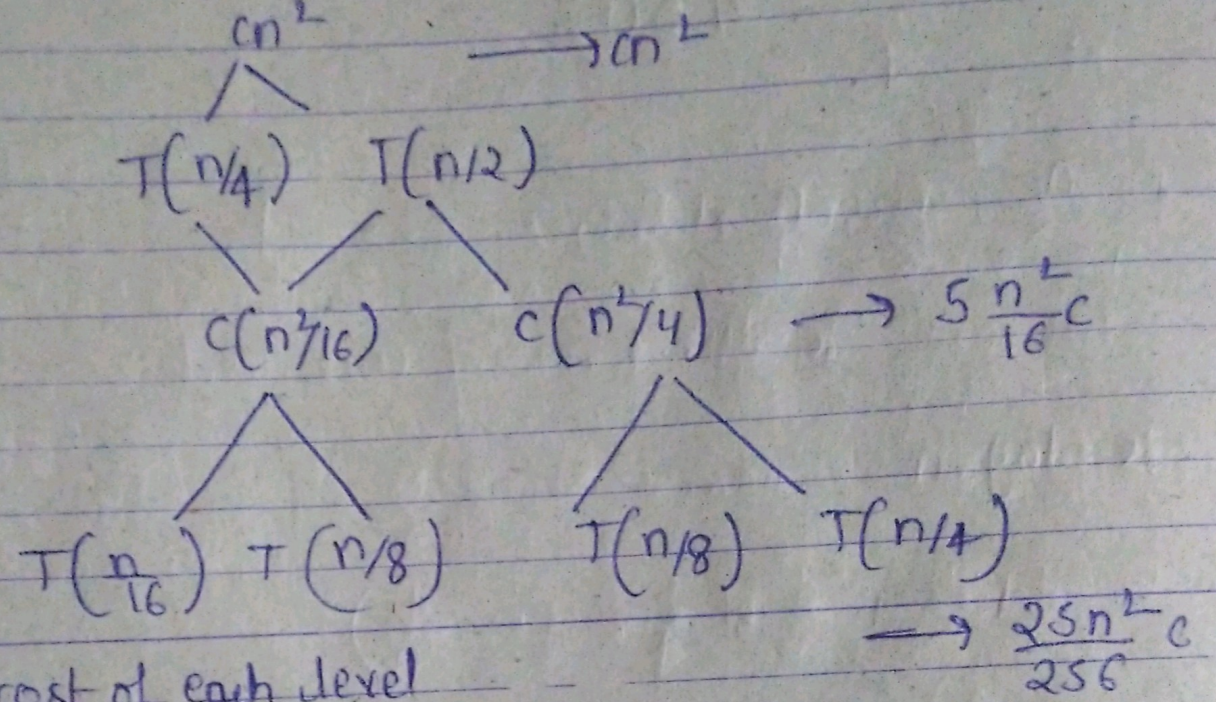
→ log (log n)

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

Q14 $T(n) = T(n/4) + T(n/2) + cn^2$

$T(1) = 0$

$T(0) = 0$



$T(n)$ = cost of each level

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

it is a G.P

with $a = n^2$

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

So, sum of G.P

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

$= 16\frac{cn^2}{11}$

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

Q15

```
for (int i to n)
{  for (int j = 1; j < n; j = j*i)
   {
      // O(1)
   }
}
```

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

k - terms

$k = \log_2 n$

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

$(n(\log n))$

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

Q16

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

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

It G.P $a = 2$

$r = 2^k$

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

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

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

$k \log_x k = \log x$

$k = \log x$ ——— (1)

$n = 2^x$

$\log_2 n = x \log_2 2$

$x = \log_2 n$

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

from (1)

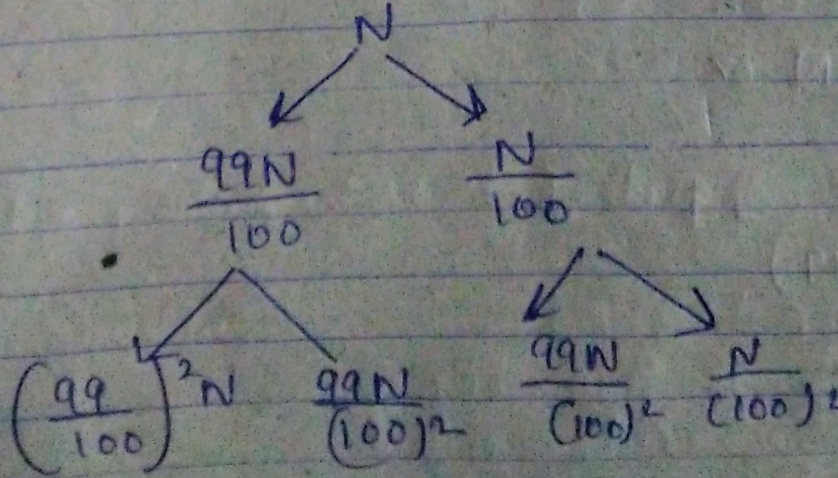
$x = \log(\log(n))$

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

Q17 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 extreme of a tree where starting point is N.



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

$$=\frac{99N}{100}+\frac{N}{100}$$

$$=N$$

So, cost of each level is N only

Total cost = height * cost of each level

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

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

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

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

$$\log N = h \log^{n}(1)$$

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

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

height of 2nd stream

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

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

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

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

$$n = \frac{\log N}{\log 100} + 1$$

$$= \log N \text{ (approx)}$$

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

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

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

and $\frac{\log N}{\log(100/99)}$ + 1 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 than height is less.

Q18

(a) $n, n!, \log n, \log \log n, \text{root}(n), n \log n$

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

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

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

(c)

Ans $O(96) < O(\log_8(n)) < O(\log_2 n) < O(\log(n!)$
$< O(n \log_6(n)) < O(n \log_2(n)) < O(5n) < O(5n^3)$
$< O(7n^3) < O(n!) < O(8^{2n})$

Q19

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

Q21

Time Complexity

	Best	Avg	worst	space
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	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(n)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$

Q22

	implace	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

Q23 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
          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)

Q24 $T(n) = \frac{T(n)}{2} + 1$