Rachit Budhani
1901135
B.Tech (CS)

DAA Assignment

Ques 1) What do you understand by Asymtotic notations. Define different Asymtotic notation with examples.

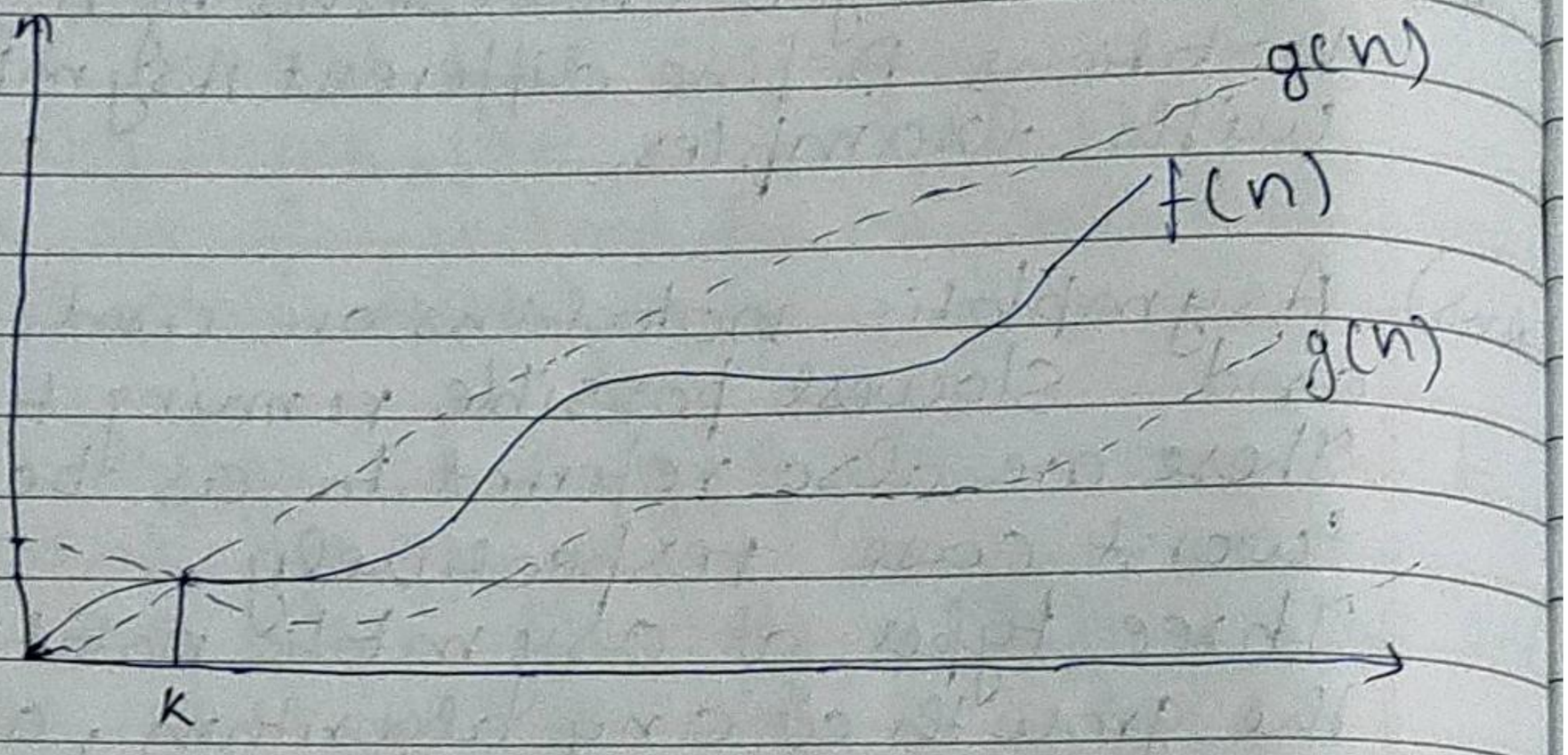
Ans) Asymptotic notations are used to write fastest and slowest possible running time for an algorithm. These are also referred to as 'best case' and 'worst case' respectively.
Three types of asymtotic notations to represent the growth of any algorithm, as input increases.

1) Big Theta (Θ)
2) Big oh (O)
3) Big omega (Ω)

Big Theta Notation (Θ)
The time complexity represented by the Big Θ notation is like the average value or range within which the actual time of execution of the algorithm will be.

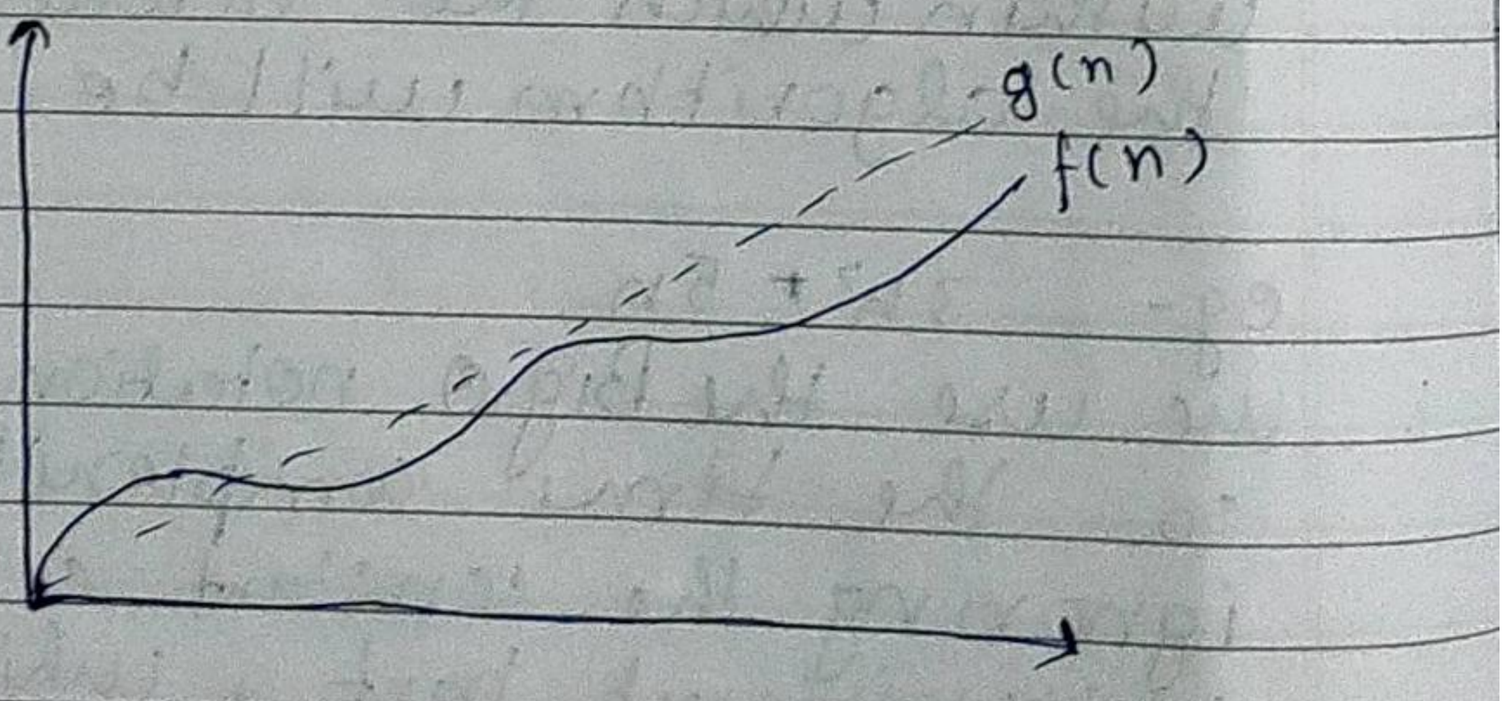
eg- $3n^2 + 5n$
We use the Big Θ notation to represent this, so the time complexity would be $\Theta(n^2)$ ignoring the constant coffecient and removing insignificant part, which is $5n$.

$\Theta(f(n)) = \{ g(n)$ if and only if $g(n) = O(f(n))$ and $g(n) = \Omega(f(n))$ for all $n > n_0 \}$.



2) Big oh Notation (O)

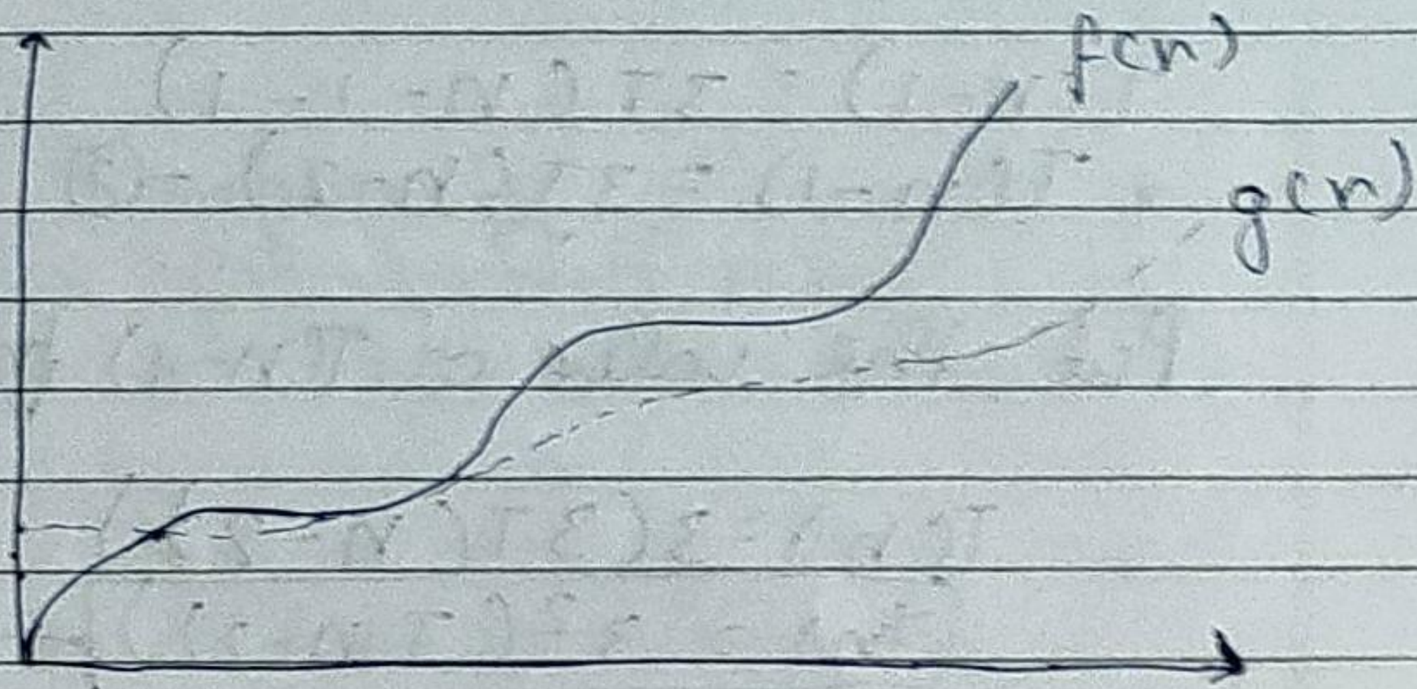
It is the formal way to express the upper bound of an algorithm running time. It measures the worst case time complexity or the longest amount of time an algorithm can possible take to complete.



eg: $O(f(n)) = \{ g(n)$: there exists $c > 0$ and n_0 such that $f(n) \leq c \cdot g(n)$ for all $n > n_0 \}$

3) Omega Notation(Ω)

The notation Ω(n) is the formal way to express the lower bound of all algorithm's running time. It measures the best case time an algorithm can possibly take to complete



Ω(f(n)) ≥ {g(n) : there exists c>0 and no and such that g(n) ≤ c.f(n) for all n>no }

Q2) What should be time complexity of
for (i = 1 to n) { i = i * 2 }

for (i=1 ; i<=n ; i = i*2)

1, 2, 3, 4, - - - .

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

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

Ans Let us solve this using substitution.

$$\boxed{T(n) = 3T(n-1)} \quad -①$$

Put $n = n-1$ in eq ①, we get

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

Put the value of $T(n-1)$ from ② in ① we get

$$T(n) = 3(3T(n-2))$$
$$\boxed{T(n) = 3^2(T(n-2))} \quad -③$$

Put $n = n-2$ in eqn ①, we get

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

$$\boxed{T(n-2) = 3T(n-3)} \quad -④$$

Put the value of $T(n-2)$ from ④ to ③, we get

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

So

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

$= 3^n T(0)$

$= 3^n(1)$

$\boxed{T_n = 3^n}$

So time complexity of this function is $O(3^n)$

Q4) Find the complexity

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

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

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

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

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

$= 2^3 T(n-3) - 2^2 - 2^1 - 2^0$

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

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

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

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

$= 1$

Time Complexity is $O(1)$.

Q 5) Find the complexity of the below program

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

Ans) We can define the terms 's' according to relation $s_i = s_{i-1} + i$

If k is total number of iterations taken by the program.

then while loop terminates if

$$1+2+3+\cdots+k = \left[\frac{k(k+1)}{2}\right] > n$$

So $k = O(\sqrt{n})$

Time complexity of the above function $O(\sqrt{n})$

Q6) Time complexity of

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

Ans) if k is the total no: of iterations taken by program.

∴ then loop terminates

$$(1)^2, (2)^2, (3)^2, \ldots (\sqrt{n})^2$$

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

Q7) $T(n) = O(n \cdot \log_2 n \cdot \log_2 n)$

$$T(n) = O(n (\log_2 n)^2)$$
$$T(n) = O(n (\log_2 n)^2)$$

Q8) Time complexity of

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

$T(n) = T(n-3) + n^2$ — (1)

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

$T(n-1) = T(n-4) + n^2$ — (2)

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

$T(n) = T(n-6) + n^2 + (n-1)^2 + (n-2)^2$

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

$T(n-k) = 1$

$k = n-1$

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

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

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

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

$T(n) = n^3$

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

9) Time complexity of -

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

3

3

i=1 n terms
i=2 1, 3, 5, ... n/2
i=3 1, 4, 7 - - - n n/3

i=n 0

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

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

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

$k \geq 1$ and $a \geq 1$

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

Q11) Void func (int n)
{
int i=1, j=0;
while (i<n)
{
j= j+i;
i++;
}
}

* 0, 3, 6, 10, 15 - - - - n

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

$k \approx \sqrt{n}$ $\quad T = O(\sqrt{n})$

Q12) Recurrence Relation of Fibnacci 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 $T(n-2k) = T(0)$

$n = 2k$

$k = n/2$

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

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

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

Hence Space complexity of fibnacci series is O(n) as it depends on height of recurrance and it is equal to n in fabnacci series

Q13) A) $n(\log n)$

```
for(int i=0; i<n; i++)
{
    for(int i=0; i<n; i=i*0)
    {
        print("*")
    }
}
```

```
Void main()
{
    fun();
}
```

$\rightarrow n^3$

```
#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++)
            {
                k++
            }
        }
    }
}
```

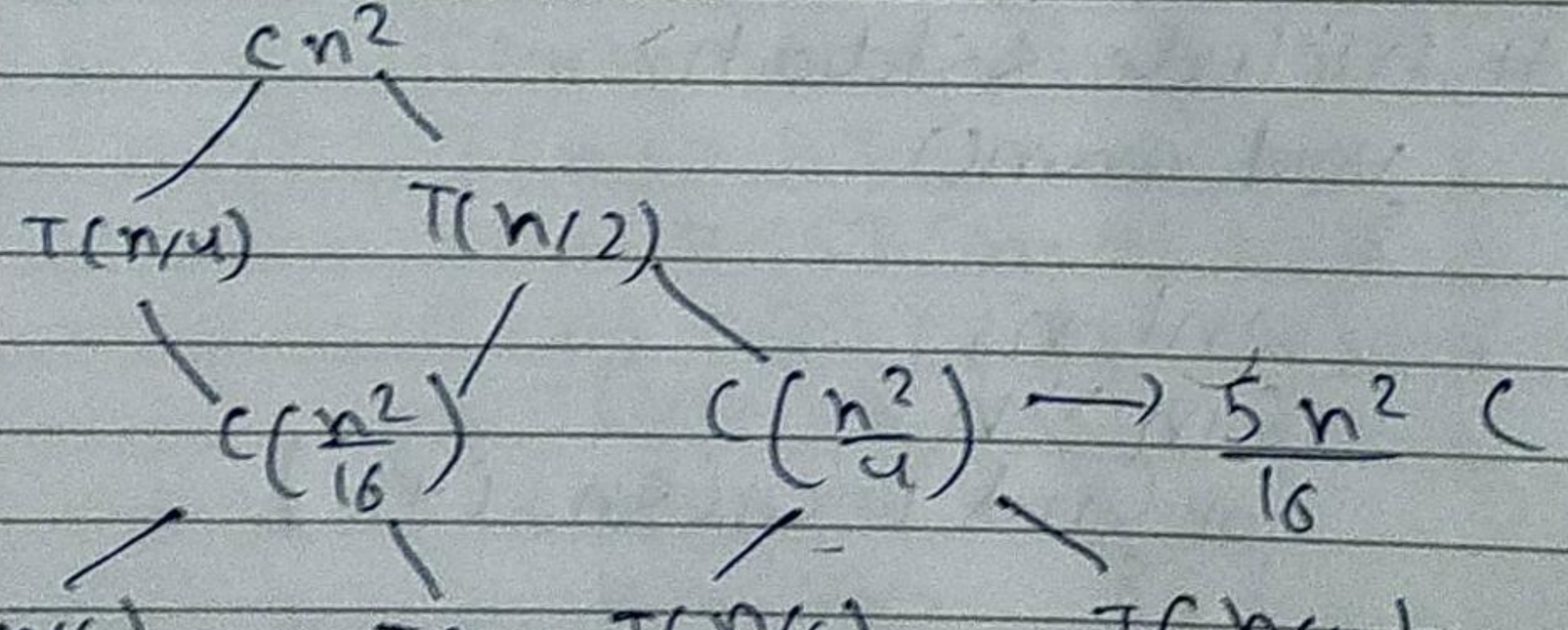
→ log (Log n)

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

(n-1)/6) $T(n) = T(n/4) + T(n/2) + cn^2$

$T(1) = 0$

$T(0) = 0$



T(n) = cost of each level

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

It is a G.P

with $a = n^2$

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

So sum of GP

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

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

Q15 Ans

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

$$n, \frac{n}{2}, \frac{n}{3}, \frac{n}{4}, \frac{n}{5} \cdots \frac{n}{7}$$

k Lines

$$k = \log_2 n$$

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

$$n(\log n))$$

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

Q16) Ans) for (int i = 2; i <= n; i = pow(i, k))

{

O(1)

}

2, 2^k, 2^{k2}, 2^{k3}, ---- n

GP a = 2

$r = 2^k$

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

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

Let $k^{k-1} = x$

$k \log_k k = \log x$

$k = \log x$ — ①

$n = 2^x$

$\log_2 n = x \log_2 2$

$x = \log n$

$\text{Log } x = \log(\log n)$

from ①

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

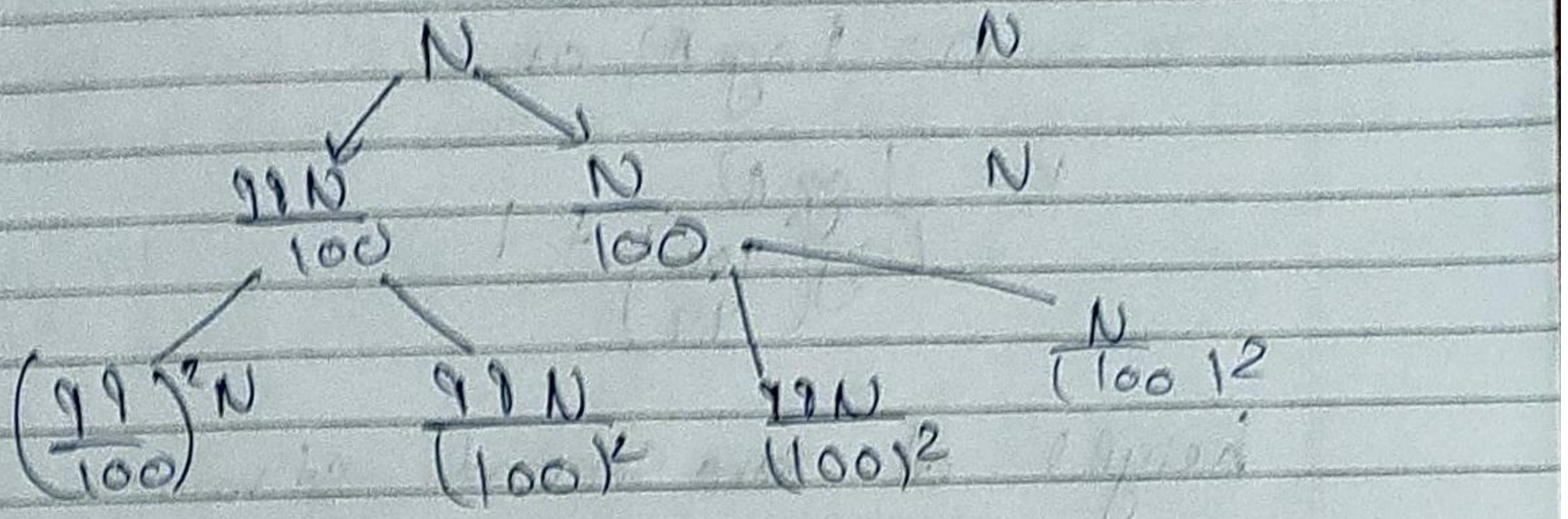
$T(n) = O(\log(\text{Log}(n)))$.

Ans) Pwed is divided in 99% and 1%

So

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

Now so here we can use 2 extreme of a bee where starting point is N.



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

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

$$= N$$

So cost of each level is N only.

Total cost = height * cost of each level

So for first steam = $N, \frac{99}{100}N, \left(\frac{99}{100}\right)^2 N \ldots$

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

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

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

$$\text{Log } N = n \text{ Log}(1)$$

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

$$n = \frac{\text{Log } N}{\text{Log}\left(\frac{100}{99}\right)} + 1$$

height of ~~st~~ second st raw.

$$N, \frac{N}{100}, \frac{N}{(100)^2}, \frac{N}{(100)^3} \quad + \cdots + 1$$

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

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

$$(n-1) \log 100 = \text{Log } N$$

$$n = \frac{\text{Log } N}{\text{Log } 100} + 1 \quad \text{or} \quad n = \text{Log } N \text{ (approx)}$$

$$T(n) = O(N \text{ Log } N)$$

So time complexity is $O(N \text{ Log } N)$

height of both extreme is $\frac{\log N}{\log \frac{1}{n}}$ + log $\frac{1}{100}$

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

So we conclude that if division is done more than height of tree will be more and when division ratio is less than height is less

Ans 8) $n, n!, \log n, \text{Log Log} n, \text{root}(n), n \log n, 2^n, 3^n, 4^n, 100.$

$O(100) < O(\log \log N) < O(\log N) < O(\sqrt{n}) < O(n) < O(n \log N) < O(n^2) < O(2^n) < O(3^n) < O(4^n)$

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

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

c) $O(96) < O(\log(n) \times O \log_n(n) < O(\log n!) < O(n \log(n)) < O(n \log_2(n)) < O(5n!) < O(8n^3) < O(7^0 n^3) < O(n!)^2 < O(8^{\wedge} 2^n)$.

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	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$

Q22) Ans)

	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	Yes	No	No
Heap Sort	Yes	No	No

```
(HW23) 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
                end = mid - 1
        }
    }
```

Time complexity of linear search: O(n)
Space " " " " - O(1)

Time complexity of Binary search = O(log n)
Space " " " " = O(n).