Name → Geetika Paliwal

University Roll no. → 1961053

Section → A

Course → B.Tech. CSE

Subject → DAA

FREEMIND

Date

Page

Assignment - 01

Q1. What do you understand by Asymptotic notation. Define different Asymptotic notations with examples.

Asymptotic Notations

Asymptotic means towards infinity. Here infinity describes very large input.

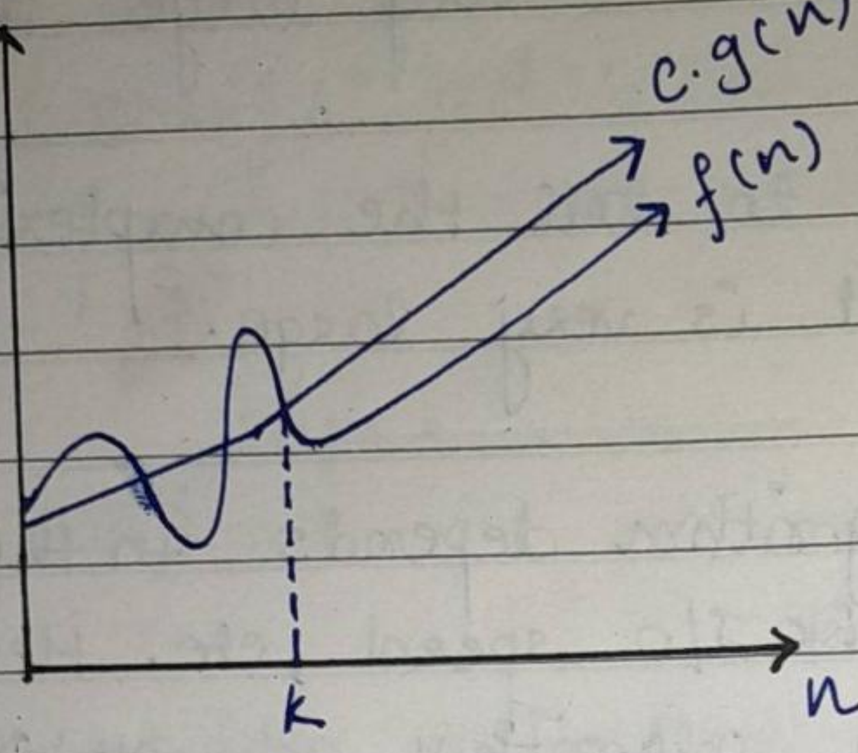
These notations are used to tell the complexity of an algorithm when the input is very large.

Execution time of an algorithm depends on the instruction set, processor speed, disk I/O speed, etc. Hence, we estimate efficiency of an algorithm asymptotically.

Complexity of an algorithm in analyzed in two perspectives time and space

Following notations will be used-

1] Big-oh (O) -



n → input values

t → time

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

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

$\forall\ C > 0$

$n \geq k$

$k \geq 0$

k is some threshold after which $C \cdot g(n)$ will always be greater than or equal to $f(n)$.

→ worst case

→ Least upper bound (At most)

Example → If $f(n) = 2n^2 + n$ then $f(n) = O(?)$

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

So, $2n^2 + n = O g(\ )$

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

$2n^2 + n \leq C \cdot g(n^2)$ ← since we have to take least upper bound. we could have also taken n^3, n^4, n^5 .... but then it won't satisfy least upper bound rule and anything less than n^2 would make it smaller than $2n^2+n$.

Now, for C, if we take 2 it becomes $2n^2 + n \leq 2 \cdot n^2$ which is wrong. So, C has to be 3.

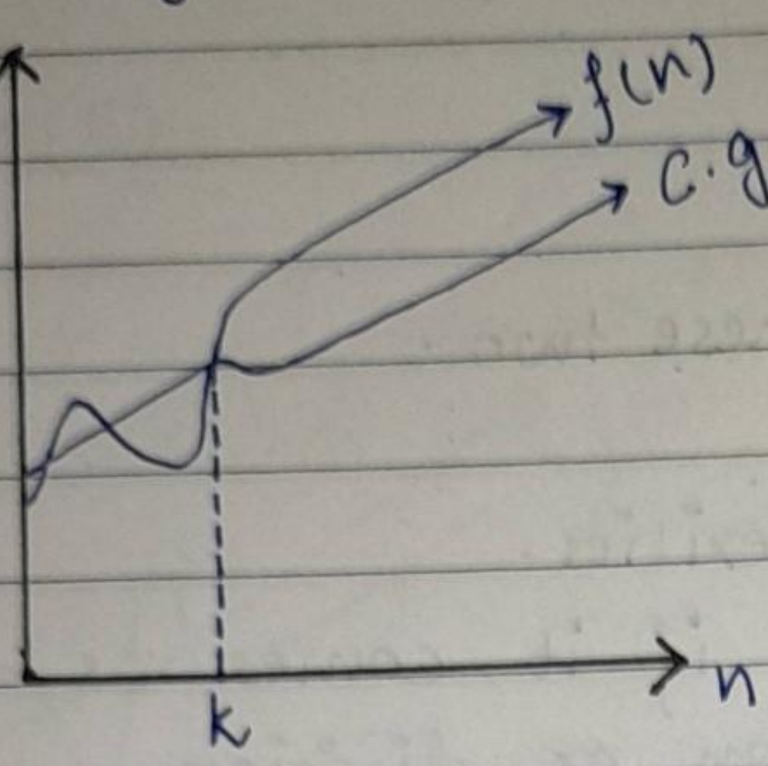
Therefore, $2n^2 + n \leq 3n^2$ ←

$n \leq n^2$

$1 \leq n \Rightarrow n \geq 1$ So, for all the values of inputs greater than 1 this condition will always hold. Applying c has to be 3.

Hence, complexity of f(n) can be represented as $O g(n)$ i.e. $O(n^2)$

2] Big-Omega (Ω)



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

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

$\forall\ c > 0$

$n \geq k$

→ Best case

→ Greatest lower Bound (At least)
or
closest

example → If $f(n) = 2n^2 + n$ then $f(n) = \Omega(?)$

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

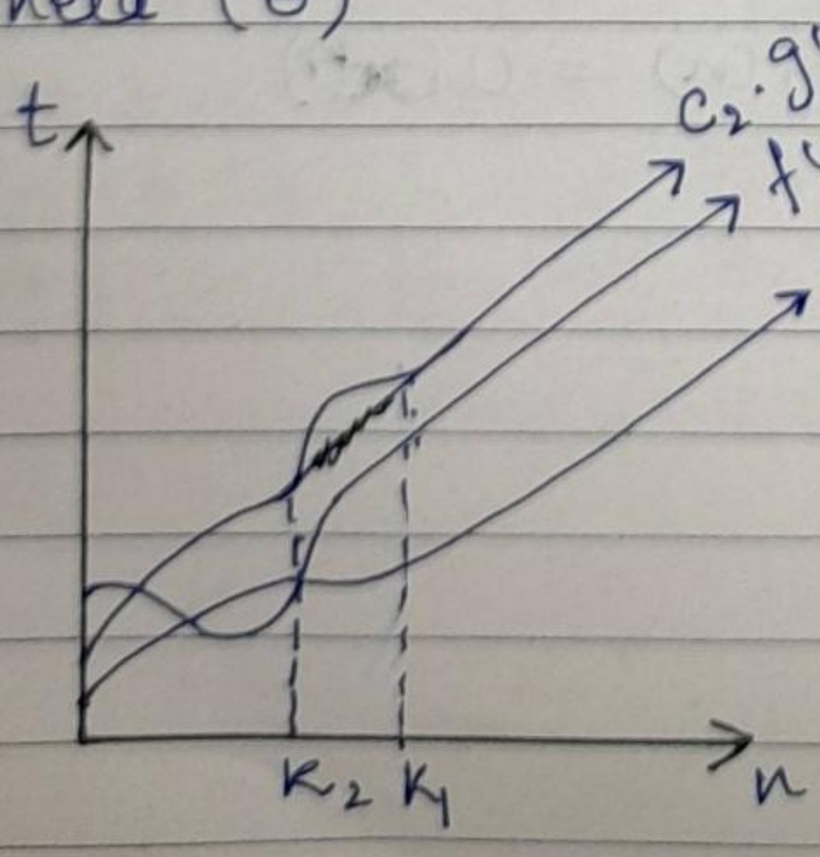
$2n^2 + n \geq c \cdot g(n^2)$

Now choose c carefully.

$2n^2 + n \geq 2n^2$ ✓

$n \geq 0$

3] Theta (Θ)



$f(n) = \Theta\, g(n)$

if

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

$\forall\ n \geq \underbrace{\max(k_1, k_2)}$ because b/w k_1 and k_2 either c_1 or c_2 can fluctuate based on value of k_1 and k_2

c_1 and $c_2 > 0$

Θ gives least upper bound and greatest lower bound both

$f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$ & $f(n) = \Omega(g(n))$

4] Small-oh (o)

o gives us upper bound

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

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

$$\forall \quad n > k$$

$$c > 0$$

5] Small-omega (ω)

→ lower bound

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

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

$$\forall \quad n > k_0$$

$$c > 0$$

Q2. What should be time complexity of -

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

→ 1, 2, 4, 8, 16, ---- n

G.P.

$a = 1 \quad r = t_2/t_1 = 2/1 = 2$

$t_k = a r^{k-1}$

$n = 1 \cdot 2^{k-1}$

$n = 2^k/2$

$2^k = 2n$

$k = \log_2(2n)$

$k = \log_2 2 + \log_2 n$

$k = 1 + \log_2 n$

Time complexity = $O(\log_2 n)$

Q3. $T_n = \begin{cases} 3T(n-1) & \text{if } n > 0 \\ \text{otherwise } 1 \end{cases}$

Page

→ $T(n) = 3T(n-1)$ -①

$T(n-1) = 3T(n-2)$ -②

$T(n-2) = 3T(n-3)$ -③

Putting ② in ① by backward substitution-

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

$T(n) = 9T(n-2)$ -④

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

Putting ③ in ④

$T(n) = 9(3T(n-3))$

$T(n) = 27T(n-3)$

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

⋮

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

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

$n-k = 0$

$n = k$

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

$T(n) = 3^n \cdot 1$

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

Q4. $T(n) = \begin{cases} 2T(n-1)-1 & \text{if } n>0, \\ \text{otherwise } 1 \text{ i.e. } 1 & \text{if } n=0 \end{cases}$

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

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

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

Putting (2) in (1)

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

$T(n) = 4T(n-2)-(2+1) \quad -(4)$

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

Putting (3) in (4)

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

$T(n) = 8T(n-3)-(4+2+1)$

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

$\vdots$

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

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

$n = k$

$T(n) = 2^nT(n-n)+(1+2+4+\cdots 2^{n-1})$

$= 2^nT(0)+(\underbrace{1+2+4+\cdots}_{G.P.})$

$a = 1$

Sum of n terms of G.P. $= \frac{a(r^n-1)}{r-1}$

$r = 2/1 = 2$

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

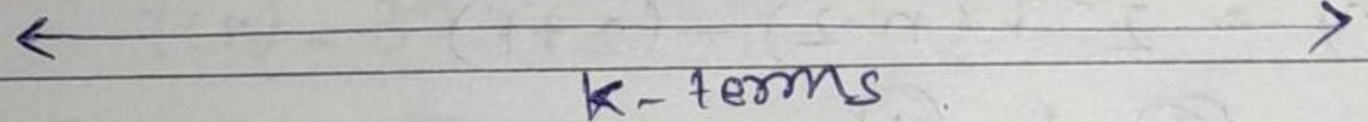
$T(n) = 2^n - 2^n + 1$

$T(n) = O(1)$

Q5. What should be time complexity of -

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

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



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

$2n = k^2 + 1$ (ignoring constants)

$n = k^2$

$k = \sqrt{n}$

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

Q6. Time complexity of -

```
void function (iut n)
    {
        iut 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 ++
    }
```

Page

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

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

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

Q8. Time complexity of -

```
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)

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

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

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

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

$\cdots (k-2)$ terms)

$T(n-k) = 1$

$k = n-1$

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

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

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

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

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

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

Q9. Time complexity of -

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

3+

$i=1$	j	n times
$i=2$	1, 3, 5, ---- n	n times
$i=3$	1, 4, 7, ---- $n/3$	$n/2$ times
$i=4$	1, 5, 9, ---- $n/4$	$n/3$ times
:		
$i=n$	0	$n(n + n/2 + n/3 + \cdots)$

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

G.P.

$\Rightarrow a = n \quad r = n/2 \,/\, n \to 1/2$

ms)

$$t_k = a \cdot r^{k-1}$$

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

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

$$2^k = 2n$$

$$k = \log_2 n + 1$$

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

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

FREEMIND

Date

Page

Q11. What is the time complexity of below code and why?

```
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 $= \frac{k(k+1)}{2}$

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

$n = k^2$

$k = \sqrt{n}$

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

Q12. Write recurrence relation for the recursive function that prints fibonacci series. Solve the recurrence relation to get the time complexity of the program. What will be the space complexity of this program and why?

Recurrence relation of fibonacci 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 $T(n-2k) = T(0)$

$n = 2k$

$k = n/2$

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

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

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

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

Q13. write programs which have complexity - $n(\log n)$, n^3, $\log(\log n)$.

($n(\log n)$)

```
void fun()
{
    for (int i=0; i<n; i++)
    {
        for (int j=0; j<n; j = j*2)
```

```
        printf("*");
    }
}
}

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

(n^3)

```
#include<stdio.h>
void main()
{
    int n;
    cin>>n;
    for(int i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
        {
            for(k=0; k<n; k++)
            {
                x++;
            }
        }
    }
```

$(\log(\log n))$

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

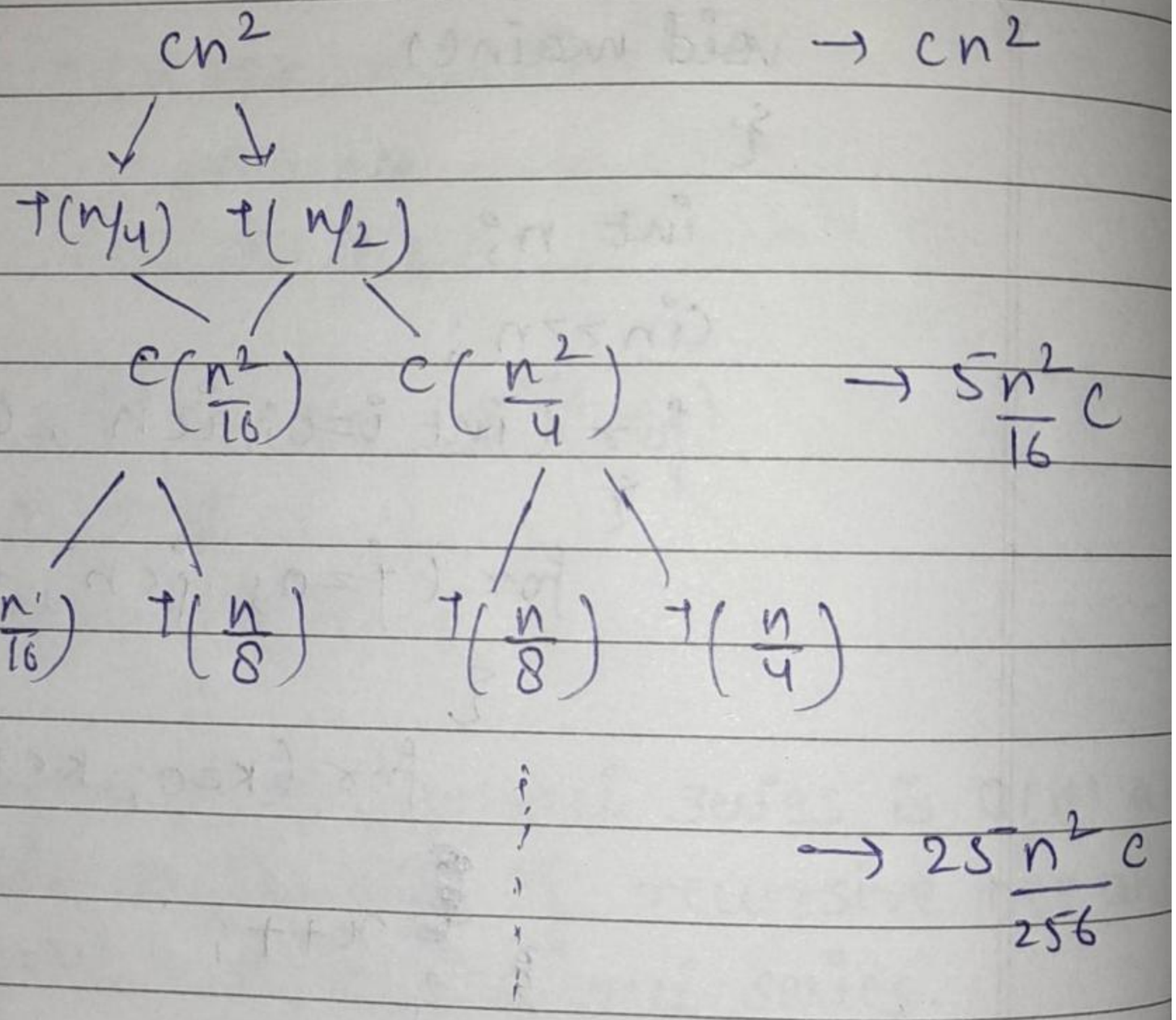
```
void main()
{
   fun(100);
}
```

Q14. Solve the following recurrence relation -

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

$T(1) = c$

$T(0) = 0$



$T(n)$ = Cost of each level

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

G.P.

$a = n^2$ $\quad r = 5/16$

So, sum of sp

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

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

FREEMIND

Date

Page

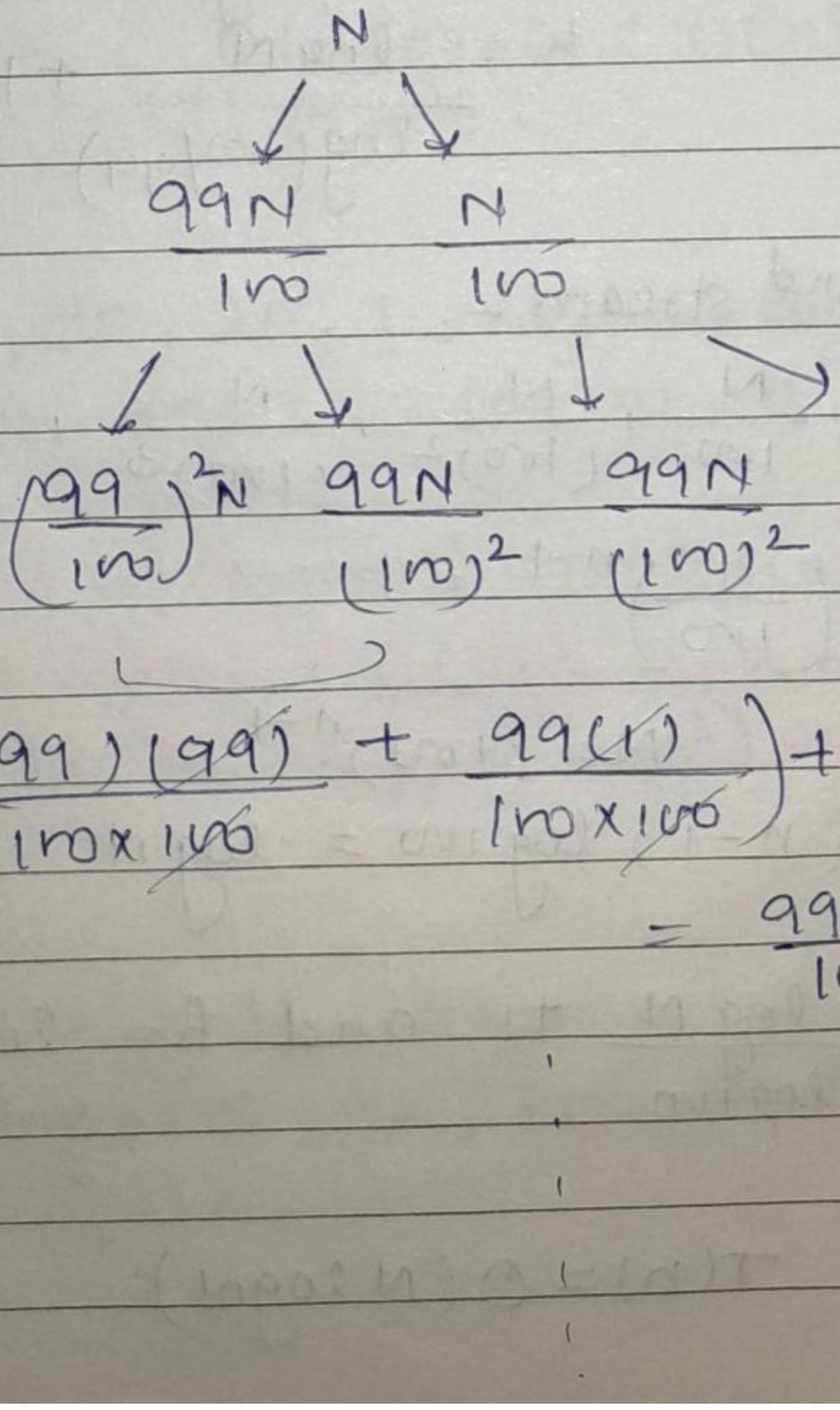
Q17. write a recurrence relation when quick sort repeatedly divides the array into two parts of 99% and 1%. Derive the time complexity in this case. Show the recursion tree while deriving time complexity and find the diff. in heights of both the extreme parts. What do you understand by this analysis?

Pivot is divided in 99% & 1%.

So,

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

Here, we can use two extremes of a tree where starting point is N



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

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

$$= N$$

$$\geq N$$

Date

Page

So, cost of each level is N only.

Total cost = height * cost of each level

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

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

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

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

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

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

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

Height of 2nd stream -

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

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

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

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

$$n = \frac{\log N}{\log 100} + 1 \text{ and } h = \log N \text{ (approx.)}$$

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

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

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

So, we can conclude that if division is done more, then height of tree will be more and when division ratio is less, height is less.

Arrange the foll. in inc. order of rate of growth:

Q18. a) $\underline{n}$, $n!$, $\log n$, $\log \log n$, $\sqrt{n}$, $\log(n!)$, $n \log n$, 2^n, 2^{2n}, 4^n, n^2, 100

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

b) $2(2^n)$, $4n$, $2n$, 1, $\log(n)$, $\log(\log(n))$, $\sqrt{\log(n)}$, $\log 2n$, $2\log(n)$, n, $\log(n!)$, $n!$, n^2, $n \log(n)$

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

c) 8^{2n}, $\log_2(n)$, $n \log_6(n)$, $n \log_2(n)$, $\log(n!)$, $n!$, $\log_8(n)$, 96, $8n^2$, $7n^3$, $5n$

$O(96) < O(\log 8(n)) < O(\log_2 n) < O(\log(n_1)) <$
$O(n \log_6(n)) < O(n \log_2(n)) < O(\sqrt{n}) < O(8n^3$
$< O(7n^3) < O(n!) < O(8^{2n})$.

Q19. Write linear search pseudocode to search an element in sorted array with minimum comparisons.

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

Q20. Write pseudo code for iterative and recursive Insertion sort. Insertion sort is called online sorting, why? What about other sorting algorithms that has been discussed in lectures.

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<=1
      return;

    insertionsort (arr, n-1)
    last = arr[n-1]
      j = n-2

while ( j >=0 and arr [j] > last)
    {
       arr [j+1] = arr [j]
       j--;
    }
    arr [j+1] = last;
}
```

FREEMIND

Date

Page

Insertion sort is called online sorting because whole input is not known here it might make decision that later turn out to be not optimal.
Other algorithm are offline algorithms that are discussed.

Page

Q22. Divide all sorting algorithms into in place/ stable / online sorting.

	inplace	stable	Online Sorting
Bubble Sort	✓	✓	×
Selection Sort	✓	×	×
Insertion sort	✓	✓	×
Merge Sort	×	✓	×
Quick Sort	✓	×	×
Heap Sort	✓	×	×

Q23. Write recursive / iterative pseudo code for Binary search. What is time and space Complexity of linear and Binary search.
(Recursive & Iterative).

```
Binary Search (arr, lutn, key)
  {
     beg = 0
     end = n-1
     while ( beg <= end )
        {
           mid = ( big + end)/2
     if [arr(mid)] == key]
       found
     else if  arr [mid] < key
          beg = mid + 1
     else
          end = mid - 1
        }
  }
```

FREEMIND

Date

Page

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

Time complexity of Binary search - O(logn)
Space complexity - O(n)

Q24. write recurrence relation for a binary recursive search.

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