DAA TUTORIALS

Ipsa Mishra
Section I
60
20022515 (Student ID)

DATE _____
PAGE _____

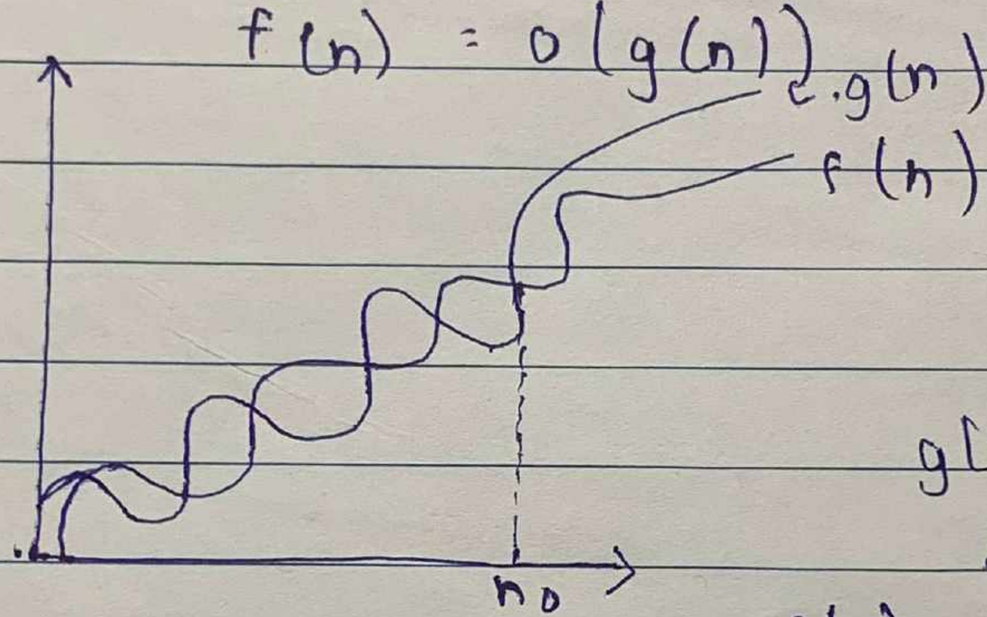
Tutorial -1

Q.1) What do you mean by Asymptotic notations. Define different type of notations along with example ?

Ans. Asymptotic Notations : means tending to infinity . They are used to tell the complexity when input is very large.

-> Different types of Asymptotic Notations :

1.) Big oh (O) Notation :



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

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

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

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

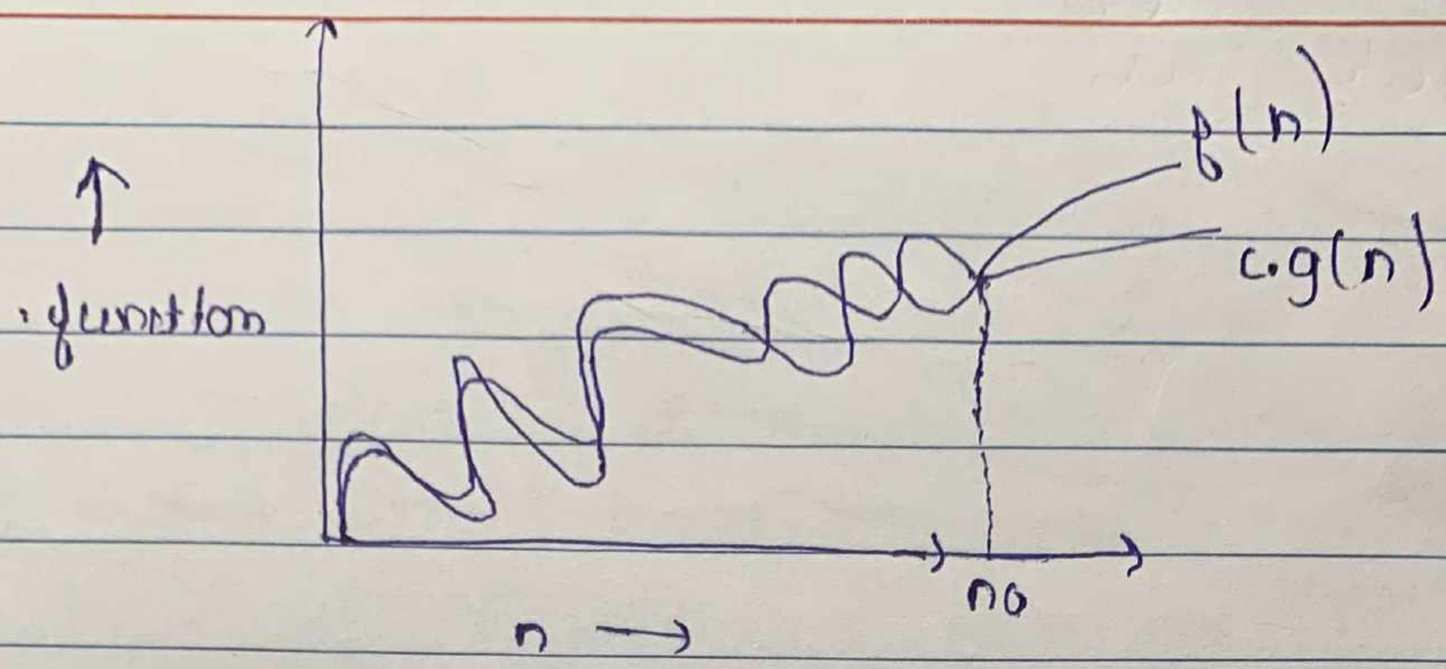
Example :-

```
for (i=1; i<=n; i++)
  { printf("*"); ——— O(1)
  }
```

=) T(n) = O(n)

2.) Big Omega (Ω) :

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



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

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

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

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

Example :

$f(n) = 2n^2 + 3n + 5 \quad , g(n) = n^2$

$0 \leq c \cdot g(n) \leq f(n)$

$0 \leq c \cdot n^2 \leq 2n^2 + 3n + 5$

$c \leq 2 + \frac{3}{n} + \frac{5}{n^2}$

on putting $n = \infty$, $\frac{3}{n} \longrightarrow 0$, $\frac{5}{n^2} \longrightarrow 0$

$\therefore \quad c = 2$

$\therefore \quad 2n^2 \leq 2n^2 + 3n + 5$

On putting $n = 1$

$2 \leq 2 + 3 + 5$

$2 \leq 10$ true.

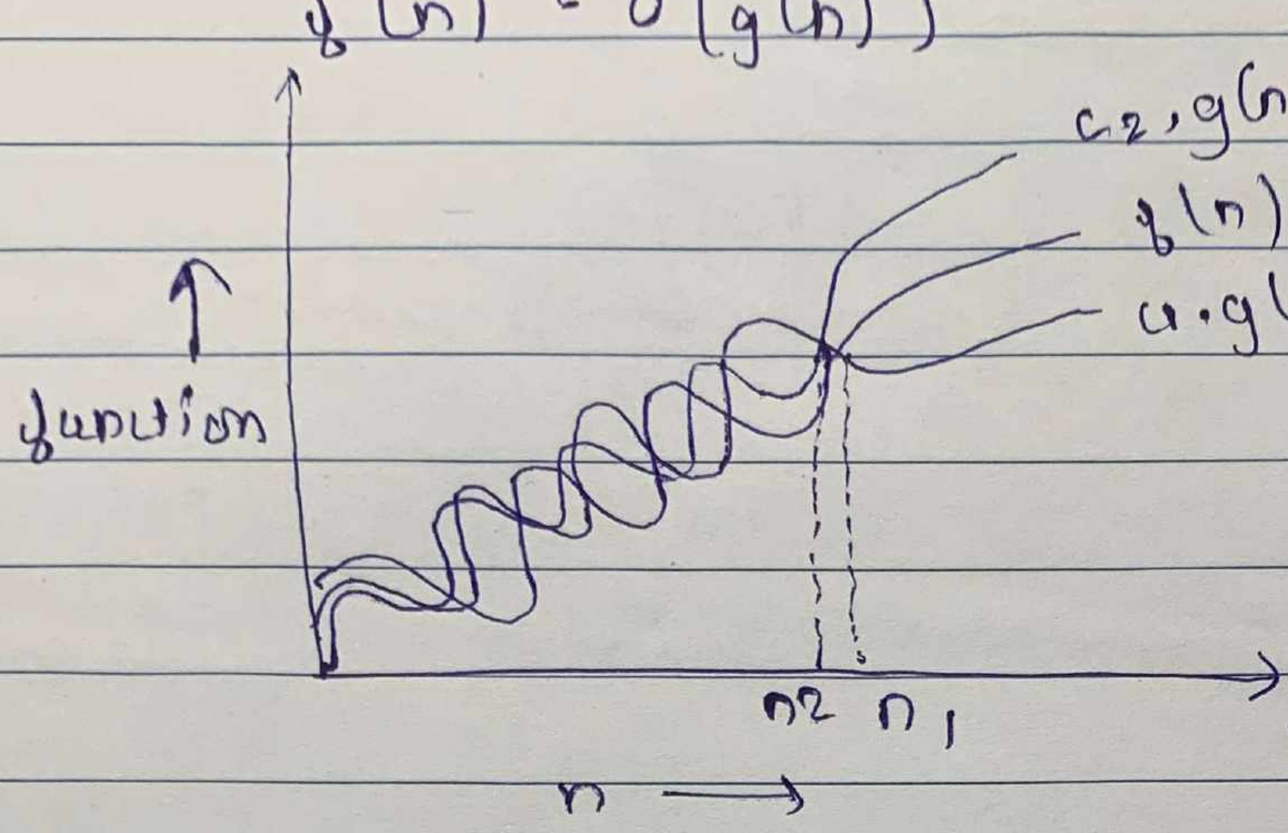
$\therefore \boxed{c = 2 \;, \; n = n_0 = 1}$

$0 \leq 2n^2 \leq 2n^2 + 3n + 5$

$\therefore f(n) = \Omega(n^2)$

3. Big Theta (Θ):

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



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

if

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

$\forall$

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

& some constant

$c_1 > 0$ & $c_2 > 0$

Example : $f(n) = 10 \log_2 n + 4$, $g(n) = \log_2 n$

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

$\Rightarrow 10 \log_2 n + 4 \leq 10 \log_2 n + \log_2 n$

$\Rightarrow 10 \log_2 n + 4 \leq 11 \log_2 n$

$c_2 : 11$

$\Rightarrow 4 \leq 11 \log_2 n - 10 \log_2 n$

$4 \leq \log_2 n$

$16 \leq n$

Here

$\forall \; n \geq 16$

$n_2 : 16$

$c_2 : 11$

$f(n) \geq c_1 \cdot g(n)$

$10 \log_2 n + 4 \geq 4 \log_2 n$

$4 > 1 \; , n > 0$

$\Rightarrow n_1 : 1 \qquad \Rightarrow n_0 \Rightarrow 16$

$\Rightarrow \log_2 n \leq 10 \log_2 n + 4 \leq 11 \log_2 n$

$c_1 = 1$, $c_2 = 1$ $\Rightarrow \Theta(\log_2 n)$

4.) Small oh (o) :

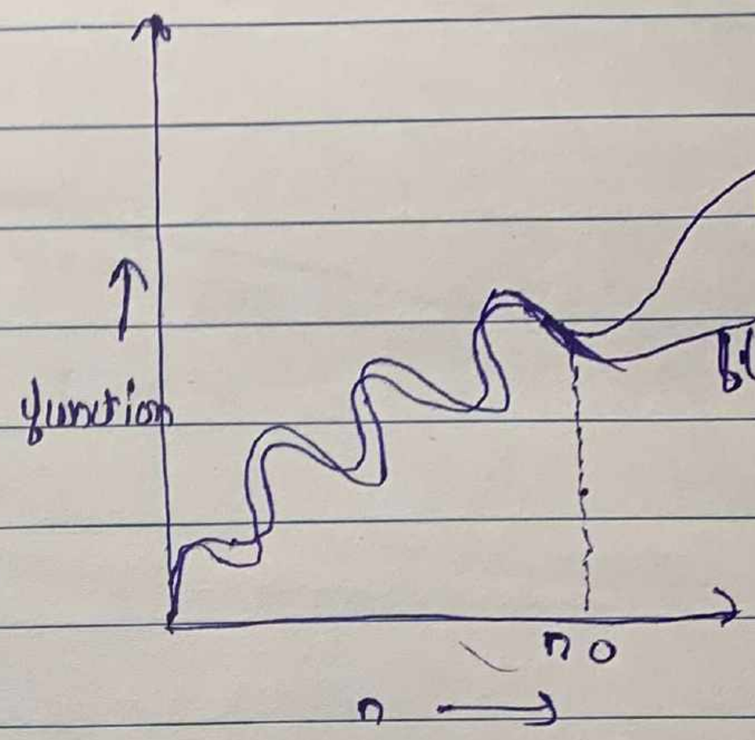
$f(n) = o(g(n))$

$g(n)$ is upper bound of $f(n)$

$f(n) < o(g(n))$

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

$\forall\ n > n_0$ & $\forall$ constant, $c > 0$



5.) Small omega (ω) :

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

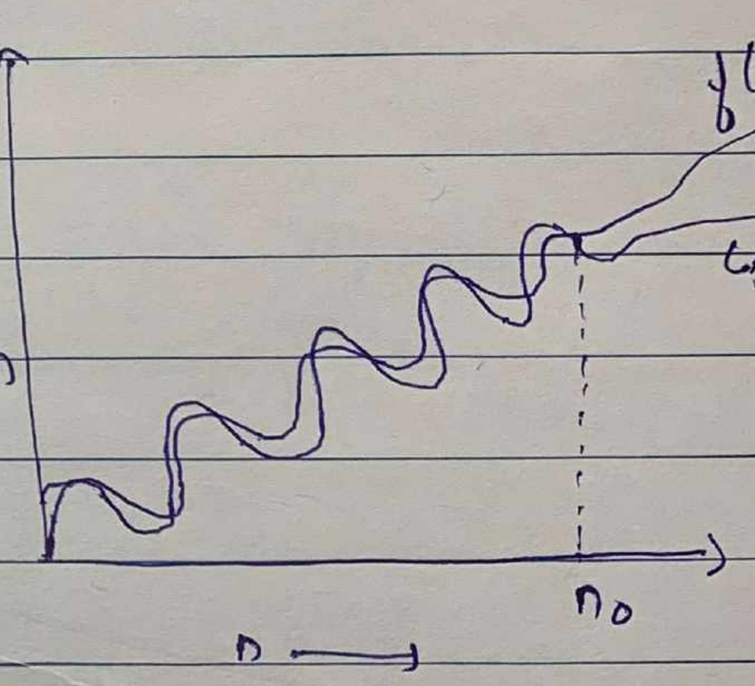
$g(n)$ is lower bound of $f(n)$

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

when :

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

$\forall\ n > n_0$ & $\forall\ c > 0$



Q2) what should be the time complexity of.

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

Sol :) Values of i : 1, 2, 4, 6, 16, ------ n,
(k terms)

as this is a G.P with $a = 1$, $r = 2$

Now,

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

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

$n = 2^{k-1}$

taking $\log_2$ on to the both sides

$\log_2 n = \log_2 2^{k-1}$

$\log_2 n = (k-1) \log_2 2$

$\log_2 n = k-1 \Rightarrow k = 1 + \log_2 n$ $[\because \log_2 2 = 1]$

$\therefore$ Time Complexity $T(n) = O(k)$

$= O(k + \log_2 n)$

$= O(\log_2 n)$.

Q3.)

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

Sol:- $T(n) = 3T(n-1)$ ——— (1)

Put $n = n-1$ in eqn (1)

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

$T(n-1) = 3T(n-2)$ ——— (2)

Put value of $T(n-1)$ from eqn (2) in eqn (1)

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

$T(n) = 9T(n-2)$ ——— (3)

Put $n = n-2$ in eqn (1)

$T(n-2) = 3T(n-3)$ ——— (4)

Put value of $T(n-2)$ in eqn (3)

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

$T(n) = 27T(n-3)$ ——— (5)

On generalising eqn (5)

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

Put $n-k = 0$

$\therefore T(n) = 3^k T(0)$

$\therefore 3^k$ $(\because T(0) = 1)$

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

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

Soln $T(n) = 2T(n-1) - 1$ ——— (1)

Put $n = n-1$ in eqn (1)

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

$T(n-1) = 2T(n-2) - 1$ ——— (2)

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

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

$Tn = 4T(n-2) - 2 - 1$ ——— (3)

Put $n = n-2$ in eqn (1)

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

Put value of $T(n-2)$ in eqn (3)

$T(n) = 4[2T(n-3) - 1] - 2 - 1$

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

On Generalising

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

Put $n-k = 0 \Rightarrow n = k$, $T(0) = 1$ (given)

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

$= 2^n - \underbrace{[2^{n-1} + 2^{n-2} - \cdots + 1]}_{k \text{ terms}}$

$\Rightarrow a = 2^{n-1}$ & $r = 2/2$

Sum of GP $= \dfrac{2^{n-1}[1 - (k_2)^{n-1}]}{1 - 1/2}$

$= 2^n - 2$

$\Rightarrow T(n) = 2^n - [2^n - 2] = 2$

$= O(2)$

$\boxed{T(n) = O(1)}$

DATE
PAGE

Q6) what should be the time complexity of -

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

Sol:-

i	S
1	1
2	3
3	6
4	10
5	15
!	!
n	n

n — k times

$S = \underbrace{1, 3, 6, 10, 15, \ldots\ldots n}_{k \text{ terms}}$

k^{th} term, $t_k = t_{k-1} + k$

$k = t_k - t_{k-1}$ ——— (1)

$\therefore k = n - t_{k-1}$

loop runs k times

Time complexity $= O(1 + 1 + 1 + n - t_{n-1})$

but $t_{n-1} = c$ (constant)

$\therefore$ time complexity $= O(3 + n - 1)$

$= O(n)$

Q7) Time complexity of

Void function (int n)

```
{ int 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 ++
}
```

Sol $i = n/2, \frac{n+2}{2}, \frac{n+4}{2}, \frac{n+6}{2}, \ldots\ldots$ upto n

$= \frac{n+0\times 2}{2} + \frac{n+1\times 2}{2} + \frac{n+2\times 2}{2}, \ldots\ldots$ upto n

General form

$$t_k = \frac{n + k * 2}{2}$$

total terms $= k+1$

$t_{k+1} = n$

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

$\Rightarrow n + 2k + 2 = 2n$

$2k = n - 2$

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

i	j	k
$n/2$	$\log_2 n$ times	$(\log_2 n)^2$
$\frac{n+2}{2}$	$\log_2 n$ times	$(\log_2 n)^2$
1	⋮	⋮
1		
1		
n		
$(n/2 - 1)$ times	$\log_2 n$ times	$(\log_2 n)^2$

s) $(n/2 - 1)(\log_2 n)^2$

c) $O(n/2 \log_2^2 n - \log_2 n)$

c) $O(n \log^2 n)$

Q6.) Time Complexity of -

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

Sol

i	i * i
1	1^2
2	2^2
3	3^2
!	!
!	!
n	n^2

$i * i = \underbrace{1^2, 2^2, 3^2, 4^2, \ldots\ldots n}_{k \text{ terms}}$

s) k^{th} term : $t_k = k^2$

$k^2 = n$

$k = n^{1/2}$

time complexity = $O(1 + 1 + 1 + n^{1/2})$

$= O(n^{1/2})$

$= O(\sqrt{n})$

Q8.) for function call

$\underbrace{n, n-3, n-6, n-9 \ldots\ldots 1}_{k \text{ terms}}$

AP with $d = -3$, $a = n$

$a_n = a + (n-1)d$

$$1 = n + (k-1)(-3)$$

$$\frac{1-n}{(-3)} = k-1$$

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

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

$$\boxed{k = \frac{n+2}{3}}$$

Hence function show a recursive call $\frac{n+2}{3}$ times

$\therefore$ Time Complexity $= \left(\frac{n+2}{3}\right)(n)(n) \Rightarrow O(n^3)$

Q 9.) Time complexity of

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

Sol

for $i = 1 \longrightarrow j = 1, 2, 3, 4, \ldots\ldots n \quad : n$

for $i = 2 \longrightarrow j = 1, 3, 5, 7, \ldots\ldots n \quad : n/2$

for $i = 3 \longrightarrow j = 1, 4, 7, \ldots\ldots n \quad : n/3$

for $i = n \Rightarrow j = 1, \ldots\ldots, n \quad : 1$

$$\Rightarrow \sum_{j=n}^{1} n + n/2 + n/3 + n/4 + \ldots\ldots + 1$$

$$\sum_{j=n}^{1} n\left[1 + 1/2 + 1/3 + \ldots\ldots + 1/n\right]$$

$$\sum_{j=n}^{1} n \log n$$

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

$T(n) : o\,[n \log n)$

Q.10.) for the functions, n^k & c^n, what is the asympto-tic notation relationship between these functions, Assume that $k >= 1$ & $c > 1$ are constants. find out the value of c & n_0 for which relation holds.

Sol:- As given n^k & c^n

relation b/w n^k & c^n is $\boxed{n^k = o(c^n)}$

as $n^k \leq a c^n \quad \forall \quad n \geq n_0$ for $\forall$ a constant $a > 0$

for $n_0 = 1$

$c = 2$

$\therefore 1^k \leq a 2^1$

$\therefore \boxed{n_0 = 1, \& \; c = 2}$

Tutorial-2

DATE
PAGE

Ques 1- what is _ _ _ _ _ _ _ _ _ how?

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

i = 0, 1, 3, 6, 10, 15, 21, - - - - - - n (K - terms)

Let the sum of above K terms is S_K

$$S_K = 1+3+6+10+15+21, - - - - - + T_K - - - (1)$$

$$S_{K-1} = 1+3+6+10+15+21+ - - - - + T_{K-1} - - - (2)$$

Subtracting 2 from 1

$$T_K = S_K - S_{K-1} = 1+2+3+4+5+6+ - - - - - + K$$

We have $T_K = n$

$$\therefore 1+2+3+4+5+ - - - - - - - + K = n$$

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

$$\Rightarrow K = \frac{-1 + \sqrt{8n+1}}{2}$$

taking only positive values we get total no. of times the loop runs for $i = K+1$ $\frac{\sqrt{8n-1}}{2}$

DATE
PAGE

∴ Time complexity, $T(n) = O\left(\frac{\sqrt{8n+1}}{2}^2\right) = O(\sqrt{n})$

ues2 Write - - - - - - - - - - - and why?

Recursive Function

```
int fib (int n)
{ if (n <=1) ——— O(1) = C
        return n;
  return fib (n-1) + f(n-2) ——— T(n-1) + T(n-2)
}
```

Recurrance Relation, $T(n) = T(n-1) + T(n-2) + C$

$T(n-1) \simeq T(n-2)$

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

$T(n-2) = 2 * (2T(n-2-2) + C) + C$

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

$T(n-4) = 2 * (4T(n-2) + 3C) + C$

$= 8T(n-3) + 7C$

Generalising

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

Put $n-k = 0$

$n = K$

Put $n = K$

$T(n) = 2^n * T(0) + (2^n - 1)C$

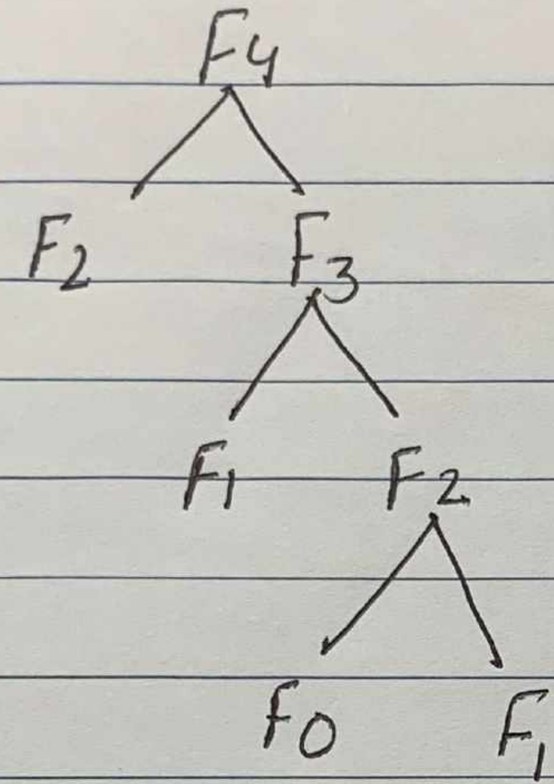
$= 2^n * 1 + 2^n C - C$

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

$= 2^n$

Time Complexity = $O(2^n)$

Space Complexity Space is proportional to the maximum depth of the recursion tree



Hence, space complexity of fibonacci recursion is $O(N)$

Ques 3 Write programs - - - - - - - complexity.

1) $n(\log n)$

```
for (i=1; K=n; i++)
  { for (j=1; j <= n; j = j*2)
  { sum = sum + i;
  }
  }
```

DATE
PAGE

2. n^3

```
for (i=0; i<n; i++)
  { for (j=0; j<n; j++)
    { for (k=0; k<n; k++)
      { sum = sum + k;
      }
    }
  }
```

3. Logn (Logn)

```
for (i=1; i<=n; i = i*2)
  { for (k=1; k<=n; k = k*2)
    { Sum = Sum + j;
    }
  }
```

Ques 4 Solve the Recurrance Relation $T(n) = T(\frac{n}{4}) + T(n/2) + cn^{\wedge}2$

$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + cn^{\wedge}2$$

$$\therefore T\left(\frac{n}{4}\right) = T\left(n/2\right)$$

$$\Rightarrow T(n) = 2T\left(\frac{n}{2}\right) + cn^2$$

DATE
PAGE

A $a \geq 1$ and $b > 1$

∴ Using master's method

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

$$c = \log_b^a$$

$$c = \log_2^2 = 1$$

$$f(n) > n^c$$

$$\therefore T(n) = O(f(n))$$
$$= O(n^2)$$

Guess what is - - - - - - - - - - - - - - function.

```
int fun(int n)
  { for(int i=1; i<=n; i++)

      for int j=1; j<n; j+=i)
        { some O(1)

        }
  }
```

sol^n

for $i = 1$, j i $1, 2, 3, 4$ - - - - run for n-times
for $i = 2$, j , $1, 3, 5$ - - - - - - upto $n/2$ times
for $i = 3$ j $1, 4, 7$ - - - - - - - run for $n/3$ times

$$T(n) = n + n/2 + n/3 + n/4 + \text{- - - - - -}$$

$n(1+1/2+1/3+1/4+\cdots)$

$= n\int_1^n dx/x$

$= \lceil \log x \rceil_1^n$

$\Rightarrow$ Time Complexity = $n\log n$

Ques 6 what should be time complexity of

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

where k is a constant

for first iteration $i = 2$

Second iteration $i = 2^k$

third iteration $i = (2^k)^k = 2^{k^2}$

n^{th} iteration, $i = 2^k$ loop ends at $2^i = n$

apply $\log n$, $= \log_2 k^i$

$k^i = \log n$

$i = \log_k (\log^n)$

Ques 7 Write a resume - - - - - - - - - - - analysis.

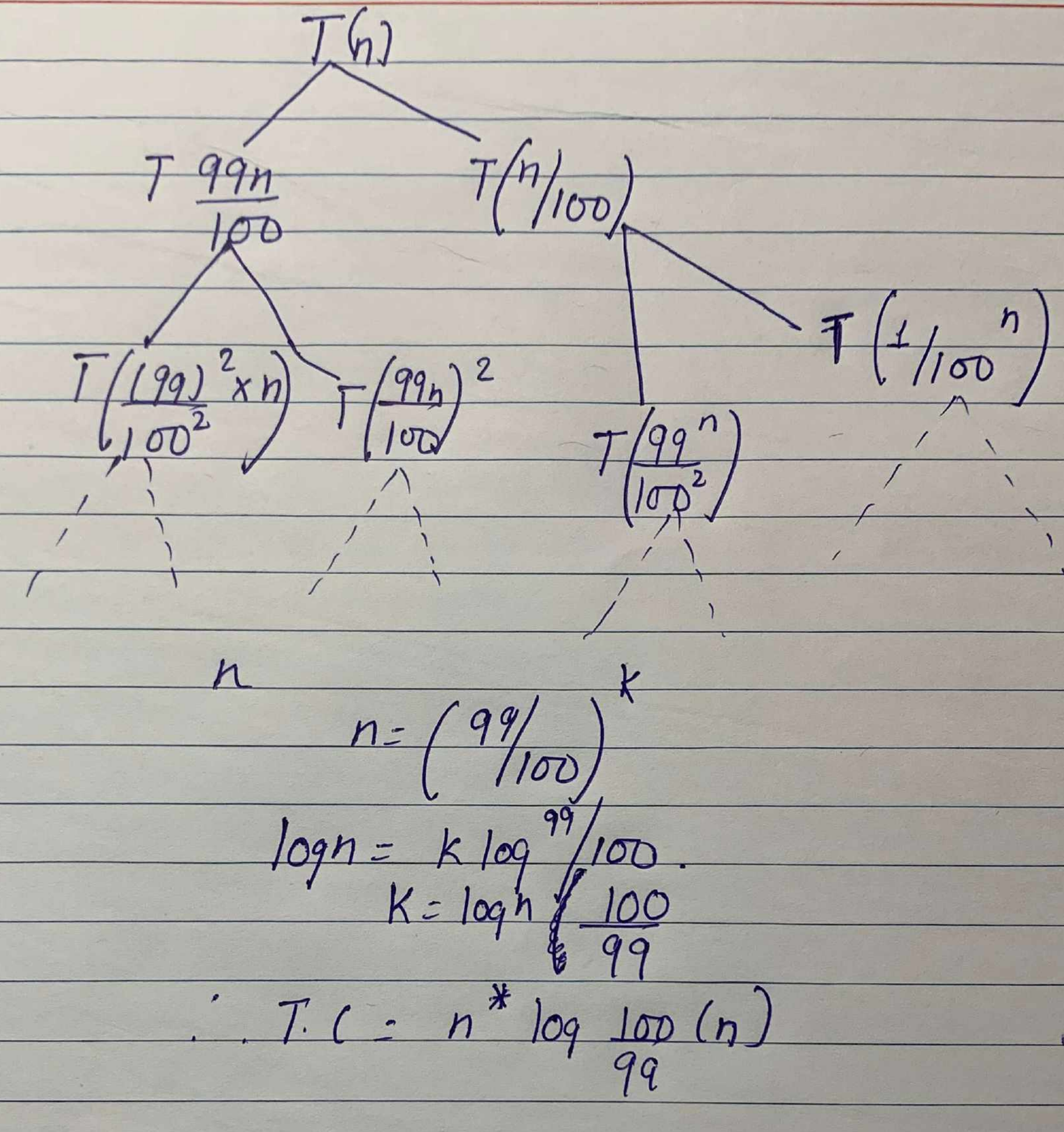
99 to 1 in quick sort

when pivot is where from front or end always

So,

$T(n) = T(99n/100) + T(n'/100) + O(n)$

$T(n) = T(99''/100) + T(''/100) + O(n)$



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

$$\log n = k \log \frac{99}{100}$$

$$K = \log n \Big/ \frac{100}{99}$$

$$\therefore T.C = n * \log_{\frac{100}{99}}(n)$$

Ques 8 Arrange - - - - - - - - - - - - - - - growth.

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

b) $1 < \log\log(n) < \sqrt{\log n} < \log(n) < 2\log(n) < \log(2n) < n$
$2n < 4n < \log n! < n\log(n) < 2(2\hat{}n)$

Tutorial - 3

Ques 1 - Write Linear Search pseudo code to search an element in sorted array with minimum no. of comparisons

Solution -
```
void linearSearch (int A[ ], int n, int key) {
    int flag = 0
    for (int i = 0; i < n; i++) {
    if (A[i] == key) {
            flag = 1;
            break; } }
    if(flag == 0)
    cout << "Not found";
    else
    cout << "found"; }
```

Ques 2 - Write pseudo code for iterative & recursive insertion sort - - - discussed.

Soln - Iterative -
```
for i=1 to n-1
    t = A[i], j = i-1
    while (j >= 0 && A[j] > t) {
        if (A[j+1] = A[j]) {
            j--; }
        A[j+1] = t; }
```

Recursive -
```
void insertionSort (int arr[], int n) {
    if (n <= 1)
        return;
    insertion sort (arr, n-1);
    int last = arr[n-1], j = n-2;
    while (j >= 0 && arr[j] > last) {
    arr[j+1] = arr[j];
        j--; }
    arr[j+1] = last; }
```

Insertion Sort is an online algo because insertion sort considers one i/p element per iteration & produces a partial soln w/o considering future elements. But in case of other sorting algorithms, we require access to the entire i/p, thus they are offline algo.

Ques 3 - Complexity of all algorithms.

Soln -

Algo	Worst Case	Best Case	Average Case
Bubble Sort	$O(n^2)$	$O(n)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n^2)$	$O(n)$	$O(n^2)$
count sort	$O(n+k)$	$O(n+k)$	$O(n+k)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
quick sort	$O(n^2)$	$O(n \log n)$	$O(n \log n)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Ques 4 - Divide all sorting algo into inplace/stable/online.

Soln -

Algo	Inplace	stable	Online
Bubble Sort	✓	✓	X
selection Sort	✓	X	X
Insertion sort	✓	✓	✓
Count Sort	X	✓	X
Merge Sort	X	✓	X
QuickSort	✓	X	X
Heap Sort	✓	X	X

Ques 5 - Write recursive / Iterative code for binary search and its time complexity.

Soln - Recursive - int binarySearch (int arr[], int l, int r,
int key) { if (r >= l)
{ int mid = l + (r - l) / 2;
if (arr[mid] == key) return mid;
if (arr[mid] > key) {
return binarySearch (arr, l, mid - 1, key);
return binarySearch (arr, mid + 1, r, key); }
return -1; }

Iterative - int binarySearch (int arr[], int l, int r, int key)
{ while (l <= r) {
int m = l + (r - l) / 2;
if (arr[m] == key)
return m;
if (arr[m] < key)
l = m + 1;
else
r = m - 1; }
return -1; }

	T(n)		S(n)	
	recursive	iterative	Recursive	iterative
Linear Search	O(n)	O(n)	O(1)	O(1)
Binary Search	O(log n)	O(log n)	O(1)	O(1)

Ques 6 - Write recurrence relation for binary recursive search.

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

Ques 7 - Find two indices such that A[i] + A[j] = k in min. time complexity.

Answ - void Sum (int A[], int k, int n) {
sort (A, A+n);
int i = 0, j = n - 1;
while (i < j) {
if (A[i] + A[j] == k) {
break;

```
else if (A[i] + A[j] > K)
    j--;
else
    i++; }
print (i, j); }
```

Here, sort func^n has $O(n \log n)$ complexity & for while loop it is $O(n)$ ∴, $T(n) = O(n \log n)$.

Ques 8 - Which sorting is best for practical use? Explain.

Sol^n We mostly prefer merge sort because of its stability & it can be best for very large data. Further the time complexity of merge sort is same in all cases that is $O(n \log n)$.

Ques 9 - what do you mean by - - - - merge sort.

Sol^n Pseudo code for Inversion count -

```
int getInvCount(int arr[], int n) {
    int c = 0;
    for (i = 0; i < n-1; i++) {
        for (int j = i+1; j < n; j++) {
            if (arr[i] > arr[j]) { c++; } } }
    return c; }
```

arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5}

Total inversions = 31

Inversion count for an array indicates how far (or close) the array is from being sorted. If the array is already sorted, inversion count is 0, but if array is sorted in reverse order, the inversion count is maximum.

Ques 10 - On which case - - - complexity.

Sol^n When the array is already sorted or sorted in reverse order quick sort gives the worst case $T(n)$ i.e. $O(n^2)$. But when the array is totally unsorted, it will give best case $T(n)$ i.e. $O(n \log n)$.

Ques 11 - Write Recurrence - - - and why!

Soln

Algo	Recurrence Reln	
	Best Case	Worst Case
Quick Sort	$2T(n/2)+n$	$T(n-1)+n$
Merge Sort	$2T(n/2)+n$	$2T(n/2)+n$

Both the algo are based on divide & conquer algorithms. Both the algo have the same time complexity in the best case & average because both the algo divides array into subparts, sort them & finally merge all the sorted parts.

Ques 12 - Selection _ _ _ _ sort.

Soln As the selection sort is not sort because it changes the relative posn of same elements after sorting. Selection sort can be made stable if instead of swapping the min element is placed in its posn by pushing every element one step forward. In simple words we use Insertion sort technique which means inserting element in its correct place.

Pseudo code for stable Selection Sort:

```
void Stable Selection (int A[], int n) {
 for(int i=0, i< n-1; i++) {
      int min = i;
       for (int j=i+1; j<n; j++){
        if (A[min] > A[j]) {
             min = j;
        int key = a[min];
   while (min > i) {
     a[min] = a[min -1];
      min --; }
     a[i] = key; } }
```

Ques 13 - Bubble sort _ _ _ _ _ _ it is sorted.

Answer - We can modify Bubble sort by placing a flag variable. If array is already sorted, we halt the process by checking the flag variable if its value changes or not.

Pseudo Code for Modified Bubble Sort :-

```
void bubble (int A[], int n) {
for (int i = 0; i<n; i++) {
  int swaps = 0;
  for(int j=0; j<n-i-1; j++) {
  if(A[j] > A[j+1]) {
    swap (A[j], A[j+1]);
      swaps++; }
    if(swaps == 0) break; } }
```

Ques 14 - Your computer ___ ___ ___ sorting.

Soln for the array of 4GB, we use external Sorting because array size is greater than RAM of our computer.

External Sorting :- These are sorting algo that can handle large data amounts which cannot fit in the main memory. ∴, only a part of array resides in the RAM during execution.

Example - K-way Merge sort.

Internal sorting :- These are sorting algo where the whole array needs to be in the RAM during execution.

Example - Bubble & Selection Sort etc.

Date

Tutorial-4

1. $T(n) = 3T(n/2) + n^2$

Sol:- $T(n) = aT(n/2) + f(n)$

$a \geq 1, b \geq 1$

on comparing

$a = 3, b = 2, f(n) = n^2$

Now,

$C = \log_b^a = \log_2^3 = 1.584$

$n^C = n^{1.584} < n^2$

$\therefore f(n) > n^C$

$\therefore T(n) = \theta(n^2)$

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

Sol:-

$a \geq 1, b > 1$

$a = 4, b = 2, f(n) = n^2$

$C = \log_2^4 = 2$

$\therefore n^C = n^2 = f(n) = n^2$

$\therefore T(n) = \theta(n^2 \log_2 n)$

5.) $T(n) = 16T(n/4) + n$

Sol:- $a = 16, b = 4$

$f(n) = n$

$C = \log_4 16 = \log_4 (4)^2 = 2$

$n^C = n^2$

$f(n) < n^C$

$\therefore T(n) = \theta(n^2)$

3. $T(n) = T(n/2) + 2^n$

Sol

$a = 1$

$b = 2$

$f(n) = 2^n$

$C = \log_b^a = \log_2^1 = 0$

$n^C = n^0 = 1$

$f(n) > n^C$

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

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

Sol:- $a = 2^n$

$b = 2, f(n) = n^2$

$C = \log_b^a = \log_2^{2^n}$

$= n$

$n^C = n^n$

$\therefore f(n) < n^C$

$\therefore T(n) = \theta(n^2 \log_2 n)$

6.) $T(n) = 2T(n/2) + n \log n$

Sol:- $a = 2, b = 2$

$f(n) = n \log n$

$C = \log_2 2 = 1$

$\therefore n^C = n^1 = n$

Since, $n \log n > n$

$\therefore f(n) > n^C$

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

7.) $T(n) = 2T(n/2) + n/\log n$

Sol: $a = 2, b = 2, f(n) = n/\log n$

$c = \log_2^2 = 1$

$\therefore n^c = n^1 = n$

Since, $\frac{n}{\log n} < n$

$\therefore f(n) < n^c$

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

8.) $T(n) = 2T(n/4) + n^{0.51}$

Sol: $a = 2, b = 4, f(n) = n^{0.51}$

$c = \log a = \log_4 2 = 0.5$

$\therefore n^c = n^{0.5}$

Since, $n^{0.5} < n^{0.51}$

$f(n) > n^c$

$\therefore T(n) = \Theta(n^{0.51})$

9.) $T(n) = 0.5T(n/2) + 1/n$

Sol: $a = 0.5, b = 2$

Since acc. to Master Theorem $a \geq 1$, but here a is 0.5 So we cannot apply master theorem.

10.) $T(n) = 16T(n/4) + n!$

Sol: $a = 16, b = 4, f(n) = n!$

$\therefore c = \log_b^a = \log_4 16 = 2$

Now, $n^c = n^2$

As $n! > n^2$

$\therefore T(n) = \Theta(n!)$

11.) $4T(n/2) + \log n$

Sol: $a = 4, b = 2, f(n) = \log n$

$a = \log_b^a = \log_2^4 = 2$

$\therefore n^c = n^2$

$f(n) = \log n$

Since $\log n < n^2$

$\therefore f(n) < n^c$

$\therefore T(n) = \Theta(n^c)$

$= \Theta(n^2)$

12.) $T(n) = \text{Sqrt}(n)\,T(n/2) + \log n$

Sol: $a = \sqrt{n}, b = 2$

$\therefore c = \log_b^a = \log_2 \sqrt{n} = \frac{1}{2}\log_2 n$

$\therefore 1/2 \log_2 n < \log(n)$

$\therefore f(n) > n^c$

$\therefore T(n) = \Theta(f(n))$

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

Date

13.) $T(n) = 3T(n/2) + n$

Sol:
$a = 3, b = 2, f(n) = n$
$c = \log_b a = \log_2 3 = 1.5849$
$\therefore n^c = n^{1.5849}$
$\therefore n < n^{1.5849}$
$\therefore f(n) < n^c$
$\therefore T(n) = \Theta(n^{1.5849})$

14.) $T(n) = 3T(n/3) + \text{sqrt}(n)$

Sol: $a = 3, b = 3$
$c = \log_b a = \log_3 3 = 1$
$\therefore n^c = n^1 = n$
As $\text{sqrt}(n) < n$
$\therefore f(n) < n^c$
$\therefore T(n) = \Theta(n)$

15.) $T(n) = 4T(n/2) + n$

Sol: $a = 4, b = 2$
$c = \log_b a = \log_2 4 = 2$
$\therefore n^c = n^2$
$\therefore cn < n^2$ (for any constant)
$\therefore f(n) < n^c$
$\therefore T(n) = \Theta(n^2)$

16.) $T(n) = 3T(n/4) + n\log n$

Sol: $a = 3, b = 4, f(n) = n\log n$
$c = \log_b a = \log_4 3 = 0.792$
$n^c = n^{0.792}$
$\therefore n^{0.792} < n\log n$
$\therefore T(n) = \Theta(n\log n)$

17.) $T(n) = 3T(n/3) + n/2$

Sol: $a = 3, b = 3$
$c = \log_b a = \log_3 3 = 1$
$f(n) = n/2$
$\therefore n^c = n^1 = n$
As $n/2 < n$
$\therefore f(n) < n^c$
$\therefore T(n) = \Theta(n)$

18.) $T(n) = 6T(n/3) + n^2\log n$

$a = 6, b = 3$
$c = \log_b a = \log_3 1 = 1.6309$
$n^c = n^{1.6309}$
As $n^{1.6309} < n^2\log n$
$\therefore T(n) = \Theta(n^2\log n)$

Date

19.) $T(n) = 4T(n/2) + n\log n$

Sol :- $a = 4, b = 2, f(n) = \frac{n}{\log n}$

$c = \log_b a = \log_2 4 = 2$

$n^c = n^2$

$\therefore \frac{n}{\log n} < n^2$

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

20.) $T(n) = 64T(n/8) - n^2\log n$

Sol :- $a = 64, b = 8$

$c = \log_b a = \log_8 64 = \log_8 (8)^2$

$c = 2$

$\therefore n^c = n^2$

$\therefore n^2 \log n > n^2$

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

21.) $T(n) = 7T(n/3) + n^2$

$a = 7, b = 3, f(n) = n^2$

$c = \log_b a = \log_3 7 = 1.7712$

$n^c = n^{1.7712}$

$\therefore n^{1.7712} < n^2$

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

22.) $T(n) = T(n/2) + n(2 - \cos n)$

Sol :- $a = 1, b = 2$

$c = \log_b a = \log_2 1 = 0$

$\therefore n^c = n^0 = 1$

$\therefore n(2 - \cos n) > n^c$

$\therefore T(n) = \Theta(n(2 - \cos n))$

Tutorial -5

Date

Ques 1- Answer- D/f B/w DFS & BFS.

BFS	DFS
1. Stands for Breadth First Search.	1. Stands for Depth First Search.
2. DFS uses queue to find shortest path.	2. It uses stack.
3. BFS is better when target is closer to source.	3. DFS is better when target is far from source.
4. As BFS consider all neighbours so it is not suitable for decision tree used in puzzle games.	4. DFS is more suitable for decision tree. As with one decision we need to traverse further to argument the decision. If we search the conclusion.
5. BFS is slower than DFS.	

~~Ques 2-~~ Applications of DFS -

1. Using DFS, we can find path b/w two vertices.
2. We can perform topological sorting which is used to scheduling of jobs.
3. We can use DFS to detect cycles.
4. Using DFS, we can find strongly connected components of a graph.

Applications of BFS -

1. BFS may also used to detect cycles.
2. finding shortest path and MST in unweighted graph.
3. In networking finding a route for packet transmission
4. finding a route through GPS navigation system.

Date

Ques 2- Soln BFS uses Queue data structure. In BFS, you mark any node in graph as source node & start traversing from it. BFS traverses all the nodes in the graph and keeps dropping them as completed. BFS visited an adjacent unvisited node, marks it as done & inserts it into Queue.

DFS uses Stack because DFS traverse a graph in a depthwise motion & uses stack to remember to get the next vertext to start a search, when a dead end occurs in any iteration.

Ques 3- Soln Sparse Graph: A graph in which the no. of edges is much less than possible no of edges.

Dense Graph: A dense graph is a graph in which the no. of edges is close to the maximal no. of edges of edges.

If the graph is sparse, we should store it as List of edges. Alternatively, if a graph is dense, we should store it as a adjacency matrix.

Ques 4- Soln DFS can be used to detect cycle in a graph. DFS for a connected graph produces a tree. There is a cycle in a graph only if there is a back edge present in the graph. A back edge is an edge that is from a node to itself or one of its ancestor in the tree produced by DFS.

BFS can also be used to detect cycles. Just perform BFS while keeping a list of previous nodes at each node visited & else constructing a tree from the starting node. If a node is visit that is already marked by BFS, It finds a cycle.

Mohan

Date

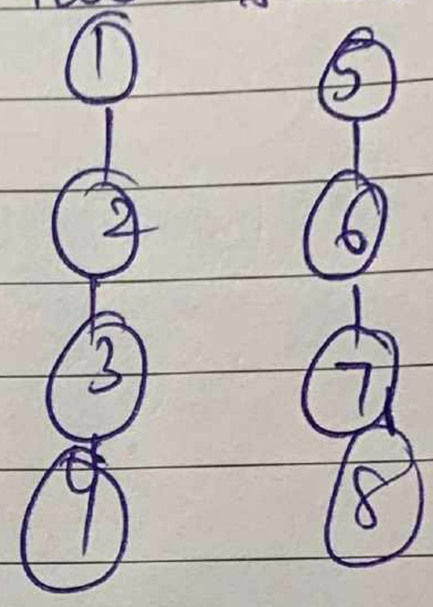
Ques 5- Soln Disjoint Set Data Structure:

It allows to find out whether the two elements are in same set or not.

A disjoint set can be defined as subsets when there is no common element b/w the two sets.

Example:- S1 = {1, 2, 3, 4}

S2 = {5, 6, 7, 8}



Operations performed:

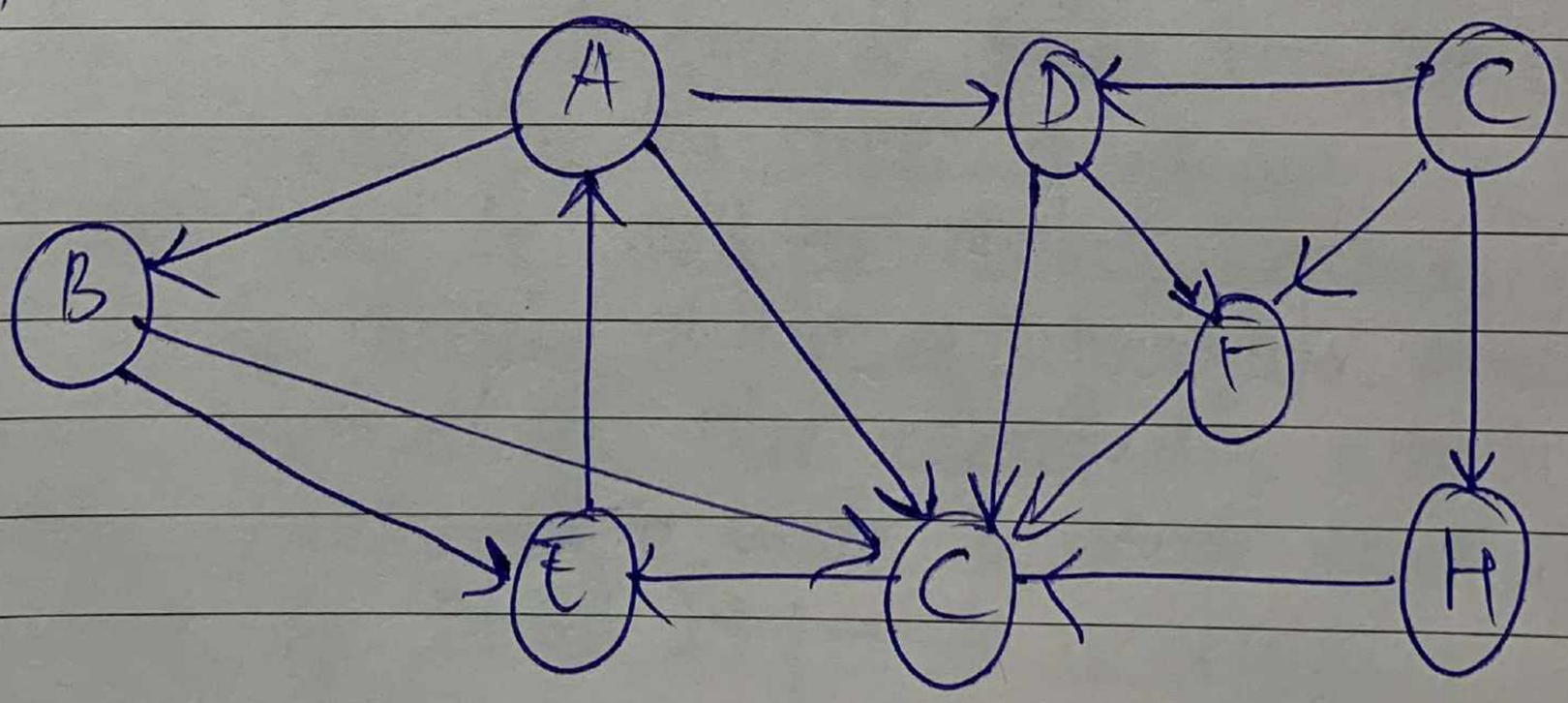
(1) Find:

```
int Find (int v) {
if (v == parent [v]) {
    return v; }
return parent [v] = find (parent [v]); }
```

(2) Union:

```
void Union (int a, int b) {
    a = find (a)
    b = find (b)
if (a != b) {
    if (size [a] < size [b]) {
        swap (a, b) }
parent [b] = a;
    size [a] += size [b]; } }
```

Ques 6- Soln



Mohan

Date

BFS :

Node : B E C A D F

parent : A B B E A D

Path : B → E → A → D → F

DFS :

Node : B B C E A D F

Stack : B CE EE AE DE FE E

Path : B → C → E → A → D → F

Ques 7- Soln

$V = \{a\}\{b\}\{c\}\{d\}\{e\}\{f\}\{g\}\{h\}\{i\}\{j\}$

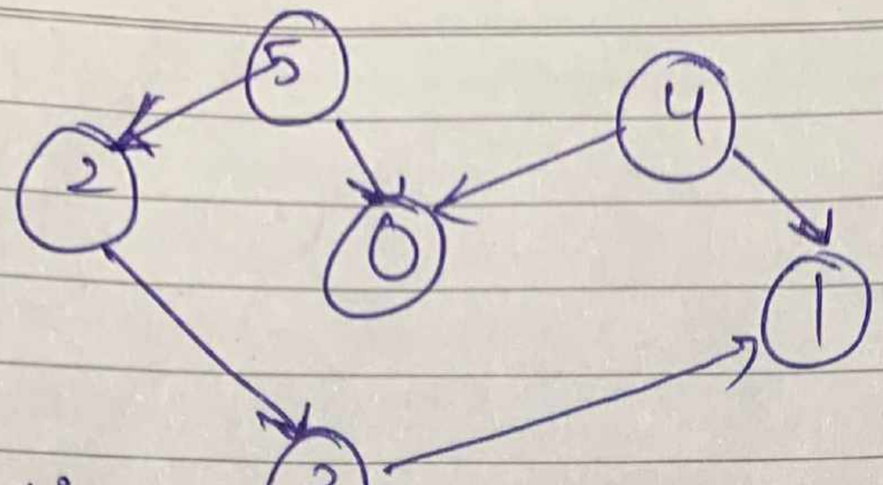
$E = \{a,b\}\{a,c\}\{b,c\}\{b,d\}\{e,f\}\{e,g\}\{h,i\}\{j\}$

(a,b)	$\{a,b\}\{c\}\{d\}\{e\}\{f\}\{g\}\{h\}\{i\}\{j\}$
(a,c)	$\{a,b,c\}\{d\}\{e\}\{f\}\{g\}\{h\}\{i\}\{j\}$
(b,c)	$\{a,b,c\}\{d\}\{e\}\{f\}\{g\}\{h\}\{i\}\{j\}$
(b,d)	$\{a,b,c,d\}\{e\}\{f\}\{g\}\{h\}\{i\}\{j\}$
(e,f)	$\{a,b,c,d\}\{e,f\}\{g\}\{h\}\{i\}\{j\}$
(e,g)	$\{a,b,c,d\}\{e,f,g\}\{h\}\{i\}\{j\}$
(h,i)	$\{a,b,c,d\}\{e,f,g\}\{h,i\}\{j\}$

Mohan No. of connected components = 3

Date

Ques 8 - Soln



Adjacency list:

0 →
1 →
2 → 3
3 → 1
4 → 0, 1
5 → 2, 0

0	1	2	3	4	5
false	false	false	false	false	false

Stack (empty)

Step 1: Topological sort [0], visited [0] = true
Stack |0|

Step 2: Topological sort (1), visited [1] = true
Stack |0|1|

Step 3: Topological sort (2), visited [2] = true

~~Step 4:~~ Topological sort (3),

visited [3] = true
Stack |0|1|3|2|

Step 4: Stack |0|1|3|2|4|

Step 5: Stack |0|1|3|2|4|5|

Step 6: Print all elements of Stack from top to bottom
→ 5, 4, 2, 3, 1, 0

Ques 9 - Soln Algorithms that Use Priority Queue :

(i) Dijkstra's Algo : when graph is stored in the form of list or matrix, priority queue can be used to extract min. efficiency while implementing,

Date

(ii) Dijkstra's Algo.

(iii) Prim's Algo : It is used to implement prims algo m to store key of nodes to extract min key nodes at every step.

(iii) Data Compression : It is used to implement Huffmann code which is used to compress data.

Ques 10 -

	Min Heap	Max Heap
1.	In Min Heap, the key present at root node must be less than or equal to among the keys present at all its children.	1. In maxheap, the key present at root node must be greater or equal to the key present at all its children.
2.	Uses the ascending priority.	2. Uses descending priority.
3.	Minimum key present at root node.	3. Maximum key present at root node

Tutorial - 6

Ques 1 - What do you mean by min spanning Tree? what is applications of MST.

Ans by - A minimum spanning tree is a subset of edges of a connected, edge-weighted undirected graph that connects all the vertices ~~together~~, w/o any cycles & with min possible total edge weight.

Applications -

- Designing local Area Network.
- Laying pipelines connecting offshore drilling sites, refineries and consumer markets.
- Suppose you want to construct highways or railroads spanning several cities, then MST is used.
- To reduce cost, you use the concept of MST to connect the houses.

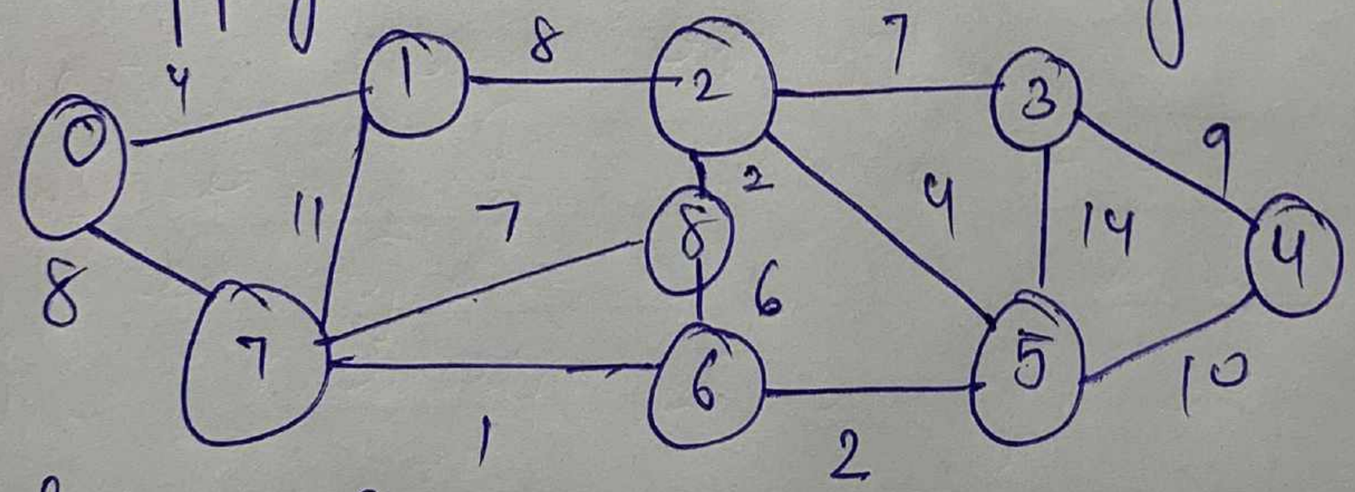
Ques 2 - Please analyze - - - algo.

Soln

Algo	T(n)	S(n)
Prims	$O(V^2)$	$O(V+E)$
Kruskal	$O(E \log V)$	$O(\log E)$
Dijkstra's	$O(V+E)$	$O(V+E)$
Bellman Ford	$O(VE)$	$O(V)$

Ques 3 - Apply - - - - weight.

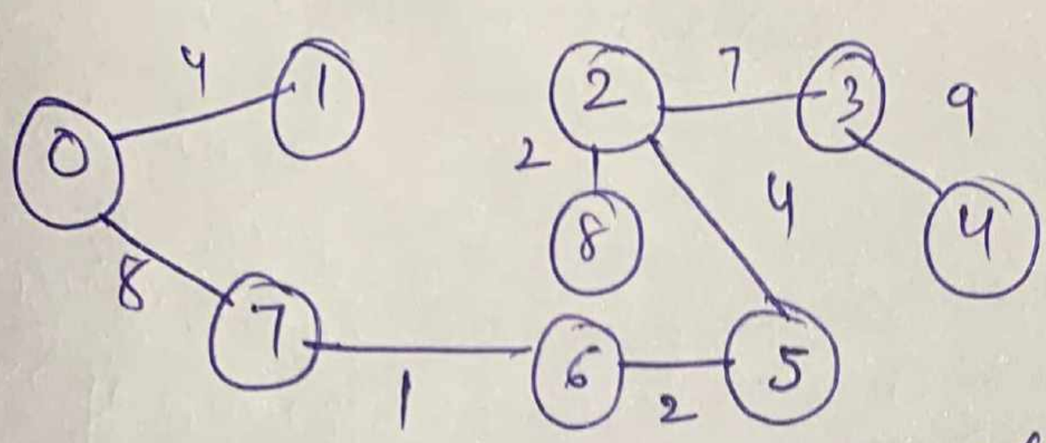
Soln



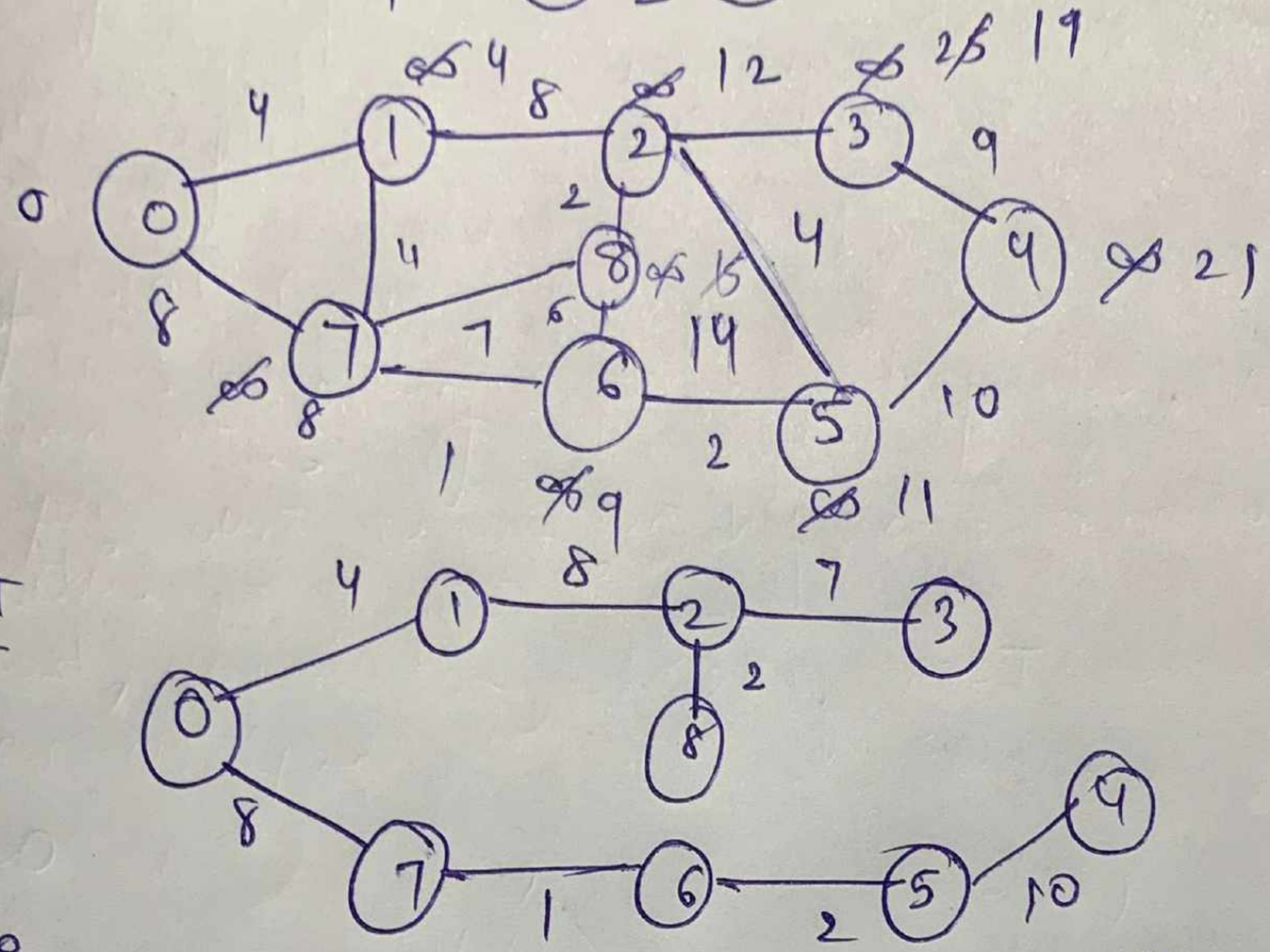
Kruskal

Path	weight	Path	weight
7 → 6	1	7 → 8	7
6 → 5	2	1 → 2	8
2 → 8	2	3 → 4	9
0 → 1	4	5 → 4	10
2 → 5	4	1 → 7	11
8 → 6	6	3 → 5	14
2 → 3	7		

TIMARIND - 6



Prism's :



MST

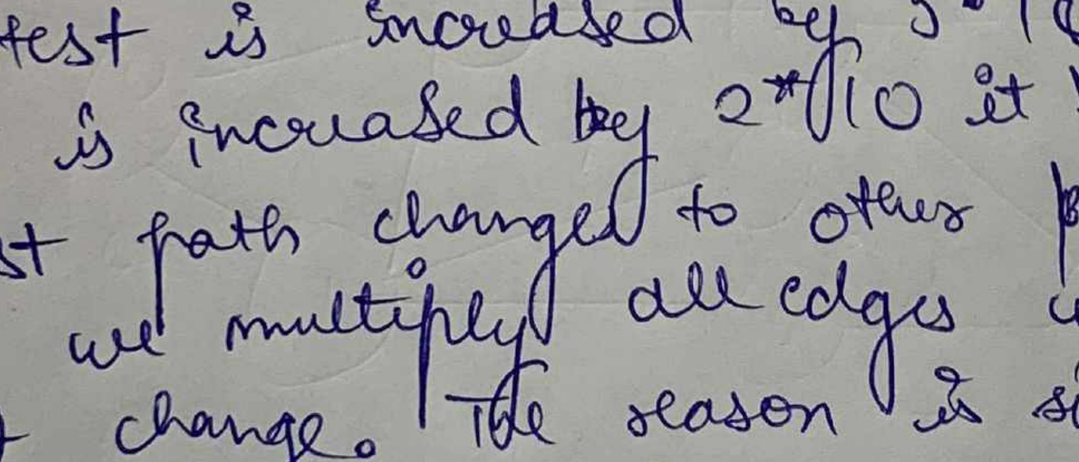
Ques 4 - Given a - - - - - 10 units.

Soln (i) The shortest path may change. The reason is that these may be diff no. of edges in diff paths from 's' to 't'. For eg, let shortest path of wt 15 & has 5 edges. Let there be another path with 2 edges & total wt 25. the wt of shortest is increased by 5*10 becomes 15+50. wt of other path is increased by 2*10 it becomes 25+20. so the shortest path changed to other path whose wt is 45.

(ii) If we multiply all edges wt by 10, the shortest path doesn't change. The reason is simple, wt of all paths from 's' to 't' get multiplied by same amount. The no. of edges on a path doesn't matter.

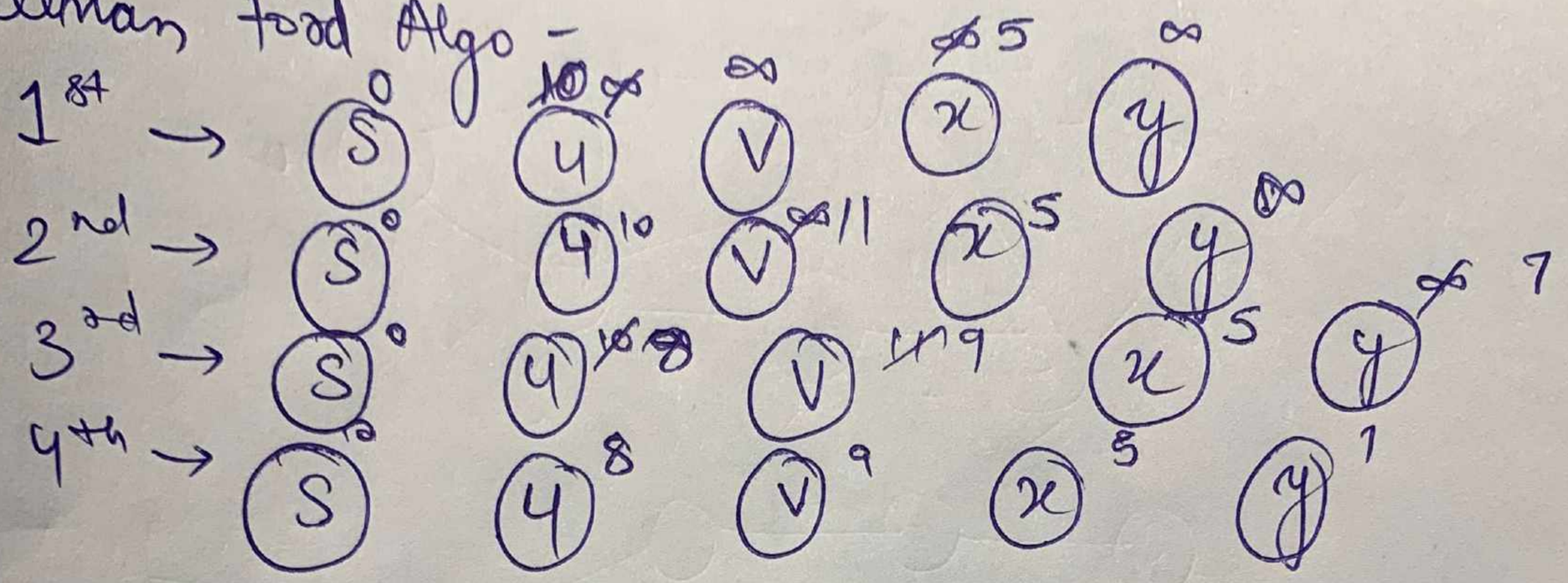
Ques 5 - Dijkstra's & Bellman Ford Algo.

Soln Dijkstra's Algo is -



node	shortest dist from source node
4	8
x	5
y	9
v	7

Bellman ford Algo -



final graph