ASSIGNMENT:- 01

Ans 01> Asymptotic Notation:- It is the mathematical notation used to describe the running-time of an Algorithm.

Different types of Notations are:-

(i) Big O Notation (Big-O) → It represents the upper bound of the algorithm.
$f(n) = O(g(n))$ iff $f(n) \leq c(g(n))$

(ii) Omega Notation (Ω) → It represents the lower bound of the algorithm.
$f(n) = \Omega(g(n))$ iff $f(n) \geq c(g(n))$

(iii) Theta Notation (Θ) → It represents upper and lower bound of the algorithm.
$f(n) = \Theta(g(n))$ iff. $c_1 g(n) \leq f(n) \leq c_2 g(n)$

Ans. 02>

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

i = 1, 2, 3, 4, 5, 6, 7
i value = 2, 4, 8, 16, ...

forms a GP; $a_n = a(2)^{n-1}$ $\quad a=1$

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

$\log n = \log 2^{k-1}$ — taking log both sides

$\log n = k-1$

$k = \log n + 1$ $\Longrightarrow$ $T(n) = O(\log n)$. — Ans.

Ans. 03> $T(n) = 3T(n-1)$ if $n > 0$; otherwise 1

$T(1) = 3T(0) = 3$

$T(2) = 3T(1) = 3 \cdot 3T(0) = 3^2$

$T(k) = 3^k$

$T(n) = 3^n$

$T(n) = O(3^n)$. — Ans.

Ans 04.) $T(n) = 2T(n-1) + 1$; if $n > 0$; otherwise 1.

$T(0) = 1$

$T(1) = 2T(0) - 1 = 1$

$T(2) = 2T(1) - 1 = 2(2(T(0) - 1)) - 1 = 2^2 T(1) - 5$

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

Substituting $k = n-1$

$T(n) = 2^{n-1} T(1) - [2^0 + 2^1 + \ldots 2^{n-2}]$

$= 2^{n-1} \times 1 - [2^{n-1} - 1]$

$T(n) = 1$

$T(n) = O(1)$ — Ans

Ans 05.)

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

i = 1 2 3 ---- loop ends
s = 1 1+2 1+2+3 ---- when s > n

Hence;

$s > n$

$1+2+3+4 \ldots + K > n$

$\frac{K(K+1)}{2} > n$

$K^2 > n$

$\Rightarrow K > \sqrt{n}$.

$T(n) = O(\sqrt{n})$. — Ans

Ans 06.)

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

i = 1, 4, 9, 16 ---- till $i^2 = n$

Hence;

$i * i < n$

$K * K < n$

$K^2 < n$

$K < \sqrt{n}$

$T(n) = O(\sqrt{n})$ — Ans

Ans-7)

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

1st loop:-
$i = n/2$ to n; $i++$
$T(i) = O(n)$

2nd loop:-
$j = 1$ to n; $j*2$
$T(j) = O(\log n)$

3rd loop:-
$k = 1$ to n; $k*2$
$T(k) = O(\log n)$

Hence:-
$T(n) = T(i) \times T(j) \times T(k)$
$= O(n) \times O(\log n) \times O(\log n)$

$\boxed{T(n) = O(n \log^2 n).}$ — Ans.

Ans 8:)

```
function(int n)
{
 if (n==1) return;        — T(1)
for(i=1 to n)
{
for(j to n)
 {
 printf("*");            —— T(n^2)
 }
 function(n-3)           —— T(n-3)
}
```

{
Hence;
Rel^n :- $T(n) = T(n-3) + n^2$; $T(1) = 1$

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

$T(7) = T(4) + n^2 = 1^2 + 4^2 + 7^2$

$\vdots$

$T(n) = 1^2 + 4^2 + 7^2 + \cdots n^2$

$= \frac{n(n+1)(2n+1)}{6} = \boxed{n^3}$ - nearest

$\boxed{T(n) = O(n^3).}$
Ans

NAME:- AYUSH RAJ

Ans. 9)

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

$n - i = 1 \quad 2 \quad \cdots \quad n$

$n - j = 1 \text{ to } 1 \quad 1 \text{ to } 2 \quad \cdots \quad 1 \text{ to } n$

Hence;

$$\boxed{T(n) = O(n^2).}$$ — Ans.

Ans 10)

$f(n) = n^k$; $k \geq 1$

$f_2(n) = c^n$. $c > 1$

Asymptotic rel^n b/w f_1 & f_2 :-

Big-O $\longrightarrow$ $f_1(n) = O(f_2(n)) = O(c^n)$

and $n^k \leq G * c^n$ [G is some constant.]

ASSIGNMENT:-02

Ans 01>

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

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

Ans.

$j=0+1$; $i=0+1$

$j=2$; $j=0+1+2$

$j=3$; $j=0+1+2+3$

$\vdots$

$j=n$; $k=n$ times

$0+1+2+\cdots+n > n$

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

$k^2 > n \quad k > \sqrt{n}$

Ans 02> Recurrence rel^n for Fibonacci series:-

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

Assume $T(n-1) \approx T(n-2)$

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

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

$= 4[2T(n-6)] = 8T(n-6)$

$\vdots$

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

$n-2k=0$

$n=2k$

$k = n/2$

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

$T(n) = 2^{n/2}$

$T(n) = 2 \cdot (2^{n/2}) = 2^{n/2}$

if $T(n-2) \approx T(n-1)$

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

$= 2(2T(n-2)) = 4T(n-2)$

$\vdots$

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

$n-1 \leq 0$

$k = n$

$T(n) = 2^k T(0)$

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

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

Ans 03)

```
for(i=0; i<n; i++)
{
for(j=1; j<n; j=j*2)
{
  // some O(1)
}
}
```
} — $O(n \log n)$

```
for(i=0; i<n; i++)
{
for(j=0; j<n; j++)
{
for(k=0; k<n; k++)
{
//some O(1);
}
}
}
```
} — $O(n^3)$.

```
for(i=1; i<=n; i=i*2)
{
for(j=1; j<=n; j=j*2)
{
  //some O(1);
}
}
```
} — $O(\log(\log n))$.

Ans 04) $T(n) = T(n/4) + T(n/2) + Cn^2$

lets assume $T(n/2) >= T(n/4)$

so; $T = 2T(n/2) + Cn^2$

Applying Master's theorm;

$a = 2$ $\quad f(n) = n^2$

$b = 2$

$c = \log_b a = 1$.

$n^c = n$

Comparing;

$f(n) > n^1$

so, $\boxed{T(n) = \Theta(n^2).}$ — Ans

Ans 05.)

```
void fun (int n)
{
for (i=1; i<n; i++)
{
for (j=1; j<n; j+=i)
{
// some O(1);
}
}
}
```

i=	1	2	3	4	...	n
j=	1	1	1	1	...	1
	2	3	4	5	...	2
	3	5	7	9	...	3
	...	7	...	...	...	...
	n	...				n
		n/2	n/3	n/4		

] — n^2

Hence; $T(n) = O(n^2) + O(n^2/2) + O(n^2/3) + \cdots$

$\boxed{T(n) = O(n^2)}$ — Ans.

Ans 06)

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

pow(i,k) = O(log n)
= log k

loop ends $2^{k^n} > n$

$\log 2^{k^n} > \log n$ — taking log both sides.

$k^n \log 2 > \log n$

$\log k^n > \log(\log n)$

$n \log k > \log \log n$

$n > \frac{\log(\log n)}{\log k}$

Hence;

$\boxed{T(n) = \log_2(\log_2 n)}$

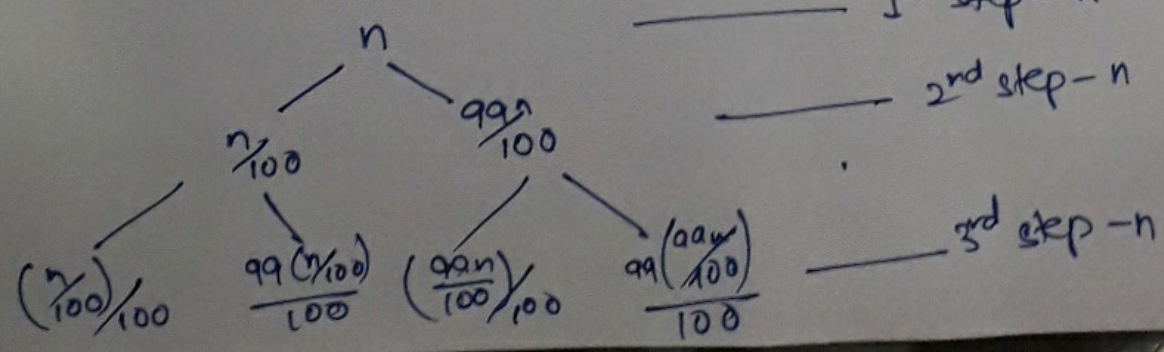
Ans

Ans 07.)

Considering the statements;

$T(n) = T(n/100) + T(99n/100) + O(n)$

where $n/100$ & $99n/100$ are for parts & $O(n)$ is partitioning algo.



So; it will remain n at each step.

So; time complexity = $O(n * \log_{10/9} n)$ if we take longer branch

$= \Omega(n * \log_{10} n)$ — TIME COMPLEXITY.

Question 03:

a) Order is :-

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

(b) Order is.-

$1 < \sqrt{\log n} < \log n < 2 \log n < \log_2 N < N < 2N < 4N < \log(\log N) < N \log N < \log N! < N! < N^2 < 2 \times 2^N$.

(c) Order is :-

$96 < \log_8 N < \log_2 N < n \log_6 N < n \log_2 N < \log n! < N! < 5N < 8N^2 < 7N^3 < 8^{2n}$.

ASSIGNMENT :- 03

Ans 1)

```
while(low <= high)
{
 mid = low+higly/2;
 if (arr[mid] == key)
   return true;
else if (arr[mid] > key)
     high = mid-1;
  else
     low = mid+1;
}
return false;
```

Ans 2) Iterative Insertion Sort

```
for(i=1; i<n; i++)
{
 j = i-1;
 x = A [i];
while (j>-1 && A[j]>x)
{
 A [j+1] = A [j]
   j--;
}
 A[j+1] = x;
 }
```

Recursive Insertion Sort

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

It is a kind of ONLINE SORTING because whenever a new element comes, insertion sort defines its right place.

Ans 3.)

Sorting	Time Complexity
Bubble Sort	$O(n^2)$
Insertion Sort	$O(n^2)$
Selection Sort	$O(n^2)$
Merge Sort	$O(n \log n)$
Quick sort	$O(n \log n)$
Count Sort	$O(n+k)$
Bucket sort	$O(n)$

Ans 4.) Online Sorting — Insertion sort

Stable sorting — Merge sort, Insertion sort, Bubble sort.

Inplace sorting — Bubble Sort, Insertion sort, Selection sort

Ans 5.) Iterative Binary Search

```
while (low <= high)
{ int mid = low + high/2;
  if (arr[mid] == key)
    return true;
  else if (arr[mid] > key)
    high = mid - 1;
  else
    low = mid + 1;
}
```

] - $O(\log n)$.

Recursive Binary Search:-

```
while (low <= high)
{ int mid = low+high/2
 if (arr[mid] == key)
 return true;
else if (arr[mid] > key)
 return BinarySearch(arr, low, mid-1);
else
  BinarySearch (arr, mid+1, high);
}
return false;
```

} - O(log n)

Ans 6) Required eq :-

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

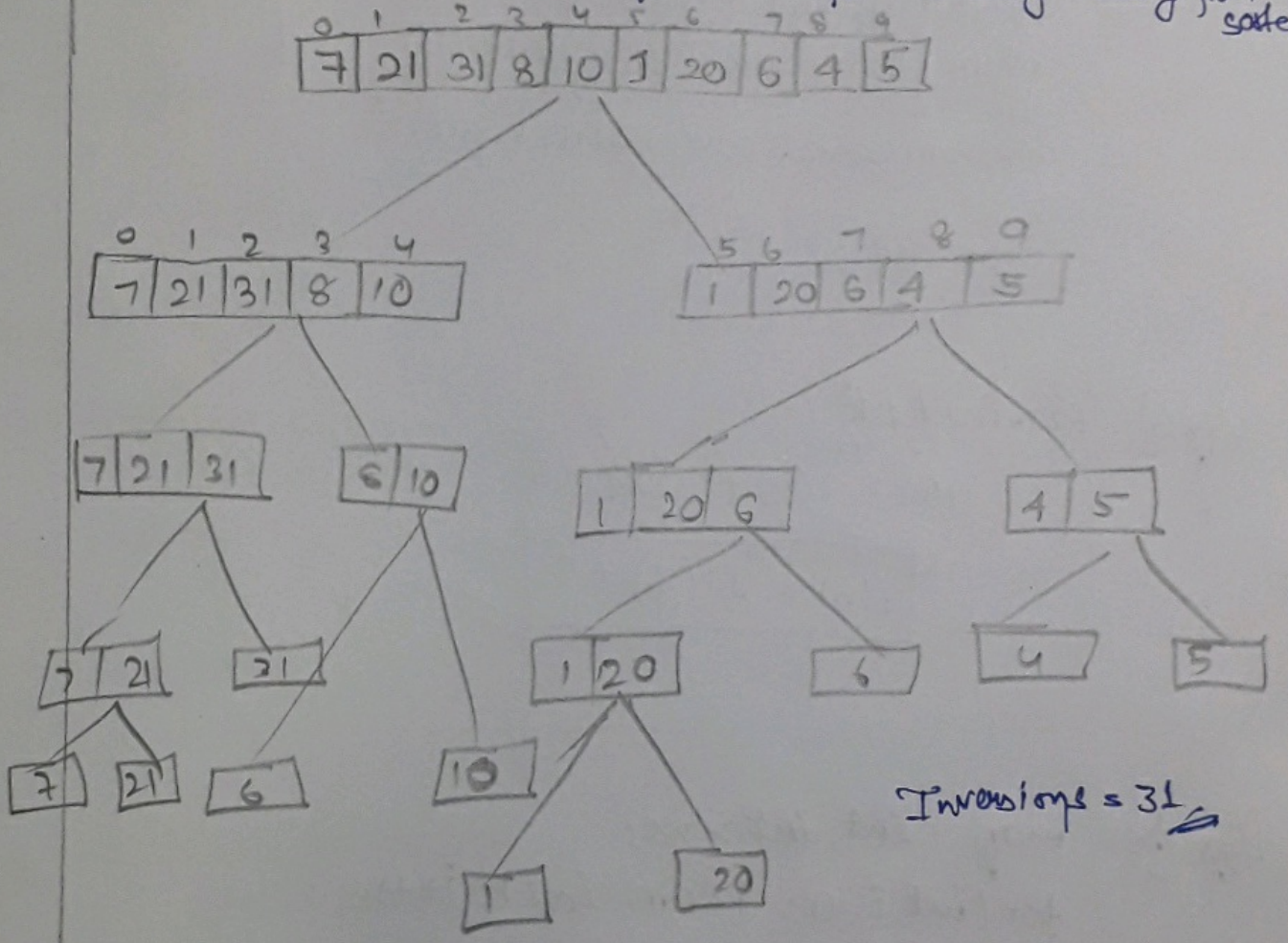
$$T(n) = 2T(n/2) + C$$

Ans 7)

```
map <int, int> m;
for(int i=0; i< arr.size(); i++)
{
  if (m.find(target - arr[i]) = m.end())
    m[arr[i]] =1;
else
  {
  cout << i << " " << map[arr[i]];
  }
}
```

Ans 8.) Quick sort is the fastest general purpose. In most practical situations, Quick sort is the method of choice. If stability is important and space is available, Merge sort is best.

Ans 9.) Inverse indicates :- It indicates how close or far the array is being from sorted.



Inversions = 31

Ans 10.) Worst case :-

It occurs when the pivot element is always an extreme (smallest or largest) element. This happens when input array is sorted and reverse sorted and either first or last element is picked as pivot.

$O(n) = n^2$

Best case :-

It occurs when pivot element is the middle element or near to the middle element.

$O(n) = n \log n$

Ans 11.)

Merge sort :- $T(n) = 2T(n/2) + O(n)$

Quick sort :- $T(n) = 2T(n/2) + n+1$

Parameter	Quick sort	Merge Sort
Partition	Splitting is done in any ratio.	Array is parted into just two halves.
Working	Smaller array	Fine on any size of array
Addl space	Less (inplace)	More (not inplace).
Efficiency	Inefficient on larger array.	effective on all types of array.
Sorting Method	Internal	External.
Stability	Not stable	Stable.

Ans 12.)

Stable selection sort :-

```
void stableselectionSort (int a[], int n)
{
for (i = 0 to n-1; i++)
{
int min = i;
for (j = i+1 to 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; } }
```

Ans 13.)

```
void BubbleSort (int arr[], int n)
{
int i,j;
bool swapped;
for (i=0 to n-1; i++)
{
swapped = false;
for (j= 0 to n-i-1; j++)
{if (arr[j] > arr[j+1])
{swap(arr[j], arr[j+1]);
swapped = true;
}
}
if (swapped == false)
    break;
}
}
```

Ans 14)

We will use Mergesort because we can divide the 4GB data into 4 packets of 1GB and sort them seperately and combine them later.

Internal Sorting:-
All the data is sorted in memory at all times during sorting is in progress.

External Sorting:-
All the data is sorted is outside memory and loaded into memory in small chunck's.

ASSIGNMENT:- 01

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

$a = 3; \; b = 2 \quad f(n) = n^2$

$n^{\log_b a} = n^{\log_2 3}$

Comparing;

$n^{\log_2 3} < n^2$ [case III]

According to master theorm;

$T(n) = \Theta(n^2)$ — Ans

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

$a = 4; \; b = 2$

$n^{\log_b a} = n^{\log_2 4} = n^2 = f(n)$ [case II]

According to Master's theorm;

$T(n) = \Theta(n^2 \log n)$ — Ans

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

$a = 1; \; b = 2$

$n^{\log_2 1} = n^0 = 1 < 2^n$ (case III)

According to Master's theorm;

$T(n) = \Theta(2^n)$ — Ans

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

Master theorm is not applicable as it's a is not a function of n.

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

$a = 16 ; b = 4 ; f(n) = n$

$n^{\log_b a} = n^{\log_4 16} = n^2 > f(n)$ — (case I)

$T(n) = \Theta(n^2)$ — Ans

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

$a = 2 ; b = 2 ; f(n) = n\log n$

$n^{\log_b a} = n^{\log_2 2} = n < f(n)$ (case III)

$T(n) = \Theta(n \log n)$ — Ans.

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

$a = 2 ; b = 2 ; f(n) = n/\log n$

$n^{\log_b a} = n^{\log_2 2} = n > f(n)$ — (case I)

$T(n) = \Theta(n)$ — Ans.

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

$a = 2 ; b = 4 ; f(n) = n^{0.51}$

$n^{\log_b a} = n^{\log_4 2} = n^{0.5} < f(n)$ — (case III)

$T(n) = \Theta(n^{0.51})$ — Ans.

Ans 9.) $T(n) = 0.8T(n/2) + 1/n$

Master's theorm is not applicable as $a < 1$.

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

$n^{\log_b a} = n^{\log_4 16} = n^4 < n!$ — (case III)

$T(n) = \Theta(n!)$ — Ans.

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

$n^{\log_b a} = n^2 > f(n)$ — (case I)

$T(n) = \Theta(n^2)$ — Ans

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

Master's theorem is not applicable as a is not a constant.

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

$a = 3 \; ; \; b = 2 \; ; \; f(n) = n$

$n^{\log_b a} = n^{\log_2 3} > n$ — (case I)

Hence; $T(n) = \Theta(n^{\log_2 3})$. — Ans

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

$a = 3 \; ; \; b = 3 \; ; \; f(n) = \sqrt{n}$

$n^{\log_b a} = n^{\log_3 3} = n > \sqrt{n}$ (case I)

Hence; $T(n) = \Theta(n)$ — Ans

Ans 15) $T(n) = 4T(n/2) + cn$

$a = 4, \; b = 2 \; ; \; f(n) = cn$

$n^{\log_b a} = n^{\log_2 4} = n^2 > cn$ [case I]

Hence; $T(n) = \Theta(n^2)$ — Ans

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

$a = 3 \quad b = 4 \; ; \; f(n) = n\log n$

$n^{\log_b a} = n^{\log_4 3} < n\log n$ [case III]

Hence; $T(n) = \Theta(n\log n)$ — Ans

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

$a = 3 \quad b = 3 \; ; \; f(n) = n/2$

$n^{\log_b a} = n^{\log_3 3} = n \Leftrightarrow n/2$ (case II)

Hence;

$T(n) = \Theta(n\log n)$ — Ans

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

$a = 6 ; b = 3 ; f(n) = n^2 \log n$

$n^{\log_b a} = n^{\log_3 6} < f(n)$ — (case III)

$T(n) = \Theta(n^2 \log n)$ — (Ans)

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

$a = 4 ; b = 2 ; f(n) = n \log n$

$n^{\log_b a} = n^{\log_2 4} = n^2 > f(n)$ — (case I)

$T(n) = \Theta(n^2)$ — (Ans)

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

Master's theorem not applicable as it is not a INCREASING FUNCTION.

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

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

$n^{\log_3 7} = n^{1.--} < f(n)$ — (case III)

$T(n) = \Theta(n^2)$ — (Ans)

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

Master's theorem is not applicable since regular condition is isolated.

ASSIGNMENT:- 05

Ans 1)

BFS	DFS
• uses queue data structure.	• uses stack data structure.
• Stands for Breadth First Search	• stands for Depth First Search.
• can be used to find single source shortest path in an unweighted path in an unweighted graph and we reach a vertex with min. no. of edges from a source vertex.	• we might transverse through more edges to reach a destination vertex from source.
• Siblings are visited before children.	• Children are visited before siblings.
Applications:-	Applications:-
• Shortest path & minimum spanning tree in unweighted graph.	• Detecting cycle in graph.
• Peer to peer networks.	• Path finding.
• Social Networking websites.	• Topographical Sorting.
• GPS Navigation systems.	• Solving puzzles with only 1 solution.

Ans 2) In BFS; we use queue data structure as queue is used when things don't have to be processed immediately, but have to be processed immediately, but have to be processed in FIFO order like BFS.

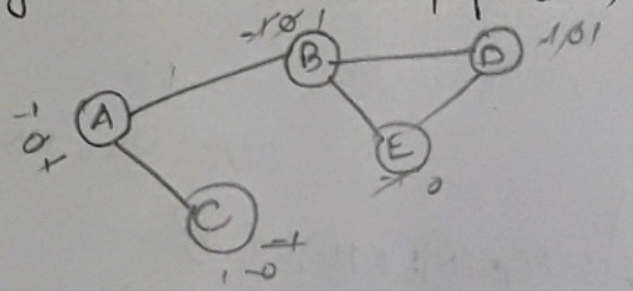
In DFS; stack data structure is used as it is beneficial for backtracking. For DFS, we retrieve it from root to the farthest node as much as possible; giving it a LIFO like approach.

Ans 3) Dense Graph is a graph in which no of edges is close to the maximal no. of edges.

Sparse Graph is a graph in which the no. of edges is close to the minimal no of edges. It can be disconnected graph.

Adjacency lists are preferred for Sparse Graph & Adjancy Matrix for Dense Graph

Ans 4.) Cycle detection in directed Graph (BFS)



-1 - unvisited.
0 - into queue.
1 - traversed.

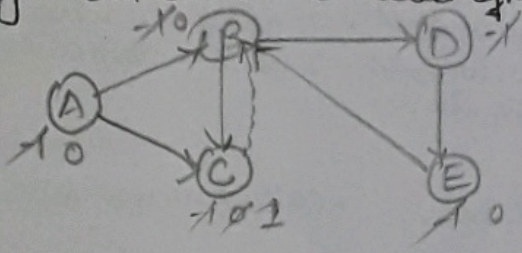
Queue: | A | B | C | D | E |

Visited set:- | A | B | C | D |

When D checks its adjacent vertices it finds E with 0.

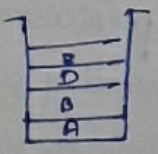
If any vertex finds the adjacent vertex with flag 0, then it contains cycle.

Cycle detection in directed graph (DFS)



-1 - unvisited.
0 - visited & stack push.
1 - visited & popped.

Stack:-



visited:-
ABCDE
→ D → D → E → B

Parent Map

Vertex	Parent
A	—
B	A
C	B
D	B
E	D

Here, E find B with 0.
⇒ Its a cycle.

Ans 5.) The DISJOINT SETS data structure is also known as Union-find data structure and merge-find set. It is a data structure that contains a collection of disjoint or non-overlapping sets.

The DISJOINT SETS means when the set of partitioned into the disjoint subsets, various operation are performed on it.

In this case, we can add new sets, merge them & also find the representative member of a set. It also allows to find out whether the two elements are of same set or not effectively

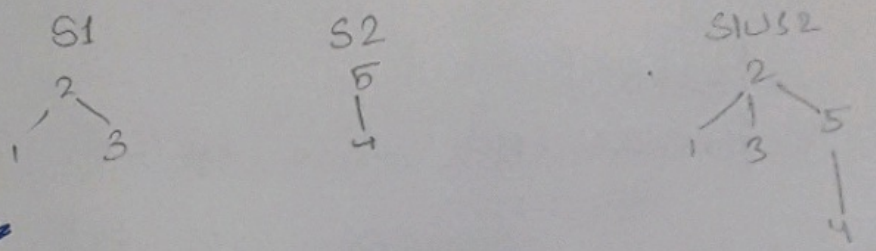
Operations on Disjoint sets :-

(1.) Union :-

(a) if sets S_1 & S_2 are 2 disjoint sets, then union S_1 & S_2 is a set of all elements x such that x is in either S_1 or S_2.

(b) As the sets should be disjoint S1 ∪ S2 replaces S1 & S2 which no longer exists

(c) It is achieved by simply making one of the trees as a subtree of other, that is; to set parent field of one of the roots of the trees to other root.



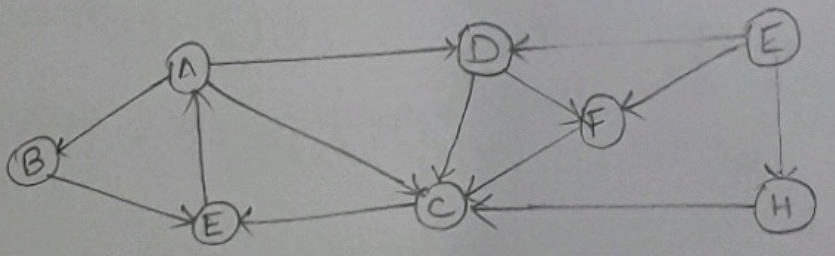
~~Merge into 1.~~

(2) Find:-
Given an element of; to find the set containing it.

Eg:- find(3) ⇒ S1
find(4) ⇒ S2

(3) Make Set(x):-
create a set containing x
makeset(1) = {1}

Ques 6)



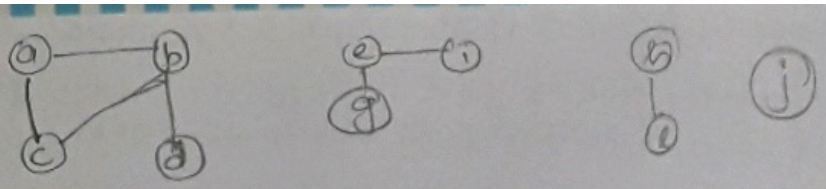
BFS:- child G D F H C E A B Path:-
Parent - G G G H C E A G → H → C → E → A → B

DFS:- Node visited Stack

~~G~~ G
D F
H C
~~F~~ E
C A
~~F~~ B
A
~~B~~

Path
G → F → ~~C~~ → E → A → B.

Ans 7)



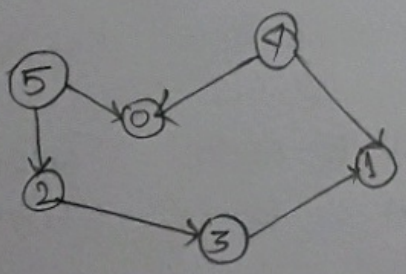
$V = \{a, b, c, d, e, f, g, h, i, j\}$

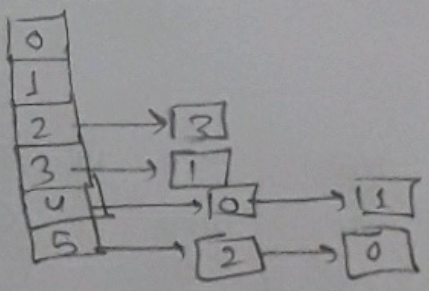
$E = \{(a,b), (a,c), (b,c), (b,d), (e,i), (e,g), (h,l), (j)\}$

Vertices → / Edges ↓	{a}	{b}	{c}	{d}	{e}		{g}	{h}	{i}	{j}	{l}
							{g}	{h}	{i}	{j}	{l}
(a,b)	{a,b}	{c}	{d}	{e}	—		{g}	{h}	{i}	{j}	{l}
(a,c)	{a,b,c}		{d}	{e}			{g}	{h}	{i}	{j}	{l}
(b,c)	{a,b,c}		{d}	{e}			{g}	{h}	{i}	{j}	{l}
(b,d)	{a,b,c,d}		{a}		{g}	{h}	{i}	{j}	{e}		
(e,i)	⋮										
(e,g)	⋮										
(h,l)	⋮										
(j)	{a,b,c,d}		{e,g,i}		{h,l}		{j}				

Finally - {a,b,c,d}, {e,i,g}, {h,l} & {j}

Ans 8)





Algo:-

(1) Go to node 0, it has no outgoing edges so push node 0 into the stack & mark as visited

(2) Go to node 1, again it has no outgoing edges so push it in stack & mark as visited.

(3) Go to node 2, process all adjacent and mark it visited.

(4) Node 3 is visited, so continue with next node.

(5) Go to node 4, all the adjacent nodes are already visited, so push it in stack & mark as visited.

(6) Go to node 5, push it in stack & stop.

Output:-
5 4 2 3 1 0.

Ans 9.) Heap is generally preferred for priority queue implementation because heaps provide better performance compared to array & linked list.

Algorithms when priority queue is used:-

(1) Dijkstra's Algo :-
shortest path algorithm ; when the graph is stored in the form of adjacency list or matrix ; priority queue can be used to extract minimum effectively when implementing it.

(2) Prim's Algo :-
To store keys of Nodes & extract minimum key node at every step.

Ans 10.)

Min Heap	Max Heap
• For every pair of parent & child node, parent node always has lower value than descendent child node.	• For every pair of the parent and descendant child node, the parent has higher value than child node.
• Value of nodes increases as we traverse from root to leaf node.	• Value of nodes decreases as we traverse from root to leaf node.
• Root node has the lowest value.	• Root node has the largest value.

ASSIGNMENT:-06

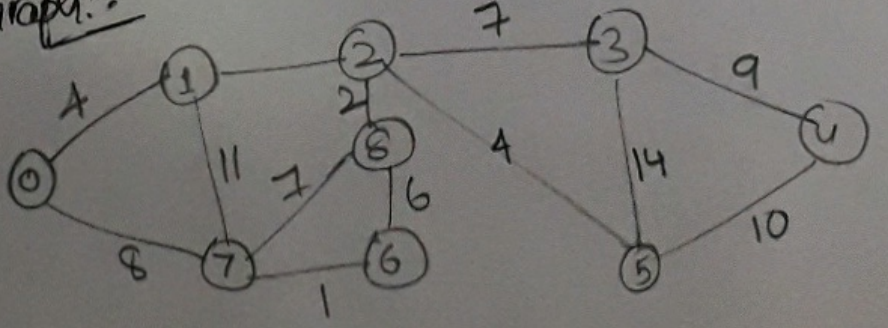
Ans 1) MINIMUM SPANNING TREE:-
It is a subset of the edges of a connected edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight.

Applications:-
- Consider n stations are to be linked using a communication network, and laying of communication link b/w any 2 stations involves a cost; so we use MST for a better output.
- same goes with roadways & highways & airlines.
- Design LAN
- Laying Pipelins connecting offshore drilling sites, refineries & markets.

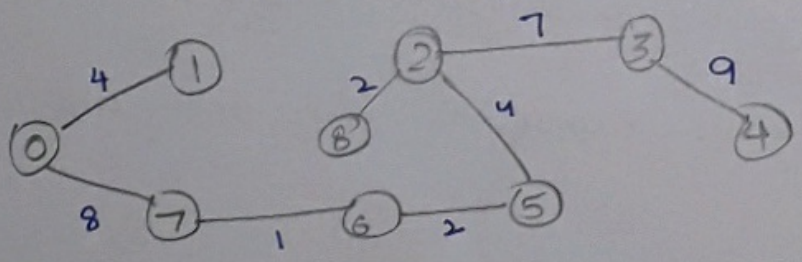
Ans 2)

Algorithm	Time Complexity	Space Complexity
Prim's Algorithm	$O(V+E)$	$O(v)$
Dijkstra's Algorithm	$O(V^2)$	$O(v^2)$
Kruskal's Algorithm	$O(E(\log V))$	$O(\|V\|)$
BellMan Ford	$O(VE)$	$O(v)$

Ans 3) Graph:-



O	V	H	
6	7	1	✓
5	6	2	✓
2	8	2	✓
0	1	4	✓
2	5	4	✓
6	8	6	×
2	3	7	✓
7	8	7	×
0	7	8	✓
1	2	8	×
4	3	9	✓
4	3	10	×
1	7	11	×
3	5	14	×



wgt. = [37] -Ans

Prim's Algo:-

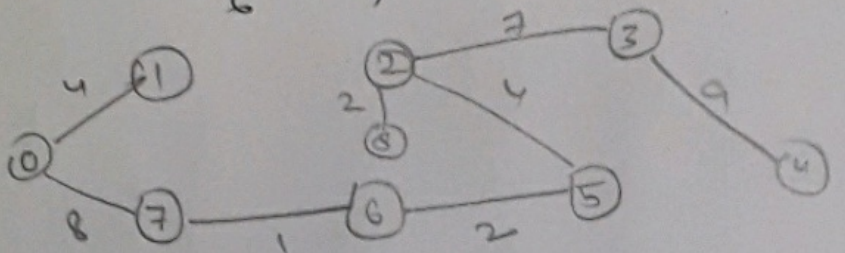
Weight:-

[0]

1	2	3	4	5	6	7	8
∞	∞	∞	∞	∞	∞	∞	∞
4	∞	∞	∞	∞	∞	[8]	∞
4	8	∞	∞	∞	[1]	8	7
11	8	7	∞	4	1	8	[2]
11	8	7	∞	2	1	8	6
4	14	1	10	2	1	8	6
4	[7]	1	10	2	1	8	6
4	7	1	[9]	2	1	8	6

Prim's:-

0	1	2	3	4	5	6	7	8
-1	~~7~~ 6	~~-1~~ 1	-1	-1	-1	~~7~~ 1	~~7~~ 1	-1

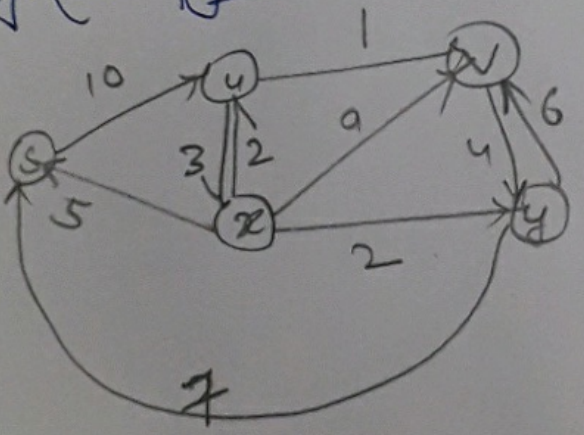


Wgt. = 37.

Ans4) i) Shortest path may change. The reason is that there may be different paths from 's' to 't'. for example:-
Let shortest path be of weight 15 & edge 5. Let there be another path with 2 edge and total weight 25. Hence, the shortest path increases. Moreover weighted of other paths are also increases.

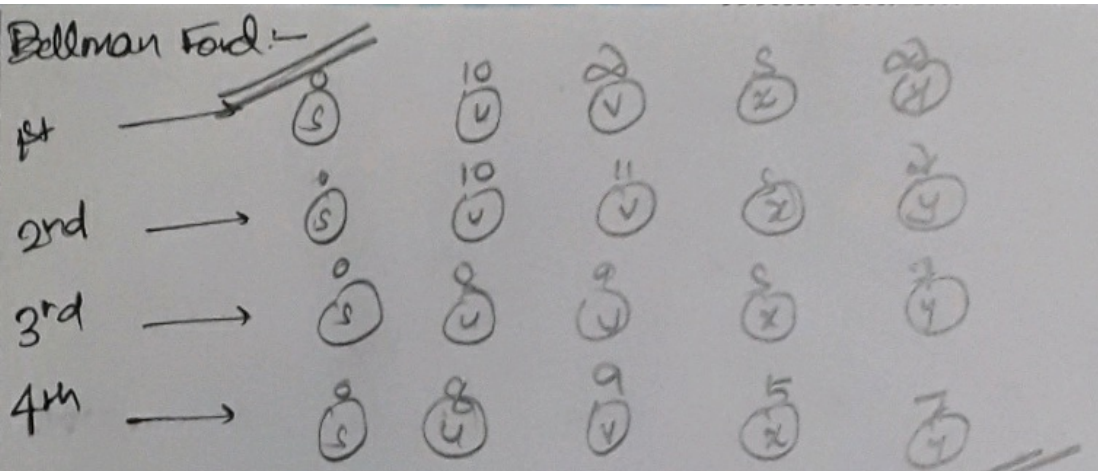
ii) If we multiply all edges wgt by 10, shortest path won't change. Only the weight will be increased by 10x otherwise it will follow the same path.

Ans5) Dijkstra's Algo:-



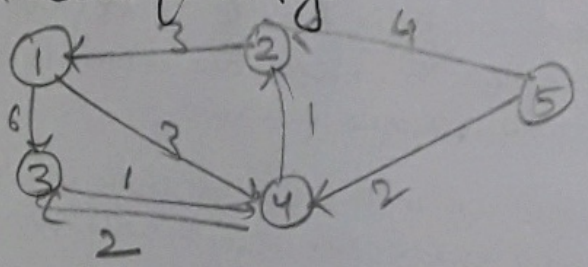
Node	Shortest distance from source node
u	8
x	5
y	9
z	7

Bellman Ford:-



Question 6

Floyd Warshall Algo:-



TC = $O(|V|^3)$.

SC = $O(|V|^2)$