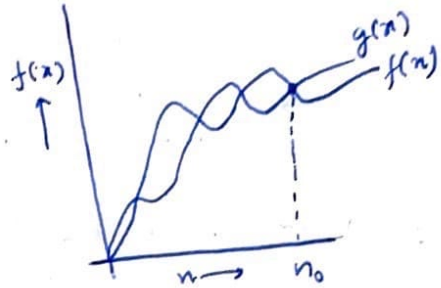
Q.1 What do you understand by Asymptotic Notation, define different asymptotic notatic with example.

TUTORIAL-1

(i) Big Oh (O)



$f(n) \Rightarrow O(g(n))$

if $f(n) \le cg(n) \quad \forall\ n \ge n_0$

for some constant, $c > 0$

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

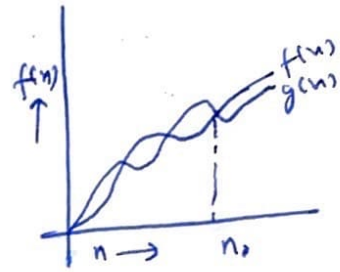
eg:- $f(n) = n^2+n$

$g(n) = n^3$

$n^2+n \le c\,n^3$

$n^2+n = O(n^3)$

(ii) Big Omega (Ω)



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

means $g(n)$ is 'tight' lower bound of $f(n)$; i.e.,

$f(n)$ can go beyond $g(n)$

i.e.; $f(n) = \Omega\, g(n)$

iff $f(n) \ge cg(n) \quad \forall\ n > n_0$ and some constant $c > 0$

Eg:- $f(n) = n^3 + 4n^2$

$g(n) = n^2$

i.e.: $f(n) \ge cg(n)$

$n^3 + 4n^2 = \Omega(n^2)$

(iii) Big Theta (θ)

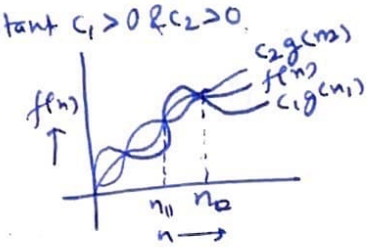
When $f(n) = \theta\, g(n)$, gives the tight upperbound and lowerbound both.

i.e; $f(n) = \theta\, g(n)$

iff $c_1\, g(n_1) \le f(n) \le c_2\, g(n_2)$

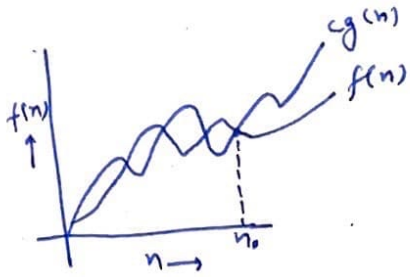
for all $n \ge \max(n_1, n_2)$ and some constant $c_1 > 0$ & $c_2 > 0$.

i.e; $f(n)$ can never go beyond $c_2\, g(n)$ and will never come down of $c_1\, g(n)$.



Eg: $3n+2 = \theta(n)$ as $3n+2 \ge 3n$.

(iv) Small Oh (o)



when $f(n) = o\,g(n)$ gives the upper bound

i.e. $f(n) = o\,g(n)$

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

$\forall\ n > n_0$ and $n > 0$

Eg: $f(n) = n^2$; $g(n) = n^3$

$f(n) < cg(n)$

$n^2 = o(n^3)$

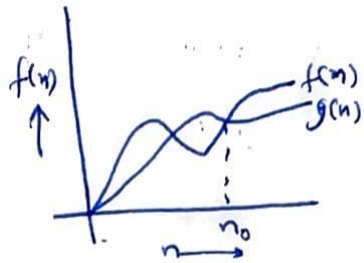
v) Small Omega (ω) :-

It gives the lower bound ;

i.e. $f(n) = \omega g(n)$

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

iff $f(n) > cg(n)$ $\forall$ $n > n_0$ and some const. $c > 0$



Q.2) What should be the time complexity of

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

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

G.P.

So, $a = 1$, $n = 2/1 = 2$

k^{th} value of GP:

$$t_k = ar^{k-1}$$

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

$$2^n = 2^k$$

$$\log_2(2^n) = k \log 2$$

$$\log_2 2 + \log_2 n = k$$

$$\log_2 n + 1 = k$$

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

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

$T(n) = 3T(n-1)$ —— ①

$T(n) = 1$

Put $n = n-1$ in ①

$T(n-1) = 3T(n-2)$ —— ②

Put ② in ①

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

$T(n) = 9T(n-2)$ —— ③

Put $n = n-2$ in ①

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

Put in ③

$T(n) = 27T(n-3)$ —— ④

$T(k) = 3^k T(n-k)$ — (5)

for k^{th} term, Let $n-k=1$

$k = n-1$

Put in (5)

$T(n) = 3^{n-1} T(1)$

$= 3^{n-1}$

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

Q.4) $T(n) = \begin{cases} 2T(n-1)-1 & \text{if } n>0, \\ \text{otherwise } 1 \end{cases}$

$T(n) = 2T(n-1)-1$ — (1)

Put $n = n-1$

$T(n-1) = 2T(n-2)-1$ — (2)

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

$= 4T(n-2)-2-1$ — (3)

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

Put in (1)

$T(n) = 8T(n-3)-4-2-1$ — (4)

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

k^{th} term:-

Let $n = k-1$

$k = n-1$

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

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

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

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

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

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

$T(n) = O(1)$

Q5) What should be time complexity of

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

$i = 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad \ldots$ —— ①

$s = 1 + 3 + 6 + 10 + 15 + 21 + \ldots$

sum of $s = 1 + 3 + 6 + 10 + \ldots \quad T_{n-1} + T_n$ —— ②

$0 = 1 + 2 + 3 + 4 + \ldots n - T_n$

$T_k = 1 + 2 + 3 + 4 + \ldots + k$

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

for k iterations

$1 + 2 + 3 + \ldots k \le n$

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

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

$O(k^2) \le n$

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

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

Q6) Time Complexity of

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

$i^2 = n$

$i = \sqrt{n}$

$i = 1, 2, 3, 4, \ldots \sqrt{n}$

$\sum_{i=1}^{n} = 1 + 2 + 3 + 4 + \ldots \sqrt{n}$

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

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

$T(n) = O(n)$

Q.7) Time Complexity of

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

Since for $k = k^2$

$k = 1, 2, 4, 8, \ldots n$

$a = 1, r = 2$

$$a\frac{(r^n - 1)}{r-1}$$

$$\frac{1(2^k - 1)}{1}$$

$n = 2^k - 1$

$n + 1 = 2^k$

$\log_2(n) = k$

i	j	k
		$\log(n) * \log(n)$
1	$\log n$	$\log(n)\log(n)$
2	$\log n$	$\log(n)\log(n)$
3	$\log n$	$\log(n)\log(n)$
⋮	⋮	⋮
n	$\log n$	$\log(n)\log(n)$

$TC \Rightarrow O(n \log(n) \log(n))$

Q.8) Time Complexity of

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

  function (n-1);
}
```

for (i=1 to n)

we get j n times every turn

$i * j = n^2$

k^{th};

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

$T(n-3) = (n-3)^2 + T(n-6)$

$T(n-6) = (n-6)^2 + T(n-9)$

and $T(1) = 1$

Now, substitute each value in $T(n)$

$T(n) = n^2 + (n-3)^2 + (n-6)^2 + \dots + 1$

let $k^n - 3k = 1$

$k = (n-1)/3$ total turns $= k+1$

$T(n) = n^2 + (n-3)^2 + (n-6)^2 + \dots + 1$

$T(n) \simeq k\, n^2$

$T(n) = (k-1)/3\ n^2$

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

Q.9) Time Complexity of

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

for $i = 1$ $\quad j = 1 + 2 + \dots \quad n \geq j+1$

$i = 2$ $\quad j = 1 + 3 + 5 + \dots \quad n \geq j+1$

$i = 3$ $\quad j = 1 + 4 + 7 + \dots \quad n \geq j+1$

n^{th} term of AP is

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

$T(n) = 1 + (n-1)d$

$(n-1)/d = n$

for $i = 1$ $\quad (n-1)/1$ times

$i = 2$ $\quad (n-1)/2$ times

$\vdots$

$i = n-1$

we get, $T(n) = i_1 j_1 + i_2 j_2 + \dots i_{n-1} j_{n-1}$

$$= \frac{(n-1)}{2} + \frac{n-2}{2} + \frac{n-3}{3} + \dots 1$$

$$= n + \frac{n}{2} + n/3 + \dots n/_{n-1} - n*1$$

$$= n\left[1 + \frac{1}{2} + \frac{1}{3} + \dots \frac{1}{n-1}\right] - n*1$$

$$= n \log n - n + 1$$

Since

$$\int \frac{1}{x} = \log x$$

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

Q.10) For the function n^{-1} & c^n. What is the asymptotic relationship b/w these functions?

Assume that $k >= 1$ & $c > 1$ are constants. Find out the value of c and no. of which relationship holds.

As given n^k & c^n.

Relationship b/w n^k & c^n is

$$n^k = O(c^n)$$

$$n^1 \le a(c^n)$$

$$\forall \; n \ge n_0 \; \& \text{ constant}, \; a > 0$$

for $n_0 = 1$; $c = 2$

$$1^k < a^2$$

$n_0 = 1$ & $c = 2$

TUTORIAL-2

Q.1) What is the time complexity of below code and how?

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

$j = 1 \qquad i = 1$

$j = 2 \qquad i = 1+2$

$j = 3 \qquad i = 1+2+3$

m-level

for (i)

$\because 1 + 2 + 3 + \ldots < n$

$1 + 2 + 3 + m < n$

$\frac{m(m+1)}{2} < n$

$m \approx \sqrt{n}$

By summation method,

$\sum_{m=1}^{m} 1 \Rightarrow 1 + 1 + \ldots \sqrt{n}$ times

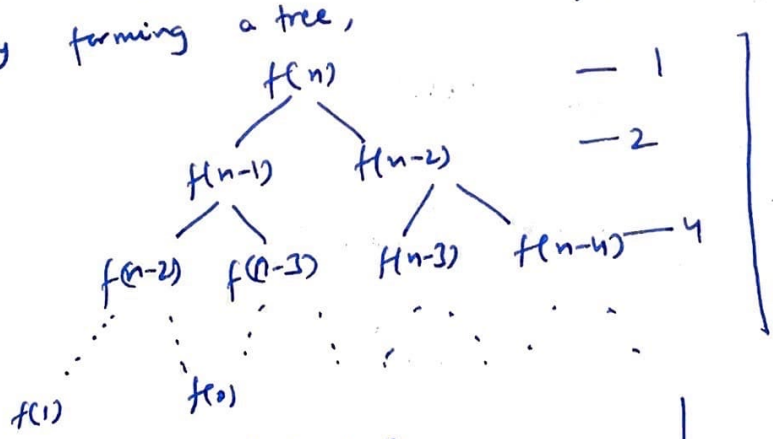
$T(n) = \sqrt{n}$

Q.2) What recurrence relation for ~~relation~~ function that prints Fib. series. Solve it to get the time complexity. What will be the space complexity and why?

For Fibonacci series,

$f(n) = f(n-1) + f(n-2)$ $\qquad f(0) = 0$, $f(1) = 1$

By forming a tree,



n-level

$1, 2, 4, 8, \ldots n$

$a = 1, r = 2$

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

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

Space Complexity :-

Recursive :- $T(n) = O(n)$

Iterative :- $T(n) = O(1)$

Q.3) Write programs which have time complexity:

$n \log n$, n^3, $\log(\log n)$

① $n\log n$ - ~~Qui~~ Mergesort

```
void mergesort (int array[], const int low, const int high)
{
    if ( low >= high) return;
    int mid = low + (high-low)/2;
    mergesort (array, low, mid);
    mergesort (array, mid+1, high);
    merge (array, low, mid, high);
}

void merge ( int array[], const int low, int mid, int high)
{
    int i, j, k;
    int n1 = mid - low +1;
    int n2 = high - mid;
    int leftArray[n1], rightArray[n2];
    for (int i=0; i < n1; i++)
        leftArray[i] = array[low+i];
    for (int j=0; j < n2; j++)
        rightArray[j] = array[j + mid +1];

    i=0, j=0, k= low;
    while ( i < n1 && j < n2)
    { if ( leftArray[i] <= rightArray [j])
            array[k] = leftArray [i];
      else
            array [k] = rightArray [j];
    }
    while (i<n1)
        array [k++] = leftArray[ i++];
    while (j<n2)
        array [k++] = rightArray[j++];
}
```

(ii) n^3 - Multiplication of two matrices

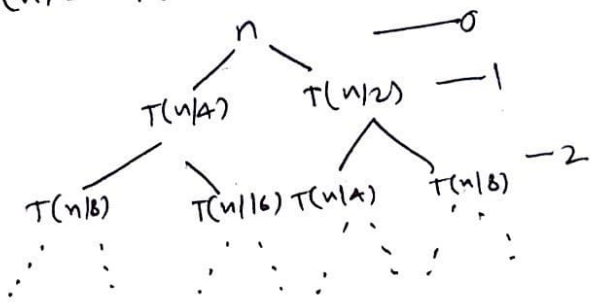
```
for(i = 0; i< n1; i++)
  for(j=0; j<n2; j++)
    for (k=0; k<n3; k++)
      res[i][j] += a[i][k] * b[k][j];
```

(iii) $\log(\log(n))$ -

```
for(i= 2; i<n; i = i*i)
    count ++;
```

Q.4) Solve the following recurrence relation:-

$$T(n) = T(n/4) + T(n/2) + c\,n^2$$



At level,

$0 \rightarrow cn^2$

$1 \rightarrow \frac{n^2}{4^2} + \frac{n^2}{2^2} = \frac{c5n^2}{16}$

$2 \rightarrow \frac{n^2}{8^2} + \frac{n^2}{16^2} + \frac{n^2}{4^2} + \frac{n^2}{8^2} = \left(\frac{5}{16}\right)^2 n^2 c$

$\vdots$

max level $= \frac{n}{2^k} = 1$

$= k = \log_2 n$

$$T(n) = c\left(n^2 + \frac{5}{16}n^2 + \frac{5}{16}^2 n^2 + \cdots \left(\frac{5}{16}\right)^{\log_2 n} n^2\right)$$

$$T(n) = cn^2\left[1 + \frac{5}{16} + \frac{5}{16}^2 + \cdots + \frac{5}{16}^{\log n}\right]$$

$$= cn^2 \times 1 \times \left[\frac{1-(5/16)^{\log n}}{1-5/16}\right]$$

$$= cn^2 \times \frac{11}{5} \times \left[1-\left(\frac{5}{16}\right)^{\log n}\right]$$

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

$$O(cn^2)$$

Q.5) What is the time complexity of following func.() ?

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

for i	j
1	1
2	1+3+5
3	1+4+7
⋮	
n	

$j = (n-1)/i$ times

$$\sum_{i=1}^{n} \frac{n-1}{i}$$

$$\because T(n) = \frac{n-1}{1} + \frac{n-1}{2} + \frac{n-1}{3} + \cdots \frac{n+1}{n}$$

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

$$= n\log n - \log n$$

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

Q.6) What should be time complexity of

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

$k \to$ const.

i
2^1
2^k
2^{k^2}
2^{k^3}
⋮
2^{k^m}

where

$$2^{k^m} <= n.$$

$$k^m = \log_2 n$$

$$m = \log k \log_2 n$$

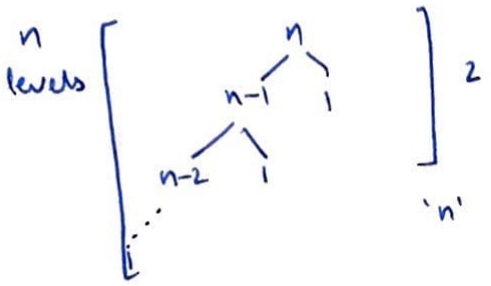
$$\because \sum_{i=1}^{m} 1 = 1 + 1 + 1 + \cdots \quad m \text{ times}$$

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

Q.7) Write a recurrence relation when quick sort repeatedly divide array into 2 parts of 99% and 1%. Derive time complexity in this case. Show the recurrence time while deriving time complexity and find difference in heights of both extreme parts. What do you understand by this analysis?

Given algorithm divides array in 99% and 1% part.

$\because T(n) = T(n-1) + O(1)$



'n' work is done at each level

$T(n) = (T(n-1) + T(n-2) + \dots T(1) + O(1)) \times n$

$= n \times n$

$\boxed{\therefore T(n) = n^2}$

Lowest height = 2

Highest height = n

$\therefore$ [difference = $\underline{n-2}$] $n > 1$

The given algorithm produces linear result.

Q.8) Arrange following in increasing order of rate of growth:

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

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

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

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

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

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

Q.1) Write linear search pseudocode to search an element in a sorted array with minimum comparisons.

TUTORIAL-1

```
for (i = 0 to n)
{
    if (arr[i] == key)
        return key;
}
return -1;
```

Q.2 Write Pseudo code for iterative and recursive insertion sort. Insertion sort is called online sort. Why? What about other sorting algorithms that has ~~been~~ ~~been~~ discussed?

Iterative:

```
void insertion sort - (int arr[], int n)
{
    for (int i=1; i<n; i++)
    {
        j = i-1;
        x = arr[i];
        while (j>-1 && arr[j] > x)
        {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = x;
    }
}
```

Recursive:

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

Insertion sort is called online sort because it does not need ~~used~~ to know anything about values it will sort and information is requested while algorithm is running.

Other ~~online~~ sorting algos:-
Bubble, selection, merge, quick, heap.

3. Complexity of all sorting algorithms that have been discussed in lectures.

Sorting Algo	Best	Worst	Avg.
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble	$O(n)$	$O(n^2)$	$O(n^2$
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$
Heap	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$
Quick	$O(n\log n)$	$O(n^2)$	$O(n\log n)$
Merge	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$

Q.4) Divide all sorting algorithms into ~~all~~ inplace | stable | online.

Inplace	Stable	Online
Bubble	Merge	Insertion
Selection	Bubble	
Insertion	Insertion	
Quick	Count	
Heap		

Q.5) Write iterative | recursive pseudocode for binary search. What is the time and space complexity of linear and binary search.

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;
    else if (key < arr[m])
        r = m-1;
    else
        l = m+1;
  }
  return -1;
}
```

Recursive:-

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

Time Complexity	Space Complexity
Linear search - $O(n)$	$O(1)$
Binary Search - $O(\log n)$	$O(n)$

Q.6) Write recurrence relation for binary search (recursive).

$T(n) = T(n/2) + 1$ — ①

$T(n/2) = T(n/4) + 1$ — ②

$T(n/4) = T(n/8) + 1$ — ③

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

$= T(n/4) + 1 + 1$

$= T(n/8) + 1 + 1 + 1$

$\vdots$

$T(n/2^k) + 1$ (k times)

let $2^k = n$

$k = \log n$

$T(n) = T(n/n) + \log n$

$T(n) = T(1) + \log n$

$T(n) = \cancel{\log n}\ O(\log n)$

Q.7) Find two index such that $A[i] + A[j] = k$ in minimum time complexity.

```
for (i=0; i<n; i++)
 {
   for(int j=0; j<=n; j++)
    {if ( a[i] + a[j] ==k)
        printf ("%d %d", i,j);
    }
 }
```

Q.8) Which sorting is best for practical uses? Explain.

Quick Sort is the fastest general-purpose sort. In most practical situations, quicksort is the method of choice. As stability is important and if space is available, merge sort might be best.

Q.9) What do you mean by inversions in an array? Count the number of inversions in Array arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5}. using merge sort.

A pair (A[i], A[j]) is said to be inverted if

- A[i], > A[j]
- i < j

Total no. of inversions in given array arr is 31 using merge sort.

Q.10) In which case, quick sort will give least and worst case time complexity.

Worst Case $O(n^2)$:- The worst case occurs when the pivot element is an extreme (smallest / largest) element. This happens when input array is sorted or reverse sorted and either first or last element is selected as pivot.

Best case $O(n \log n)$:- The best case occurs when we select pivot element as a mean element.

Q.11) Write recurrence relation of merge/quicksort in best & worst case. What are the similarities and differences b/w complexities of two algorithms and why?

Merge Sort:- Best case - $T(n) = 2T(n/2) + O(n)$
Worst Case - $T(n) = 2T(n/2) + O(n)$ } $O(n \log n)$

Quick Sort:- Best case - $T(n) = 2T(n/2) + O(n)$ — $O(n \log n)$
Worst case - $T(n) = T(n-1) + O(n)$ — $O(n^2)$

In quick sort, array of element is divided into 2 parts repeatedly until it is not possible to divide it further.

In mergesort, the elements are split into 2 subarrays (n/2) again & again until only one element is left.

Q.12) Selection sort is not stable by default. but can you write a version of stable selection code!

```
for(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-j];
          min--;
      }
      a[i] = key;
}
```

Q.13) Bubble sort scans ~~every~~ array even when array is sorted. Can you, modify, the bubble sort so that it does not scan the whole array once it is sorted.

```
void bubblesort (int arr[], int n)
{
for(int i=0; i< n-1 ; i++)
  {
    int swaps =0;
    for (int j=0; j< n-1-i ; j++)
      {
        if (arr[j] > arr[j+1])
          { int t = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = t;
            swap++;
          }
      }
    if(swap ==0)  return;
  }
}
```

TUTORIAL

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

$T(n) = aT(n/b) + f(n)$

$a \geq 1;\ b \geq 1$

On comparing,

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

$c = \log_b a = \log_2 3 = 1.584$

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

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

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

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

$a \geq 1,\ b > 1$

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

$c = \log_2 4 = 2$

$n^c = n^2 = f(n) = n^2$

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

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

$a = 1$

$b = 2$

$f(n) = 2^n$

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

$n^c = n^0 = 1$

$f(n) > n^c$

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

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

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

$c = \log_b a = \log_2 2^n = n$

$n^c \Rightarrow n^n$

$f(n) = n^c$

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

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

$a = 16,\ b = 4,\ f(n) = n$

$c = \log_4 16 = 2$

$n^c \Rightarrow n^2$

$f(n) < n^c$

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

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

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

$c = \log_2 2 = 1$

$n^c = n^1 = n$

$n\log n > n$

$f(n) > n^c$

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

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

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

$c = \log_2 2 = 1$

$n^c = n^1 = n$

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

$f(n) < n^c$

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

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

$a = 2,\ b = 4,\ f(n) = n^{0.5}$

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

$n^c = n^{0.5}$

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

$f(n) > n^c$

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

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

$a = 0.5,\ b = 2$

$a \geq 1$ but here a is 0.5,

So we cannot apply Master's theorem.

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

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

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

$n^c = n^2$

As $n! > n^2$

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

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

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

$c = \log_2 4 = 2$

$n^c = n^2$

$f(n) < n^c$

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

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

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

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

$\frac{1}{2}\log_2 n < \log(n)$

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

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

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

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

$c = \log_b a = \log_2 3 = 1.5849$

$n^c = n^{1.5849}$

$f(n) < n^c$

$\therefore T(n) = \Theta(n^{1.5849})$

Q.14) $T(n) = 3T(n/3) + \sqrt[5]{n}$

$a = 3,\ b = 3$

$c = \log_b a = \log_3 3 = 1$

$n^c = n^1 = n$

$f(n) < n^c$

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

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

$a = 4,\ b = 2$

$c = \log_b a = 2$

$n^c = n^2$

$f(n) < n^c$

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

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

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

$c = \log_b a = \log_4 3 = 0.792$

$n^c = n^{0.792}$

$f(n) = n\log n > n^c$

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

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

$a = 3,\ b = 3$

$c = \log_b a = 1$

$f(n) = n/2$

$n^c = n$

$f(n) < n^c$

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

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

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

$c = \log_3 6 = 1.6309$

$n^c = n^{1.6309}$

$f(n) > n^c$

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

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

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

$c = \log_b a = 2$

$n^c = n^2$

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

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

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

$a = 64,\ b = 8$

$c = 2$

$n^c = n^2$

$f(n) > n^c$

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

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

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

$c = \log_3 7 = 1.7712$

$n^c = n^{1.7712}$

$f(n) > n^c$

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

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

$a = 1,\ b = 2,\ f(n) = n(2 - \cos n)$

$c = \log_2 1 = 0$

$n^c = n^0 = 1$

$f(n) > n^c$

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

TUTORIAL 2-5

Q1) What is the difference between DFS and BFS. Write applications of both the algorithms.

BFS	DFS
a) It stands for Breadth First Search.	a) It stands for Depth First Search.
b) It uses queue data structure.	b) It uses stack data structure.
c) It is more suitable for searching vertices which are clear to given source.	c) It is more suitable when there are solutions away from source.
d) BFS considers all neighbours first & therefore not suitable for decision-making trees used in games and puzzles.	d) DFS is more suitable for game or puzzle problems. We make a decision then explore all paths through this decision, and if decision leads to win situation, we stop.
e) The siblings are visited before children.	e) Here, children are visited before siblings.
f) There is no concept of backtracking.	f) It is a recursive algorithm that uses backtracking.
g) It requires more memory.	g) It requires less memory.

Applications:-

- BFS - Bipartite graph and shortest graph, peer to peer networking, crawlers in search engine and GPS navigation system.
- DFS - acyclic graph, topological order, scheduling problems, soduku puzzle.

Q.2) Which data structure are used to implement BFS and DFS. Why?

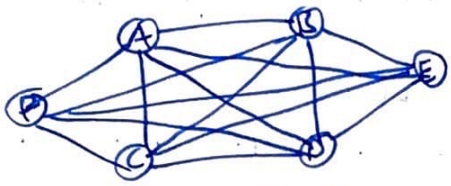
For implementing BFS, we need a queue data structure for finding shortest path between any node. We use queue because things don't have to be processed immediately, but have to be processed in FIFO order like BFS. BFS searches for nodes' level wise, i.e; it searches node wrt their distance from root (source). For this, queue is better to use in BFS.

For implementing DFS, we need a stack data structure as it traverse a graph in depth in depthward motion and uses stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

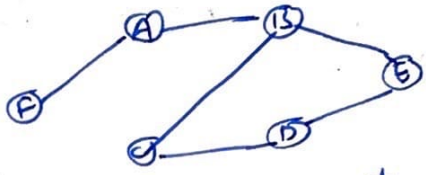
Q.3) What do you mean by sparse and dense graph? Which representations of graph is better for sparse and dense graph?

Dense graph is a graph in which no. of edges is close to maximum no. of edges.

Sparse graph is a graph in which no. of edges is very low.



Dense Graph



Sparse graph

- For sparse graph, it is preferred to use Adjacency list.
- For dense graph, it is preferred to use Adjacency Matrix.

Q.4) How can you detect a cycle in a graph using BFS & DFS?

For detecting cycle in a graph using BFS, we need to use Kahn's algorithm for topological sorting.

The steps involved are:

① Compute in degree (no. of incoming edges) for each of vertex present in graph and intialize count of visited nodes as 0.

② Pick all vertices with in-degree as 0 and add them in queue.

③ Remove a vertex from queue and then,
- increment count of visited nodes by 1.
- Decrease in-degree by 1 for all its neighbouring nodes.
- If in-degree of neighbouring nodes is reduced to zero then add to q.

④ Repeat ③ until queue is empty.

⑤ If count of visited nodes is not equal to no. of nodes in graph has cycle, otherwise not.

For detecting cycle in graph using DFS, we need to do the following: DFS for a connecting graph produces a tree, there is cycle in graph if there is a back edge present in the graph. A back cycle is an edge that is from a node to itself (self-loop) or one of its ancestor in the tree produced by DFS. For a disconnected graph, get DFS forest as output. To detect cycle, check for a cycle in individual trees by checking back edges. To detect a back edge, ~~the~~ keep track of vertices currently in recursion track for DFS traversal. If a vertex is reached that is already in recursion stack, then there is a cycle.

Q.5) What do you mean by disjoint set data structure? Explain 3 operation along with examples which can be performed on disjoint sets?

A disjoint set is a data structure that keeps track of set of elements partitioned into several disjoint subsets. In other words, a disjoint set is a group of sets where no item can be in more than one set.

3 operations:-

1. Find:- can be implemented by recursively traversing the parent array until we hit a node who is parent itself.

```
int find (int i)
{
    if(parent[i] == i])
        return i;
    else
        return find(parent[i]);
}
```

2. Union:- It takes two elements as input and finds representatives of these sets using the find operation and ~~fitto~~ finally puts either one of the trees under root node of other tree, effectively merging the trees and sets.

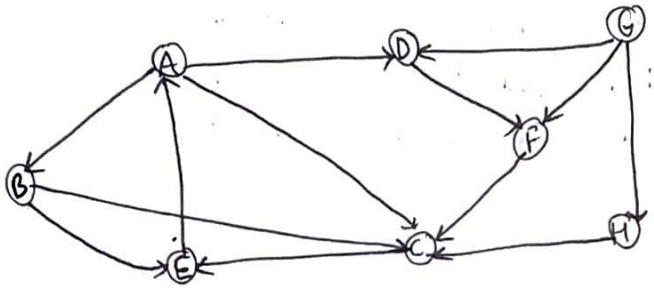
```
void union (int i, int j)
{
    int irep = this.find(i);
    int jrep = this.find(j);
    this.parent[irep] = jrep;
}
```

3. Union by Rank:- We need a new array rank[]. Size of array same as parent array. If i is the representative of set, rank[i] is height of tree. We need to minimize height of tree. If we are uniting 2-trees, we call them left and right, then it all depends on rank of left and right.

- If rank of left is less than right then it's best to move left under right and vice-versa.
- If ranks are equal, rank of result will always be one greater than rank of trees.

```
void union (int i, int j)
{
    int irep = this.Find(i);
    int jrep = this.Find(j);
    if (irep == jrep) return;
    irank = Rank[irep];
    jrank = Rank[jrep];
    if (irank < jrank) this.parent[irep] = jrep;
    else if (jrank < irank) this.parent[jrep] = irep;
    else this.parent[irep] = jrep;
    Rank[jrep]++;
}
```

Q.6) Run BFS and DFS on graph shown below.



BFS

Child	G	H	D	F	C	E	A	B
Parent		G	G	G	H	C	E	A

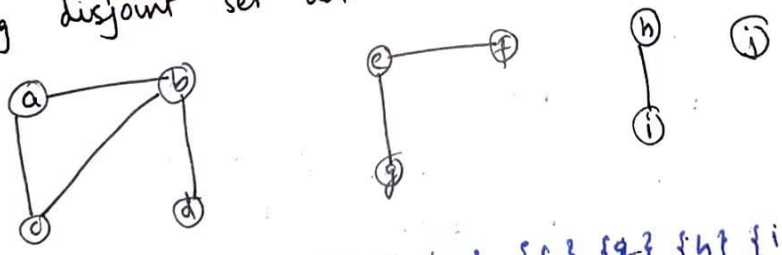
Path → G → H → C → E → A → B

DFS

Visited Nodes	Stack
~~G~~	G
D	F
H	C
~~F~~	E
~~C~~	A
~~E~~	B
~~A~~	
~~B~~	

Path → G → F → C → E → A → B

Q.7) Find out no. of connected components and vertices in each component using disjoint set data structure.



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}

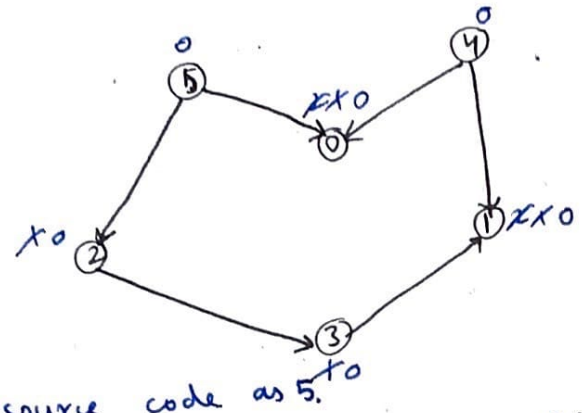
(e, f) {a, b, c, d} {e, f} {g} {h} {i} {j}

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

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

No. of connected components = 3

Q.8) Apply topological sort and DFS on graph having vertices from 0 to 5.



We take source code as 5.

Applying Topological Sort

DFS(5)
↓
DFS(0)

DFS(2)
↓
DFS(3)
↓
DFS(1)

DFS(4)
↓
Not possible

q: 5/4; Pop 5 and decrement indegree of it by 1.

q: 4/2; Pop 4 and decrement indegree and push 0

q: 2/0; Pop 2 and decrement indegree and push 3

q: 0/3; Pop 0, Pop 3, push 1

q: 1; Pop 1

5 4 2 0 3 1 ans

4
5
2
3
1
0

stack

4 → 5 → 2 → 3 → 1 → 0

Q.9) Heap data structure can be used to implement priority queue. Name few graph algorithm where you need to use priority queue and why?

Yes, heap data structure can be used to implement priority queue. It will take O(log N) time to insert and delete each element in priority queue. Based on heap structure, priority queue has two types max-priority queue based on max heap and min priority queue based on min-heap. Heaps provide better performance comparison to array and linked list.

The graphs like Dijkstra's shortest path algorithm, Prim's Minimum spanning tree use priority queue.

- Dijkstra's Algorithm:- When graph is stored in form of adjacency list or matrix, priority queue is used to extract minimum efficiently when implementing the algorithm.
- Prim's Algorithm:- It is used to store keys of nodes and extract minimum key node at every step.

.10) Differentiate between Min-heap and Max-heap.

Min-Heap	Max-Heap
In min-heap, key present at root node, must be less than or equal to among keys present at all of its children.	• In max-heap, the key present at root node must be greater than or equal to among kegs present at all of its children.
The minimum key element is present at the root.	• The maximum key almost is present at the root.
It uses ascending priority.	• It uses descending priority.
The smallest element has priority while construction of min-heap.	• The largest element has priority, while construction of max-heap.
The smallest element is the first to be popped from the heap.	• The largest element is the first to be popped from the heap.

Q.1 What do you mean by Minimal spanning Tree ? What are the applications of MST?

MST is a subset of edges of a connected edge-weighted undirected graph that connects all the vertices together without any cycles and with minimum possible edge weighted.

APPLICATIONS:-

1) Consider n stations, are to be linked using a communication network and laying of communication link between any two stations involves a cost. The ideal solution would be to extract a subgraph termed as minimum cost spanning tree.

2) Designing LAN.

3) Highways or railways b/w cities.

4) Laying pipelines connecting offshore drilling sites, refineries & consumer markets.

Q.2) Analyze time and space complexity of Prim, Kruskal, Dijkstra's, and Bellman-Ford

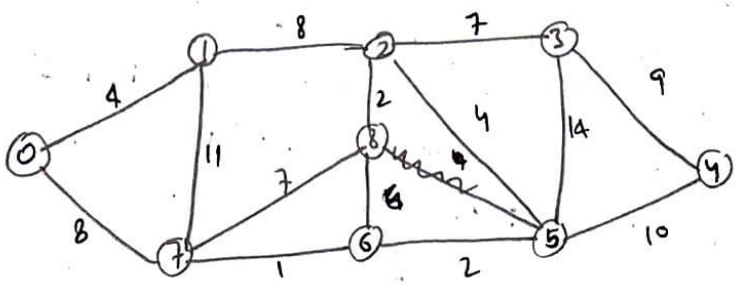
Prim's
TC - $O(|E| \log |V|)$
SC - $O(|V|)$

Kruskal's
TC - $O(|E| \log |E|)$
SC - $O(|V|)$

Dijkstra's:-
TC - $O(V^2)$
SC - $O(V^2)$

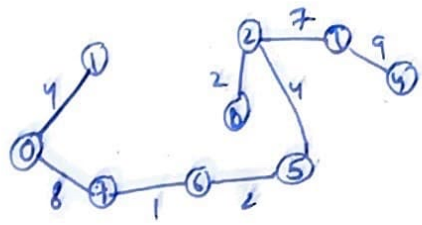
Bellman. Ford's:-
TC - $O(VE)$
SC - $O(E)$

Q.3) Apply Kruskal's and Prim's Algorithms on given graph to compare MST & its weight.



Kruskal's

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



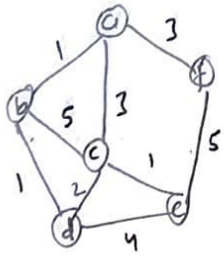
Weight = 1+2+2+4+4+7+8+9
= 37

~~Kruskal's~~ Prim's:- Weight = 4+8+2+4+2+7+9+3
= 37

Q.4) Given a directed weighted graph, You are also given the shortest path from a source vertex 's' to a destination vertex 't'. ~~Bed~~ the shortest path remain same in following cases:

i) If weight of every edge is increased by 10 units.

ii) If weight of every edge is multiplied by 10 units.

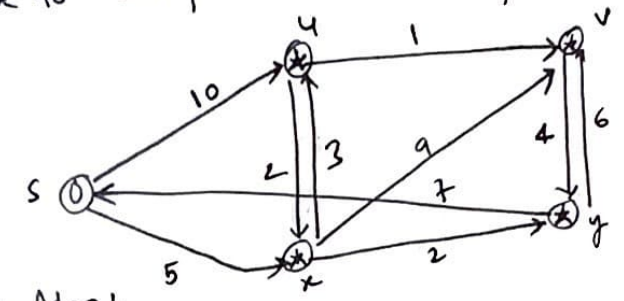


(i) The shortest path may change. The reason is that there maybe different no. of edges in different paths from 's' to 't'.

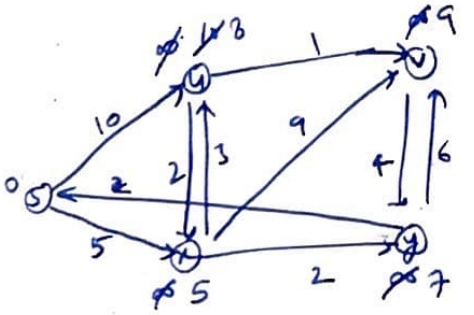
eg:- Let the shortest path of weight 15 and has edges 5. Let there be another path with 2 edges and total weight 25. The weight of shortest path is increased by 5*10 and becomes 15 + 50. Weight of other path is increased by 2*10 and becomes 25 +20. So, the shortest path changes to other path with weight as 45.

(ii) If we multiply all edges' weights by 10, the shortest path remains the same. The weights of all paths from 's' to 't' gets multiplied by same unit. The number of edges or path doesn't matter.

Q.5) Apply Dijkstra's and Bellman Ford Algorithms, on graph given right side to compare shortest path to all nodes from node s.



Dijkstra's Algo:-

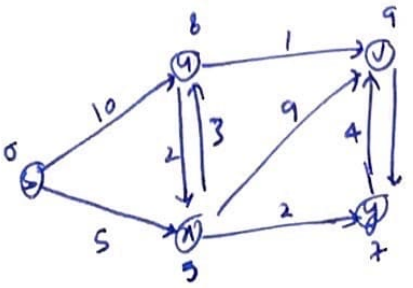


Node	Shortest dist. from source node
u	8
x	5
v	9
y	7

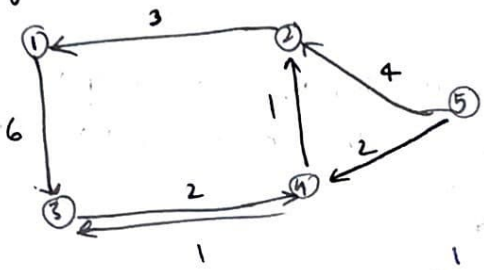
Bellman Ford's Algo:-

1st -	0 (s)	~~∞~~ 10 (u)	∞ (v)	~~∞~~ 5 (x)	∞ (y)
2nd -	0 (s)	10 (u)	~~∞~~ 11 (v)	5 (x)	∞ (y)
3rd -	0 (s)	~~10~~ 8 (u)	~~11~~ 9 (v)	5 (x)	~~∞~~ 7 (y)
4th -	0 (s)	8 (u)	9 (v)	5 (x)	7 (y)

Graph does not have -ve cycle.



Q6) Apply all pair shortest path algorithm - Floyd Warshall on below mentioned graph. Also, analyze space and time complexity of it. 4.



$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & \infty & 6 & 3 & \infty \\ 2 & 2 & 0 & \infty & \infty & \infty \\ 3 & \infty & \infty & 0 & 2 & \infty \\ 4 & \infty & 1 & 1 & 0 & \infty \\ 5 & \infty & 4 & \infty & 2 & \infty \end{array}$$

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & \infty & 6 & 3 & \infty \\ 2 & 2 & 0 & 8 & 5 & \infty \\ 3 & \infty & \infty & 0 & 2 & \infty \\ 4 & \infty & 1 & 1 & 0 & \infty \\ 5 & \infty & 4 & \infty & 2 & 0 \end{array}$$

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & \infty & 6 & 3 & \infty \\ 2 & 2 & 0 & 8 & 5 & \infty \\ 3 & \infty & \infty & 0 & 2 & \infty \\ 4 & 3 & 1 & 1 & 0 & \infty \\ 5 & 6 & 4 & 12 & 2 & 0 \end{array}$$

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & \infty & 6 & 3 & \infty \\ 2 & 2 & 0 & 8 & 5 & \infty \\ 3 & \infty & \infty & 0 & 2 & \infty \\ 4 & 3 & 1 & 1 & 0 & \infty \\ 5 & 6 & 4 & 12 & 2 & 0 \end{array}$$

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

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