Name: Garima Singh
Section: F
Class Roll no: 02
University Roll no: 2016744

DAA

Tutorial - 1

Name : Garima Singh
Section : F
Roll no : 02
University Roll no: 2016744.

Q.1 →

Ans → **Asymptotic Notations :-**

Asymptotic Notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.

big O, big θ, big Ω are the different types of Asymptotic Notations.

Sol^n 2 → $i = 1, 1\times2, 2\times2, 4\times2, \ldots\ldots 2^k$ (k times) the loop will run

$\therefore 2^k \leq n$

$k \log_2 2 = \log_2 n$

$k = \log_2 n$

$\therefore$ the T.C. is - $O(k) = O(\log_2 n)$

(3) Ans: $T(n) = 3T(n-1)$ for $n > 0$

$T(0) = 1$ for $n = 0$

Let $n = n-1$

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

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

$T(n) = 9T(n-2)$

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

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

∴ it can be concluded from above eqn.

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

Let $n-k = 0$

$n = k$

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

$T(n) = 3^k (1)$ ∵ $T(0) = 1$

∴ $T(n) = 3^k$

$= 3^n$

$= O(3^n)$

Soln 4⇒ $T(n) = 2T(n-1) - 1$ for $n > 0$

$T(0) = 1$ for $n = 0$

⇒ $T(n-1) = 2T(n-2) - 1$

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

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

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

$T(n) = 8T(n-3) - 6 - 1 = 8T(n-3) - 7$

$\therefore T(n) = 2^K T(n-K) + -\{2^{K-1} + 2^{K-2} + \cdots 2^2 + 2 + 1\}$

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

Let $n - k = 0$

$n = k$.

$= 2^n T(0) - [2^n - 1]$

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

$= 1$

$T(n) = O(1) = T.C.$

Soln 5 ⇒ $1 = 1, 3, 6, 10, \ldots$

$= 1, (1+2), (1+2+3), (1+2+3+4), (1+2+3+4+5)$

$\cdots n$ (K times)

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

So, $\rightarrow 2n = K(K+1)$

$\Rightarrow K^2 + K = 2n$

$K^2 + K - 2n = 0 \rightarrow$

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

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

Soln 6 → ~~i = 1, 1, 1~~ -

$i = 1, 2, 3, 4 \ldots$ (K times)

$i^2 <= n$

$\Rightarrow$ ~~k=2~~ $i^2 = n$

$i = \sqrt{n}$

∴ i will increment till $\sqrt{n}$.

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

∴ T.C. = $O(\sqrt{n})$

Soln 7 →

```
for (i = n/2; i <= n; i++) {→ Loop 3
    for (j = 1; j <= n; j = j * 2) {→ Loop 2
        for (k = 1; k <= n; k = k * 2) }
            count ++
```

Loop 1

Loop 1 → $k = 1, 2, 4, \ldots n$ (K times)

$n >= 2^K$

$k = \underbrace{1, 2, 4, 8 \ldots 2^K}_{K \text{ times}}$.

$2^K = \cancel{\log n}\ n$

$\log 2^K = \log n$

$K = \log n$

Loop 2 → $j = 1, 2, 4, \ldots \frac{2^k}{n}$ (k times)

$n = 2^k$

$k = \log n$

Loop 3 → $i = n/2, (\frac{n}{2}+1), (n/2+2) + \ldots n$

$\therefore \frac{n}{2}$ times.

∴ T.C. of the entire Nested loop will be

$O\left(\frac{n}{2} \times \log n \times \log n\right)$

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

Ans 8 ⇒ $T(n) = n^2 + T(n-3)$

& $T(1) = 1$

⇒ $T(n-3) = (n-3)^2 + T(n-6)$

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

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

$T(n) = n^2 + (n-3)^2 + (n-6)^2 + T(n-9)$

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

Let ~~$n-3k=1$~~ $n - 3k = 1$

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

∴ Total Term

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

$T(n) \simeq kn^2$

$T(n) \simeq k[(n-1)/3 \cdot n^2$

So, $T(n) = \frac{n^3 - n^2}{3}$

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

Soln 9 :- for

$i = 1$ $j = 1, 2, 3, \cdots$ $(n \geq j+i)$

$i = 2$ $j = 1, 3, 5, 7 \cdots$ $(n \geq j+i)$

$i = 3$ $j = 1, 4, 7, 10 \cdots$ $(n \geq j+i)$

$T(n) = a + nd$ $\rightarrow$ $n = \frac{n-a}{d} +$

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

for $i = 2$ $\rightarrow$ $(n-1)/2$

for $i = n-1$ $\rightarrow$ 2 times

for $i = n$ $\rightarrow$ 1 time

$T(n) = i_1 j_1 + i_2 j_2 + \cdots + i_{n-1} j_{n-1}$

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

$= n + \frac{n}{2} + \frac{n}{3} \cdots \frac{n}{n-1} - n \times 1$

$= n[1 + 1/2 + 1/3 + 1/n-1] - n$

$\Rightarrow n \times \log n - n$

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

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

Ans 10 $\Rightarrow$ $n^k = O(c^n)$

$\therefore n^k \leq a(c^n)$

$\forall \; n > n_0$ & constant $a > 0$

for $n_0 = 1$ & $c = 2$

$\Rightarrow 1^k < 2^n$

$\Rightarrow n_0 = 1$ & $c = 2$

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

$c = \log_3 7 = 1.7712$

$n^c = n^{1.7712}$

$\because n^2 > n^{1.7712}$

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

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

$a = 1, \quad b = 2$

$c = \log_2 1 = 0$

$n^c = n^0 = 1$

$f(n) = n(2 - \cos n)$

$\because 1 < n(2 - \cos n)$

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

Tutorial - 2

Name → Garima Singh
Section → F
Rollno → 02
Univ Roll.No → 2016744

Q.1 ⇒

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

] m - level

for (i)

$\therefore 1 + 2 + 3 + \cdots + < n$

$\therefore 1 + 2 + 3 + m < n$

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

$m \approx \sqrt{n}$

By summation method

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

$$\boxed{T(n) = \sqrt{n}} \rightarrow \text{Ans}$$

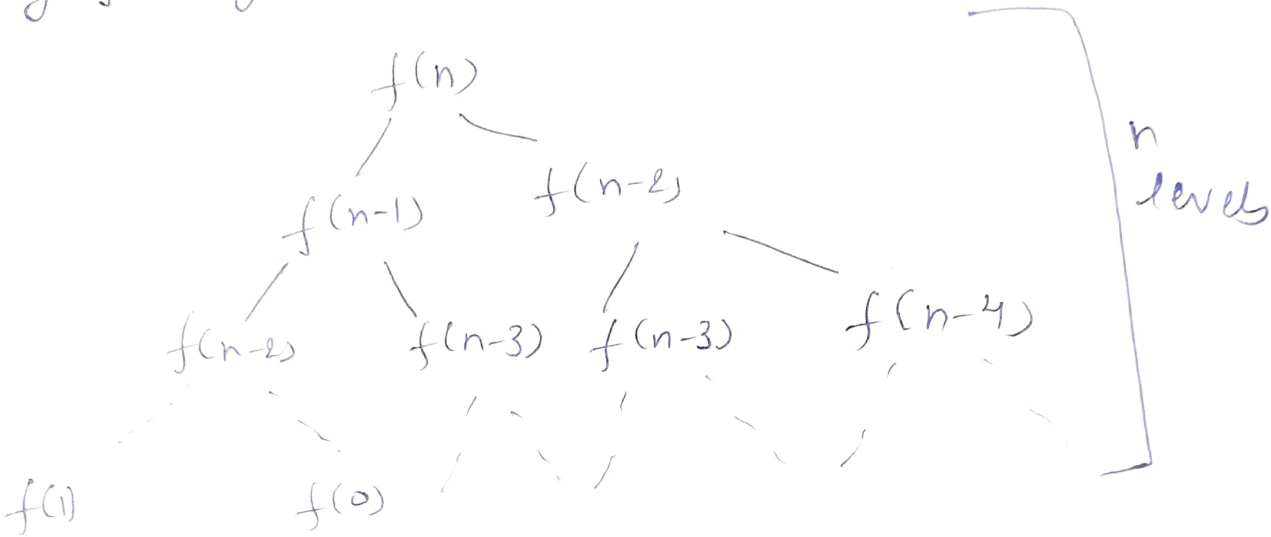
Soln 2 - for Fibonacci series

$f(n) = f(n-1) + f(n-2)$

$f(0) = 0$

$f(1) = 1$

By forming a tree



∴ At every function call we get 2 function calls

∴ for n levels

We have = 2 × 2 ... n times

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

Maximum Space

Considering Recursion Stack:

no of calls maximum = n

For each call we have space complexity O(1)

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

Without considering Recursive stack:

each call we have time complexity O(1)

$\therefore$ $T(n) = O(1)$

Sol^n 3 →

1) nlogn → Quick Sort

```
void quicksort (int arr[], int low, int high)
{
    if (low < high)
    {
        int pi = partition (arr, low, high);
        quicksort (arr, low, pi-1);
        quicksort (arr, pi+1, high);
    }
}

int partition (int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low-1);
    for (int j = low; j <= high-1; j++)
    {
        if (arr[j] < pivot)
        {
            i++;
            swap (&arr[i], &arr[j]);
        }
    }
    swap (&arr[i+1], &arr[high]);
    return (i+1);
}
```

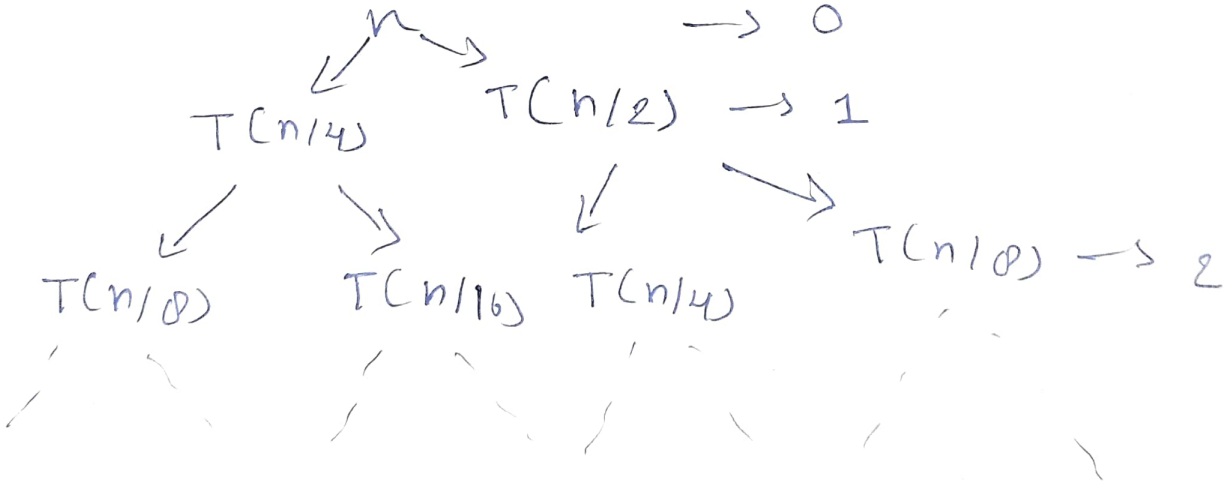
2) n^3 → Multiplication of 2 square matrix

```
for (i=0; i<n1; i++)
  for (j=0; j<c2; j++)
    for (k=0; k<c1; k++)
    {
      res[i][j] += a[i][k] * b[k][j];
    }
```

3) log (log n)

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

Q04⇒ $T(n) = T(n/4) + T(n/2) + cn^2$



At level

$0 \rightarrow cn^2$

$1 \rightarrow \frac{n^2}{4^2} + \frac{n^2}{2^2} = \frac{c\,5n^2}{16}$

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

$\vdots$

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

$$= k = \log_2 n$$

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

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

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

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

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

$$= \underline{O(cn^2)} \qquad \underline{\text{Ans}}$$

Soln 5 → for

i	j
1	1
2	1 + 3 + 5
3	1 + 4 + 7
$\vdots$	
n	1 + 5 + 9

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

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

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

$$T(n) = n[1 + 1/2 + 1/3 + \cdots 1/n] - 1[1 + 1/2 + 1/3 \cdots + 1/n]$$

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

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

Sol^n 6 -> for

i
2^1
2^k
2^{k^2}
2^{k^3}
$\vdots$
2^{k^m}

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

$k^m \leq \log_2 n$

$m = \log_k \log_2 n$

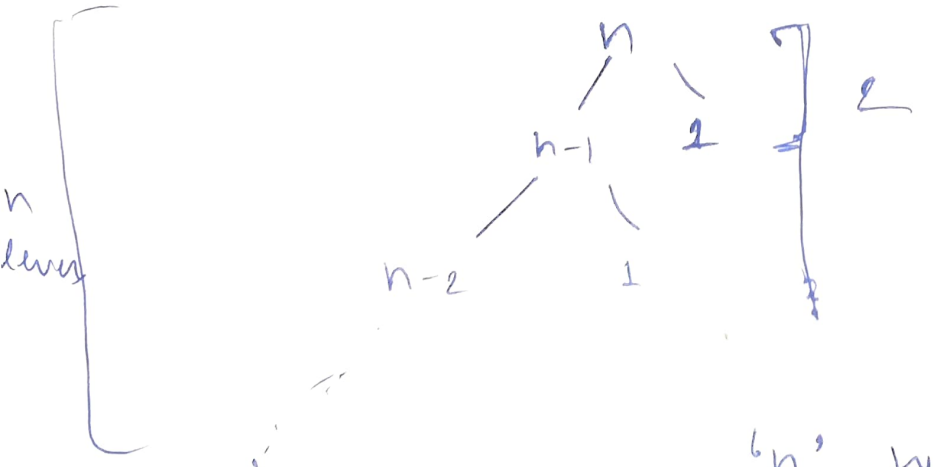
$\because \sum_{i=1}^{m} 1$

$1 + 1 + 1 + \cdots \cdots$ m times

$T(n) = O(\log_k \log n)$ -> Ans

Sol^n 7 -> Given algorithm divides array in 99% and 1% part.

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



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

$= n \times n$

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

Lowest height = 2

highest height = n

$\therefore$ difference $= n-2$ $\quad n \geq 1$

The given algorithm produces linear result

Sol^n a →

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

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

c) $2^6 < \log_6 n < \log 2n < 5n < n\log_6(n) < n\log_2 n$
$< \log(n!) < 8n^2 < 7n^3 < n! < 6^{2n}$

Tutorial - 3

Name - Garima Singh
Section - F
Roll no - 02

Ans -> 1 ->

```
for( i= 0 to n)
{
    if (arr[i] == value)
        // element found
}
```

Ans 2 -> 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 = j-1;
        }
        arr[j+1] = x;
    }
}
```

Recursive

```
void insertion_sort (int arr[], int n)
{
    if(n<=1)
        return;
    insertion_sort (arr, n-1);
    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 to know anything about what values it will sort and information is requested while algorithm is running.

Other Sorting Algorithm :-

- Bubble Sort
- Quick Sort
- Merge Sort
- Selection Sort
- Heap Sort

Ans 3 →

Sorting Algorithm	Best	Worst	Average
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick Sort	$O(n \log n)$	$O(n^2)$	$O(n \log n)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Ans 4 →

Inplace Sorting	Stable Sorting	Online Sorting
Bubble sort	Merge sort	Insertion sort
selection sort	Bubble sort	
Insertion sort	Insertion sort	
Quick sort	Count sort	
Heap sort		

Ans 5-> Iterative ->

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

Recursive ->

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

Time complexity

- Linear Search - O(n)
- Binary Search - O(log n)

Ans 6 → $T(n) = T(n/2) + 1$ — (1)

$T(n/2) = T(n/4) + 1$ — (2)

$T(n/4) = T(n/8) + 1$ — (3)

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

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

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

⋮

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

Let $2^k = n$

$k = \log n$

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

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

$T(n) = O(\log n)$ → Answer

Ans 7 →

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

Ans 8 → Quicksort is fastest general-purpose sort. In most practical situations quicksort is the method of choice as stability is important and space is available, mergesort might be best.

Ans 9 → A pair $(A[i], A[j])$ is said to be inversion if

- $A[i] > A[j]$
- $i < j$
- Total no. of inversions in given array are 3 using merge sort.

Ans 10 → Worst Case $O(n^2)$ → The wosst case occuss 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.

Base Case $O(n \log n)$ → The best case occurs when we will select pivot element is a mean element

Ans 11 → Merge sort →

Best Case → $T(n) = 2T(n/2) + O(n)$

Worst Case → $T(n) = 2T(n/2) + O(n)$ } $O(n \log n)$

Quicksort →

Best Case → $T(n) = 2T(n/2) + O(n) \rightarrow O(n \log n)$

Worst Case → $T(n) = T(n-1) + O(n) \rightarrow O(n^2)$

In quicksort, array of element is divided into 2 subarray $(n/2)$ again and again until only one element is left.

Ans 12 →

```
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>j)
    {
        a[min] = a[min-j];
        min--;
    }
    a[i] = key;
}
```

Ans 13 → A better version of bubble sort, known as the bubble sort, includes a flag that is set if a exchange is made of ten an entire pass over. If no exchange is made after an entire pass over then it should be called the array is already order because no two elements need to be switched.

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

Tutorial-4

Name - Garima Singh
Section - F
Rollno - 2016744
Class Rno - 02

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

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

$a \geq 1, \quad b \geq 1$

On Comparing

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

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

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

$f(n) > n^c$

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

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

$a \geq 1, \quad b \geq 1$

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

$c = \log_2 4 = 2$

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

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

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

-> $a = 1, \quad b = 2$

$f(n) = 2^n \quad \rightarrow c = \log_b a = \log_2 1 = 0$

$n^c = n^0 = 1 \quad \rightarrow f(n) > n^c$

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

Ans 4 → $T(n) = 2^n T(n/2) + n^n$

→ $a = 2^n$

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

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

$n^c \rightarrow n^n$

$f(n) = n^c$

∴ $T(n) = \Theta(n^2 \log n)$

Ans 5 → $T(n) = 16T(n/4) + n$

$a = 16$, $b = 4$

$f(n) = n$

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

$n^c = n^2$

$f(n) < n^c$

∴ $T(n) = \Theta(n^2)$

Ans 6 → $T(n) = 2T(n/2) + n \log n$

→ $a = 2$, $b = 2$

$f(n) = n \log n$

$c = \log_2 2 = 1$

$n^c = n^1 = n$

$n \log n > n$

$f(n) > n^c$

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

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

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

$c = \log_2 2 = 1$

$n^c = n^1 = n$

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

$\therefore f(n) \not\geq n^c$

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

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

$a = 2,\ b = 4.\ f(n) = n^{0.51}$

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

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

$f(n) > n^c$

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

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

$a = 0.5,\ b = 2$

$a \geq 1$ but here $a = 0.5$.

So we cannot apply master's Theorem

10) $T(n) = 16T(n/4) + 2n$

$a = 16,\ b = 4,\ f(n) = 2n$

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

$n^c = n^2 < 2n \quad \therefore f(n) > n^c$

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

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

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

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

$n^c = n^2 > \log n$

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

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

$a = \sqrt{n}, \quad b = 2, \quad f(n) = \log n$

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

$n^c = n^{\frac{1}{2}\log_2 n}$ $\quad \Delta \; f(n) = \log n$

$\therefore n^{\frac{1}{2}\log_2 n} > \log n$

$\therefore T(n) = \theta(\sqrt{n}\, n^{\frac{1}{2}\log n})$

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

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

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

$n^c = n^{1.5849}$

$n < n^{1.5849}$

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

$T(n) = \theta(n^{1.5849})$

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

$a = 3,\ b = 3$

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

$n^c = n$ & $f(n) = \sqrt{n}$

$\therefore\ n > \sqrt{n}$

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

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

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

$a = 4,\ b = 2$

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

$n^c = n^2$

$n < n^2$ (for any constant)

$f(n) < n^c$

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

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

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

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

$n^c = n^{0.792}$

$n^{0.792} < n \log n$

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

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

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

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

$n^c = n$

$A_2 \rightarrow \quad n/2 < n$

$f(n) < n^c$

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

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

$a=6, \quad b=3$

$c = \log_b a = \log_3 6 = 1.6309$

$n^c = n^{1.6309}$

$n^{1.6309} < n^2 \log n$

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

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

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

$c = \log_b a = 2$

$n^c = n^2 \rightarrow \frac{n}{\log n}$

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

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

$c = \log_8 64 = 2 \quad \rightarrow \quad n^c = n^2$

$n^2 < n^2 \log n$

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

Tutorial - 5

DAA

Name: Garima Singh
Section: F
Rollno: 2016744
Class Rollno: 02

1. What is difference b/w DFS and BFS. Please write the applications of both the algorithms.

BFS	DFS
1) It stands for Breadth First Search.	1) It stands for Depth First Search.
2) It uses Queue data Structure	2) It uses stack data Structure.
3) It is more suitable for searching vertices which are closer to given source.	3) It is more suitable when there are solutions away from source.
4) BFS considers all neighbour first & therefore not suitable for decision making trees used in games & puzzles	4) DFS is more suitable for game or puzzle problems. We make a decision, then explore. And if decision leads to win ~~children are visited~~ situations we stop.
5) Here sibling are visited before children	5) Here children are visited before siblings.
6) There is no concept of backtracking.	6) It is a recursive algorithm that uses backtracking.
7) It requires more memory	7) It requires less memory.

Applications :-

BFS → Bipartite graph and shortest path, peer to peer networking, crawlers in search engine & GPS navigation system.

DFS → cyclic graphs, topological order, scheduling problems, sudoku puzzle.

Q02 ⇒ Which Data structures are used to implement BFS and DFS and why?

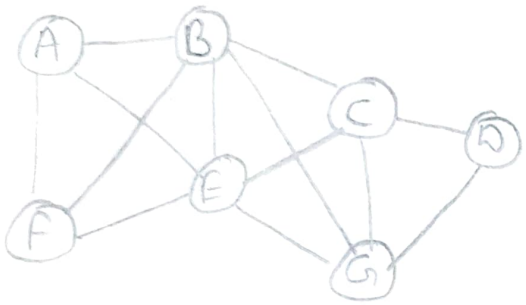
Ans ⇒ BFS → Queue
DFS → Stack.

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 nodes w.r.t. their distance from root (source). For this queue is better to use in BFS.

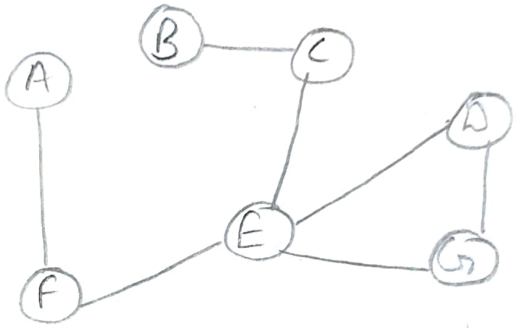
For implementing DFS we need a stack data structure as it traverse a graph in depth mand motion and uses stack to remember to get the next vester to start a search, when a dead end occurs in any iteration.

Q.3 → What do you mean by sparse and dense graphs? Which representation of graph is better for sparse and dense graph?

Ans 3 → Dense graph is a graph in which no. of edges is close to maximal no. of edges.
Sparse graph is graph in which no. of edges is very less.



Dense Graph
(many edges b/w nodes)



Sparse graph (few edges b/w nodes)

- 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 and DFS?

Ans 4 → 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 & initialize 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 queue.

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 following:

DFS for a connected graph produces a tree. There is cycle in graph if there is a back edge present ~~pres~~ in the graph. A back edge is an edge that is from a node to itself (self-loop) or one of its ancestors 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, keep track of vertices currently in recursion stack for DFS traversal. If a vertex is reached that is already in recursion stack, then there is a cycle.

Qo 5 ⇒ What do you mean by disjoint set data structure? Explain 3 operations along with examples which can be performed on disjoint sets?

Ans ⇒ A disjoint set is a data structure that keeps track of set 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 to itself.

Eg:- int find (int i){
    if (parent[i] == i){
        return i;
    }
    else {
        return find (parent[i]);
    }
}

2) Union → It takes 2 elements as input and find representatives of their sets using the find operations. And finally puts either one of it's trees under root node of other trees, effectively merging the trees and sets.

Eg → 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 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.

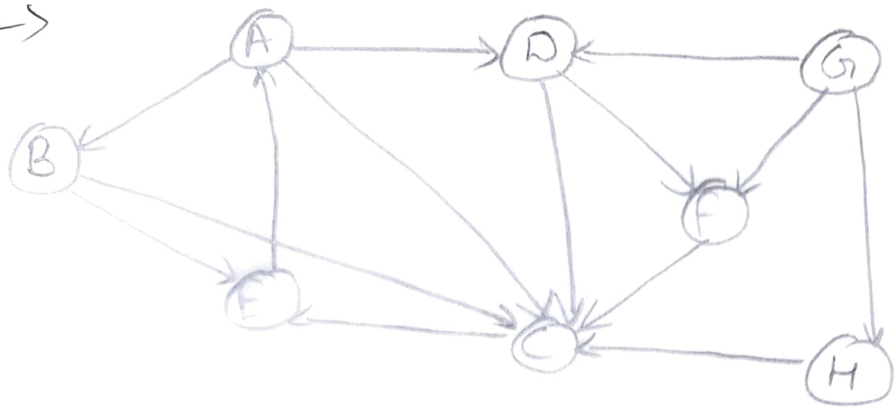
- If rank of left is less than right then it's best to move left under right & vice versa.
- If ranks are equal, rank of result will always be one greater than rank of trees.

Eg:- 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]++;
  }
}
```

Q06⇒ 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

G
D
H
F
C
E
A
B

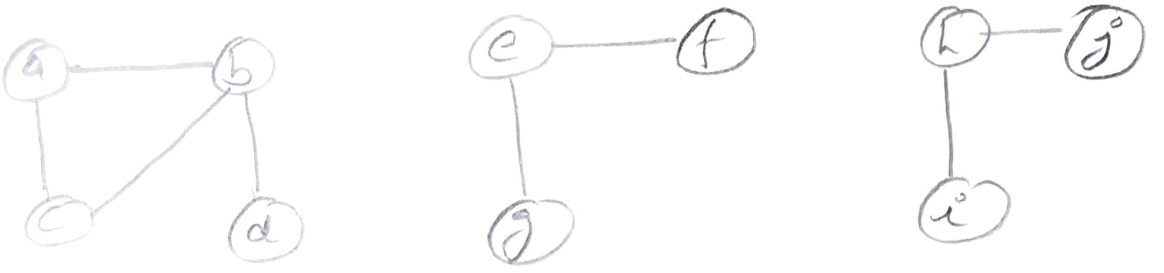
Nodes visited

G
F
C
E
A
B

Stack

Path → G → F → C → E → A → B

Q07⇒ 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}.

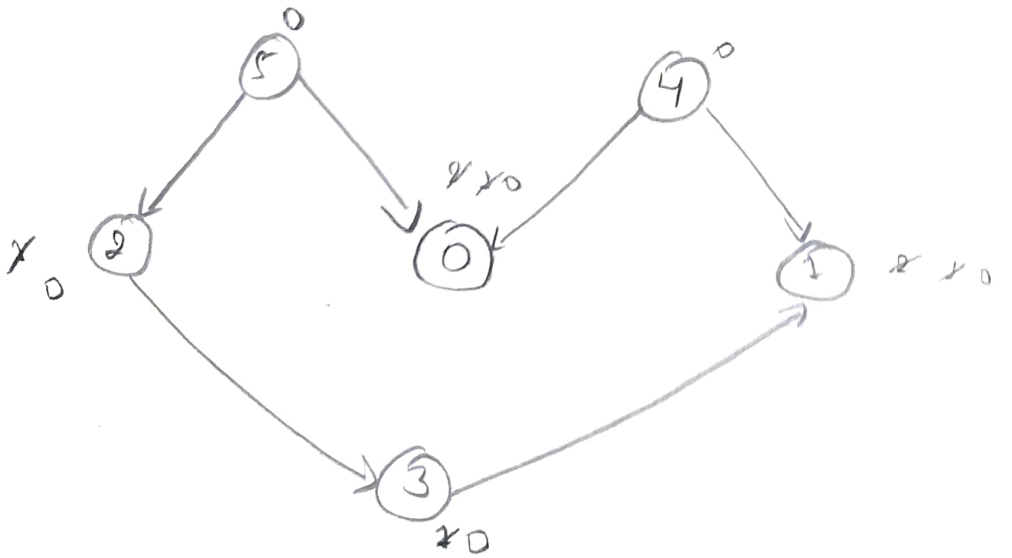
(e,f) {a,b,c,d} {e,f}, {g} {h} {i} {j}

(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 ⟶ Ans.

Q03⇒ Apply topological sort & DFS on graph having vertices from 0 to 5.



We take source node as 5

Applying Topological Sort → q: 5/4; Pop 5 & decrement indegree of it by 1

q: 4/2; Pop 4 & decrement indegree & push 0.

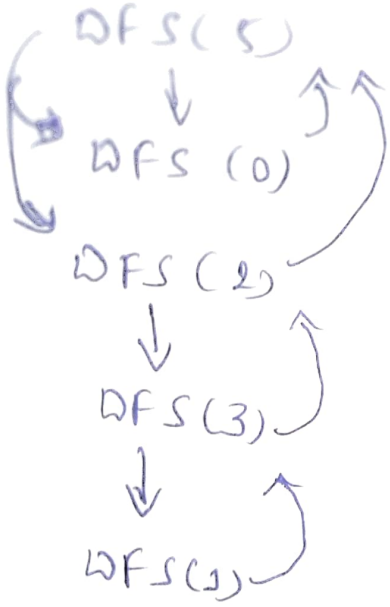
q: 2/0 Pop 2 & decrement indegree & push 3.

q: 0/3 Pop 0, Pop 3 Push 1

q: 1; Pop 1

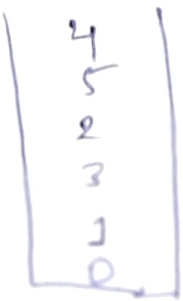
Ans = 5 4 2 0 3 1

Topological sort.



DFS(4)
↓
Not possible

DFS



4 → 5 → 2 → 3 → 1 → 0

Ans

Q9 = ~~Heap~~

Ans = 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 2 types max priority queue based on max heap and min priority queue based on min-heap.

Heaps provide better performance (comparison to array & LoL.

The graphs like Dijkstra's shortest path algo, 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.

Q) → Differentiate b/w Min-heap and Max heap.

Min-heap	Max-heap
1) In min-heap, key present at root node must be less than or equal to among keys present at all of its children.	1) In max-heap the key present at root node must be greater than or equal to among keys present at all of its children.
2) The minimum key element is present at the root.	2) The maximum key element is present at the root.
3) It uses ascending priority.	3) It uses descending priority.
4) The smallest element has priority while construction of Min-heaps.	4) The largest element has priority, while construction of Max heaps.
5) The smallest element is the first to be popped from the heap.	5) The largest element is the first to be popped from the heap.

DAA Tutorial-6

Name: Garima Singh
Section: F
Rollno: 2016744
Class Rollno: 02

Q01⇒ What do you mean by Minimum Spanning Tree? What are the application of MST?

Ans⇒ Minimum Spanning Tree is a subset of edge of a connected edge-weighted undirected graph that connects all the vertices together w/o any cycles and with minimum posible edge weighted.

Applications

1) Consides n stations are to be linked using a communication network and lying of communication link blw any station involves a cost. The ideal solution would be to extract a subgraph termed as minimum cost spanning tree.
2) Designing LAN
3) Suppose you meant to construct highways or railroads spanning several cities, then we can use concept of MST.
4) Laying pipeline connecting offshore drilling sites, refineries & consumer markets.

Q02⇒ Analyze time and space complexity of Prim, kruskal and Dijkstra and Bellman Ford Algorithm.

Ans⇒ Time complexity of Prim's algorithm: $O(|E| \log |V|)$
Space complexity of Prim's Algorithm: $O|V|$

Time complexity of Kruskal's Algorithm : $O(E \log E)$

Space complexity of Kruskal's : $O(V)$

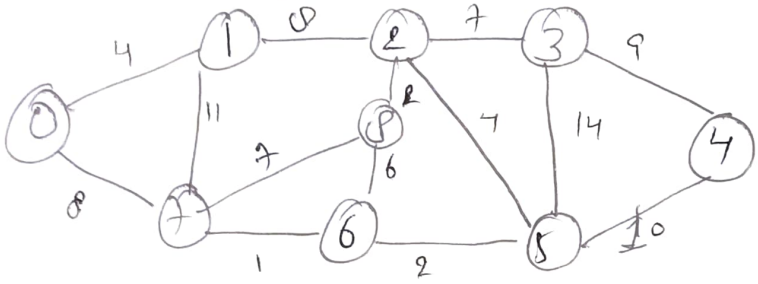
Time " " Dijkstra's : $O(V^2)$

Space " " " : $O(VE)$

Time " " Bellman Ford : $O(VE)$

Space " " " : $O(E)$

Q03: Apply Kruskal's & Prim's Algorithm on given graph to compute MST and its weight.



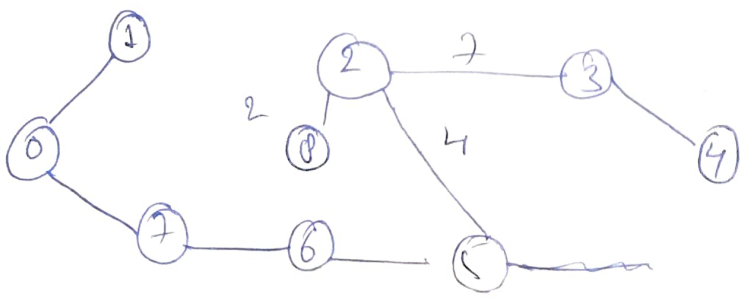
Ans 3>

Kruskal's Algorithm

u	v	w	
0	7	1	✓
5	6	2	✓
2	8	2	✓
0	1	4	✓
2	5	4	✓
6	8	6	✗
2	3	7	✓
7	8	7	✗
0	2	8	✓
1	3	8	✗
4	5	9	✓
4	7	10	✗
1	2	11	✗
3	5	14	✗

Weight Algorithm

weight = 4+8+2+4+2
+ 7+ 9+ 3
= 37

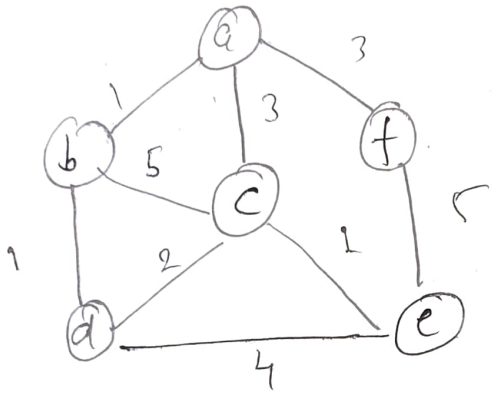


weight = 1+2+2+4+4+7+8
+ 9
= 37

Q04> Given a directed weighted graph. You are also given the shortest path from a source vertex 's' to a destination vertex 't'. Does the shortest path remain same in following cases.

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

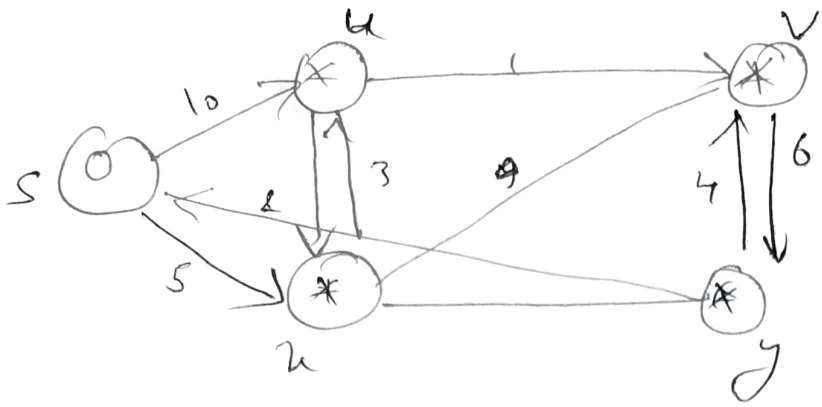
(ii) If weight of every edge is multiplied by 10 units.



Ans (i) The shortest path may change. The reason is that there may be different no. of edge in different paths from 's' to 't'. For eg -> Let the shortest path of weight is and has edge is 5. Let there is another path with 3 edge 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 & becomes 25+20. So, the shortest path changes to other path with weight as 15.

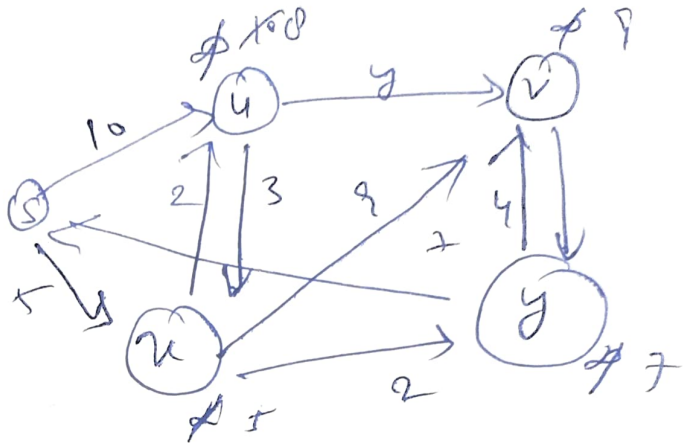
(ii) If we multiply all edges weight by 10, the shortest path doesn't change. The reason is that weight of all path from 's' to 't' gets multiplied by same unit. The numbers of edges or path doesn't matter.

Ques →



Ans → Dijkstra's Algorithm :-

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



Bellman ford Algorithm :-

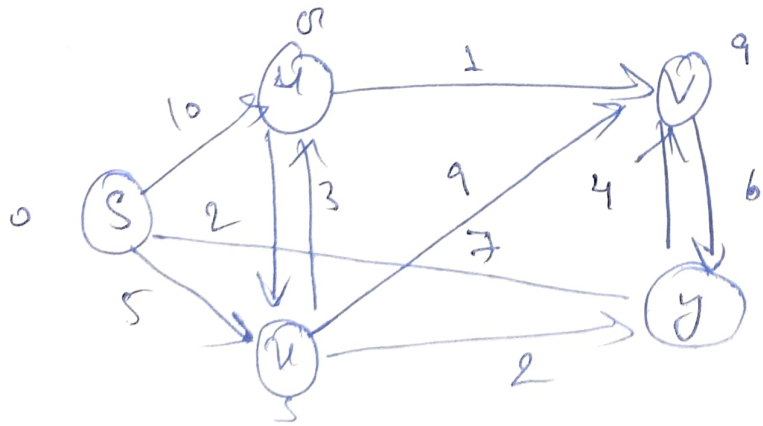
1st → s(0) u(∞ 10) v(∞) x(∞ 5) y(∞)

2nd → s(0) u(10) v(∞ 11) x(5) y(∞)

3rd → s(0) u(10 8) v(11 9) x(5) y(∞ 7)

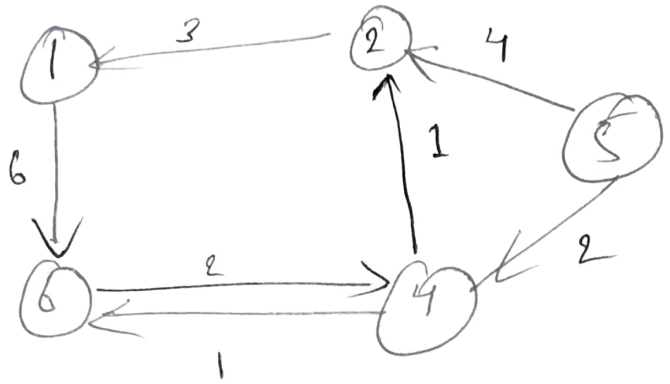
4th → s(0) u(8) v(9) x(5) y(7)

graph does not have negative cycle.



→ Find graph

Q.6 →



	1	2	3	4	5
1	0	∞	6	3	∞
2	2	0	∞	∞	∞
3	∞	∞	0	2	∞
4	∞	1	1	0	∞
5	∞	4	∞	2	0

⇒

	1	2	3	4	5
1	0	∞	6	3	∞
2	2	0	∞	5	∞
3	∞	∞	0	2	∞
4	∞	1	1	0	∞
5	∞	4	∞	2	0

⇒

	1	2	3	4	5
1	0	∞	6	3	∞
2	2	0	∞	5	∞
3	∞	∞	0	2	∞
4	3	1	1	0	∞
5	6	4	12	2	0

⇒

	1	2	3	4	5
1	0	∞	6	3	∞
2	2	0	∞	5	∞
3	∞	∞	0	2	∞
4	3	1	1	0	∞
5	6	4	12	2	0

Time Complexity ⇒ $O(|V|^3)$

Space Complexity ⇒ $O(|V|^2)$