Tutorial - 1 (DAA)

Ans 1. **Asymptotic Notation**: Asymptotic Notation are the mathematical notation used to describe the running time of an algorithm.

Different types of Asymptotic Notation.

1. **Big-O Notation (O)**: It represents upper Bound of algorithm. $f(n) = O(g(n))$ if $f(n) < c * g(n)$
2. Omega Notation (Ω): It represents lower bound of algorithm. $f(n) = \Omega(g(n))$ if $f(n) > c * g(n)$
3. Theta Notation (Θ): It represents upper and lower bound of algorithm.

$f(n) = \theta(g(n))$ if $c_1 g(n) < f(n) \leq c_2 g(n)$

Ans 2.

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

i = 1
i = 2
i = 4
i = 8
i = 16
i = n

It is forming up

$a_n = a r^{n-1}$

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

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

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

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

$\boxed{k = \log n + 1}$

$(a_n = n, r = 2, a = 1)$

$O(\log n)$

Ans 3 $T(n) = 3T(n-1)$ if $n>0$, otherwise 1

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

$T(1) = 3 \times 1$

$T(2) = 3T(1) = 3 \times 3 \times 1$

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

$T(n) = 3 \times 3 \times 3 \ldots$

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

Ans 4. $T(n) = 2T(n-1) - 1$ if $n>0$, otherwise 1

$T(0) = 1$

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

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

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

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

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

$= 2 - 1 = 1$

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

Ans 5.

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

i = 1
i = 2
i = 3
i = 4

s = 1
s = 1 + 2
s = 1 + 2 + 3
s = 1 + 2 + 3 + 4

Loop end when $S > n$

$1 + 2 + 3 + 4 \dots k > n$

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

$k^2 > n$

$k > \sqrt{n}$

$= O(\sqrt{n})$

Ans 6 -

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

Loop ends when $i * i > n$

$k * k > n$

$k^2 > n$

$k > \sqrt{n}$

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

Ans 7

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

- 1st loop : $i = \frac{n}{2}$ to n , $i++$

$= O(\frac{n}{2}) = O(n)$

- 2nd Nested loop: $j = 1$ to n , $j = j * 2$

$j = 1$
$j = 2$
$j = 4$
$j = n$

$= O(\log n)$

- 3rd Nested loop : $k = 1$ to n , $k = k * 2$

$k = 1$
$k = 2$
$k = 4$

$= O(\log n)$

Total complexity $= O(n \times \log n \times \log n) = O(n \log^2 n)$

Ans 8.

```
Function (int n)
{
    if (n==1) return;
        for (int i=1 to n)
        {
            for (int j = 1 to n)
            {
                Printf ("*");
            }
        }
    Function (n-3) — T(n-3)
}
```

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

$T(1) = 1$

$T(1) = 1$

$T(4) = T(4-B) + 4^2$

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

$T(7) = T(7-3) + 7^2$

$= 1^2 + 4^2 + 7^2$

$T(10) = T(10-3) + 10^2$

$= 1^2 + 4^2 + 7^2 + 10^2$

So, $T(n) = 1^2 + 4^2 + 7^2 + 10^2 \ldots n^2 = \frac{n(n+1)(2n+1)}{6}$

also for term like $T(2), T(3), T(5)$ $= O(n^3)$

So, $T(n) = O(n3)$

Ans 9.

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

i = 1 — j = 1 to n
i = 2 — j = 1 to n
i = 3 — j = 1 to n
i = 4 — j = 1 to n

So, for i upto n it will take
So, $T(n) = O(n^2)$

Ans 10

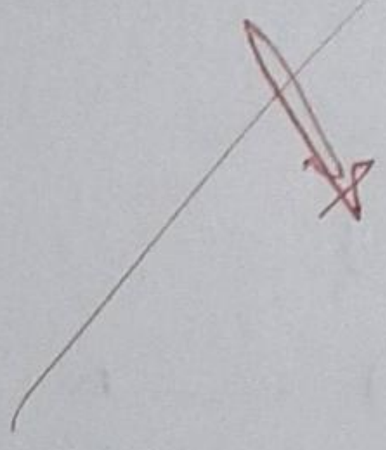
$f_1(n) = n^k$ $\quad$ $f_2(n) = c^n$

Asymptotic relation between f_1 & f_2

is Big O i.e $f_1(n) = (f_2(0)) = O(c^n)$

i.e $n^k < G * c^n$

[G is some constant)



Tutorial-2

Ans 1.

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

j=1 ; i = 0+1
j=2 ; i = 0+1+2
j=3 ; i = 0+1+2+3.
:
Loop ends when i>n

$0+1+2+3\cdots n>n$

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

$k^2 > n$

$k > \sqrt{n}$

$O(\sqrt{n})$

Ans 2.

Recurrence Relation For Fiborocci series.

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

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

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

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

(Lower Bound)

$T(n) = 2\{2T(n-4)\} = 4T(n-4)$

$= 4(2T(n-6))$

$= 8T(n-6)$

$= 8(2T(n-8))$

$= 16T(n-8)$

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

$n-2k = 0$

$n = 2k$

$k = \frac{n}{2}$, $T(n) = 2^{n/2} + T(0)$

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

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

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

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

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

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

$n - k = 0$

$\boxed{k = n}$

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

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

Ans 3. $O(n \log n)$ ⇒

```
for (int i=0; i<n; i++)
{
for (int j=1; j<n; j=j*2)
{
// same O(1)
}
}
```

- $O(n^3)$ ⇒

```
for (int i=0; i<n; i++)
{
for (int j=0; j<n; j++)
{
for (int k=0; k<n; k++)
{
// same O(1)
}
}
}
```

• O(log(log n)) ⇒ for(int i = i; i <= n · j i= i * 2)

{

for(int j = 1 ; <= n ; * j = j * 2)

{

// some O(1)

}

}

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

Lets assume $T(n/2) >= T(N/4)$

So, $T(n) = 2T(N/2) + cn^2$

applying master theorm $T(n) = a*(\frac{n}{b}) + f(n))$

$a = 2, b = 2$ $f(n) = n2$

$c = \log_b a = \log_2 2 = 1$

$n^c = n$

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

$f(n) > n^c$ So, $T(n) = \theta(n^2$

Ans 5. int fun ~~(int n)~~

{

for(int i = 1; i <= n; i++)

{

for(int j = 1; j < n ; j += 1)

{ //some O(1)

}

}

}

i = 1 — j = 1
j = 2 — n times.
j = 3
j = n

$i = 2$ —— $j = 1$, $j = 3$, $j = 5$, $j = 7$ — loop ends with $j \geq n$

$k \geq \frac{n}{2}$

— n times.

$i = 3$ —— $j = 1$, $j = 4$, $j = 7$ — $1 + 4 + 7 > n$

$k > \frac{n}{3}$

$i = 4$ —— $k > n/4$

$i = n$

So Total complexity = $O(n^2 + n^2 + n^2 \ldots)$

$= O(n^2)$.

Ans 6.

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

Complexity Pow(i, k) —— $O(\log N)$

$= \log(k)$

$i = 2$

$i = 2^k$

$i = 2^{k^2}$

$i = 2^{k^3}$

$i = 2^{k^4}$

$j = 2^{k^m}$.

loop ends when $i > n$

$2^{k^m} > n$

$\log(2^{k^m}) > \log n$

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

$k^m > \log n$

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

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

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

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

Q#8 a) $100 < \log n < \sqrt{n} < n < \log(\log n) < n \log n$
$< \log n! < n! < n^2 < \log 2^n < 2^n < 2^{2n} < 4^n$

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

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

Tutorial-3

Ans 1

```
while (low<= High)
{
    mid = (low + High) /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(int 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 arr[], int n)
{
    if (n<=1)
        return;
    insertionSort(arr, n-1);
    int last = arr[n-1];
    j = n-2
while(j>=0 && arr[j] > last)
{
    arr[j+1] = arr[j];
        j--;
}
    arr[j+1] = last;
}
```

Insertion sort is online sorting because wherever a new element come, insertion sort define its right place.

Ans 3 -

Bubble sort $- O(n^2)$
Insertion sort $\rightarrow O(n^2)$
Selection sort $\rightarrow O(n^2)$
Merge sort $- O(n * \log n)$
Quick sort $- O(n \log n)$
Count sort $\rightarrow O(n)$
Bucket sort $\rightarrow O(n)$

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)
        Binary search (arr, low, mid-1);
    else
        Binary - search (arr, mid+1, High);
}
return false;
```

O (log n)

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

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]] = i
else
{
cout << i << " " << mp [arr[i]];
}
}
```

Ans 8- Quicksort is the fastest general purpose sort In most Practical situation, quicksort is the method of choice. If stability is important and space is available, mergesort might be best.

Ans 9. Inversion indicates — how far or close the array is from being sorted.

Ans 10 **Worst-Case**: The worst case occurs when the Picked Pivot is always an extreme (smallest as largest) element. this happens when input array is sorted as reverse sorted and either first or last element is Picked as Pivot $O(n^2)$.

Best Case: Best case occur when Pivot element is the middle element as near to the middle element. $O(n \log n)$.

Ans 11 **merge sort**: $T(n) = 2T(n/2) + O(n)$

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

Basis	Quick sort	Merge sort
• Partition	Splitting is done in any ratio	array is Parted into Just 2 halves.
• Work work on	Smaller arrays	Fine on any size of array.
• Additional space	Less (in place)	More (Not in place)
• Efficient	inefficient for large array.	More efficient
• sorting method	Internal	External
• Stability	Not stable.	Stable

Ans 14 We will use Merge sort 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 to sort is stored in memory at all times while sorting is in Progress.
- External sorting: all the data is stored outside memory and only loaded into memory in small chunks.

Tutorial - 4

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

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

$n^{\log_b a} = n^{\log \frac{3}{2}}$

Comparing $n^{\log \frac{3}{2}}$ and n^2

$n^{\log_2 3} < n^2$ (Case 3)

∴ according to master theorm

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

(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 2)

∴ according to master theorem

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

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

$a = 1, b = 2$

$n^{\log_2 1} = n^0 = 1$.

$1 < 2^n$ (Case 3)

∴ According to master theorm

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

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

∴ Master's theorem is not applicable as a is function of n.

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

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

$n^{\log_b a} = n^{\log_4 16} = n^2$

$n^2 > f(n)$ (Case 1)

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

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

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

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

Now $f(n) > n$

$\therefore$ According to master $T(n) = \theta(n \log n)$

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

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

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

$n > f(n)$

$\therefore$ According to master theorem $T(n) = \theta(n)$

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

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

$n^{\log_b a} = n^{\log_4 2} = n^{0.5}$

$n^{0.5} < f(n)$

$\therefore$ According master theorem $T(n) = \theta(n^{0.5})$

(9) $T(n) = 0.5 + (n/2) + \frac{1}{n}$

∴ Master's Not applicable as $a < 1$

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

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

$n^{\log_b a} = n^{\log_4 16} = n^2$

$n^2 < n!$

∴ According to master $T(n) = \Theta(n!)$

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

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

$n^{\log_b a} = n^{\log_2 4} = n^2$

$n^2 > f(n)$

∴ According to master's $T(n) = \Theta(n^2)$

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

∴ Master's Not applicable as a is not constant.

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

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

$n^{\log_b a} = n^{\log_2 3} = n^{1.58}$

$n^{1.58} > f(n)$

∴ According to master's theorem
$T(n) = O(n^{\log_2 3})$

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

$a=4, b=2, f(n) = c*n$

$n^{\log_b a} = n^{\log_2 4} = n^2$

$n^2 > c*n$

$\therefore$ According to master's theorem,

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

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

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

$n^{\log_b a} = n^{\log_4 3} = n^{0.79}$

$n^{0.79} < n \log n$

$\therefore$ According to master's theorem, $T(n) = O(n \log n)$

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

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

$n^{\log_b a} = n^{\log_3 3} = n$

$O(n) = O(n/2)$

$\therefore$ According to master's theorem.

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

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

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

$n^{\log_b a} = n^{\log_3 6} = n^{1.63}$

$n^{1.63} < n^2 \log n$

$\therefore$ According to master's theorem

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

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

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

$n^{\log_b a} = n^{\log_2 4} = n^2$

$n^2 > n/\log n$

∴ According to master's theorm

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

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

Ans Master's theorm is not applicable as $f(n)$ is not increasing function.

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

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

$n^{\log_b a} = n^{\log_3 7} = n^{1.7}$

$n^{1.7} < n^2$

∴ According to master's $T(n) = O(n^2)$

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

Ans - Master's Theorm is n't applicable since regularity condition is isolated in case 3.

DAA - Tutorial 5

Q1.
Ans.

	BFS	DFS
Definition	Stands for Breadth First Search.	Stands for Depth First Search
Datastructure	It uses queue for finding the shortest path.	It uses stack to find the shortest find.
sources	It is better when target is closer to source	It is better when target is far from source
Decision Tree	It is consider all neighbour so it is not suitable for decision tree used in puzzle game	It is more suitable as with one decision. we need to traverse further to augment the decision.
speed Time complexity	It is slower than DFS. O(V+E) where V is vertices and E is edge	It faster than BFS O(V+E).

Q2. Ans stack is used to implement DFS, because in it we first traverse the whole branch of the tree and later on visit the adjacent branch, since this is similar to LIFO, therefore stack is used.

Queue is used to implement BFS, it is because queue is used as a FIFO instead because BFS is to test the immediate children first and after all immediate children are tested, to then return to those children and check their children and so forth.

Q3. Ans. Sparse Graph: Graph where no. of edges is much less than the possible no. of edges.

Dense Graph: where no. of edges is much more close to maximum no. of edges.

If graph is dense, it should be represented by adjacency matrix.

If graph is sparse, it should be represented by adjacency list.

Q4. Ans. BFS

In undirected graph, do a BFS traversal on given graph, for each visited vertex v, if there is an adjacent 'u' such that 'v' is already visited and 'u' is not parent of 'v' then there is cycle in graph.

DFS

Run DFS from a node and mark this node as visited, now for any other vertex if it is neighbour is already visited and that neighbour is not the parent of that current node then there exist a cycle in the graph.

Q5. Ans. Disjoint set Data Structure

The distaint set can be defined as the subsets where there is no common element b/w two sets.

operation are:

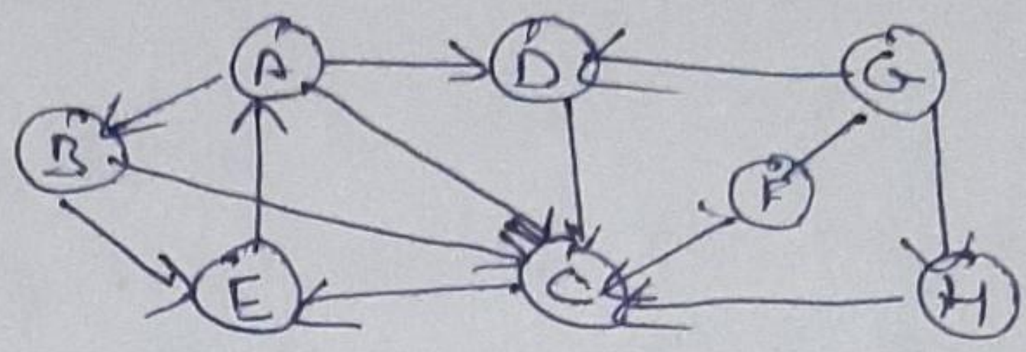
1) Union
2) make one set
3) find

Q6

Ans BFS

A → B → C → D → E,

G → H → F



DFS

A → D → C → B, G → F → H

Q7

Ans connected components → 4

vertices = 10

Q8

Ans. Topological sort → 0 — 1 - 2 - 3 - 4 - 5

DFS → 5 → 2 → 3 → 1 → 0

4 can't be reached.

Q9.

Ans: Yes, heap data structure can be used to create priority queue.

- Dijkstra's to find shortest path.
- Prism's Algo.
- Huffman Algo.

Q10)

Ans. min. heap → root element is the smallest.

max. heap → root element is the largest.

DAA - Tutorial - 06

Q1. Ans. Minimum Spanning Tree

It is a spanning tree which has minimum total cost. If we have linked undirected graph with a weight combine with each edge, then the cost of spanning would be the sum of the cost of its edge.

Application: in design of networks including computer networks, telecommunication networks transportation networks.

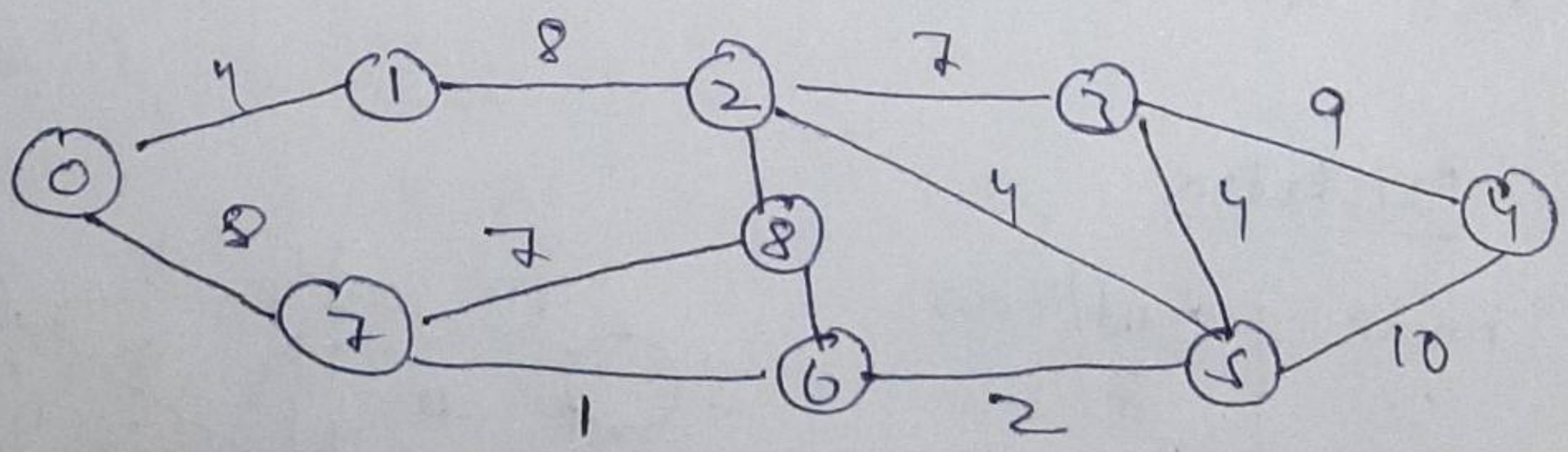
Q2

Ans.

Time complexity	Prism.	Dijkstra	BellmonFord
	$O((V+E)\log V)$	$O(E\log V)$	$O(VE)$
Space	$O(V+E)$	$O(V^2)$	$O(N)$

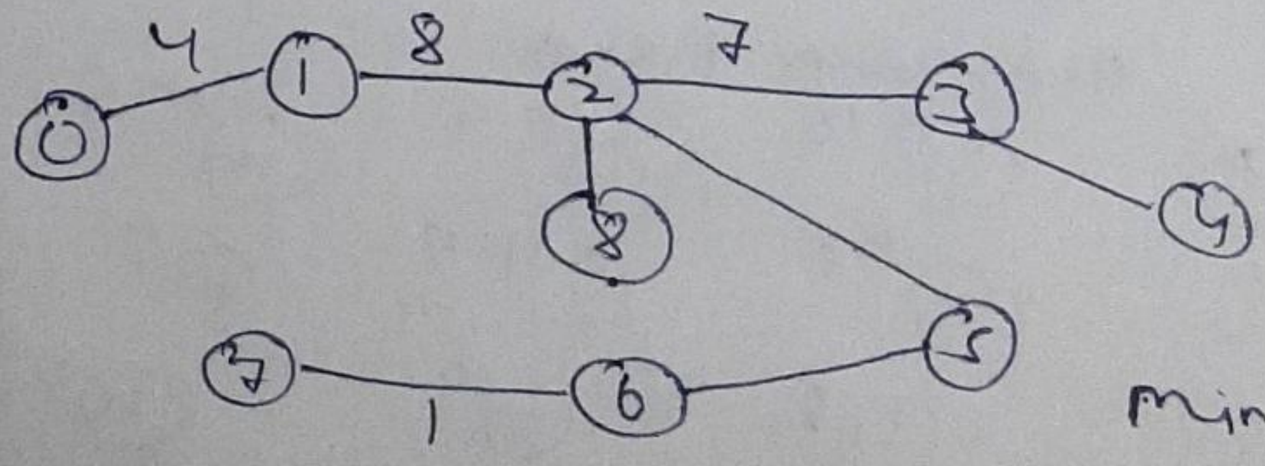
Q3.

Ans.



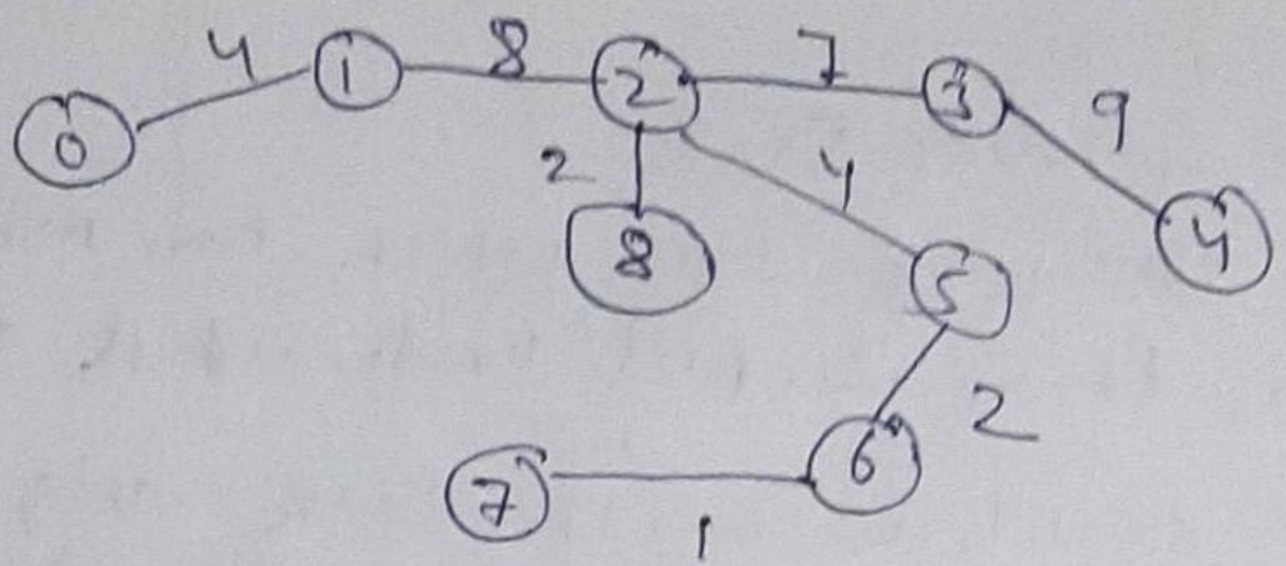
(i) Kruskals

[1, 2, 2, 4, 4, 6, 7, 7, 8, 8, 9, 10, 11, 14]



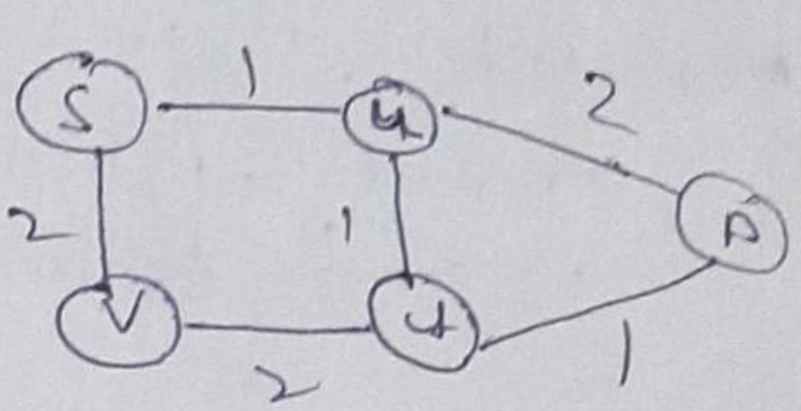
Min wt = 37

(iii) Prim



min wt = 37.

Q4
Ans. Let we have
initial shortest path
$S \to V \to t$

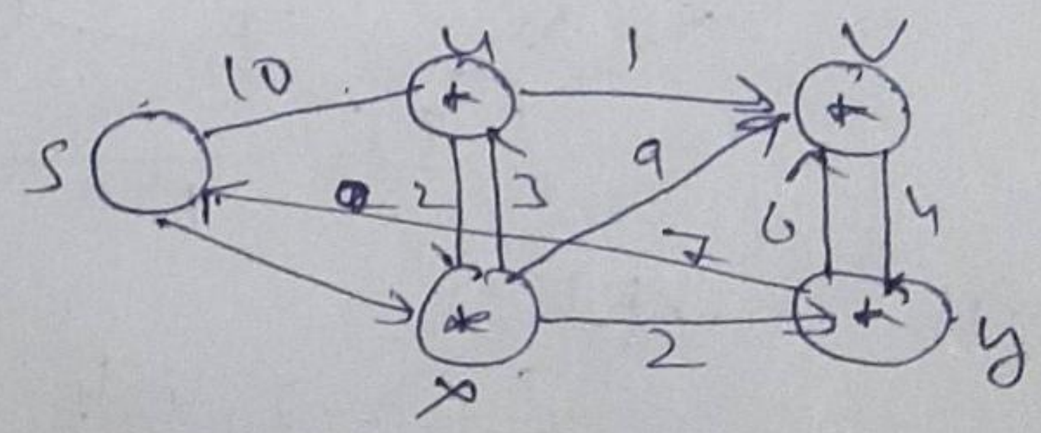


a) If we increase every edge by 10 units then also shortest path is same.

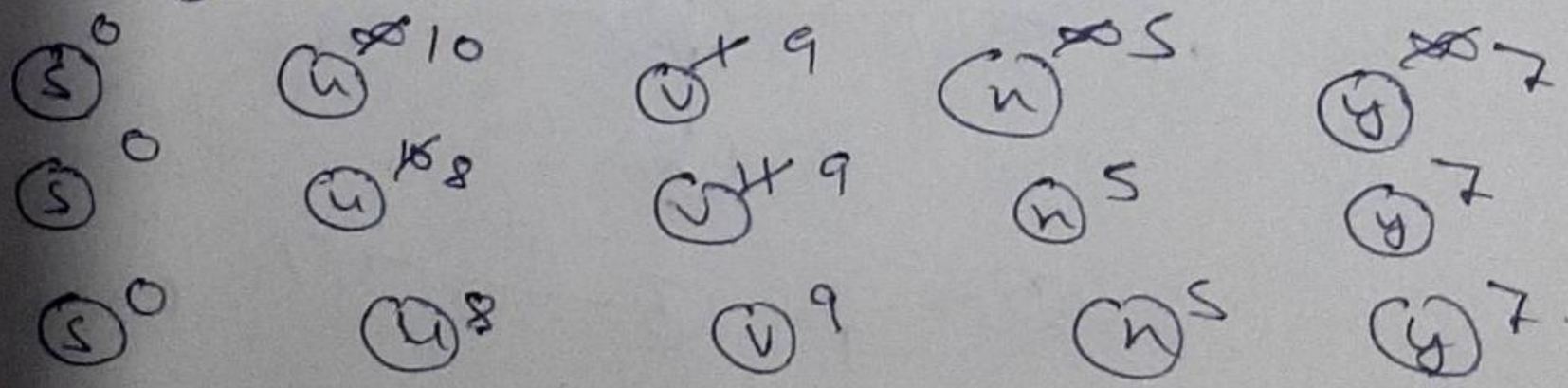
b) If we multiply every edge by 10 units then also the shortest path is same.

Q5
Ans. Dijkstra

node	dist/cost
u	8
v	9
x	5
y	7



Bellmann Ford



Q6

Ans

$$A_0 = \begin{bmatrix} 0 & \infty & 6 & 3 & \infty \\ 3 & 0 & \infty & \infty & \infty \\ \infty & \infty & 0 & 2 & \infty \\ \infty & 1 & 1 & 0 & \infty \\ \infty & 4 & \infty & 2 & 0 \end{bmatrix}$$

$$A_1 = \begin{bmatrix} 0 & \infty & 6 & 3 & \infty \\ 3 & 0 & 9 & 6 & \infty \\ \infty & \infty & 0 & 2 & \infty \\ \infty & 1 & 1 & 0 & \infty \\ \infty & 4 & \infty & 2 & 0 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 0 & \infty & 6 & 3 & \infty \\ 3 & 0 & 9 & 6 & \infty \\ \infty & \infty & 0 & 2 & \infty \\ \infty & 1 & 1 & 0 & \infty \\ \infty & 4 & 13 & 2 & 0 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 0 & \infty & 6 & 3 & \infty \\ 3 & 0 & 9 & 6 & \infty \\ \infty & \infty & 0 & 2 & \infty \\ \infty & 1 & 1 & 0 & \infty \\ \infty & 4 & 13 & 2 & \infty \end{bmatrix}$$

$$A_4 = \begin{bmatrix} 0 & 4 & 4 & 3 & \infty \\ 3 & 0 & 7 & 6 & \infty \\ \infty & 3 & 0 & 2 & \infty \\ \infty & 1 & 1 & 0 & \infty \\ \infty & 3 & 3 & 2 & 0 \end{bmatrix}$$

$$A_5 = \begin{bmatrix} 0 & 4 & 4 & 3 & \infty \\ 3 & 0 & 7 & 6 & \infty \\ \infty & 3 & 0 & 2 & \infty \\ \infty & 1 & 1 & 0 & \infty \\ \infty & 3 & 3 & 2 & 0 \end{bmatrix}$$