Name - Riya Saini
H
11

Tutorial - 1

Q What do you understand by asymptotic notation, Define them with example.

Symptotic notations are used to tell the complexity of an algorithm when input is high

① Big O (n)

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

if $(f(n) \leq g(n) * c \quad \forall n \geq n_0$

for some constant, $c > 0$

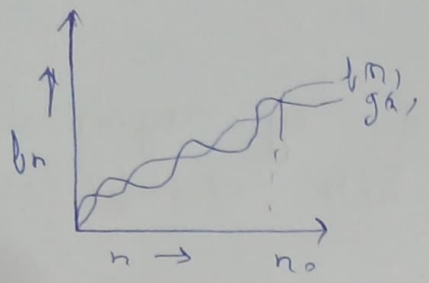
$g(n)$ is 'tight' upperbound of $f(n)$

② Big Omega (Ω)

when $f(n) = \Omega g(n)$ means $g(n)$ is 'tight' lower bound of $f(x)$ i.e $f(x)$ can go beyond $g(n)$ i.e $f(n) = \Omega g(n)$

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

$\forall n2 > n_0$ & c = constant > 0

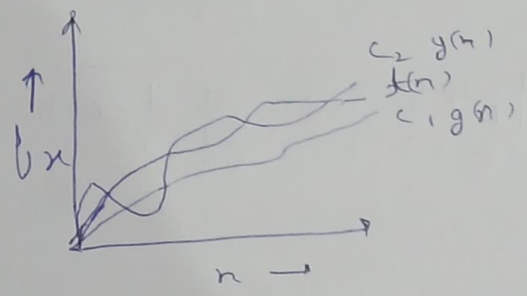


③ Big Thetha(Θ) - when $f(n) = \Theta(g(n))$ gives the tight upper bound & lower bound both i.e $f(n) = \Theta(g(n))$

iff $c_1 * g(n) \leq f(n) \leq c_2 * g(n)$

$\forall n \geq \max(n_1, n_2)$ source constant $c_1 > 0$ & $c_2 > 0$

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

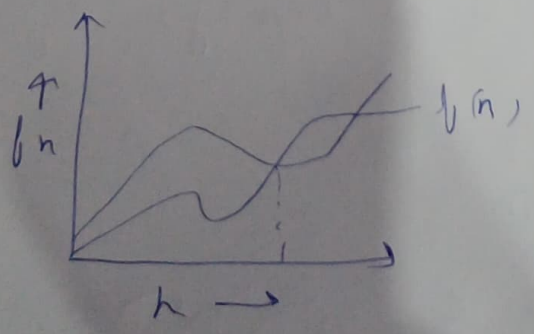


④ Small oh(o)

when $f(n) = og(n)$ gives the upper bound

$f(n) = og(n)$ iff

$f(n) < c * g(n)$ $\forall n > n_0$ & $n > 0$

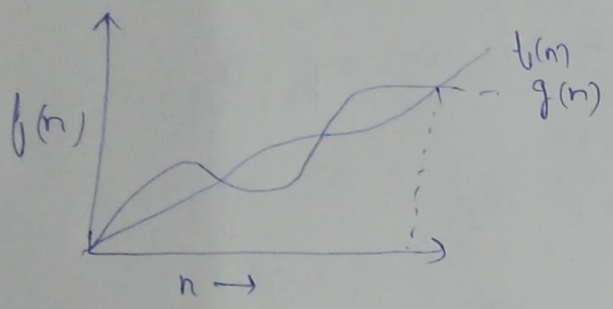


5) Small Omega (ω)

It give the lower bound i.e $f(n) = \omega(g(n))$

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

iff $f(n) > c_n g(n)$ $\forall$ $n > n_0$, $c > 0$



6) for (i=1 to n)

{ i = i * 2; O(1)

}

for i = 1, 2, 4 ... n times

i.e series is a GP

So $a = 1$ $r = 2$

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

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

$2^n = 2^k$

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

$\log_2 2 + \log_2 n = k$

$\log_2 n + 1 = k$

So time complexity $T(n) = O(\log_2 n)$

3]

$T(n) = 3T(n-1)$ — ①

$T(n) = 1$

Put $n = n-1$ in eq ①

$T(n-1) = 3T(n-2)$ — ②

Put value of $T(n-1)$ in eq ①

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

$T(n) = 9T(n-2)$ — ③

Put $n = n-2$ in eq ①

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

Put value of $T(n-2)$ in eq ③

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

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

⋮

So, $T(n) = 3^k T(n-k)$ — ⑤

for k^{th} term, let $n-k=1$ [least eqn]

$k = n-1$

Put $k = n-1$ in eq ⑤

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

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

$= 3^{n-1}$

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

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

Put $n = n-1$

$T(n-1) = 2T(n-2) - 1$ — ②

Put in eq ①

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

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

$T(n) = 4T(n-2) - 3$ — ③

Put $n = n-2$ in eq ①

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

Put in eq ③

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

$T(n) = 8T(n-3) - 4 - 2 - 1$ — ④

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

k^{th} term =>

Let $n - k = 1$

$k = n - 1$

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

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

GP

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

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

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

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

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

5)

$i = 1, 2, 3, 4, 5 \ldots$

$S = 1 + 3 + 6 + 10 + 15 \ldots$

Sum of $s = 1 + 3 + 6 + 10 \ldots n$ ①

Also $S = 1 + 3 + 6 + 10 \ldots T_{k-1} + T_n$ ②

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

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

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

for $k, \; 1 + 2 + 3 + \ldots k <= n$

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

$$\frac{k^2 + k}{2} <= n$$

$$O(k^2) \leq n$$

$$O(k^2) \leq n$$

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

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

Q 6 As $i^2 = n$

$i = \sqrt{n}$

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

$\sum_{i=1} (1 + 2 + 3 + \ldots + \sqrt{n})$

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

$T(n) = O(n)$

Q 7) Since for $n = k^2$

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

$\therefore$ series is in GP

By $a = 1 \quad r = 2$

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

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

$n = 2^k - 1$

$n + 1 = 2^k$

$\log_2(n) = k$

n) (DFS)

thankyou sir

i	j	k
1	log(n)	log(n) + log(n)
2	log(n)	log(n) * log n
⋮	⋮	⋮
n	log(n)	log(n) * log n

T.C = O(n * log n * log n)

= O(n log²(n))

Q6 for(i=1 to n)

we get j = n times every time

∴ i * j = n²

Now,

T(n) = n² + T(n-3);

T(n-3) = (n-3)² + T(n-6)

& T(1) = 1

now, put these values in T(n)

T(n) = n² + (n-3)² + (n-6)² + ... + 1

n - 3k = 1

k = (n-1)/3 total terms k+1

without we leave

Ex:

~~the vice chancellor sir~~
~~traffic~~

$$T(n) = n^2 + (n-5)^2 + (n-6)^2 + \ldots 1$$

$$T(n) \leq kn^2$$

$$T(n) \leq (k-1)\,3n^2$$

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

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

(1) 10

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

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

$\forall\ n \geq n_0$ & constant, $a > 0$

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

$$1^k = a^2$$

$n_0 > 1$ & $c = 2$

Riya Sahni
n
@11

Tutorial - 2

Q1) write time complexity of below code

```
i = 1                    j = 1
i = 2                    i = i + 2
j = 3                    i = j + 2 + 3
```

for (i)

$\therefore 1 + 2 + 3 + \dots + t < n$

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

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

$m \leq \sqrt{n}$

By Summation method

$$\sum_{i=1}^{m} = 1 + 1 + \dots + \sqrt{n} \text{ times}$$

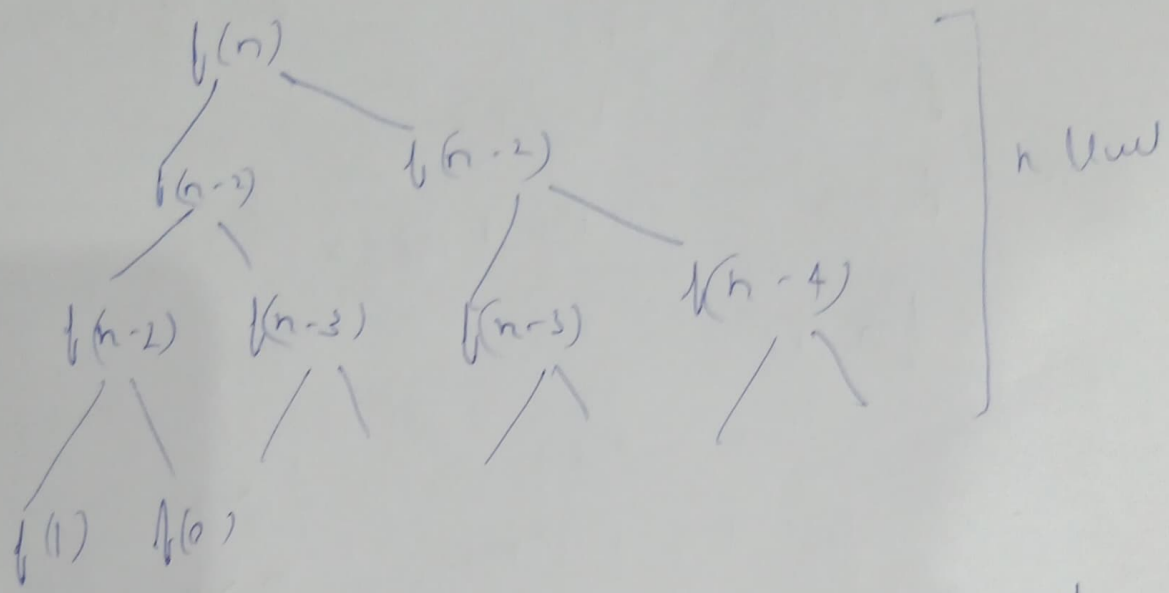
$T(n) = \sqrt{n}$

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

int (int

Q 2) for fibonacci series

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



∴ At every function call we get 2 function calls.

∴ for n levels

we have = $2 \times 2 \cdots$ n times

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

Space complexity

considering recursive

Stack:

no of calls maximum size

for each call we have S.C $O(1)$

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

without considering recursive stack space time complexity is O(1)
without we have
∴ T(n) = O(1)

Q2)

```
1) void quicksort (int arr[], int l, int h)
   {
     if (l < h)
     { int p = partition (arr, l, h);
       quicksort (arr, l, p-1);
       quicksort (arr, p+1, h);
     }
   }
int partition (int arr[], int l, int h)
  { int pivot = arr[h];
    int i = l-1;
    for (j = l; j <= h-1; j++)
    { if (arr[j] < pivot)
      { i++;
        swap (&arr[i], &arr[j])
      }
    }
```

for 2^1

k k

ii) n^3

```
for (i=0; i<n; i++)
  for (j=0; j<<1; j++)
  for (k=0; k<<b; k++)
res[i][j] += arr[i][k] * b[k][j];
```

iii) log (log n)

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

5)

for	i	t
1		1
2		1+3+5
3		1+4+7
:		:
:		i
n		1+5+9

t = (n-1) times

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

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

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

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

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

i where

1 2 $2^k < n$

2^1 $k^m = \log n$

2^{k^2} $m = \log^k \log_2 n$

2^{k^3}

2^{k^n}

$$\sum_{i=1}^{m} 1 + 1 + 1 + \dots \text{ m time}$$

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

Q 6 a) $1 < \log \log n < \log n < (\log n)^2 < \sqrt{n}$
$< \sqrt{n} < n \log n < \log(n!) < 2^n < n^n < 2^{2n}$

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

c) $ab < \log_b n < \log 2n < 5n < n \log_6(n) < n \log_2 n$
$< \log(n!) < 6n^2 < 7n^3 < n^{10} < 6^{2n}$

Tutorial - 3

Name - Riya Saini

1]

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

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--;
        }
        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[i];
        j--;
  }
arr[j+1]= last;
  }
```

Insertion sort is called 'online sort' as it does need to know anything element what value it is next & information is sequential

Other sorting Algo

1) Bubble
2) Quick
3) Merge
4) Selection
5) Heap

Que 6)

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

$$T(n/2) = T(n/4) + 1 \quad - (2)$$

$$T(n/4) = T(n/8) + 1 \quad - (3)$$

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

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

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

$$T(n/2^n) + 1 \quad (k \text{ Times})$$

Let $2^k = n$

$k \log n$

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

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

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

7]

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

Q3)

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

Q4)

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		

$a = 3$, (n) , .

b) Quick Sort is fastest general purpose sort. In most practical situation quicksort is method of choic as stability is imp. & space is available, mergesort might be best

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

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

Q-10] Worst case $O(n^2)$ - The worst case occurs when the pivot element is an extreame (smallest / largest) element This happens when input array is sorted or reversed sorted & either first or last element is selected as Pivot

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

Q 11 Merge sort

Best case → $T(n) = 2T(n/2) + O(n)$

worst case → $T(n) = 2T(n/3) + O(n)$

Quick sort

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 merge element are split into 2 sub array n/2 again until only one element is left.

Q 12)

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

Q 11 Merge sort

Best case → $T(n) = 2T(n/2) + O(n)$

worst case → $T(n) = 2T(n/3) + O(n)$

Quick sort

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 merge element are split into 2 sub array $n/2$ again until only one element is left.

Q 12)

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

Q 11 Merge sort

Best case → $T(n) = 2T(n/2) + O(n)$

worst case → $T(n) = 2T(n/3) + O(n)$

Quick sort

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 merge element are split into 2 sub array $n/2$ again until only one element is left.

Q 12)

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

Tutorial - 4

Riya Soni
G
11

Q1) $T(n) = 3T\left(\frac{n}{2}\right) + n^2$

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

$c = \log_b a = \log_2 3$

$= 1.584$

$n^2 = n^{1.33} < n n^2$

$f(n) > n^c$

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

Q2) $a = 4 \quad b = 2 \quad f(n) = n^2$

$c = \log_2 4 = 2$

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

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

Q3)

$a = 1$

$b = 2$

$f(n) = 2^n$

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

$n^c = n^0 = 1$

$f(n) \geq n^0$

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

Application

Q 4) $a = 2^n$

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

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

$= n$

$n^c = n^2$

$f(n) = n^c$

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

Q 5) $a = 16 \quad b = 4$

$f(n) = n$

$c = \log_4 16 = 2$

$n^c = n^2$

$f(n) < n^c$

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

Q 6) $a = 2 \quad b = 2$

$f(n) = n \log n$

$c = \log_2 2 = 1$

$n^c = n^1 = n$

$n \log n > n$

$f(n) \Rightarrow n^c$

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

only on

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

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

$n^c = n^{0.5}$

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

$f(n) > n^c$

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

Q g) $a = 0.5 \quad b = 2$

$a \not\geq 1$ but here a is 0.5 so we cannot apply Masters Theorem

Q d) $a = 16 \quad b = 4$

$f(n) = n!$

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

$n^c = n^2$

As $n! > n^2$

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

Q 14)

$a = 3 \quad b = 3$

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

$n^c = n^1 = n$

as $\text{sqrt}(n) < n$

$f(n) < n^2$

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

Q 11) $a = 3 \quad b = 3$

$c = \log_b a = \log_3 3$

$= 1$

$f(n) = n/2$

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

As $n/2 < n$

$f(n) < n^c$

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

Q 20) $a = 64 \quad b = 8$

$c = \log_b a = \log_8 64$

$c = 2$

$n^c = n^2$

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

Q 22) $a = 1 \quad b = 2$

$c = \log_b a = \log_2 1$

$= 0$

$n^c = n^c = 1$

$n(1 - \cos x) \Rightarrow n^c$

$T(n) = \Theta(n(1 - \cos x))$

Riya Saini
(DFS)

Tutorial-5

Ans-1

BFS	DFS
i) Uses queue data structure.	i) Uses stack data structure
ii) Stands for Breadth first search	ii) Stands for depth first search
iii) Can be used to find single source shortest path in an unweighted graph, as we reach a vertex with min. no. of edges from a source vertex	iii) we might traverse through more edges to reach a destination vertex from a source
iv) sibling are visited before the children	iv) Children are visited before the sibling
Applications:	Applications:
i) Shortest Path & Minimum spanning Tree for unweighted graph	i) Detecting cycle in a graph
ii) Peer to Peer Networks	ii) Path finding
iii) Social Networking Websites	iii) Topological sorting
iv) GPS Navigation System	iv) Solving puzzles with only one solution.

Ans-2 In BFS we use Queue data structure as queue is used when things do not have to be processed immediately, but have to be processed in FIFO order like BFS.

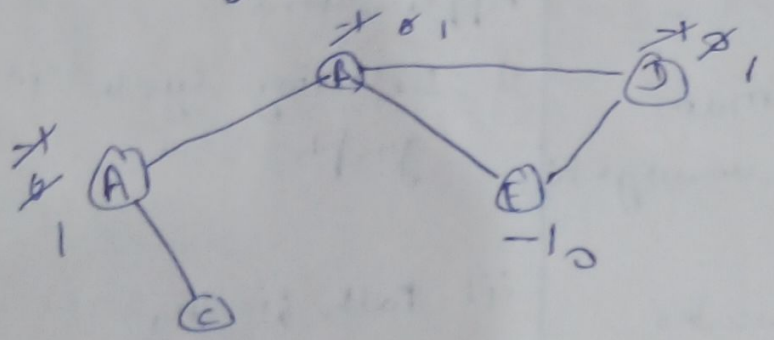
In DFS stack is used as DFS uses backtracking. For DFS we retrieve it from

root to the farthest node as much as possible, this is the same idea as LIFO [used by stack].

Ans 3 Dense graph is a graph in which no. of edges is close to the maximal no. of edges is close to the minimal no. of edges. It can be disconnected graph.

* Adjacency lists are preferred for sparse graph & Adjacency matrix for dense graph.

Ans. 4 Cycle Detection in Undirected graph (BFS)



-1 = unvisited
0 = into the queue (nodes)
1 = traversed.

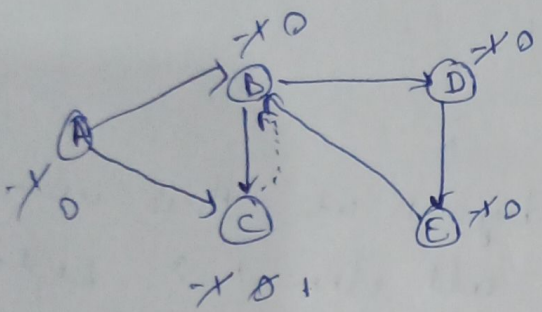
Queue : | A | B | C | D | E |

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

when D checks its adjacent vertics it finds E with 0

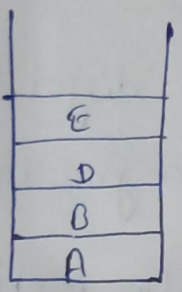
If any vertex finds the adjacent vertex with flag 0, then it contains cycle.

Cycle ~~Destination~~ Detection in Directed Graph (DF



-1 = unvisited
0 = visited & in stack
1 = visited & popped out from stack

Stack :



visite set
ABCDE

Parent Map

Vertex	Parent
A	—
B	A
C	B
D	B
E	D

⇒ B → D → E → B

there E finds B (adjacent vertex of E) with 0.

⇒ it contains a cycle

Ans 5 The disjoint set data structure is also known as union-find data structure & merge-find set. It is a data structure that contains a collection of disjoint set means that when the set is partioned into the disjoint subsets various optins can be performed on it. In this case, we can add new sets, we can merge the sets. It also allows to find out whether the two elements are in the same set

on root efficiency.

Operations on disjoint set.

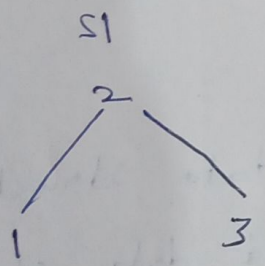
1) Union

a) If S1 & S2 are two disjoint sets, their union S1 ∪ S2 is a set of all element x such that x is in either S1 or S2

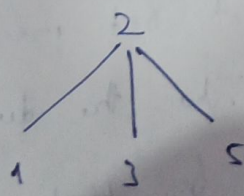
b) As the sets should be disjoint S1 ∪ S2 replaces S1 & S2 which no longer exists.

c) Union is achieved by simply making one of the trees as a subtree of other i.e to set parent field of one of the roots of the trees to other root.

Ex



S1 ∪ S2

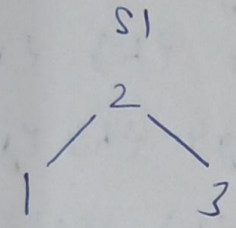


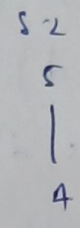
Merg the sets containing X & containing Y into one

2) Find

Given an element x, to find the set containing it:

Ex





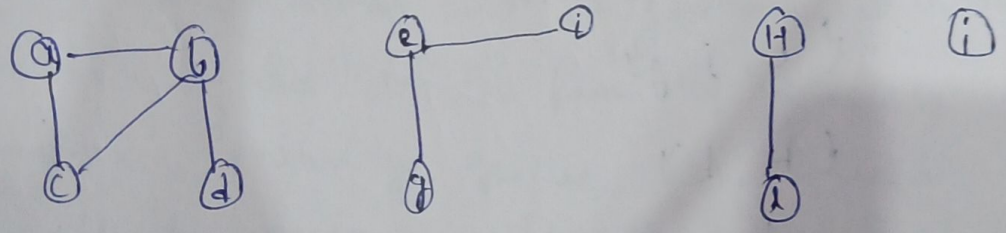
find (3) ⇒ S1

find (5) ⇒ S2

return in which set x belongs

3) Make-set(x) : Create a set containing x.

Ans-7



V = { a, b, c, d, e, g, h, i, j, l }

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

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

we have

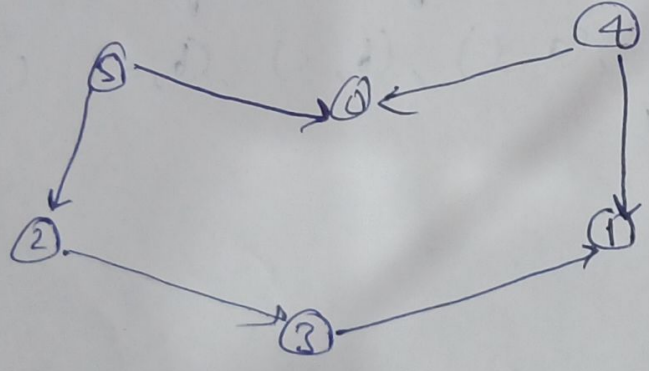
{a, b, c, d}

{e, i, g}

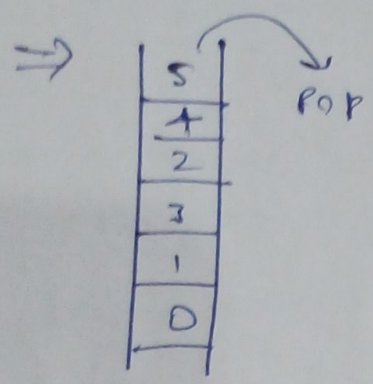
{h, l}

{j}

Ans. B



A

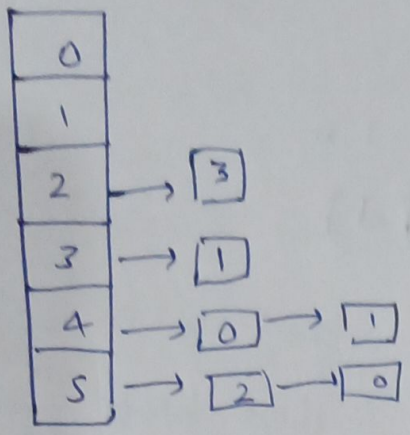


5 4 2 3 1 0
(output)

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

Algorithms where priority queue is used:

1. Dijkstra's shortest Path Algorithm: when the graph is stored in the form of adjacency list or matrix, priority queue can be used to extract minimum efficiently when implementing Dijkstra's algorithm.
2. Prim's algorithm: To store keys of nodes & extract minimum key node at every step



Algo:

1. Go to node 0, it has no outgoing edges so push node 0 into the stack & mark is visited.
2. Go to node 1, again it has no outgoing edges, so push node 1 into the stack & mark it visited.
3. Go to node 2, process all the adjacent nodes & mark node 2 visited.
4. Node 3 is already visited so continue ~~no push with next~~ node. 4 into the stack & mark it visited.
6. Go to node 5, all its adjacent nodes are already visited so push node 5 into the stack & mark it visited

Ans-10

Min Heap	Max Heap
1) For every pair of the parent & descendant child node, the parent node always has lower value than descended child node	1) For every pair of the parent & descendant child node, the parent nod has greater value than descended child node.
2) The value of nodes increases as we traverse from root to leaf node	2) The value of nodes decreases as we traverse from root to leaf node
3) Root node has the lowest value	3) The root node has the greatest value.

TUTORIAL 6

Name - Riya Saini
U
11

Ans 1

Minimum spanning tree: A minimum spanning tree (MST) or minimum weight spanning tree is a subset of the edges of a connected edge-weighted undirected graph that connects all the vertices together, without any cycle and with the minimum possible total edges weights.

Applications

i) Consider n station are to be linked using a communication network and laying of communication link between any two station involved a cost. The ideal solution would be exact a subgraph formed as minimum cost spanning tree.

ii) Suppose you meant to construct highways or railroads spanning several cities then we can use the concept of minimum spanning tree.

iii) Design LAN

iv) Laying pipelines connecting offshore drilling sites, refineries and consume markets.

Ans-2 Time complexity of Prims algorithm

$O((V+E)\log)$

Space complexity of Prim's algorithm $O(V)$.

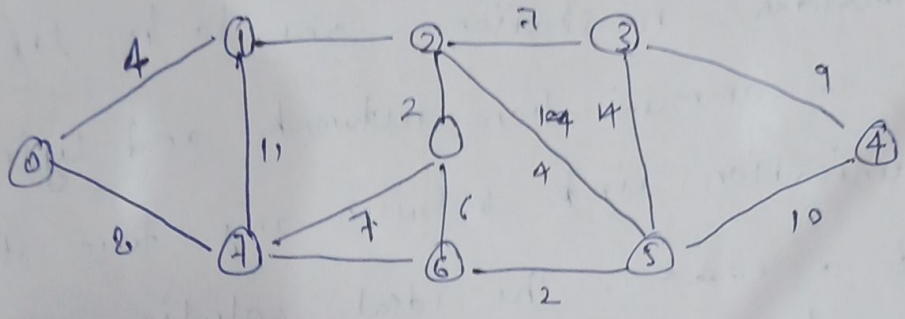
Time complexity of Kruskal's Algo : $O(E(\log V))$

Space complexity of Kruskals Algo : $O(|V|)$

Time complexity of Dijkstra Algo : $O(V^2)$

Space complexity of Bellman ford : $O(E)$.

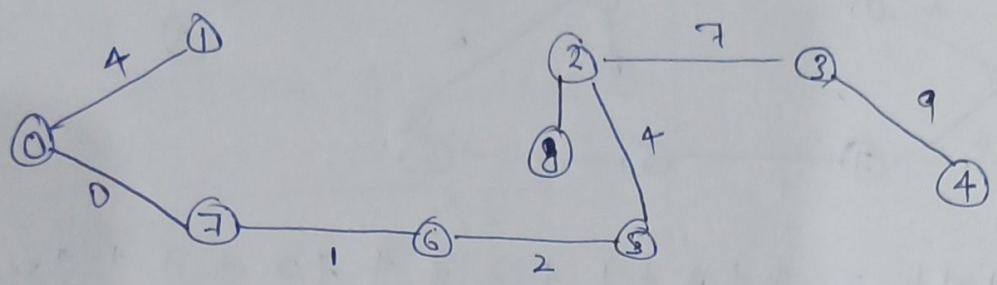
Ans-3



Kruskal Algorithm

0	v	w	
6	7	1	✓
5	6	2	✓
2	8	2	✓
0	1	4	✓
2	5	4	✓
6	8	6	X
2	3	7	✓
7	8	7	X
0	7	8	✓
1	2	8	X
4	3	9	✓
4	5	10	X

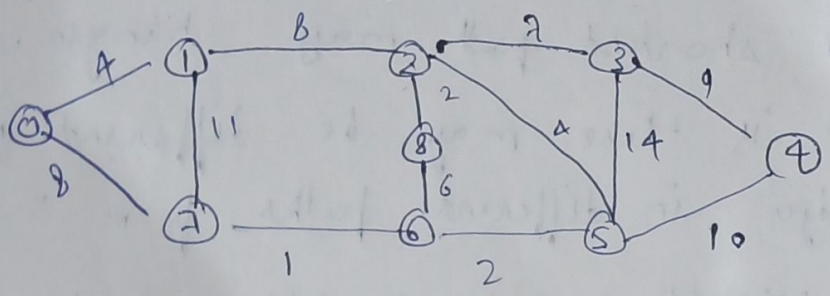
0	v	w	
1	7	11	X
3	5	14	X



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

~~37~~

Prim's Algorithm

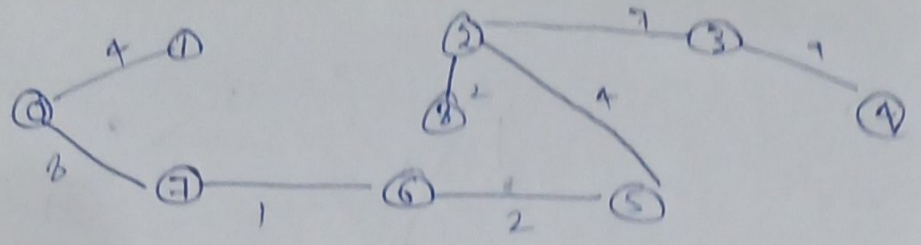


weight :-

0	1	2	3	4	5	6	7	8
[0]	∞	∞	∞	∞	∞	∞	∞	∞
	4						[8]	
		8				[1]		7
			7		4	1		6
			7		2			
	4	14	1	10				
	[7]							
			[9]					

Parent :

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



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

Solution = 4

Ans- 4

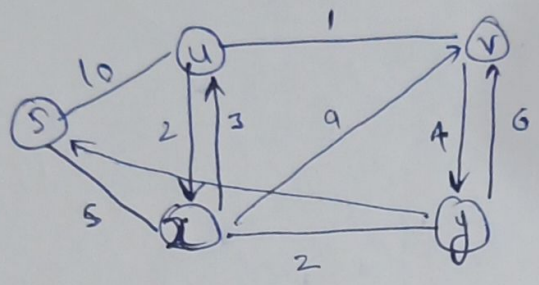
i) The shortest path may changes. The reason is there may be different number of edges in different paths from 's' to 't' for example: Let shortest path be of weight 15 & has edge 5. Let there be another path with 2 edge and total weight 25. The weight of the shortest path increased by 5*10 and become 15 + 50 weight of the other path is increased by 2*10 and become 25 + 20 so the shortest path changes to the other path with weight a, 45

ii) If we multiply all edges weight by 10, the shortest path don't change. The reason is simple, weight of all path from 's' to 't' & multiply by same amount. The no. of edges on a path don't matter. It is like

changes limits of weight

Ans 5

Dijkstra 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 0	→ graph does not have cycle
2nd	s 0	u 10	v 11-1	x 5	y 0	
3rd	s 0	u 8	v 9	x 5	y 7	
4th	s 0	u 8	v 9	x 5	y 7