Amey Thakur B-50

Analysis of Algorithm

Assignment No. 1

23/1/2020

Q.1. List and explain different growth function (types of complexity)

Ans:

Time complexity of algorithm is a measure which evaluates the order of the count of operations performed by a given algorithm. as function of size of input data.

Complexity of algorithms can be constant, O(1), linear O(n), Logarithmic O(log n), quadratic, exponential, etc.

① Constant O(1):

It takes constant no. of steps for performing a given operation. This count does not depend of size of input.

② Linear :

The no. of steps required and no. of input elements are linearly dependent.

Ex - Code containing single loop and index variable varying by increment or decrement factor.

eg.

	s/e	Frequency	Total
for (i=0; i<n ; i++)	1	n+1	n+1
{			
statement;	1	n	n
}			2n+1

$\therefore T(n) = 2n+1$

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

③ Logarithmic :

It takes the order of log (N) steps, where base of log is most often 2. It happens when the looping index is varying by multiplying or dividing factor.

PAGE NO.
DATE / /

Eg.
```
for ( i=0; i<n ; i++)
{
  code ....
  i = i * 2 ;
}
```
$\therefore T(n) = O(\log n)$

Q7. Explain the time complexity notations.

Ans:

The notations used to represent complexity are called asymptotic notations.

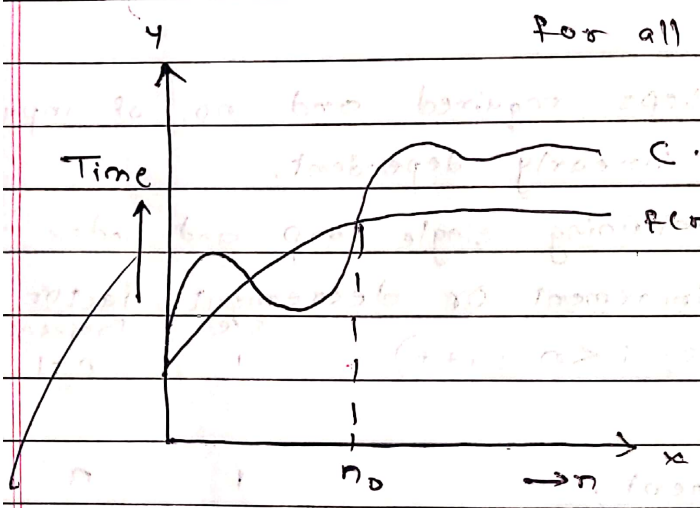
Three types of notations O, Ω, Θ.

① (Big oh) O -

It represents upper bound of a function denoted as

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

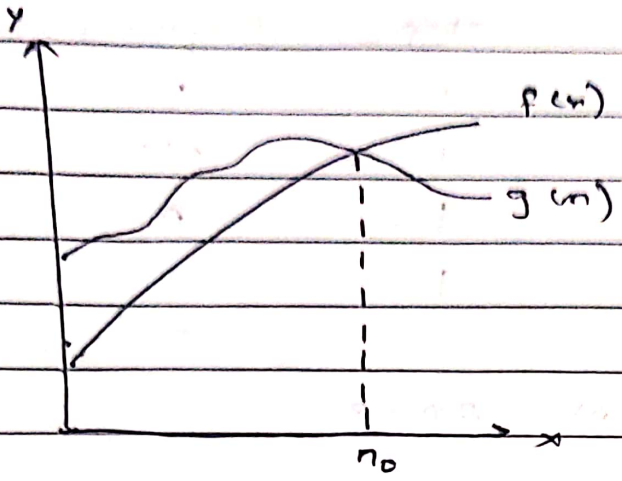
$O(g(n)) = f(n)$ { There exists +ve constants c and n_0 such that $0 \leq f(n) \leq c \cdot g(n)$ for all $n > n_0$ }



② Ω (Omega) -

It represents lower bound of functions.

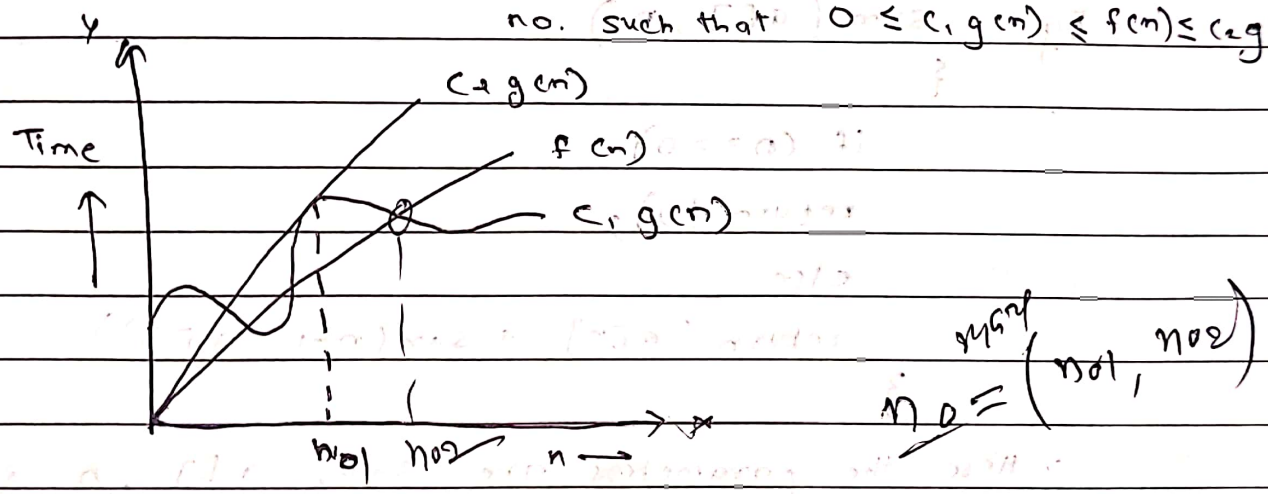
Denoted as $\Omega(g(n)) = f(n)$ { There exists +ve constants such that $c \cdot g(n) \leq f(n)$ for all $n > n_0$



③ Θ (Theta) -

It is used to represent lower upper bound from set f(n). It is denoted as Θ(g(n))

Θ(g(n)) = { f(n) : There exists the constants c_1, c_2 no. such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$



Q.3. What are different ways to derive complexity

Ans:

① Step count method -

In this method we count the no. of times one instruction is executing.

i] Define executing statements as steps.

ii] Count no. of times those steps are executed

iii] Assign one unit to simple statement.

iv] Assign one unit to each iteration

v] Assign one unit to function call

Eg.	step	Frequency	Total
s = 0	1	1	1
for i = 0 to n	1	n+1	n+1
s = s + a[i]	1	n	n
print s	1	1	1
			2n+3

$\therefore T(n) = 2n + 3$

② Recursive method -

This type requires two fields.

i] No. of parameters

ii] No. of recursive calls.

```
Eg.  sum ( a[] , n )
     {
        if (n == 0)
        return (0);
        else
        return ( a[n] + sum (n-1, a[]))
     }
```

∴ Here the parameters are sum, a[] , n = 3,

No. of recursive calls = 3

③ Master method -

```
Eg -  Sum ( int n)
      {
        if n == 1
        return (1)
        else
        return ( n + sum (n-1))
      }
```

∵ Sum of 1 is computed using fixed no. of operations

$T(1) = 1$

If $n > 1$ function will be performed for n no. of times also it will make recursive calls to Sum $(n-1)$

$\therefore T(n) = 1 + T(n-1)$

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

$T(n) = 1 + T(n-1) \quad n > 1$

$\therefore T(n) = 1 + T(n-1)$

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

$= 2 + T(n-2)$

$\Rightarrow 2 + (1 + T(n-3))$

$\Rightarrow 3 + (T(n-3))$

$\vdots$

$\Rightarrow K + T(n-K)$

Q4. Bubble sort

Ans:

	S/e	Frequency	Total
bubble sort (a[], n)	1	1	1
{			
for i = 1 to n-1	1	n	n
{			
for j = 1 to n-i-1	1	$\frac{(n-1)}{(n-1)}$ / 2	$\frac{(n-1)}{(n-1)}$ / 2
{			
if a[j] > a[j+1]	1	$(n-1)$	$(n-1)$
swap (a[j], a[j+1])		$\left(\frac{n-1}{2}\right)$	$\left(\frac{n-1}{2}\right)$
}			
}			
}			

∴ The inner loop is dependent on the outer loop

Frequency $= (n-1) + (n-2) + (n-3) + \ldots + 1$

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

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

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

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

$$= \frac{2n + 2n^2 - 4n + 2}{2}$$

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

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

ii] Binary search

	s/e	Frequency	Total
low ← 1	1	1	1
high ← n	1	1	1
while (low < high)	1	log (n+1)	log (n+1)
mid (low + high) / 2			
if a (mid) = = key	1	log (n)	log (n)
return (mid);			
else if a (mid) < key	1	log (n)	log (n)
low ← mid + 1;			
else			
high ← mid - 1;			
end			
return 0	1	1	1

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

Time Complexity $O(n) = \log N$

For each iteration n is divided by 2.

if loop terminates at 'k'

∴ executed 'k' time

$\therefore \quad n = \frac{n}{2^k}$

$\therefore \quad a + i = k \qquad n = 1$

$n = 2^k$

$\log_2 n = \log_2 2^k$

$= k \log_2 2$

$\log_2 n = k$

$O(n) = \log n$

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

AOA

Assignment 2

6/2/20

Q1. Write Algorithm and derive complexity

Ans:

1] Linear search:

Linear search (Array A, Value x)

S1: set i to 1

S2: if i>n then go to S7.

S3: if A[i] = x then go to S6.

S4: set i to i+1

S5: Go to S2.

S6: Print element x found at index i and go to S8.

S7: Print element not found

S8: Exit.

$$T(n) = \begin{cases} 0 & n < 1 \\ T(n-1)+1 & n > 1 \end{cases}$$

This eqn is recurrence relation of linear search.

$$T(n) = T(n-1)+1$$
$$= [T(n-1-1)+1]+1$$
$$= [T(n-2)+2]$$
$$= [T(n-3)+1]+2$$
$$\vdots$$
$$= T(0)+n \quad \leftarrow \text{for } n^{th} \text{ step}$$
$$T(n) = 0+n$$
$$T(n) = O(n) \quad \therefore \text{Linear Complexity}$$

Program:

```
#include <stdio.h>
#include <conio.h>
void search (int a[] , int n, int p)
{
   int i;
   for(i=0 ; i<n; i++)
   if (a[i] == x)
    return i;
   return -1;
}
int main (void)
{
   int a[] = {2,3,4,5,10,40}
   int n=10;
   int n = sizeof (a) / sizeof (a[0]);
   int result = search (a, n, x);
   (result == -1) ? print ( "Element is not present"):print("Element is
                    present at index %d", result);
   return 0;
}
```

ii] Binary search

S1: START

S2: a[0] = low, a[n-i] = high

S3: while (low <= high) find mid = (low + high)/2;

S4: Compare if (search = a[mid])

return (mid);

break;

```
else if (search > a[mid])
    low = a[mid]+1;
else
    high = a[mid]-1;
    print (a[mid])
55: Print (absent)
56: END
```

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/2) + 1 & \text{otherwise} \end{cases}$$

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

$$= T\left(\left(\frac{n}{4} + 1\right)\right) + 1$$

$$= T\left(\frac{n}{8} + 1\right) + 2$$

$$T(n) = T\left(\frac{n}{8}\right) + 3$$

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

$$= \vdots$$

$$= T\left(\frac{n}{2^k}\right) + k \quad \ldots \text{ in } k^{th} \text{ step}$$

$$= T(1) + k$$

$$= 1 + k$$

$$k = \log n$$

$$1 + \log_2 n$$

$$T(n) = O(\log_2 n)$$

Assume $n = 2^k$; Taking log on both the sides

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

$\log_1^n = k(\log_0^2)$

$\log_2 n = k(1)$

$k = \log_2 n$

Program:

```
#include <stdio.h>
int binary search (int arr[], int l, int r, int x)
{
 if (r >= l) {
 int mid = l + (r-1) /2;
 if (arr [mid] == x)
 return mid;
 if (arr [mid] > x)
 return binary search (arr, l, mid-1, x);
 return binary search (arr, mid+1, r, x);
 }
 return -1;
 }
int main (void)
{
int arr [] = { 2,3,4,10, 40};
int n = size of (arr) / size of (arr [0]);
int x = 10;
int result = binary search (arr, 0, n-1, x);
(result == -1) ? printf ("Element is not present")
 : printf ("Element is present at index %d", result);
 return 0;
}
```

Q2. Algorithm for selection sort for given n numbers

Ans:

S1: Set min to location 0

S2: Search the minimum element in the list

S3: Swap with value at location min

S4: Increment min to point to next element

S5: Repeat until list is sorted

```
void selectionsort (int arr[], int start)
{
 if (start >= array.length -1)
 return;
 int min.index = start;
 for(int i = start +1 ; i<arr.length ; i++)
 {
  if (arr[i] < arr [mid index])
   min index = index;
 }
 int temp = arr [start];
 arr [start] = arr [min index];
 arr [min index] = temp;
  selectionsort (arr, start +1);
 }
```

Time complexity

$T(n) = O(n^2)$ ∴ Quadratic complexity

Modified bubble sort

```
#include <stdio.h>
void swap (int *b, int *c)
{
 int x;
 x = *b;
 *b = *c;
 *c = x;
}
void main ( )
{
 int a[10], i, j, flag;
 printf ("Enter the unsorted element");
 for (i=0; i<5; i++)
 scanf ("%d", &a[i]);
 for (i=0; i<5; i++)
 {
    flag = 0;
  for (j=0; j<4-i; j++)
  {
    if (a[j] > a[j+1])
    {
      flag = 1;
      swap (&a[j], &a[j+1]);
    }
  }
 if (flag == 0)
   break;
}
printf ("Sorted elements : ");
for (i=0; i<5; i++)
printf ("%d", a[i]);
}
```

Complexity :

i) $T(n) = O(1)$ is complexity for best case

ii) $T(n) = O(n^2)$ is complexity for worst case.

Q3. Merge sort

Merge sort is a sorting algorithm which commonly used in computer. Merge sort is a divide and conquer algorithm. It works by recursively breaking down the problem into two or more sub program of the same or related types, until these becomes simple enough to be solved directly.

The solution to the sub problem are then combined to give a solution to the original problem. So merge sort first divides the array into equal halves and then combined them into sorted manner.

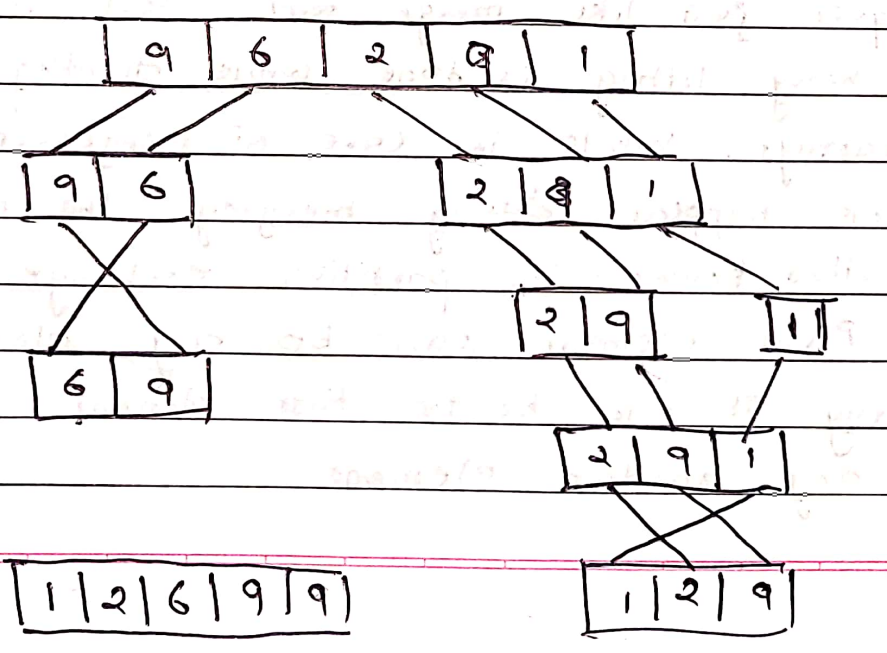
Algorithm :

i] If it is only one element in the list, It is already sorted, return.

ii] Divide the list recursively into two halves until it can no more divided.

iii] Merge the smaller list into new list in sorted order.

Eg



Time complexity

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

Solving using master method

$f(n) = n$

$a = 2$

$b = 2$

$n \log_b a = n (\log_2 2)^1 = n$

$\therefore n \log_b a = n$

$\therefore f(n) = n$

Compare $f(n)$ & $n \log_b a$

$n = n$

$\therefore$ This eq^n falls in case 2.

Complexity will be $T(n) = \Theta(n \log n)$

Q4. Quick sort

Ans:

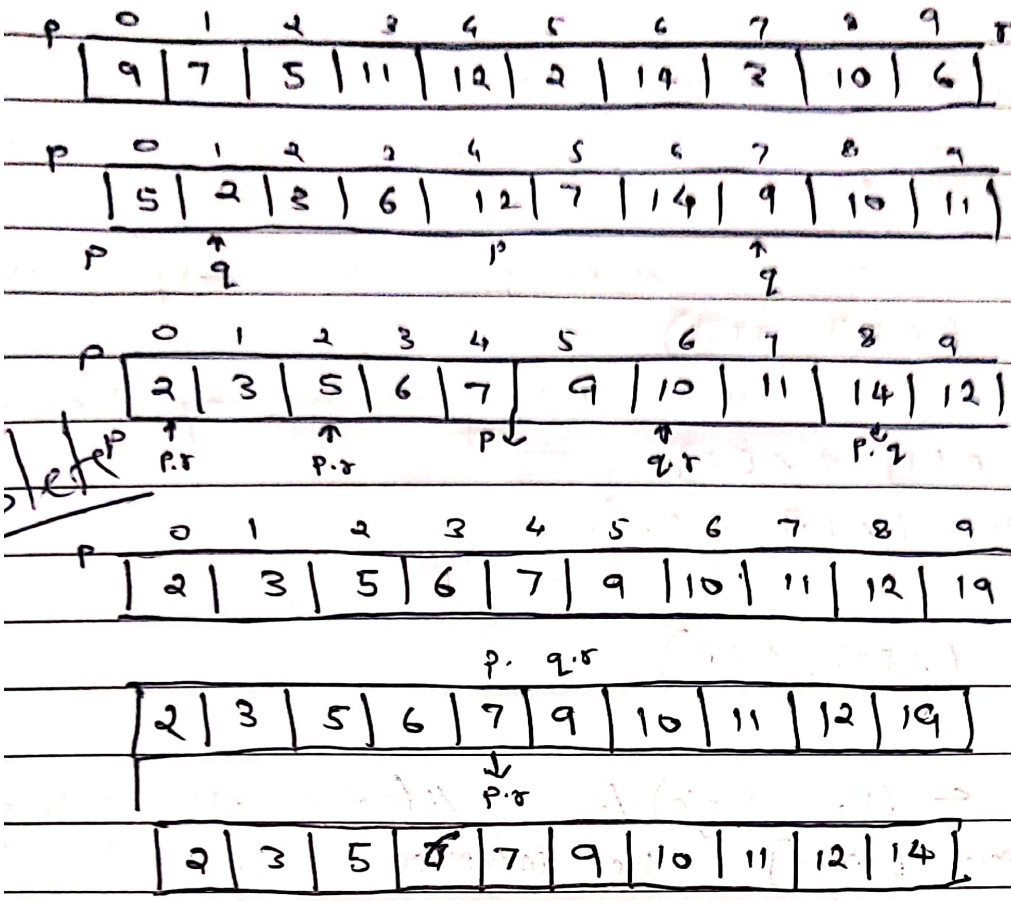
Quick sort is based on the concept divide and conquer., just like merge sort. But in quick sort all the heavy lifting is done while dividing the array into subarrays. While in case of merge sort, all the real work happens during merging the sub array

Also known as partition exchange sort.

Pivot element can be any element from the array it can be the first element, the last element of any random element.

Let's consider array

{ 9, 7, 5, 11, 12, 2, 14, 3, 10, 6 }



Time complexity for best case

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

$F(n) = n$

$a = 2$

$b = 2$

Find $n^{\log_b a}$

$= n^{\log_b a}$

$= n$

$F(n) = n^{\log_b a}$

$n = n$

Eqn fall in case 2, so time complexity is

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

AOA

Assignment - 3

6/2/20

Q1. Algorithm for minmax :

// Input : array A of length n and indices low = 0 & high = n-1

```
if n == 1    then    // Array contains only 1 element
return (A[i], A[i])
else if n == 2    then // Array contains only 2 element
    if A[1] < A[2]
    return (A[1], A[2])
else
  return (A[2], A[1])
  else
    mid ← (low + high) /2    // (solve left sub problem)
    [L min, L max] = DC_MAXMIN (A, low, mid)
    [R min, R max] = DC_MAXMIN (A, mid+1, high) →
                        // Solve right sub problem
If Lmax > R max    then    // Combine soln.
  max ← L max
else
  Max ← R max
end
 if  L min < R min    then // Combine soln.
    min ← L min
 else
    min ← R min
 end
 return (min, max)
 end
```

Complexity Analysis

The conventional algorithm takes $2(n-1)$ comparisons in worst, best and average case

DC-MAX. MIN does two comparison to determine Min and Max element and creates two problem of size $n/2$, so the recurrence can be stated as.

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 1 & \text{if } n = 2 \\ 2T\left(\frac{n}{2}\right) + 2 & \text{if } n > 2 \end{cases}$$

Now,

$$T(n) = 2T\left(\frac{n}{2}\right) + 2 \quad \text{— (1)}$$

Substituting n by $\frac{n}{2}$ in eq^(1)

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + 2$$

$$T(n) = 2\left[2 + \left(\frac{n}{4}\right) + 2\right] + 2$$

$$= 4T\left(\frac{n}{4}\right) + 4 + 2 \quad \text{— (2)}$$

By substituting n by $\frac{n}{4}$ in eq^(1) $T(n) = 2T\left(\frac{n}{8}\right) + 2$

Now, substitute in (2)

$$\therefore T(n) = 4\left[2T\left(\frac{n}{8}\right) + 2\right] + 4 + 2$$

$$= 8T\left(\frac{n}{8}\right) + 8 + 4 + 2$$

$$= 2^3 T\left(\frac{n}{2^3}\right) + 2^3 + 2^2 + 2^1$$

⋮

After k-1 interations

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

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

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

Let $n = 2^k \rightarrow 2^{k-1} = \left(\frac{n}{2}\right)$

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

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

For $n=2$, $T(n) = 1$ (Two elements require only 1 comparison to determine min max)

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

Hence, it is observed that D & C approches does only $\left(\frac{3n}{2} - 2\right)$ comparisons compared to $2(n-1)$ comparisons by conventional approach

Amey Thakur B-60

Q2. Algo for Strassen's Matrix multiplication

```
SMAT_MUL (int *A, int *B, int *C, int n)

// A & B are i/p matrices
// C is the output matrix
// All matrices are of size n x n

if n == 1 then
  *C = *C + (*A) * (*B)
else
SMAT_MUL (A, B, C, n/4)
SMAT_MUL (A, B + (n/4), B, C + 1 * (n/4), n/4)

SMAT_MUL (A + 2 * (n/4), B + (n/4), C + 2 * (n/4),
n/4)
SMAT_MUL (A + (n/4), B + 2 * (n/4), C, n/4)

SMAT_MUL (A + (n/4), B + 3 * (n/4), C + (n/4), n/4)

SMAT_MUL (A + 3 * (n/4), B + 2 * (n/4), C + 2 * (n/4),
n/4)
SMAT_MUL (A + 3 * (n/4), B + 3 * (n/4), C + 3 * (n/4),
n/4)

end
```

⟹ Complexity Analysis:

Conventional approach performs eight multiplications to multiply two matrices of size 2 x 2 matrices using addition. Hence, to solve problem of size n, Strassen's approach creates 7 problems of size (n-3)

Amey Thakur B-50

PAGE No.
DATE / /

Recurrence relation for this approach is

$$T(n) = 7T\left(\frac{n}{2}\right)$$

Two matrices of size 1x1 needs only 1 multiplication So base would be, $T(1) = 1$

Let's find solution using iterative approach by substituting $A = \left(\frac{n}{2}\right)$ in eq^n above.

$$T\left(\frac{n}{2}\right) = 7 \cdot T\left(\frac{n}{4}\right)$$

$$\therefore T(n) = 7^2 \cdot T\left(\frac{n}{2^2}\right)$$

$$\vdots$$

$$T(n) = 7^k \cdot T\left(\frac{n}{2^k}\right)$$

Let's assume

$$n = 2^k$$

$$T(2^k) = 7^k T\left(\frac{2^k}{2^k}\right) = 7^k \cdot T(1) = 7^k = 7^{\log_2 n}$$

$$= n \cdot \log_2 7 = n^{2.81} < n^3$$

Thus, running time of Strassen's multiplication algo is $O(n^{2.81})$, which is less than traditional approch

Q.3 $$T(n) = \begin{cases} 1 & n = 1 \\ T(n/2) + cn & n > 1 \end{cases}$$

$$\therefore T(n) = T(n/2) + cn$$

$$= \left(T\left(\frac{n}{4}\right) + \frac{cn}{2}\right) + cn$$

$= T\left(\frac{n}{8}\right) + \frac{cn}{4} + \frac{cn}{2} + cn$

$= T\left(\frac{n}{2^3}\right) + cn\left(\frac{1}{2}\right)^2 + cn\left(\frac{5}{2}\right) + cn$

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

$\ldots k^{th}$ step.

$= 1 + cn\left[\left(\frac{1}{2}\right)^{k-1} + cn\left(\frac{1}{2}\right)^{k+2} + \ldots + \left(\frac{1}{2}\right)\right]$

$= 1 + cn\left[\frac{(1/2)^{k-1}}{1/2 - 1}\right]$

$= 1 - 2\,cn\left[\left(\frac{1}{2}\right)^{k-1}\right]$

$= 1 - 2\,cn\left[\left(\frac{1}{2}\right)^{\log_2 n} - 1\right]$

$= 1 - 2\,cn\left[n^{\log_2 1/2} - 1\right]$

$= 1 - 2cn\left[n^{\log 1 - \log 2} - 1\right]$

$= 1 - 2\,cn\left[n^{0-1} - 1\right]$

$= 1 - 2\,cn\left[\frac{1}{n} - 1\right]$

$= 1 - 2c + 2n$

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

DATE / / /

Q4. $T(n) = \begin{cases} d & n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & n > 1 \end{cases}$ [Merge sort]

$T(n) = 2T\left(\frac{n}{2}\right) + cn$

$= 2\left(2T\left(\frac{n}{4}\right) + \frac{cn}{2}\right) + cn$

$= 4T\left(\frac{n}{4}\right) + cn + cn$

$= 2\left[4T\left(\frac{n}{8}\right) + \frac{cn}{4}\right] + cn$

$= 8T\left(\frac{n}{8}\right) + 3cn$

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

$\vdots$

$= 2^k T\left(\frac{n}{2^k}\right) + k\,cn$

$= n \cdot d\ T(1) + \log_2 n\ cn$

$= nd + cn \cdot \log_2 n$

$\therefore T(n) = O(n \log_2 n)$

Assignment (2)

Quicksort :

Eg. [50, 31, 71, 38, 77, 81, 12, 33]

Fix pivot, low, high

until A[low] > A[pivot] and A[high] ≤ A[pivot]

if low < high then swap A[low] and A[high]

if low ≥ high then swap A[pivot] and A[high]

① | 50 | 31 | 71 | 38 | 77 | 81 | 12 | 33 |

high

② | 50 | 31 | 33 | 38 | 77 | 81 | 12 | 71 |

pivot low high

③ | 50 | 31 | 33 | 38 | 77 | 81 | 12 | 71 |

high

④ | 50 | 31 | 33 | 38 | 77 | 81 | 12 | 71 |

⑤ | 50 | 31 | 33 | ~~38~~ | 12 | 81 | 77 | 71 |

low

⑥ | 50 | 31 | 33 | 38 | 12 | 81 | 77 | 71 |

⑦ | 50 | 31 | 33 | 38 | 12 | 81 | 77 | 71 |

Pivot j low high

left sublist Right sublist

⑧ | 12 | 31 | 33 | 38 | 50 | 81 | 77 | 71 |

⑨ Left sublist to be sorted

12	31	33	38

Pivot low high

Right sublist to be sorted

81	77	71

Pivot low high

⑩

12	31	33	38

Pivot low high

81	77	71

Pivot high low

⑪

12	31	33	38

Pivot low high

71	77	81

Pivot high low

⑫

31	33	38

Pivot low high

71	77

Pivot high low

⑬

31	33	38

Pivot low high

71	77

Pivot low high

⑭

33	38

Pivot low high

~~Sorted list is~~

⑮

33	38

pivot low high

1

Sorted list is

12	31	33	38	50	71	77	81

AOA

Assignment 4

17.02.2020

07/02/ MONDAY

Q1. Write an abstract solution of greedy problem and also list applications of greedy method.

Ans:

Abstract solution:

```
Greedy (a[], n)
{
  Solution = empty ;
  for i = 1 to n :
  {
    x = select (a[i])
    feasible ( solution, x)
    solution = Union (solution, x)
  }
}
```

- A greedy method is the most straight forward design technique.
- It can be apply to a wide variety problems.
- most of the problems having 'n' input and requires to obtain a subset that satisfy some constant.

Applications of greedy approach:

- Greedy algorithms are used to find an optimal solution to many real life problems.

1. Machine scheduling
2. Container loading
3. Knapsack problem
4. Job sequencing with deadlines

DATE / /

⑤ Minimum cost spanning Tree
⑥ Optimal storage on tape
⑦ Optimal merge pattern
⑧ Huffman Code generation

Q2. Explain knapsack problem with example.

Ans:

Knapsack problem

- No. of object. i.e. n and weight i.e w_i and profit i.e p_i for knapsack of capacity M where i is vary from 1 to n
- Using knapsack problem find the subset $\{x_i\}$ which satisfy the given constrains. Where $0 \le x_i \le 1$ all subset that satisfy the given constrains are coalled feasible solution
- A subset that maximize the objective of function is called optimal solution. i.e Feasible solution

① $\sum_{i=1}^{n} p_i x_i$ must be maximum

② $\sum_{i=1}^{n} w_i x_i \le$ Capacity M

③ $1 \le i \le n$ & $0 \le x_i \le 1$

Observation:

① In case, sum of the all weight are $w_i \le x_i = 1$ for $\forall i = 1$ to n.

② All optimal solution will fill the knapsack exactly (upto the capacity M)

Example -

$n = 3 \qquad m = 20$

$(P_1, P_2, P_3) = (25, 24, 50)$

$(w_1, w_2, w_3) = (18, 15, 10)$

Soln:

① As per sequence,

$(x_1, x_2, x_3) = (1, 2/15, 0)$

$\Sigma p_i x_i = 25 \times 1 + 24 \times 2/15 + 50 \times 0 = 28.2$

$\Sigma w_i x_i = 18 \times 1 + 15 \times 2/15 + 10 \times 0 = 20$

② Weight as descending order

$(x_1, x_2, x_3) = (1, 2/15, 0)$

$\Sigma p_i x_i = 25 \times 1 + 24 \times 2/15 + 50 \times 0 = 28.2$

$\Sigma w_i x_i = 18 \times 1 + 15 \times 2/15 + 10 \times 0 = 20$

③ Profit as descending order

$(x_1, x_2, x_3) = (5/9, 0, 1)$

$\Sigma p_i x_i = 25 \times 5/9 + 24 \times 0 + 50 \times 1 = 32.8$

$\Sigma w_i x_i = 18 \times 5/9 + 15 \times 0 + 10 \times 1 = 20$

④ Each object half in size

$(x_1, x_2, x_3) = (1/2, 1/2, 3/10)$

$\Sigma p_i x_i = 25 \times 1/2 + 24 \times 1/2 + 50 \times 3/10 = 39.5$

$\Sigma w_i x_i = 18 \times 1/2 + 15 \times 1/2 + 10 \times 3/10 = 19.5$

⑤ Weight as ascending order

$(x_1, x_2, x_3) = (0, 2/3, 1)$

$\Sigma p_i x_i = 25 \times 0 + 24 \times 2/3 + 50 \times 1 = 66$

$\Sigma w_i x_i = 18 \times 0 + 15 \times 2/3 + 10 \times 1 = 20$

⑥ Profit as ascending order

$(x_1, x_2, x_3) = (0, 2/3, 1)$

$\Sigma p_i x_i = 25 \times 0 + 24 \times 2/3 + 50 \times 1 = 30$

$\Sigma w_i x_i = 18 \times 0 + 15 \times 2/3 + 10 \times 1 = 20$

⑦ Object in descending order of p_i / w_i

$(x_1, x_4, x_5) = (0, 1, 1/2)$

$\Sigma p_i x_i = 25 \times 0 + 24 \times 1 + 50 \times 1/2 = 49.33$

$\Sigma w_i x_i = 18 \times 0 + 15 \times 1 + 10 \times 1/2 = 20$

⑧ gives maximum profit

∴ Optimal solution

Amey Thakur B-50

Q3. Explain job sequencing with deadline

Ans:

- There are 'n' jobs to be processed on a machine
- Each job 'i' has a deadline $d_i \geq 0$ and profit $p_i > 0$.
- Profit is earned if and only if the job is completed by its deadline
- The job is completed if it is processed on a machine for a unit time
- Only one machine is available for processing jobs
- Only one job is processed at a time on machine
- A feasible solution is a subset of jobs J such that each job is completed by its deadline
- An optimal solution is a feasible solution with maximum profit value.

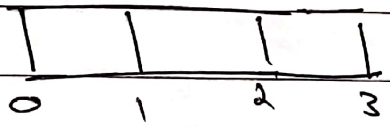
Example -

Job	→	1	2	3	4	5	6
Profit	→	20	15	10	7	5	3
Deadline	→	3	1	1	3	1	3

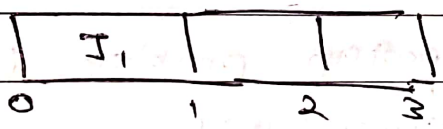
Soln:

① The given profit value of job is already is descending order.

② select maximum deadline D = 3



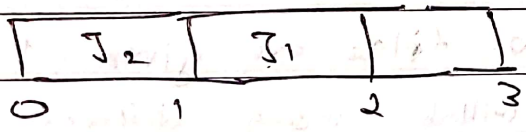
③ Select job J_1 = 20, D = 3



P = 20

D = 3

④ Select job J2 = 20, D = 1



P = 15	P = 20
D = 1	D = 3

⑤ Reject J3

⑥ Select job J4 : 7 , D : 3

P : 15	P : 20	P : 7
D : 1	D : 3	D : 3

Total profit = 15 + 20 + 7

= 42

Job sequence = $J_2 - J_1 - J_4$

Q4. Explain optimal merge pattern and also gives its complexity

Ans:

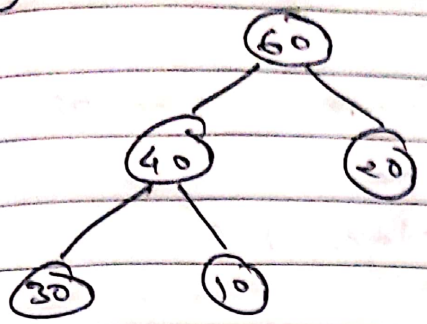
- An optimal merge pattern problem is used to merge files in one in optimal way
- We are given 'n' no. of sorted file that to be sorted in one sorted file
- When more than 2 sorted files are merged together then total time required to merge will be the size of the all merge files
- When 'n' no. files are given then to be merge in one their will be wong different way of merging

Example -

There will be 3 sorted files. Merge in one, Find feasible solution and optimal solution

Input of file size: 30, 10, 20

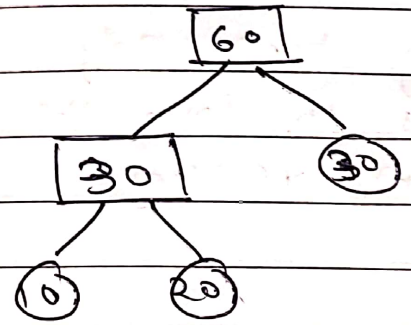
(1)



60
+ 40
100

Total comparison = 100

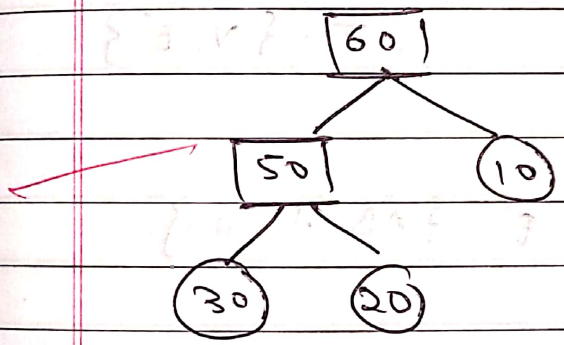
(2) Take size in ascending order 10, 20, 30



60
+ 30
90

Total comparison = 90

(3) Descending order. 30, 20, 10



60
+ 50
110

Total comparisons = 110

- All above are feasible solution
- (2) is the optimal solution

Time complexity

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

DATE / /

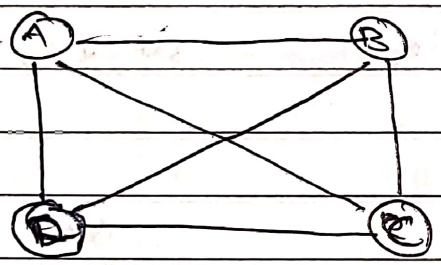
Q5. Explain minimum cost spanning tree

Ans:

- Spanning Tree $T = (V', E')$ is a tree of connected, undirected, weighted graph $G = (V, E, W)$ or all edges of G.
  So $V' = V$ and $E' = E$.
  Graph G can have many spanning tree with a different cost.
- Minimum Spanning Tree is a spanning Tree with minimum cost
- It is called spanning tree only when it contains n-1 edges to form the tree for a graph g(n) vertices

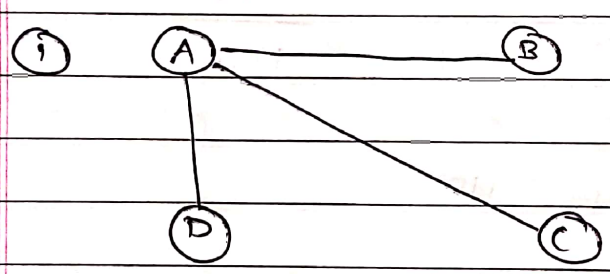
Example -

$V = \{A, B, C, D\}$



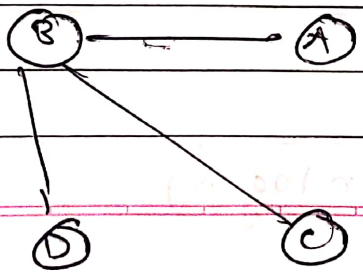
$G\{V, E\}$.

$T = \{V, E'\}$

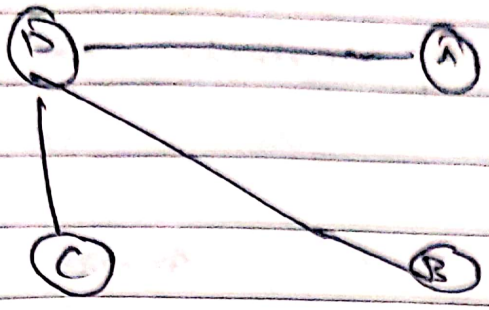
(1) 

$E' = \{AB, AC, AD\}$

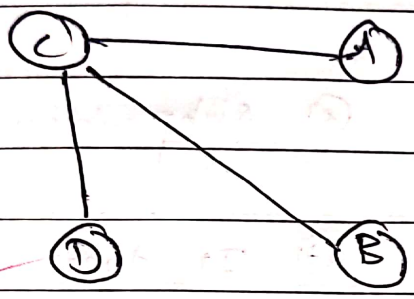
(2) $E' = \{BA, BC, BD\}$



③ E' = { DA, DB, DC }



④ E' = { CA, CB, CD }



Applications:

① Minimum spanning Tree is used in network design

② It is used in implementation of efficient routing algorithm

③ It is used to solve travelling salesman problem

Amey Thakur B-50

PAGE No.
DATE / /

Assignment 5

Q1. Differentiate between greedy methods and dynamic approach.

Dynamic Programming	Greedy Approach
① It guarantees optimal solution.	① It does not guarantee an optimal solution
② Subproblems overlap	② Subproblems don't overlap.
③ It does more work	③ It does little work
④ It considers the future choices	④ Only considers the current choice
⑤ There is no specialized set of feasible solutions	⑤ Construct the solution from set of feasible solutions
⑥ Select choice which is globally optimum	⑥ Select choice which is locally optimum.
⑦ Employ memorization	⑦ There is no concept of memorization

Q2. Write note on 0/1 knapsack. Explain with an example

- This problem is similar to knapsack problem in greedy approach, except the values of x_i can be either 0 or 1.
- The condition for knapsack problem are

① $\sum_{i=1}^{n} w_i x_i \leq m$

② $\sum_{i=1}^{n} p_i x_i$ must be maximum

③ $1 \leq i \leq n$, $x_i = 0$ or $x_i = 1$

- Using the principle of optimality, solution of knapsack of capacity m is

$$g_0(m) = \max \{ g_1(m), g_1(m - w_1) + p_1 \} \quad - ①$$

This is a solution for knapsack $(1, n, m)$

- Replace this function when we are starting from $(j+1)$ index to 1.

Knapsack $(j+1, n, m)$

$$g_j(m) = \max \{ g_{j+1}(m), g_{j+1}(m - w_{j+1}) + p_{j+1} \}$$

- Eq^ ① is a recursive sol^ for knapsack $(1, n, m)$

To solve eq^n. Remember following instructions

$g_n(y) = 0$ for all $y > 0$

$g_n(y) = \infty$ for all $y < 0$

eg. For 0/1 knapsack $n = 4$, $m = 21$

$(w_1, w_2, w_3, w_4) = (10, 15, 6, 9)$

$(p_1, p_2, p_3, p_4) = (2, 5, 8, 1)$

Find $g_0(21)$

$g_0(21) = \max \{ g_1(21), g_1(21-10) + 2 \}$

$= \max \{ g_1(21), g_1(11) + 2 \}$ - (1)

$g_1(21) = \max \{ g_2(21), g_2(6) + 5 \}$ - (2)

$g_2(21) = \max \{ g_3(21), g_3(15) + 8 \}$ - (3)

$g_3(21) = \max \{ g_4(21), g_4(12) + 1 \}$ - (4)

$= \max \{ 0, 0 + 1 \}$

$= \max \{ 0, 1 \}$

$= 1$

$g_3(15) = \max \{ g_4(15), g_4(6) + 1 \}$

$= \max \{ 0, 0 + 1 \}$

$= \max \{ 0, 1 \}$

$= 1$

Putting the values of $g_3(21)$ & $g_3(15)$ in eq^n 3

$\therefore g_3(21) = \max \{ 1, 1 + 8 \}$

$= 9$

$g_2(6) = \max \{ g_3(6), g_3(4) + 8 \}$ — ①

$g_3(6) = \max \{ g_4(6), g_4(3) + 1 \}$

$= \max \{ 0, -\infty + 1 \}$

$= 0$

$g_3(0) = \max \{ g_4(0), g_4(-9) + 1 \}$

$= \max \{ 0, -\infty + 1 \}$

$= 0$

$\therefore g_2(6) = 8$

$g_1(11) = \max \{ g_2(11), g_1(-4) + 5 \}$

$g_2(11) = \max \{ g_3(11), g_3(5) + 8 \}$

$g_3(11) = \max \{ g_4(11), g_4(2) + 1 \}$

$= \max \{ 0, 0 + 1 \}$

$= 1$

$g_3(5) = \max \{ g_4(5), g_4(-4) + 1 \}$

$= \max \{ 0, -\infty + 1 \}$

$= 0$

$g_2(-4) = \max \{ g_3(-4), g_3(-10) + 8 \}$

$\therefore g_3(-4) = \max \{ g_4(-4), g_4(-15) + 1 \}$

$= \max \{ -\infty, -\infty + 1 \}$

$= -\infty$

$\therefore g_2(11) = \max \{ 1, 8 \} = 8$

$\therefore g_1(11) = 8$

$g_1(21) = \max\{7, 8+5\}$

$= 13$

$g_0(21) = \max\{13, 10\}$

$= 13$

$\therefore$ max profit is 13

Q3. Design an algorithm to find all pair shortest path and derive its complexity

```
All pairs  P(n)
{
   for i = 1 to n
   for j = 1 to n
   A[i,j] = w[i,j],
        /* ------ */
        for k = 1 to n
        {
            for i = 1 to n
            for j = 1 to n
            A^k[i,j] = min{ A^(k-1)(i,j), A^(k-1)(i,k) + A^(k-1)(k,j) }
        }
        /* print A^1, A^2, ..... A^n */
}
```

Complexity Analysis

- The above algorithm uses 3 nested loops. Innermost loop has only one statement which complexity is $O(1)$

Its complexity can be computed as

$$T(n) = \sum_{k=1}^{n} \sum_{j=1}^{n} \sum_{i=1}^{n} O(1) = \sum_{k=1}^{n} \sum_{i=1}^{n} n = \sum_{k=1}^{n} n^2 = O(n^3)$$

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

Q4. What is multistage graph. What are the objectives of multistage graph How multistage graph problem is solved

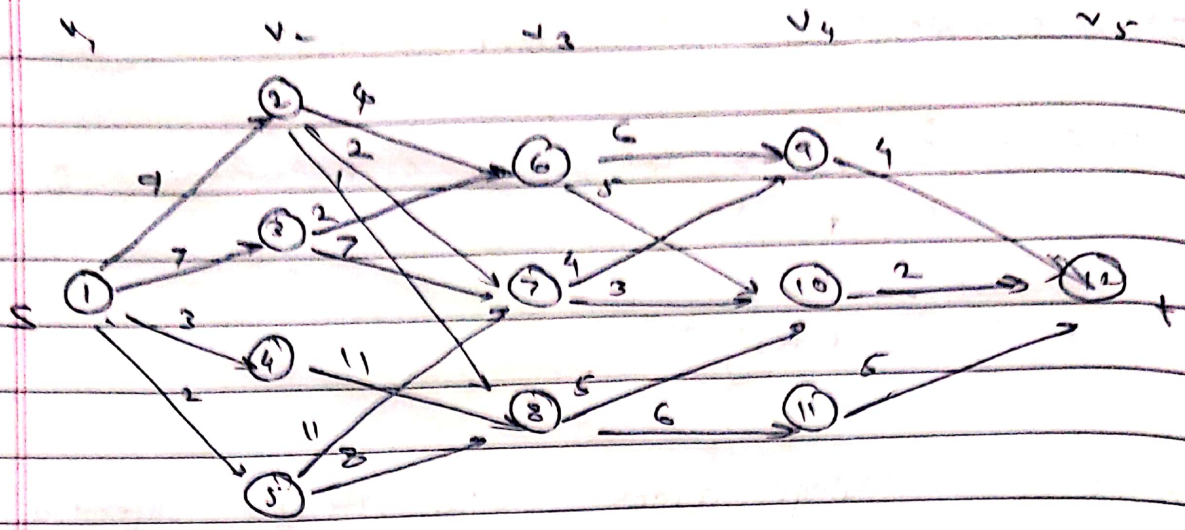
- A multistage graph $G = (V, E, W)$ is a weighed directed graph in which vertices are partitioned into $k \geq 2$ disjoint subsets $V = \{v_1, v_2, v_3, \ldots v_k\}$ such that if the edge (u,v) is present in E then $u \in V$ & $v \in V_{i+1}$, $1 \leq i \leq k$

- Let s be the source vertex and t be the target vertex and let $c(i,j)$ be cost of edge (i,j).
The cost of path from s to t is sum of cost of all edges on that path.

- In multistage graph problem, we have to find minimum cost path from source to target

$$\text{cost}(i,j) = \min \{ c(j,l) + \text{cost}(i+1, l) \}$$
$$l \in V_{i+1}$$
$$(j,l) \in E$$

where i = stage number
j = vertex on that edge



- In graph, there are 5 stages V_1, V_2, V_3, V_4, V_5 = 'k' stages, here k = 5 & cost of vertices is 'k-1' stage.

cost (k-1, j) = c (j, t)

To find cost of vertices in k-1 stage k = 4

① cost (4, 9) = c (9, 12) = 4

② cost (4, 10) = c (10, 12) = 2

③ cost (4, 11) = c (11, 12) = 5

To find cost of vertices in k-2 stages = k = 3

① cost (3, 6) = min { c (6, 9) + cost (4, 9), c (6, 10) + cost (4, 10) }
= min { 6 + 4, 5 + 2 }
= 7

② cost (3, 7) = min { c (7, 9) + cost (4, 9) + c (7, 10) + cost (4, 10) }
= min { 4 + 4, 3 + 2 }
= 5

③ cost (3, 8) = min { c (8, 10) + cost (4, 10), c (8, 11) + cost (4, 11) }
= min { 5 + 2, 6 + 5 }
= 7

To find cost of vertices in k-3 stages, k = 4

① cost(3,2) = min { c(3,6) + cost(4,6), c(3,7) + cost(4,7), c(3,8) + cost(4,8) }
= min { 4 + 7, 2 + 5, 1 + 7 }
= 7

② cost(3,3) = min { c(3,6) + cost(4,6), c(3,7) + cost(4,7) }
= min { 2 + 7, 7 + 5 }
= 9

③ cost(3,4) = min { c(4,8) + cost(4,8) }
= min { 11 + 7 }
= 18

④ cost(3,5) = min { c(5,7) + cost(4,7), c(5,8) + cost(4,8) }
= min { 11 + 5, 8 + 7 }
= 15

To find the cost of vertices in k-4 stage i.e 1

① cost(1,1) = min { c(1,2) + cost(2,2), c(1,3) + cost(2,3), c(1,4) + cost(2,4), c(1,5) + cost(2,5) }
= min { 9 + 7, 7 + 9, 3 + 18, 2 + 15 }
= 16

cost from source to destination is 16

Possible paths are

① 1 - 2 - 7 - 10 - 12

② 1 - 3 - 6 - 10 - 12