DELTA Page No. 1
Date 20/12/16

Analysis and Designing Algorithms (ADA)

Books :- 1. Algorithm by CORMEN (PHI) [TEXT]
Reference 2. Algorithm by SARTAJ SAHNI (GALGOTIA)
3. Schaum Series (TMH)
4. a. _____ 60% Numerical
b. _____ 40% Theory

22/12/16

Algorithm

used to reduce complexity in developing the program. Factors of complexity are:
Time
Memory

→ Algorithm is well defined list of steps for solving a particular problem.

Characteristics:-

1. Input :- Algorithm should be able to accept zero or more inputs. I^* (zero or more i/p)
2. Output:- Algorithm should be able to produce output. $I^+ O^+$ (1 or more o/p)
3. Finiteness :- Algo should be finite.
4. Effectiveness :- Each & every instruction or step should play effective role in the algorithm. Each step should have a meaning.
5. Unambiguity - No redundancy, no duplicacy.
6. Efficient :- Each & every steps should play efficient role.

~~Types of Statements~~

We can measure the effeciency of algorithm by measuring the complexity of algorithm. It is two types
(i) Space complexity &
(ii) Time complexity

Space Complexity:- It is the amount of space taken by any algorithm for complete execution of program.

Time Complexity:- It is the amount of time taken by any algorithm for complete execution of program

Time Complexity is more important than space complexity

Asymptotic Complexity: It is a fundamental measurement of performance of the algorithm. This measurement can be represented in terms of efficiency.
Efficiency has 3 categories:
1. Best case
2. Average case
3. Worst case.

Best Case:- when resources used are minimum
Worst Case:- when resources used are maximum
Average :- nor minimum nor maximum

* Note: To maximize the efficiency of algorithm we should always consider the worst case.

→ The complexity of algorithm not only depends on algorithm but also the number of instructions, the quality of compiler and the skill of programmer.

Growth of Functions:

If any function depends on the number of input n then its growth is represented by using some notations. These notations are known as asymptotic notations
They are as follows:

1. Big oh (O) notation - used for upper bound
2. Big omega (Ω) notation - used for lower bound
3. Big theta (Θ) notation - tight bounds.

Complexity is always in terms of number of inputs (n)

$O(n)$, $O(n^2)$, $O(n^3)$, $O(2^n)$, $O(\log n)$.

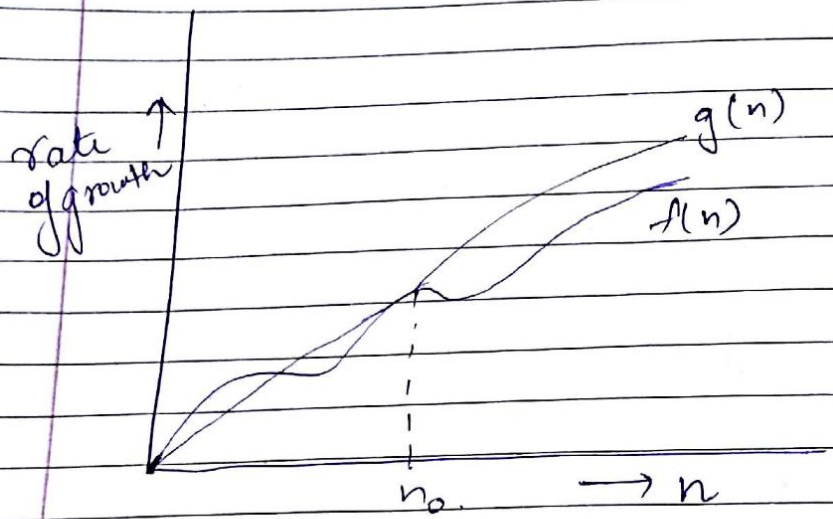
3rd Jan 17

Upper Bound

Big Oh This notation is used to represent the upper bound of a function. For any given function $f(n)$ we can define upper bound as follows

$O(g(n)) = f(n)$: $\exists$ a constant c and integers n_0 in such a way that

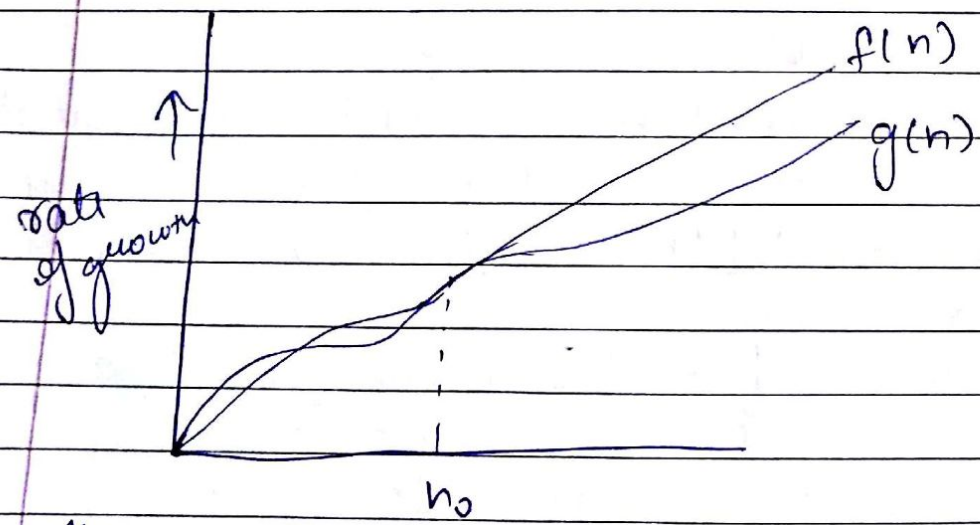
$[0 \leq f(n) \leq c * g(n)$ for $n \geq n_0]$



After n_0 f(n) and g(n) never intersected that is why
$[0 \leq f(n) \leq c * g(n)$ for $n \geq n_0]$

Lower Bound.

Big Ω Notation :



After n_0 f(n) & g(n) $n \to$ do not in intersect
$[0 \leq c * g(n) \leq f(n)]$ for $n \geq n_0$

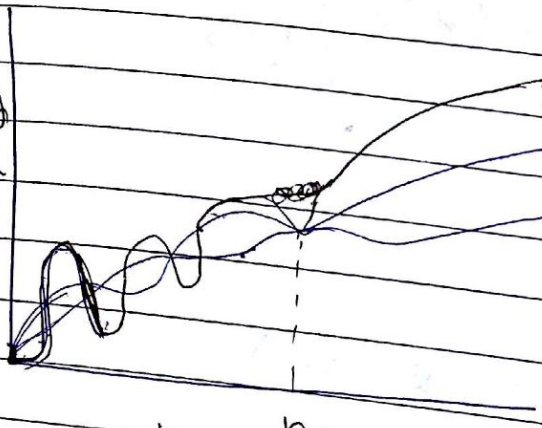
Big Ω is used to represent lower bound for any given function f(n). This bound is represented by Ω(n) and is defined as follows:

$0 \le c * g(n) \le f(n)$ for $n \ge n_0$

For any given function $\Omega_n = f(n)$

Big Thetha Notation (Tight bound)

This is used to represent tight bound ($\Theta(g(n))$



It can be defined as follows $\Theta(g(n)) = f(n) : \exists, c_1, c_2$ for

$c_2 * g(n) \le f(n) \le c_1 * g(n)$ and $n \ge n_0$.

not for exam

Q. Consider the following equation and find upper bound lower bound & tight bound for

$f(n) = 27n^2 + 8n + 5$.

Solⁿ:- For upper bound: $f(n) \le c * g(n)$

$f(n) = 27n^2 + 8n + 5$

$\le 27n^2 + 8n + n$ for $5 \le n$.

$\le 27n^2 + 9n$

$\le 27n^2 + n^2$ for $9n \le n^2$.

$f(n) \le 28n^2$.

$9 \le n \quad n \ge n_0$

So, $f(n) = O(n^2)$. Ans

↘ This is complexity ($O(n^2)$)

$c = 28$
$n_0 = 9$.

For lower Bound : $f(n) \geq c * g(n)$.

$27n^2 + 8n + 5 \geq 27n^2$.
$\Rightarrow f(n) \geq c * g(n)$
$f(n) = \Omega(n^2)$
$c = 27$

Ans

For tight bound

$27n^2 \leq f(n) \leq 28(n^2)$.
$c_1 = 28$
$c_2 = 27$
$n_0 = 9$.
$f(n) = \Theta(n^2)$ Ans

Rules for Calculating Complexity:

1. The complexity of input & output statement is $O(1)$
   eg. Hello world.
2. The complexity of assignment statement ($i = 5$) is $O(1)$
   eg
3. The complexity of calling function is $O(1)$.
4. The complexity of ~~decision statement~~ is that is if else statement.

4. The complexity of if else statement that is decision making is equal to the complexity of condition checking + the complexity of true block and false block.

5. Complexity of checking the condition is O(1)

Sum Rule: If we have two or more complexity in a block then the largest complexity will be considered as the complexity of block.

If $f(n) + g(n) + \dots$ then
$T(n) = \max(O(f(n)), O(g(n)), \dots)$.

for eg. $O(n) + O(n^2)$ then
complexity = $O(n^2)$ Sum Rule.

Loop:- In the case of loop the total complexity is the complexity of checking the condition plus the complexity of body of loop over the no. of times the execution of loop.

Q: for i = 1 to m
    C[i] = A[i] + B[i]
END for

(O(1) : Assignment statement)

complexity → $\sum_{i=1}^{m} O(1) = O(n)$

Q: for j = 1 to n
    for i = 1 to m
        C[i] = A[i] + B[i]
    End for
End for

Complexity = $\sum_{j=1}^{n} \sum_{i=1}^{m} O(1)$
$= \sum_{j=1}^{n} O(n)$
$= O(n \times m)$

```
for i=1 to n
for j=1 to i
    C[i] = D[i] + E[i]
  END for
End for
```

Complexity $= \sum_{i=1}^{n} \sum_{j=1}^{i} O(1)$

$= \sum_{i=1}^{n} O(i) = O(1) + O(2) + \cdots + O(n)$

$= O(\Sigma n)$

$= O\left(\frac{n(n+1)}{2}\right)$ $\frac{1}{2}$ is constant

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

$= O(n^2)$

Bubble Sort -

(5) (1) -15 3 5 5 100 11

i=1

1	(5)	(1)	3	5	100	11
1	-15	(5)	(3)	5	100	11
1	-15	3	(5)	(5)	100	11
1	-15	3	5	(5)	(100)	11
1	-15	3	5	5	(100)	(11)
1	-15	3	5	5	11	100

Comparisons = 6

Assignment statements = 12

Similarly sort other elements in passes

Bubble (a, n)

```
for i=1 to n-i      // pass
   for j=1 to n-i   // Comparison
      if (a[j] > a[j+1]
         t = a[j]
         a[j] = a[j+1]
         a[j+1] = t
      end if
   end for
end Algo.
```

Complexity

(i) $O(1)$

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

(iii) $\sum_{i=1}^{} \sum_{j=1}^{n-i} O(1)$

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

$$= O(n-1) + O(n-2) + O(n-3) + -- + O(2) + O(1)$$

$$= O\left(\Sigma(n-1)\right)$$

$$= \Sigma O\left(\frac{(n-1)n}{2}\right)$$

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

Q. Calculate complexity of selection sort & insertion sort.

DELTA Page No. 11
Date 6 / Jan / 17

Recurrence Relation Those relations which cover the problems related to the recursion are known as recurrence relation. Such relations are solved by various methods like

(i) Master Theorm Method
(ii) Recursion Tree method
(iii) Substitution method.

The running time of recursive algorithm can be obtained by the recurrences in such algorithms we can make the functions in terms of recursive relations.

Master Theorm Method: This method is used to solve all type of eq^n/relation)

~~$T(n) = aT\left(\frac{n}{b}\right) + f(n)$ --- (i)~~

$$T(n) = aT\left(\frac{n}{b}\right) + f(n) \quad \text{--- (i)}$$

where a is the number of subproblem, b is size of subproblem.

To obtain solution of eq (i) we have 3 cases:

Case 1: If $f(n) = O(n^{\log_b - \epsilon})$ where $\epsilon > 0$.

then $T(n) = \Theta(n^{\log_b^a})$ Ans.

Note: If $f(n) < n^{\log_b^a}$ then

Case 2: If $f(n) = \Theta(n^{\log_b^a})$ then ~~$T(n) = \Theta(n^{\log_b}$~~

$$T(n) = \Theta(n^{\log_b^a} \times \log n) \text{ Ans}$$

* This is when $f(n) = n^{\log_b^a}$.

If $f(n) = \Omega(n^{\log_b a + \epsilon})$ then $T(n) = \Theta(f(n))$ Ans

This is when $f(n) > n^{\log_b a}$.

$T(n) = 9T\left(\frac{n}{3}\right) + n$ - (1).

Comparing it with this $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

$a = 9$ $b = 3$ $f(n) = n$	$n^{\log_b a}$ $n^{\log_3 9} = n^2$

$f(n) < n$ so by case I of Master theorem.

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

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

$T(n) = aT\left(\frac{n}{b}\right) + f(n)$ Comparing with this

$a = 1$ $b = \frac{3}{2}$ $f(n) = 1$	$n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1.$ $= \Theta(\log_{3/2} 1)$

$\because f(n) = n^{\log_b a}$ So by case II of Master Theorem

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

DELTA Page No. 13
Date / /

Q. $T(n) = 3T\left(\frac{n}{4}\right) + n\log n$ — (1)

$a = 3$

$b = 4$

$f(n) = n \log n$.

$f(n) > n^{\log_b a}$ so by case III of Master Theorem

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

Recursion Tree Method: This method is also used to solve the recurrence relation. Here we generate a tree where the function $f(n)$ becomes the root of the tree, then we expand the tree in the form of size of sub problem (denoted by s).

→ If size is equal of sub problem is even then the node will be divided into equal parts

→ If size of sub problem is odd then we have odd number of children for the node

H.W General tree, Binary tree, heap, spanning tree, Binary Search Tree.
level, height

Steps to calculate complexity using RTM:

(i) Define $f(n)$ as a root.

(ii) Calculate the height of the tree by extending the nodes according to no. of subproblem & size of sub problem.

Merge Sort

(iii) Calculate the cost of tree at each level.

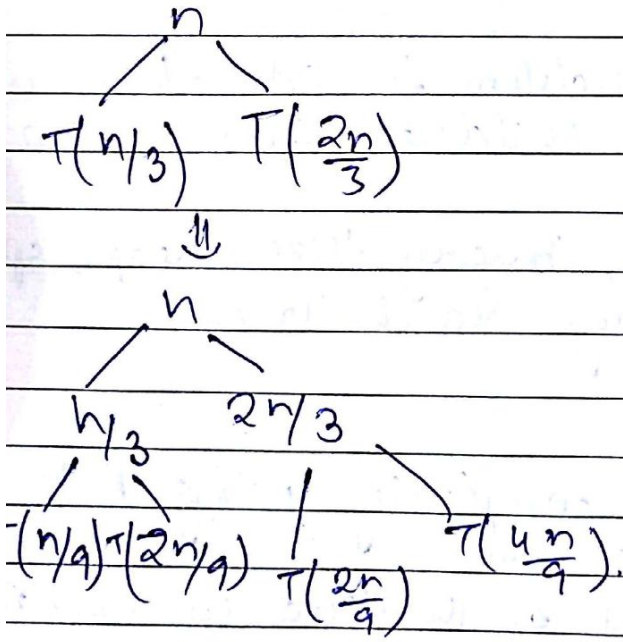
If cost is constant at each level, then
$T(n)$ = cost × height of tree

If cost is not constant at each level, then
$T(n) = C_1 + C_2 + \cdots$
= Σ cost of each level upto the height of the

If nodes have unequal weight, then height of the tree will be calculated by observing the node of heaviest weight.

Find the running time of the following:
$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n$ ... (1)

(i) $T(n)$



Height of tree

(i) Size of subproblem at level 0 = $(2/3)^0 \cdot n$

(ii) " " " " 1 = $(2/3)^1 \times n$

" " " " " 2 = $(2/3)^2 \times n$

" " " " " i = $(2/3)^i \times n$ -(2).

" " " " " = 1 -(3).

Size of subproblem at level = 1 -(3).

By (2) & (3)

$(\frac{2}{3})^i \times n = 1$

$n = (\frac{3}{2})^i$

$\log n = i \log 3/2$

height => $i = \log_{3/2} n$

Cost: Cost at level 0 is n

cost at level 1 is $n/3 + \frac{2n}{3} = n$

cost at level 2 is = n

cost at last level = n.

Since cost at each level is constant so,

$T(n)$ = cost * height

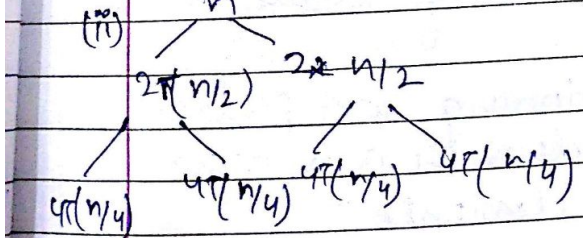
$T(n) = \Theta(n \log_{3/2} n)$

Q. Consider the foll. & find its complexity:

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

(i) $T(n)$

(ii)



$(\because T(n/2) = 4T(n/4) + n/2)$

DELTA Page No 16
Date 19 / 1 / 17

Quick Sort

P.E → (9) 3 (17) 1 5 10 16 (7)
i j

→ first element will be considered as pivot.

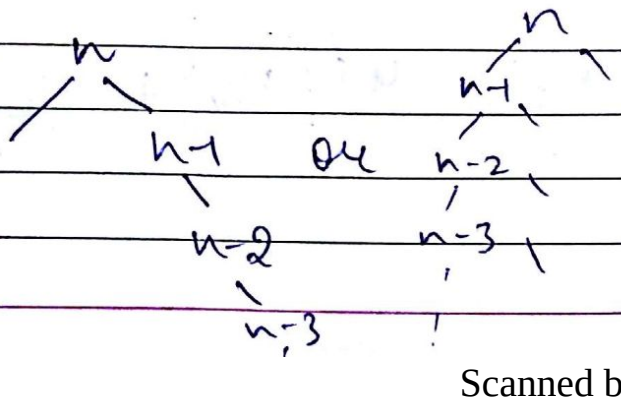
(9) 3 7 1 (5) 10 16 17
i j i j

5 3 7 1 (9) 10 16 17
new P.E i j P.E

Analysis:

P.E → (44) 33 11 55 77 90 40 60
i j
99 22 88 66

(i) WORST CASE - It will occur if all elements are either in ascending order or in descending order.
In this case partioning will produce one sub array with 0 elements and other subarray with n-1 elements.



So, time complexity in this case will be
$T(n) = T(n-1) + \Theta(n) + \Theta(1)$ —①
where the total cost of partioning n elements is $\Theta(n)$, combining cost is $\Theta(1)$.

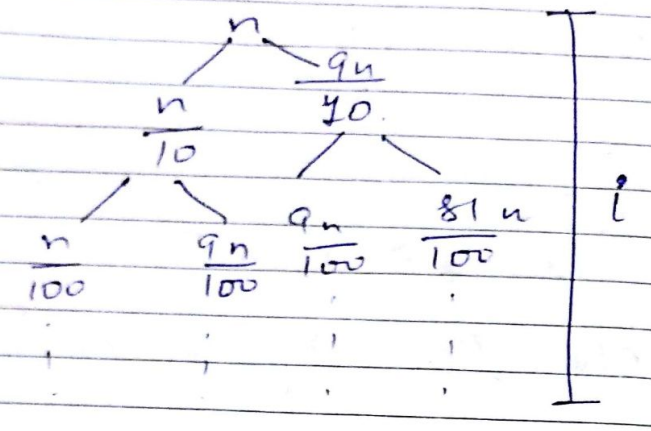
$$T(n) = \Theta\left(\sum(n-1)\right) + \Theta(n)$$
$$= \Theta\left(\frac{(n-1)n}{2}\right) + \Theta(n)$$
$$= \Theta\left(\frac{n^2}{2}\right) - \Theta\left(\frac{n}{2}\right) + \Theta(n)$$
$$= \Theta(n^2) \text{ Ans} \quad [\text{By Sum Rule}]$$

(ii) BEST CASE - It will occur if the partitioning produces two sub arrays each of size not more than $n/2$ elements. i.e nearly balanced tree.
In this case time complexity is given by
$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) + \Theta(1)$

So, by case 2 of Master theorem method we can write
$T(n) = \Theta(n \log n)$

(iii) AVERAGE CASE: When ~~the position of pivot element is random~~ there array is partitioned into any number of children in left and right hand sides then this will be average case.
In this case partioning is not fixed and the position of pivot element may be at any place

Suppose the array is partitioned into 1:9 ratio, then the tree will look like



$i = \log_{\frac{10}{9}} n$

Cost = n

Since cost is fixed at each level so multiply i & cost.

Time complexity will be,

$$T(n) = \theta\left(n \times \log_{\frac{10}{9}} n\right)$$

DELTA Page No. 19
Date 17/1/17

BINARY SEARCH

MERGE SORT:

Divide and Conquer

D & C

Solve the problem in fall 3 steps -

1. Divide: dividing a problem into no. of sub problems.
2. Conquer: Here, we solve the subproblems in recursive manner.
3. Combine: In this step, we combine all the sub solutions to get the final solution. In such problems, time complexity can be obtained by the fall formula -

$$T(n) = aT\left(\frac{n}{b}\right) + D(n) + C(n)$$

- $T(n)$: time to div. problem into sub problem
- a: no. of subprob
- $\frac{n}{b}$: size of subprob
- $D(n)$: time to combine the sub prob.

Merge Sort

90 3 5 2 2 11 75 8

90 3 5 2

```
i=1, k=1, j=(n/2+1)
while (i <= n/2 of j <= n)
  {
if (a[i] < b[j])
{
 c[k]= a[i]
   i++, k++}
else
{
c[k] = b[j]
   k++
   j++
   }
if (i > n/2)
  {
  while (b <= n)
     {
     c[k++] = b[j++]
        }
else
{
while (i <= n/2)
{
(n++) = a[i++]
      }
```

Complexity of Merge Sort

$T(n) = a + \left(\frac{n}{b}\right) + D(n) + C(n) - (1)$

(1) Here, $a = 2$

$b = 2$

(2) Now, Time to div the prob into subprob = $\Theta(1)$

(3) time to combine all the n elements will be $\Theta(n)$ so eqn(1) can be written as -

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

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

Now, by master theorem case 2 -

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

20/11/17 Binary Search

```
BINSRCH(a, lo, hi, item)
BEGIN//
  while(lo <= hi)
    mid = (lo+hi)/2
    if (item = a[mid])
        return(mid) EXIT
    else
    if( item < a [mid])
```

return BINSRCH (a, mid+1, hi, item)
END IF
END IF
END WHILE

Q. 15 22 33 35 47 60 78 88 900.
↑ Mid

Item = 33
lo = 1
hi = 9.
mid = 5

lo = 1
hi = 4
mid = (1+4)/2

Q. 80
lo = 1
hi = 9
mid = 5
mid+1 = lo
lo = 6
hi = 9
mid = 6+9/2 = 8

Analysis of Binary Search

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

from the algorithm of binary search it is clear that the time complexity $T(n)$ = same as above.

For $n>1$

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

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

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

$\vdots \qquad \vdots$

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

$\because \frac{n}{2^x} = 1$

$2^x = n$

$x \log 2 = \log n.$

$x = \log_2 n.$

Putting in eq (1).

$\Rightarrow T(1) + xk.$

$= \theta(\log_2 n)$ Ans.

$x = \log_2 n.$

GREEDY TECHNIQUE: We can solve imp problems like knapsack problem, travel Sales person problem, PRIM's KRUSKAL'S proble.

→ In this technique we consider solution of the problem in such a way that we are able to get feasible or optimum solution.

Following are the steps in greedy approach:

(1) Feasibility check: in this step we check that whether selected approach satisfies all the problem constraints or not.

(2) Local optimum choice:

3/11/17

Fractional eq^n: In this technique items are arranged in the ratio of $R_i = \frac{V_i}{w_i}$

we will arrange all the items in descending order of ~~their~~ above ratio and then we will solve the napsack problem as usual.

Q. Consider the foll. information about napsack.

$I = \langle I_1, I_2, \ldots I_5 \rangle$ $W = 60$.

$w = \langle 5, 10, 20, 30, 40 \rangle$

$v = \langle 30, 20, 100, 90, 160 \rangle$

INITIAL

	v	w	R_i
I_1	30	5	6
I_2	20	10	2
I_3	100	20	5
I_4	90	30	3
I_5	160	40	4

Now, by arranging all the elements in descending order of their ratio we get.

	v	w	R_i
I_1	30	5	6
I_3	100	20	5
I_5	160	40	4
I_4	90	30	3
I_2	20	10	2

$W = 60$

$S = \langle I_1 \rangle$

$v = 30$

$w = 5$

Now $W = 55$

Now 20 <= 55

So, $S = \langle I_1, I_3 \rangle$

$v = 30 + 100 = 130$

$w = 5 + 20 = 25$

$W = 35.$

40 → 160

$35 \rightarrow \frac{35 \times 160}{40} = 35 \times 4. = 140.$

So, $S = \langle I_1, I_3, I_5^* \rangle$

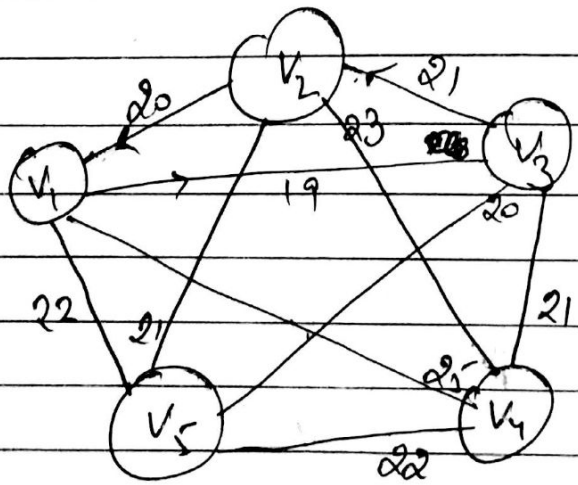
$v = 30 + 100 + 140 = 270$ } Ans

$w = 5 + 20 + 35 = 60$ }

TSP problem

Travelling Salesperson problem.

Consider the graph and find the total cost by applying greedy approach



	V_1	V_2	V_3	V_4	V_5
V_1	0	20	19	25	22
V_2	20	0	21	23	21
V_3	19	21	0	21	20
V_4	25	23	21	0	22
V_5	22	21	20	22	0

$V_1 \xrightarrow{19} V_3 \xrightarrow{20} V_5 \xrightarrow{21} V_2 \xrightarrow{23} V_4 \xrightarrow{25} V_1$

T.C = 108.

Optimum solution = 106.

$V_1 \rightarrow V_2 \rightarrow V_3 \rightarrow V_4 \rightarrow V_5 \rightarrow V_1$.

2/17

MST - Minimum Spanning Tree