1) What is an Algorithm, list its Properties

Ans:

An algorithm is an unambiguous step-by-step procedure to solve given problem in finite number of steps by accepting set of legitimate inputs to produce the desired output.

Every algorithm must satisfy the following criteria:

i) Input:
Should accept one or more inputs

ii) Output:
Should Produce atleast one output

iii) Effectiveness:
Each instruction should transform given I/P to desired O/P

iv) Finiteness:
Algorithm should terminate after finite number of steps

v) Definiteness
Each Instruction should be clear and unambiguous

2. What is Analysis of Algorithm? Name the factors affecting it.

Ans:

The Process of finding the efficiency of an algorithm is called Analysis of algorithm.

Factors affecting it are:

i) Time complexity
ii) Space complexity

Time complexity depends on

-> Speed of computer
-> Choice of Programming Language
-> Compiler used
-> Choice of Algorithm design technique
-> Size and number of Inputs / Outputs.

Order of growth

$1 < \log_2 n < n < n\log_2 n < n^2 < n^3 < 2^n < n!$

⇒ Asymptotic notation

1) O (Big oh) worst case
2) Ω (Big omega) best case
3) Θ (Big thetha) average case

1) O (Big-oh)
It is a measure of longest amount of time it could possibly take for algorithm to complete
It deals with worst case

$f(n) <= c.g(n)$

2) Ω (Big-omega)
It is a measure of minimum amount of time it could possibly take for algorithm to complete.
It deals with worst case
$f(n) \geq c.g(n)$

3) Θ (Big-thetha)
It deals with average case

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

Express following f(n) using big oh notation

1) $f(n) = 100n + 5$

$f(n) <= c \cdot g(n)$

$c = 100 + 1$
$= 101$

$100n + 5 <= 101n$

Let $n = 0$;

$100(0) + 5 \leq 101(0)$

$5 \leq 0$ false

Let $n = 1$;

$100(1) + 5 \leq 101(1)$

$105 \leq 101$ false

$n = 2$;

$100(2) + 5 \leq 101(2)$

$205 \leq 202$ false

$n = 3$;

$100(3) + 5 \leq 101(3)$

$305 \leq 303$ false

$n = 4$;

$100(4) + 5 \leq 101(4)$

$405 \leq 404$ false

$n = 5$;

$100(5) + 5 \leq 101(5)$

$505 \leq 505$ True

$\therefore O(n) \forall n \geq 5$

2) $f(n) = 10n^3 + 5$

$f(n) \leq c \cdot g(n)$

$g(n) = n^3$

$c = 11$

$10n^3 + 5 \leq 11n^3$

$n = 0$:

$10(0) + 5 \leq 11(0)$

$5 \leq 0$ False

$n = 1$

$10(1)^3 + 5 \leq 11(1)$

$15 \leq 11$ False

$n = 2$

$10(2)^3 + 5 \leq 11(2)^3$

$85 \leq 88$ True

$\therefore O(n^3)$ $\forall n \geq 2$

3) $f(n) = 6 * 2^n + n^2$

$c = 7 \quad g(n) = 2^n$

$f(n) \leq c \cdot g(n)$

$6 * 2^n + n^2 \leq 7(2^n)$

Let $n = 0$

$6 * 2^0 + 0^2 \leq 7(2^0)$

$6 * 1 + 0 \leq 7(1)$

$6 \leq 7$ True

$\therefore O(2^n)$ $\forall n \geq 0$

Recurrence Time Complexity

1) $$M(n) = \begin{cases} 0 & \text{if } n=0 \\ M(n-1)+1 & \text{if } n>0 \end{cases}$$

$M(n) = M(n-1) + 1$

$M(n-1) = M(n-1-1)+1+1$

$= M(n-2)+2$

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

$= M(n-3)+3$

$M(n-3) = M(n-1-3)+3+1$

$= M(n-4)+4$

ith term

$= M(n-i)+i$

$n-i = 1$

$i = n-1$

$= M(n-(n-1))+n-1$

$= M(n-n+1)+n-1$

$= 1+n-1$

$= n$

$\therefore O(n)$

Towers of Hanoi

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

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

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

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

$= 2^3 M(n-3) + 4 + 2 + 1$

$= 2(2^3 M(n-3-1) + 4 + 2 + 1) + 1$

$= 2^4 M(n-4) + 2^3 + 2^2 + 2 + 1$

i^{th} term

$= 2^i M(n-i) + 2^{i-1} + 2^{i-2} + 2^{i-3} + \dots + 1$

$i = n-1$

$= 2^{n-1} M(n-n+1) + 2^{n-2} + 2^{n-3} + \dots + 2^2 + 2 + 1$

$= 2^{n-1} + 2^{n-2} + 2^{n-3} + \dots + 2^2 + 2 + 1$

$= \sum_{i=0}^{n-1} 2^i$

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

$= 2^n - 1$

$O(2^n - 1)$

Algorithms

1. Bubble sort $T = O(n^2)$

```
for (i=0; i<n; i++)
  for (j=0; j<n; j++)
  {
    if (a[j] > a[j+1])
      swap (a[j], a[j+1]);
  }
```

2. Selection sort

// Input: An array A[0.... n-1] of orderable elements
// Output: Array A[0... n-1] sorted in ascending order

```
for i ← 0 to n-2 do                    T = O(log n)
  min ← i
  for j ← i+1 to n-1 do
      if A[j] < A[min]
        min ← j
  swap A[i] and A[min]
```

3) Binary search

// Input: sorted array a[i] < ... < a[j], Key x;

```
m ← (i+j)/2;
while i<j and x != a[m] do
  if x < a[m] than j ← m-1
              else i ← m+1;
  if x = a[m]
  then output a[m];
```

$T = O(\log n)$

4) Sequential / Linear search

```
// Searches for a given value in given array
// Input : Array A[0...n-1] and search key K
// Output : The index of first element of A
// that matches K or -1 if no matching elements
    i ← 0
    while i < n and A[i] ≠ K do
        i ← i + 1
    if i < n return i
    else return -1
```

$T = O(n)$

5) Non recursive factorial

```
fact = 1
for (i = 1 → n)
{
    fact = fact * i;
}
return fact;
```

$T(n) = O(n)$

6) GCD of 2 numbers using recursion

```
gcd (a,b)
  if b=0
    return a;
  else
    return gcd (b, a mod b);
```

7) Maximum element

```
// Determines value of largest element in given array
// Input : Array A[0...n-1] of real numbers
// Output : The value of largest element in A
    maxval ← A[0]
    for i ← 1 to n-1 do
```

```
    if A[i] > maxval
        maxval ← A[i]
return Maxval
```

$T = O(n)$

7. Element uniqueness (To find duplicate element in array)

```
// Determines whether all element in array are distinct
// Input: Array A[0...n-1]
// Output: Returns 'true' if all elements in A are distinct
        and 'false' otherwise
for i ← 0 to n-2 do
    for j ← i+1 to n-1 do
    if A[i] = A[j] return false
else return true
```

8. Factorial using recursion

```
// To compute n! recursively
// Input: A non negative integer n
// Output: value of n!
    fact(n)
{
    if n == 0 then
        return 1
    else
        return n * f(n-1)
}
```

9) Towers of Hanoi

```
Procedure Hanoi (disk, source, dest, aux)
if disk == 1 then
 Mov disk from source to dest
else
Hanoi (disk - 1, source, aux, dest)
Mov disk from source to dest
Hanoi ( disk-1, aux, dest, source)
end if
end procedure
```

10) Finding min-max element

```
set min to a[0]
   for i=1 to n-1
    if a[i] > max
    then
        set max to a[i]
   else if a[i] < min
   then
        set min to a[i]
end if-else
end for
print max & min element
```

4. String Matching

Algorithm

BruteForceStringMatch (T[0...n-1], P[0...m-1])
// Implements brute-force string matching
// Input: An array T[0...n-1] of n characters representing text
// and array P[0...m-1] of m characters representing a pattern
// Output: The index of first character in text that starts
// a matching substring or -1 if search is unsuccessful
for i ← 0 to n-m do
    j ← 0
while j < m and P[j] = T[i+j] do
    j ← j+1
if j = m return i
return -1

---

n = length of characters in text ; m = pattern length
Best case = Ω(m)
Average case = Θ(n+m)
Worst case = O(nm)

Code:

```
for (i=0; i<n-m; i++)
{
  for(j=0; j<m; j++)
  {
    if (P[j]==T[i+j])
    continue;
  }
  Break;
}
```

if (j == m)
return 1;

$$T(n) = \sum_{i=0}^{n-m} \sum_{j=0}^{m-1} 1$$

$$= \sum_{i=0}^{n-m} m-1-0+1 = \sum_{i=0}^{n-m} m$$

$$= m \sum_{i=0}^{n-m} 1$$

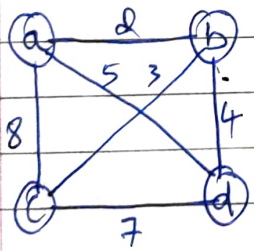
$$= m(n-m-0+1) = mn - m^2 + m$$

$$\approx mn$$

$\therefore O(mn)$

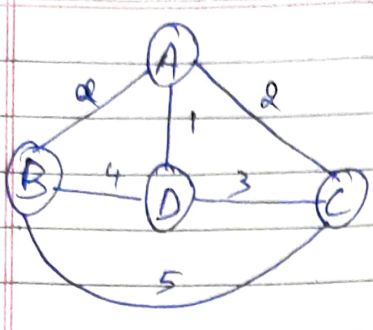
Exhaustive Search
Uses Brute force

→ Travelling Salesman Problem (TSP)
To find shortest Tour that passes through all points exactly once before returning to starting point
$O((n-1)!)$



Tour	Cost
A→B→C→D→A	2+3+7+5 = 17
A→C→B→D→A	8+3+4+5 = 20
A→D→B→C→A	5+4+3+8 = 20
A→B→D→C→A	2+4+7+8 = 21
A→C→D→B→A	8+7+4+2 = 21
A→D→C→B→A	5+7+3+2 = 17

The shortest route is
A→B→C→D→A (17)
A→D→C→B→A



Tour	Cost
A→B→C→D→A	2+5+3+1 = 11
A→B→D→C→A	2+4+3+2 = 11
A→D→C→B→A	1+3+5+2 = 11
A→D→B→C→A	1+4+5+2 = 12
A→C→B→D→A	2+5+4+1 = 12
A→C→D→B→A	2+3+4+2 = 11

→ Knapsack Problem

→ Uses Exhaustive Search

Finds out which combination can give more Profit

Ex:

1. Knapsack capacity = 10

item	Weight	Profit
1	5	20
2	3	40
3	4	30

$O(2^n)$

Sol:

item	wt	Profit
{1}	5	20
{2}	3	40
{3}	4	30
{1,2}	5+3	60
{1,3}	5+4	50
{2,3}	3+4	70
{1,2,3}	5+3+4	NF

Hence {2,3} gives max profit: 70

2.

item	Wt	Value
1	2	20
2	5	30
3	10	50
4	5	10

Knapsack capacity W=16

item	Wt	Profit
{1}	2	20
{2}	5	30
{3}	10	50
{4}	5	10
{1,2}	2+5	50
{1,3}	2+10	70
{1,4}	2+5	30
{1,2,3}	2+5+10	NF
{1,3,4}	2+10+5	NF
{2,3,4}	5+10+5	NF
{2,3}	5+10	80
{2,4}	5+5	40
{3,4}	10+5	60
{1,2,4}	2+5+5	60
{1,2,3,4}	2+5+10+5	NF

Hence {2,3} gives highest value 80

-> Greedy Technique
A technique used to solve optimization Problems.

-> Optimal Solution
For a problem, which satisfies both constraints and gives a solution for our problem is called optimal solution.

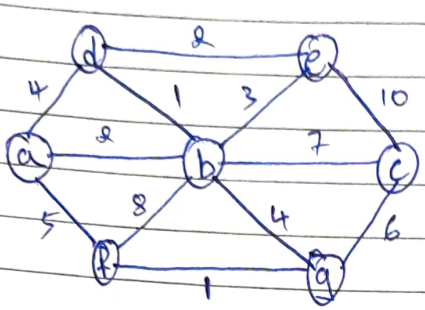
-> Feasible Solution
The solution which will be beneficial to us in the optimal solutions.

→ Prim's Algorithm For Minimal Spanning Tree (MST)

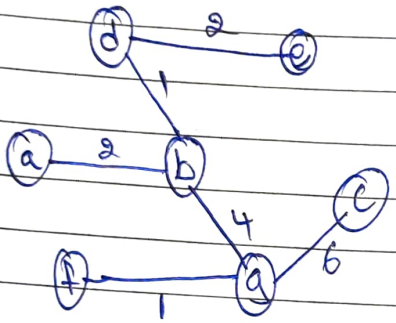
1. no. of vertex will be same as original graph
2. It should be connected
3. No cycles allowed
4. remove the loops, adjacent vertexes (in first step)

Ex:

1.

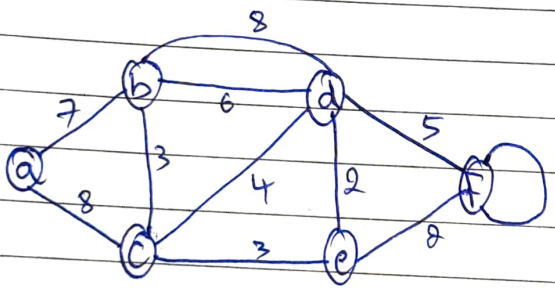


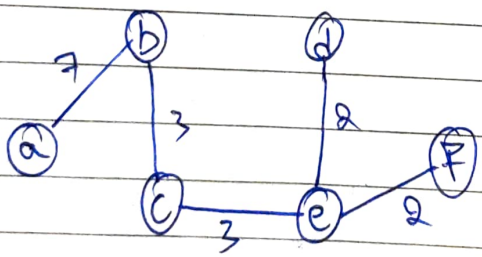
Sol:



Wt = 16

2.





Wt = 17

Algorithm

VT ← {v0}
ET ← NULL
for i ← 1 to |v| - 1 do
find a minimum weight edge of e* = {v*, u*} among all the edges (v,u) such that v is VT and u is in V - VT VT ← VT ∪ {u*}
ET ← ET ∪ {e*}
return ET

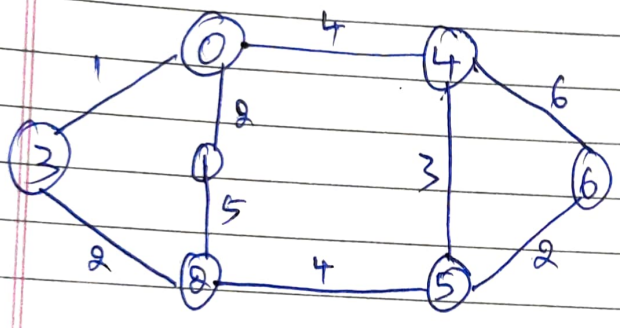
O(n²) for weight matrix
O(m log n) for adjacency

Kruskal

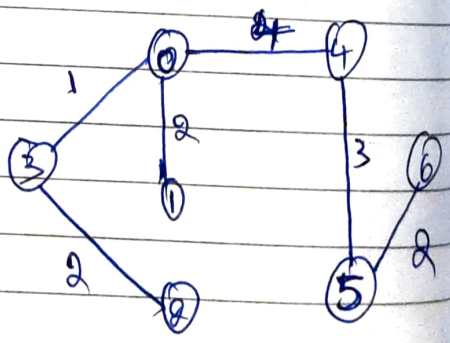
Algorithm

```
Parent = [ ]  Key [ ]  MST = [ ]
For each value V in graph
Key[V] = int max ; MST[V] = F
Key[0] = 0 , Parent[0] = -1
For each vertex V in graph
K = min Key[V] && MST[V] = F
MST[K] = T
For Each vertex V in graph
if (graph[K][V] > 0 && graph[K][V] < Key[V])
{
    Key[V] = graph[K][V]
    Parent[V] = K
}
```

Ex:



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



Wt = 14

Fractional Knapsack Problem

$W = 15$

$n = 7$

object	1	2	3	4	5	6	7
Profit (P)	5	10	15	7	8	9	4
Weight (W)	1	3	5	4	1	3	2
P/W	5	3.3	3	1.7	8	3	2
x_i					1		

Sol:

x_1 x_2 x_3 x_4 x_5 x_6 x_7

$15 - 1 = 14$

$14 - 1 = 13$

$13 - 3 = 10$

$10 - 5 = 5$

$5 - 3 = 2$

$2 - 2 = 0$

$\sum x_i w_i = 1 + 1 + 3 + 5 + 3 + 2 = 15 \text{ kg}$

$\sum x_i p_i = 8 + 5 + 10 + 15 + 9 + 4 = 51$

$\therefore$ Max Profit $= 51$ //

Master Theorem

$$T(n) = a\,T\left(\frac{n}{b}\right) + \Theta(n^k \log^p n)$$

where $a >= 1$ & $b > 1$ & $k >= 0$
p is a real number

case 1: if $a > b^k$
Then $T(n) = \Theta(n^{\log_b^a})$

case 2: if $a = b^k$
a) if $p > -1$ then $T(n) = \Theta(n^{\log_b^a} \log^{p+1} n)$
b) if $p = -1$ then $T(n) = \Theta(n^{\log_b^a} \log\log n)$
c) if $p < -1$ then $T(n) = \Theta(n^{\log_b^a})$

case 3: i) if $a < b^k$
a) if $p >= 0$ then $T(n) = \Theta(n^k \log^p n)$
b) if $p < 0$ then $T(n) = \Theta(n^k)$

Problems.

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

$a = 4, b = 2, k = 1, p = 0$
$4 > 2^1$

$T(n) = \Theta(n^{\log_2^4})$
$= \Theta(n^2)$

2. $T(n) = 16^n T\left(\frac{n}{3}\right) + \frac{1}{n}$

Cannot use Master Theorem

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

$a = 2 \quad b = 2 \quad k = 0 \quad p = 0$

$2 > 2^0$

$T(n) = \theta(n^{\log_b a})$

$= \theta(n)$

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

$a = 8 \quad b = 2 \quad k = 1 \quad p = 0$

$8 > 2^1$

$T(n) = \theta(n^{\log_b a})$

$= \theta(n^{\log_2 8})$

5. $T(n) = 9T\left(\frac{n}{3}\right) + n^2$

$a = 9 \quad b = 3 \quad k = 2 \quad p = 0$

$9 = 3^2$

$T(n) = \theta(n^{\log_b a} \log^{p+1} n)$

$= \theta(n^{\log_3 9} \log^{0+1} n)$

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

6. $T(n) = 2T\left(\frac{n}{2}\right) + n \log n$

$a = 2 \quad b = 2 \quad k = 1 \quad p = 1$

$2 = 2^1$

$T(n) = \theta(n^{\log_b^a} \log^{p+1} n)$
$= \theta(n^{\log_2^2} \log^{-1+1} n)$
$= \theta(n^1 \log^0 n)$
$= \theta(n \log^0 n)$

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

$a = 2 \quad b = 2 \quad k = 1 \quad p = -1$
$2 = 2^1$

$T(n) = \theta(n^{\log_b^a} \log\log n)$
$= \theta(n^{\log_2^2} \log\log n)$
$= \theta(n^1 \log\log n)$
$= \theta(n \log\log n)$

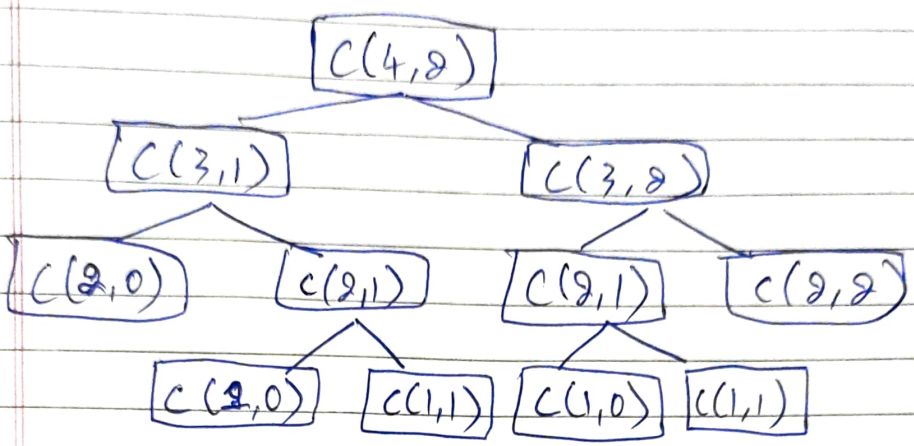
8. $T(n) = 3T\left(\frac{n}{2}\right) + \frac{n^2}{\log n}$

$a = 3 \quad b = 2 \quad k = 2 \quad p = -1$
$3 < 2^2$

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

→ Read Quicksort & Binary Search Algorithm

→ Binomial coefficient of n=4 and K=2



→ Read Warshall
Floyd
Dijkstra's
Sum of Subset
} Problems