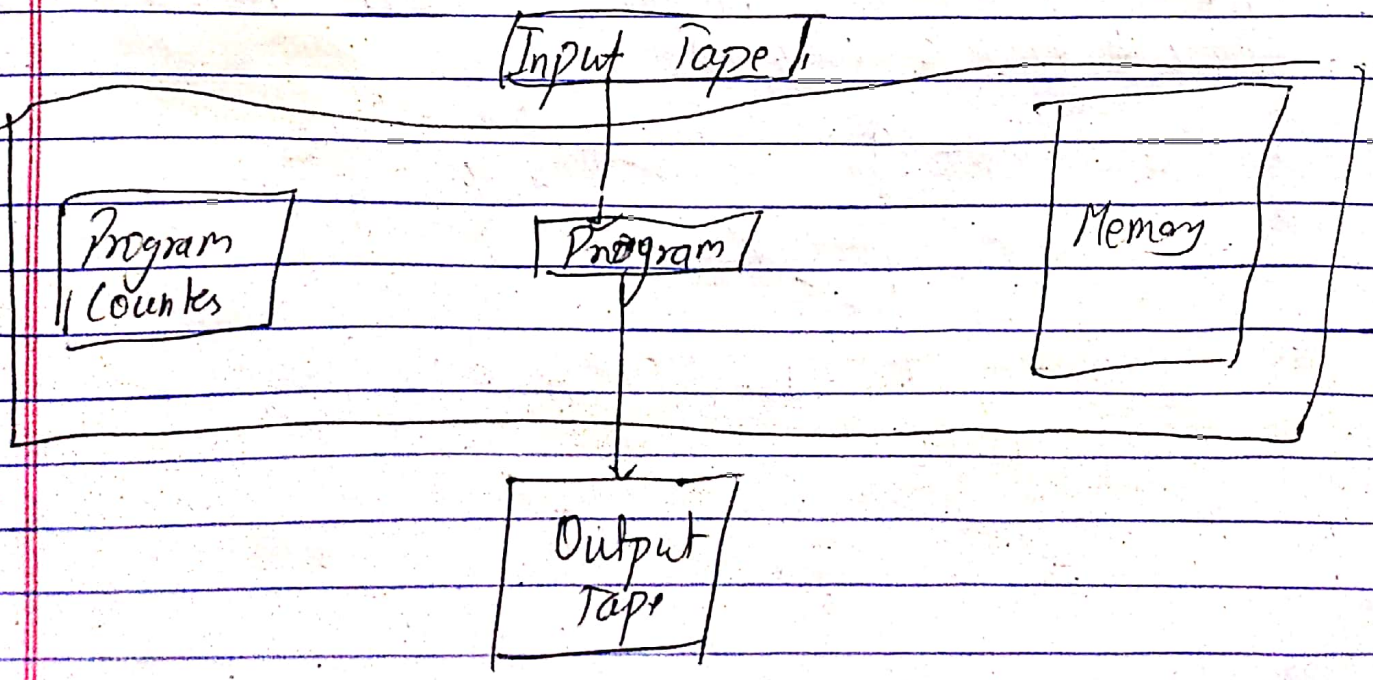
1. Why do you need the algorithm analysis? Discuss about RAM model for analysis of algorithms. Also discuss about Big Oh, Big Omega and Big theta with examples.

→ We need algorithms analysis for following :-

- To understand the basic idea of the problem.
- To find an approach to solve the problem.
- To improve the efficiency of existing techniques.
- To understand the basic properties principles of designing the algorithms.
- To compare the performance of the algorithm.
- To understand the flow of the ~~program~~ problem.

RAM Model

Algorithms can be measured in a machine-independent way using Random Access Machine (RAM) model. This model assumes a single processor



→ It is a generic machine with 5 components
→ There are two types of operations

* Primitive
  - Assign value to variable
  - Perform arithmetic operation
  - Call / return functions
  - Comparisons

* Non - Primitive operation
  - loops

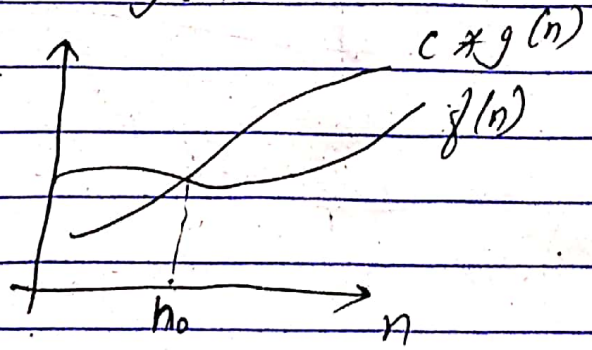
For example :-

a = 5	1
b = a	1
c = a+b	1
T(n) =	O(1)

Big Oh

→ It is used to represent upper bound of algorithm.
→ A function $f(n) = O(g(n))$ iff there exists two +ve constants n_0 and c such that $\forall n \geq n_0$,

$$0 \leq f(n) \leq c * g(n)$$

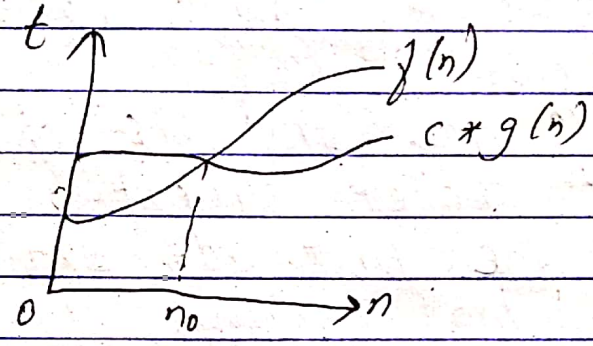


Big Ω

→ It is use to represent lowes bound of algorithm

→ A function $f(n) = \Omega(g(n))$ iff there exist two +ve constants n_0 and c such that for all $n \geq n_0$,

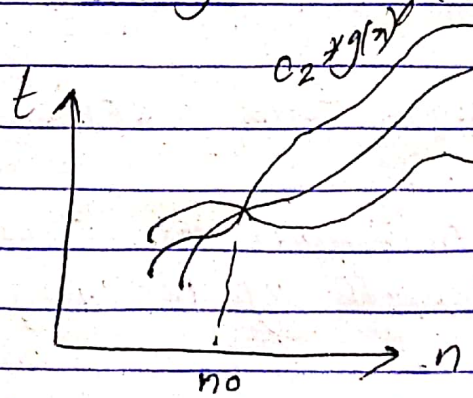
$$0 \leq c * g(n) \leq f(n)$$



Big Θ

→ It is used to represent assymptotically right bound of algorithm.

→ A function $f(n) = \Theta(g(n))$ iff there exists three +ve constants c_1, c_2 and n_0 such that $\forall\ n \geq n_0$, $c_1 * g(n) \leq f(n) \leq (c_2 * g(n))$



2.) Discuss the order statistics. Explain about the worst case linear time selection algorithm and analyze its time complexity.

→ Order statistics are sample value placed in ascending order.

The i^{th} order statistics of a set of n elements is the i^{th} smallest element.

Thus, the first order statistics ($i=1$) indicates the maximum element of set and n^{th} order statistics ($i=n$) indicates the maximum element of the set.

The median is referred as the halfway point of the set of n element. It is given by

$$i = \left(\frac{n+1}{2}\right) \text{ for odd}$$

$$i = \frac{n}{2} \text{ and } \frac{n}{2}+1 \text{ for even}$$

Worst Case linear time selection Algorithm

→ This algorithm determines the i^{th} smallest of an input array of n elements by executing the following steps:

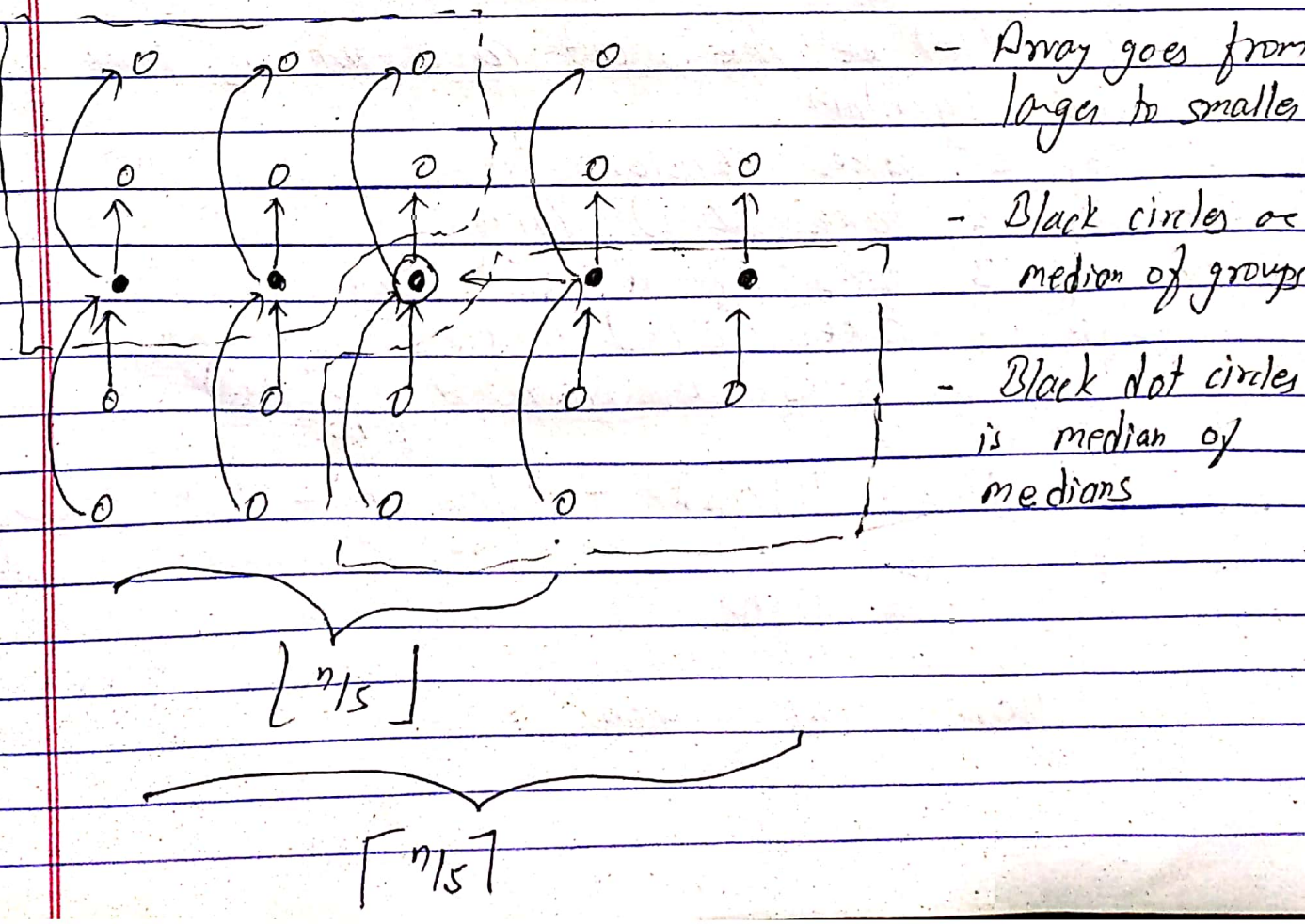
i) Divide the n elements of input array into $\lfloor \frac{n}{5} \rfloor$ groups of 5 elements each at most one group contains remaining n mod 5 elements.

ii) Find the median of each of $\lceil n/5 \rceil$ group.

iii) Use select recursively to find the median x of the $\lceil n/5 \rceil$ medians found in step 2.

iv) Partition the input array around medians of median x
let k = rank (k)

v) if $i == k$, then return x
if $i < k$, then select recursively to find i^{th} smallest element in first partition
if $i > k$, then use select recursively to find i^{th} smallest element in last partition.



From the above figure at least half of the group medians are greater than x, thus half of the $\lceil n/5 \rceil$ group ~~contain~~ contribute 3 elements that are greater than x except the group that has fewer than 5 elements and the group containing x itself.

So, no. of elements greater than x is at least

$$3\left(\frac{\lceil n/5 \rceil - 2}{2}\right) \geq \frac{3n}{10} - 6$$

Similarly, at least $(3n/10 - 6)$ element are less than x. Thus, in worst case, step 5 calls select recursively on atmost

$$n - \left(\frac{3n}{10} - 6\right) = \frac{7n}{10} + 6$$

Now

let $T(n)$ be the worst case running time of select algorithm

→ Step 1 takes constant time
→ Step 2 takes $O(n)$ Time
→ Step 3 take $T(\lceil n/5 \rceil)$ time
→ Step 4 takes $O(n)$ Time
→ Step 5 takes ad most $T\left(\frac{7n}{10} + 6\right)$ time

$$\therefore T(n) = T(\lceil n/5 \rceil) + T\left(\frac{7n}{10} + 6\right) + O(n) \; ; \; n > 5$$

$$= O(n)$$

Hence $T(n) = O(n)$

3. Explain in brief about the Dynamic Programming Approach for algorithm design. How it differs with recursion? Explain the Floyd Warshall algorithm to ~~compute~~ compute the all pair shortest path in graph and analyse its time ~~its~~ complexity.

→ Dynamic Programming is a technique for solving problem with overlapping subproblem. Like divide & conquer approach, it solves the problem by combining the solutions of subproblems. This is a bottom up problem solving technique. It is typically used for solving the optimization problem. The basic elements that characterize a dynamic ~~program~~ algorithm are:

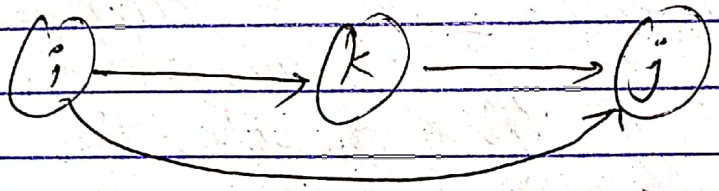
i) Substructure
ii) Table Structure
iii) Bottom up computation.

Dyanamic Programming	Recursion
→ It reduces the line code.	It takes more line of code.
→ It speeds up the processing.	Processing speed is less than DP.
→ It takes lot of memory.	It takes less memory.
→ It leads to unnecessary memory utilization.	Memory is utilized properly.

Floyd Warshall Algorithm

→ To compute all pairs shortest path in bottom up manner for finding $D^1, D^2, D^3, \ldots D^n$, Floyd's Warshall algorithm use the relation

$$D^k(i,j) = \min\{D^{k-1}(i,j), D^{k-1}(i,k) + D^{k-1}(k,j)\}$$

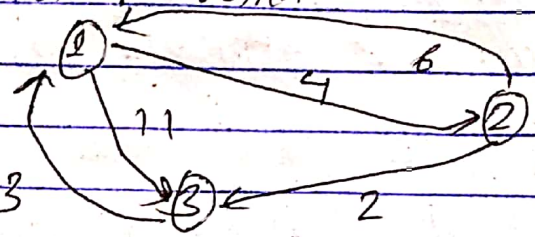



```
FloydWarshwal (W, D, n)
{
    for (i=1; i<=n; i++)
        for (j=1; j<=n; j++)
            D[i][j] = W[i][j]

    for (k=1; k<=n; k++)
        for (i=1; i<n; i++)
            for (j=2; j<=n; j++)
                if ( D[i][j] > D[i][k] + D[k][j]
                    D[i][j] = D[i][k] + D[k][j]
}
```

The running time of the Floyd-Warshall Algorithm is determined by the triply nested for loops. Each loop executes at most $O(n)$ time. The algorithm thus runs in time $O(n^3)$

For eg:- Find the shortest path from every vertex to other vertex



Sol:-

Adjancy matrix of given graph is

D^0	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

Case I When vertex (1) as intermediate vertex

$D^1(1,1) = 0$

$D^1(1,2) = \min\{D^0(1,2), D^0(1,1) + D^0(1,2)\} = 4$

$D^1(1,3) = \min\{D^0(1,3), D^0(1,1) + D^0(1,3)\} = 11$

$D^1(2,1) = \min\{D^0(2,1), D^0(2,1) + D^0(1,1)\} = 6$

$D^1(2,2) = 0$

$D^1(2,3) = \min\{D^0(2,3), D^0(2,1) + D^0(1,3)\} = 2$

$D^1(3,1) = \min\{D^0(3,1), D^0(3,1) + D^0(1,1)\} = 3$

$D^1(3,2) = \min\{D^0(3,2), D^0(3,1) + D^0(1,2)\} = 7$

$D^1(3,3) = 0$

Then, adjacency matrix is modified as

D^1	1	2	3
1	0	4	11
2	6	0	2
3	3	7	0

D^2	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

D^3	1	2	3
1	0	4	6
2	5	0	2
3	2	7	0

} which is the shortest path

4. Write the algorithm for Binary Search with divide and conquer approach and explain its complexity.

```
BinarySearch (A, low, high, key)
{
    if (low == high)
    {
        if (key = A[low])
            return low;
        else
            return -1;
    }
    else
    {   mid = (low + high) / 2;
        if (key == A[mid])
            return mid;
        else if (key < A[mid])
            return BinarySearch (A, low, mid-1, key)
        else
            return BinarySearch (A, mid+1, high, key)
    }
}
```

Let n be the number of element in the array and T(n) be the time required to search the value in the list of n element by BinarySearch. The running time of binarySearch algorithm can be expressed by following recurrence relation

$$T(n) = 1 \quad \text{for } n = 1$$
$$= T(n/2) + 1 \quad \text{for } n > 1 \quad \Rightarrow O(\log n)$$

In the best case output is obtained at one run ie $O(1)$ time if the key is at middle.

In the worst case the output is at the end of the array so running time is $O(\log n)$ time.
In the average case also running time is $O(\log n)$

5. Solve the following recurrence relations using master method.

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

So,

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

$$n^{\log_b a} = n^{\log_2 3} = n^{1.5849}$$

From case I of master method

$$T(n) = O(n^{1.5849}) \quad [\because f(n) < O(n^{\log_b a})]$$

2.) $T(n) = 2T(n/4) + \sqrt{n}$

So,

Here, $a = 2$, $b = 4$, $f(n) = \sqrt{n} = n^{1/2}$

$$n^{\log_b a} = n^{\log_4 2} = n^{1/2}$$

From case II of master method

$$T(n) = \Theta(n^{\log_b a} \cdot \log n) \quad [\because f(n) = \Theta(n^{\log_b a})]$$
$$= \Theta(n^{1/2} \cdot \log n)$$
$$= \Theta(\sqrt{n} \log n)$$

6.) What is prefix code ? Explain Huffman algorithm to compute the prefix codes.

→ Prefix code is a binary code given to each character in a file such that no any code is prefix of another code.

Eg :-

a = 0, b = 101, c = 100, d = 111, e = 1101

f = 1100

Huffman algorithm

→ To generate a Huffman code, we should creat a binary tree called huffman coding tree as

i) Maintaining priority queue in ascending order of frequencies representing a tree of single node for each character.

ii) At each step, extract two min code, delete from queue, merge two tree into a single tree and insert into a priority queue until all of the partial huffman trees were combined into one.

For eg :-

a : 45 , b : 13 , c : 12 , d : 16 , e : 9 , f : 5

so,

Step 1

Initial priority queue

f : 5 e : 9 c : 12 b : 13 d : 16 a : 45

i = 1

c : 12 b : 13 (14) d : 16 a : 45

f : 5 e : 9

i = 2

(14) d : 16 (25) a : 45

f : 5 e : 9 c : 12 b : 13

i = 3

(25) (30) a : 45

c : 12 b : 13 (14) d : 16

f : 9 e : 9

i = 4

a : 45 (55)

(25) (30)

c : 12 b : 13 (14) d : 16

f : 9 e : 9

i = 5

(100)

a : 45 (55)

(25) (30)

c : 12 b : 13 (14) d : 16

f : 9 e : 9

∴ x = 45

y = 55

z = 100

7. Write the algorithm for insertion sort and explain its time complexity

```
InsertionSort (A, n)
{
    for i=1 to n
    {
        temp = A[i]
        j = i-1
        while (j>=0 && A[j] > temp)
        {
            A[j+1] = A[j]
            j = j-1
        }
        A[j+1] = temp
    }
}
```

Best Case

If the input array is already in sorted order, insertion sort compares O(n) elements and performs no swap. Therefore, in best case, it runs in O(n) time.

Worst and Average Case

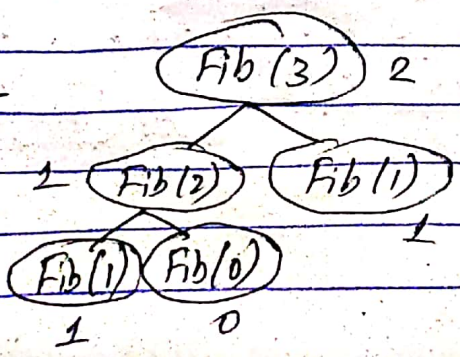
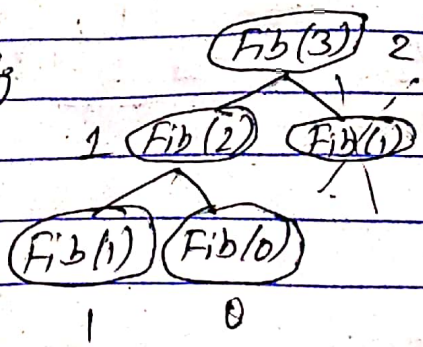
The worst case for insertion sort will occur when the input list is in decreasing order.

To insert last element, comparisons = n-1 swaps = n-1
To insert 2nd last element, comparisons = n-2, swaps = n-2
⋮

$$2 \times (1+2+3+\dots+(n-2)(n-1))$$
$$= 2\left[\frac{(n-1)(n-1+1)}{2}\right] = n^2 - n = O(n^2)$$

5.) What do you mean by memoization strategy? Compare memoization with dynamic programming.

→ The method of recording the ~~records~~ results of earlier calculations so that we don't have to repeat the calculation later is known as memoization strategy.

Dynamic Programming	Memoization
→ DP is bottom up (build the table incrementally)	Memoization is top-down (recurse from the top but with caching)
→ DP is a solution strategy which asks you to find similar subproblems so as to solve big subproblems.	It is a technique to avoid repeated computation on the same problems.
→ In DP, table is maintained from bottom up for the subproblem solutions	There is no table maintained but has extensive recursive calls.
→ DP solves all the sub-problems.	It only solves those that are needed to be solved.
→ Eg:- 	Eg 

9.) Explain Backtracking with suitable example.

→ The Backtracking is an algorithm - method to solve a problem with an additional way.

→ The Backtracking uses recursive approach to solve the problems.

→ We can say that the backtracking is used to find all possible combination to solve an optimization problem.

→ If an algorithm backtracks with intelligence, it is called backtracking algorithm.

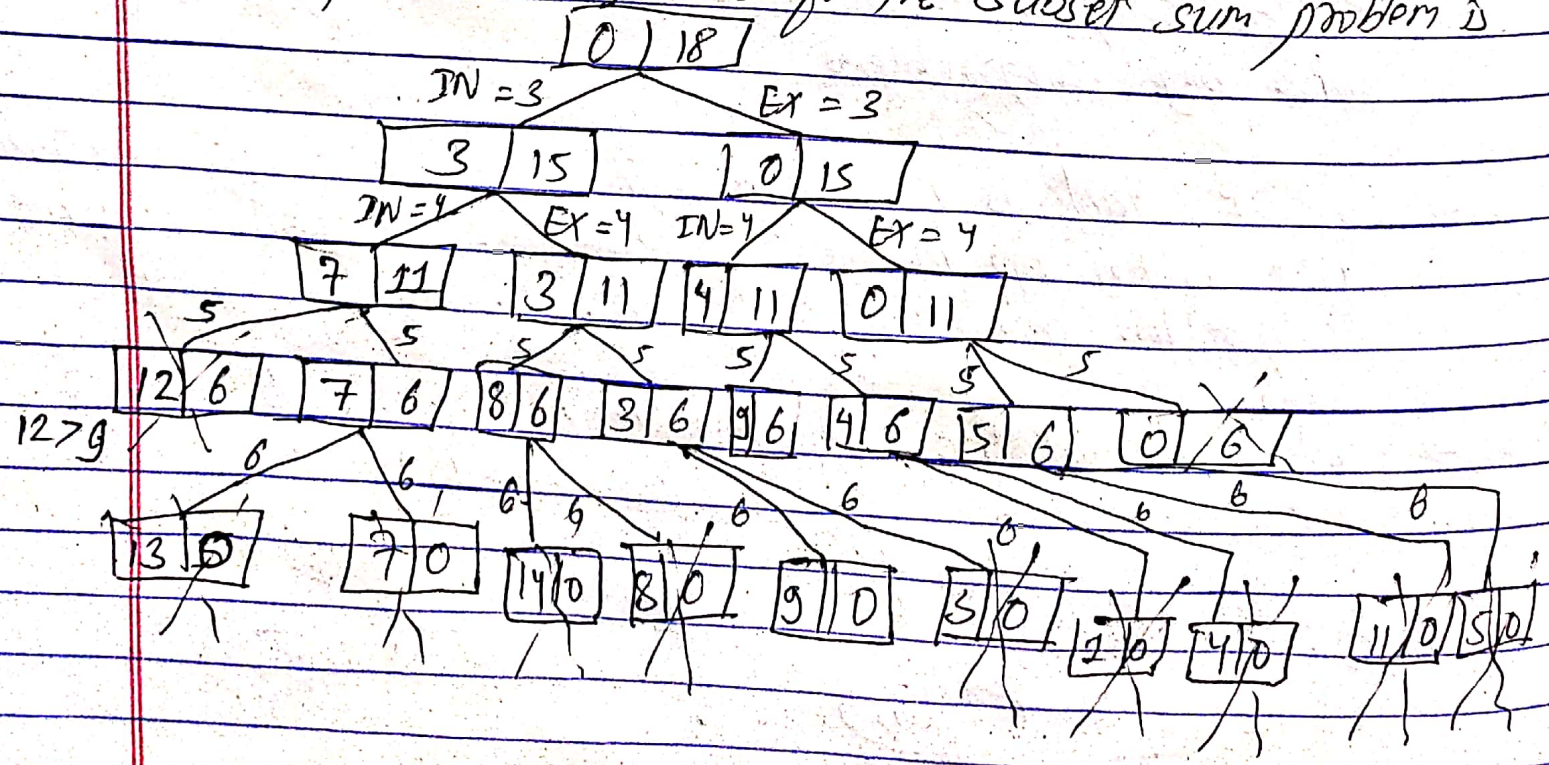
For eg:-

Subset sum using backtracking

$S = \{3, 4, 5, 6\}$, and $x = 9$

$S' = \{\} \subseteq S$

The implicit binary tree for the subset sum problem is



10.) Explain the Euclid's method to solve the modular linear equations with examples.

Modular Linear Equations

Consides the problem of finding solutions to the equation

$$ax \equiv b \pmod{n}$$

The algorithm is

```
Modular linear Eqn (a, b, n)
{
    (d, x', y') = Extended-Euclid (a, n)
    if d | b
        x0 = x' (b/d) mod n
        for i=0 to d-1
            print (x0 + i(n/d)) mod n
    else
        print 'no solutions'
}
```

→ The equation $ax \equiv b \pmod{n}$ is solvable for the unknown x iff $\gcd(a,n) \mid b$

→ The equation either has d distinct solutions modulo n where $d = \gcd(a,n)$ or it has no solutions

→ For any $n > 1$, if $\gcd(a,n) = 1$, then the equation $ax \equiv b \pmod{n}$ has a unique solution, modulo n

→ For any $n > 1$, if $\gcd(a,n) = 1$, then the equation $ax \equiv 1 \pmod{n}$ has a unique solution, modulo n Otherwise, it has no solution.

11.) Explain in brief about the complexity classes P, NP and NP Complete.

P

The class P consists of those problems that can be solved by a deterministic Turing Machine in Polynomial time. P problems are obviously tractable. Eg :- Adding two numbers

NP

NP is set of decision problems that can be solved by a non-deterministic Turing Machine in polynomial time. P is subset of NP. It consists of those problems that are verifiable in polynomial time. Every problem in this class can be solved in exponential time using exhaustive search.

Eg :- Prime Factorization

NP Complete.

These are the problems that are at least as hard as the NP-complete problems.

NP-complete problem is a complexity class which represent the set of all problems X in NP for which it is possible to reduce any other NP problem Y to X in polynomial time. It is the hardest problems in NP set. A decision problem L is NP-complete if

- L is in NP
- Every problem in NP is reducible to L in polynomial time

- Eg :- Knapsack problem, Vertex covers, Subset sum

12.) Write short Notes on

a) Tractible and Intractable Problems.

The problem which can be solved using polynomial time algorithms are called Tractable problem.

The problems that cannot be solved in polynomial time but requires supes-polynomial time algorithm are called intractable or hard problem

Eg. of tractable problem
- Searching an unordered list
- Sorting a list
- Multiplication of integers

Eg of Intractable problem
- Tower of Hanoi
- List all permutations of n numbers.

b) Approximation Algorithms.

An approximation algorithm is a way of approach NP-completness for the optimization problem. This technique does not guarantee the best solution. The goal of it is to come as possible to the optimum values in a reasonable amount of time which is at the most polynomial time.