I NOTATION OF AN ALGORITHM :-

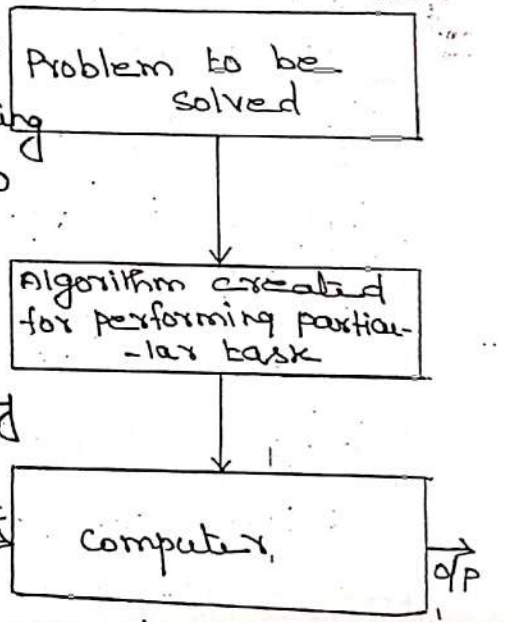
ALGORITHM :-

The Algorithm is defined as a collection of unambiguous instructions occurring in some specific sequence and such an algorithm should produce output for given set of input in finite amount of time.

After understanding the problem statement we have to create an algorithm carefully for the given problem.

→ The algorithm is then converted in to some programming language & then given to some computing device.

→ Then executes this algorithm which is actually submitted in the form of source program.



→ During the process of execution it requires certain set of input. with the help of algorithm & input set, the result is produced as an output.

→ If the given output is invalid then it should raise appropriate error message. Otherwise correct result will be produced as an o/p.

PROPERTIES OF ALGORITHMS:-

1. Non-ambiguity:-
   ↳ Each step in an algorithm should be non-ambiguous.
→ That means each instruction should be clear and precises.
→ This property also indicate the effective-ness of algorithm.

2. Range of input:-
→ The range of input should be specified this is because normally the algorithm is input driven & if the range of the input is not been specified then algorithm can go in an infinite state.

3. multiplicity:-
   The same algorithms can be represented in several different ways.
→ That means we can write simple english the sequence of instructions or we can write it in the form of pseudo code.

4. Speed:-
   It should be efficient and should produce the output with fast speed.

5. Finiteness:-
   The algorithm should be finite. That means after performing required operations it should terminate.

HOW TO WRITE AN ALGORITHM

Algorithm

Algorithm heading (name of algorithm, problem description, input & output)

Algorithm body [logical body of algorithm, by making use of various pgm Construct assignment statement].

→ Algorithm is a procedure consisting of heading and body.

→ The heading consists of keyword, Alg and name of the algorithm and parameter list.

Algorithm name $(P_1, P_2 \text{ ----} P_n)$

↓ This keyword should be written first

↓ name of an Algorithm

↓ Write parameters if any.

→ Then in the heading section we should write

// problem description.
// input;
// output:

→ Then body of an algorithm is written in which various programming construct like if, for, while or some assignment statement may be written.

→ The compound statements should be enclosed within { and } brackets.

→ The identifier should begin by letters not by digit. An identifier can be a combination of alphanumeric string.

→ Using assignment operator ← an assignment statement can be given,

syntax,

Variable ← expression

→ There are other types of operators such as boolean operators such as true or false. logical operators such as AND, OR, NOT & relational operators such as <, <=, >, >=, =, ≠.

→ The array indicates are stored with in square brackets [ ]. The index of array usually start at zero.

The multidimensional arrays can also be used in algorithm.

→ The input & output can be done using read & write.

ex: Write ("This message will be displayed on console");
read (Val);

→ The conditional statements such as if-then or if-then else are written.

if (condition) then statement

if (condition) than statement else statement.

→ While statement can be written as;

```
While (condition) do
 {
 statement 1
 statement 2
      :
 statement n
 }
```

→ While the condition is true the block enclosed with { }, gets executed otherwise statement after } will be executed.

→ The general form for writting for loop is,

```
for Variable ← Value1 to Valuen do.
    {
     statement -1
     statement - 2
          :
     statement -- n
    }
```

→ here Value1 is initialization condition & Valuen is a terminating condition.
The step indicates the increments or decrements in Value1 for executing the for loop.

→ The repeat until statement can be,

```
repeat
 statement 1
 statement -2
      :
 statement . n
 until (condition).
```

→ The break statement is used to exit from inner loop.

→ The return statement is used to return control from one point to another.

→ While is exiting from function

IMPORTANT PROBLEM TYPES:-

There is large number of computing problems & some of them can be classified as

1. sorting
2. searching
3. Numerical problems
4. Geometric problems,
5. Combinational problems
6. Graph problems
7. String processing problems.

These are important problem types.

1. SORTING:-

→ sorting means arranging the elements in increasing & decreasing order called as sorting.

→ sorting can be done on numbers, characters strings or employee records.

EX: For keeping the employee record in sorting order, we will arrange the employee record as per employee ID.

2. SEARCHING:-

→ Searching is an activity by which we can find out the desired elements from the list.

→ The element which is to be searched is called search key.

Two types of searching are,

1. sequential search.
2. binary search.
3. Fibonaci search.

3. NUMERICAL PROBLEMS:-

→ It based on mathematical equations, systems of equations, computing definite integrals, evaluating functions & so on.

4. GEOMETRIC PROBLEMS:-

→ It is one type of problem solving area in which various operations can be performed on geometric objects such as points, line, polygon.

Ex, computer graphics, robotics.

5. COMBINATIONAL PROBLEMS:-

→ It related to the problems like computing Permutations & combinations.

→ The combinational problems are most difficult problems in computing area.

1. As problem size grows the combinational objects grow rapidly & reach to a huge value.
2. There is no algorithm available which can be solve these problems in finite amount of time
3. many of these problems fall in the category of unsolvable problems.

6. GRAPH PROBLEMS

→ Graph is a collection of vertices & edges

→ It involves graph traversal algorithms, shortest path algorithms and topological sorting and so on.

Ex; Travelling sales man problem, graph coloring problems.

7. STRING PROCESSING PROBLEMS:-

→ string is a collection of characters.

→ The typical string processing algorithm is pattern matching algorithm

→ These algorithms used ~~word~~ in Particular word is searched from the text.

→ It is simple to implement.

FUNDAMENTALS OF ALGORITHMIC PROBLEM SOLVING:-

The following steps are,

1. understanding the problem
2. Decision making on
3. Capabilities of computational devices
4. Choice for either exact or approximate Problem solving method.
5. Data structures
6. Algorithmic Strategies

3. specification of algorithm
4. Algorithmic verification
5. Analysis of algorithm.
6. Implementation or loading of algorithm.

1. UNDER STANDING THE PROBLEM:.

→ To understand the problem statement completely.
→ To understand the problem statements, read the problem description carefully.
→ To ask questions for clarifying the doubt about the problem.
→ Some types of problems which are commonly occurring & to solve such problems, they are typical algorithms which are already available.
→ After carefully understanding the problem statements find out what are the necessary inputs for solving that problem.
→ The input to the algorithm is called instance of the problem.
→ It is very important to decide the range of inputs, so, that the boundary values of algorithm get fixed.
→ The algorithm should work correctly for all valid inputs.

2. DECISION MAKING:.

→ After finding the required input set for the given problem we have to analyze

→ The input & need to decide certain issues such as capabilities of computational devices. whether to use exact or approximate Problem solving, data structures are used.

a] CAPABILITIES OF COMPUTATIONAL DEVICES:-

↳ The devices on which the algorithm will be running.

→ Globally we can classify an algorithm from execution point of view as sequential algorithm and Parallel algorithm.

→ The sequential algorithm specifically runs on the machine in which the instructions are executed one after another.
This machine is called Random Access Machine (RAM).

→ The Parallel algorithms are run on the machine in which the instructions are executed in parallel.

→ It is space and time efficient.

b] CHOICE FOR EITHER EXACT OR APPROXIMATE PROBLEM SOLVING METHOD

→ If the problem needs to be solved correctly then we need exact algorithm.

→ If the problem is so complex that we won't get the exact solution then in that situation we need to choose approximation algorithm

C] DATA STRUCTURES:-

→ Data structures and algorithm work together and these are interdependent

→ The proper data structure is required before designing the actual algorithm.

D] ALGORITHMIC STRATEGIES:-

→ It is a general approach by which many problems can be solved algorithmically.

→ These problems may belong to different areas of computing. It is also called as algorithmic techniques (or) algorithmic paradigm.

3] SPECIFICATION OF ALGORITHM:-

There are various ways by which we can specify an algorithm.

Algorithm → Using Natural language
→ Pseudo code
→ flow chart

→ It is very simple to specify an algorithm using natural language.

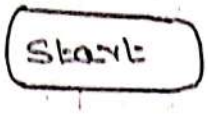
→ in many time specification of algorithm by using natural language is not clear & there by we get brief specification.

EX:

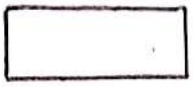
step:1⇒ Read the 1st number say a.
2⇒ Read the 2nd number say b.
3⇒ Add the 2 numbers & store the result in var c.
4⇒ Display the result.

→ Such a specification creates difficulty such a space while actually implement it.

→ many programmers prefer to have specification of algorithm by means of pseudo code.

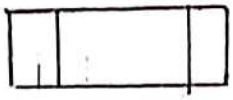


→ start state

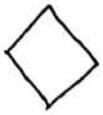


→ Transition.

→ processing or assignment statement



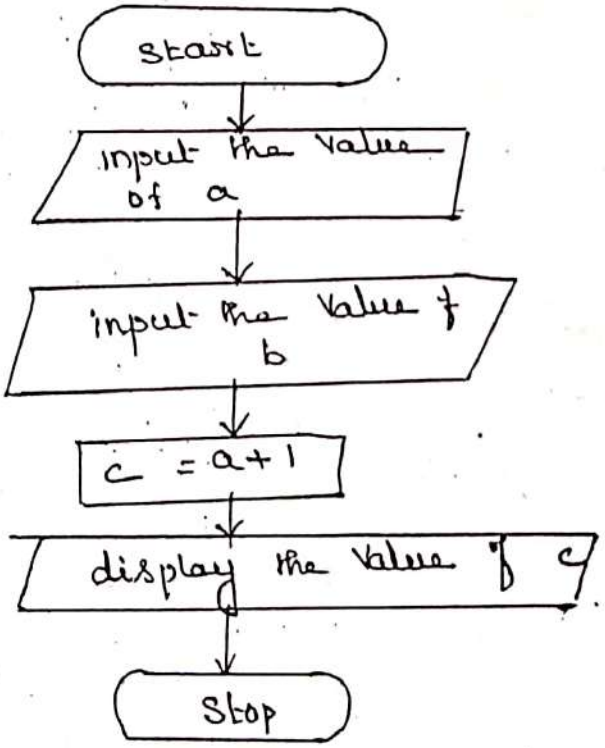
→ input - output statement



→ conditional statement

→ Stop.

Ex:



4. ALGORITHMIC VERIFICATION:-

→ It means checking correctness of an algorithm.

→ After specifying an algorithm we go for checking its correctness.

→ To check the algorithm gives correct output in finite amount of time for a valid set of input.

→ A common method of proving the correctness of an algorithm is by using mathematical induction.

5. ANALYSIS OF ALGORITHM

The following step's are,

1. Time efficiency of an algorithm
2. space efficiency of an algorithm
3. simplicity of an algorithm
4. Generality of an algorithm
5. Range of Input.

1) Time complexity ⇒

→ It means the amount of time taken by an algorithm to run.

→ by computing time complexity algorithm is slow or fast.

2) space efficiency:-

→ It means the amount of space taken by an algorithm. to analyze these algorithm require more or less space.

3] SIMPLICITY:-

→ It generating sequence of instructions which are easy to understand.

→ while, simplifying an algorithm we have to compute any predefined computations or some complex mathematical derivation.

→ finding out bugs from algorithms or debugging the program becomes easy when an algorithm is simple.

4] GENERALITY:-

→ It becomes easier to design an algorithm.

→ in more general way rather than designing it for particular set of input.

5] Range of inputs:-

→ It comes in picture when we execute an algorithm.

→ The design of an algorithm should be handle the range of input which is most natural to corresponding problem.

→ It means checking the characteristics & time complexity, space complexity, simplicity, generality & range of input.

6] IMPLEMENTATION OF ALGORITHM

→ If an algorithm consist of objects & related methods then it will be better to implement such algorithm using some object oriented programming language like C++ or Java.

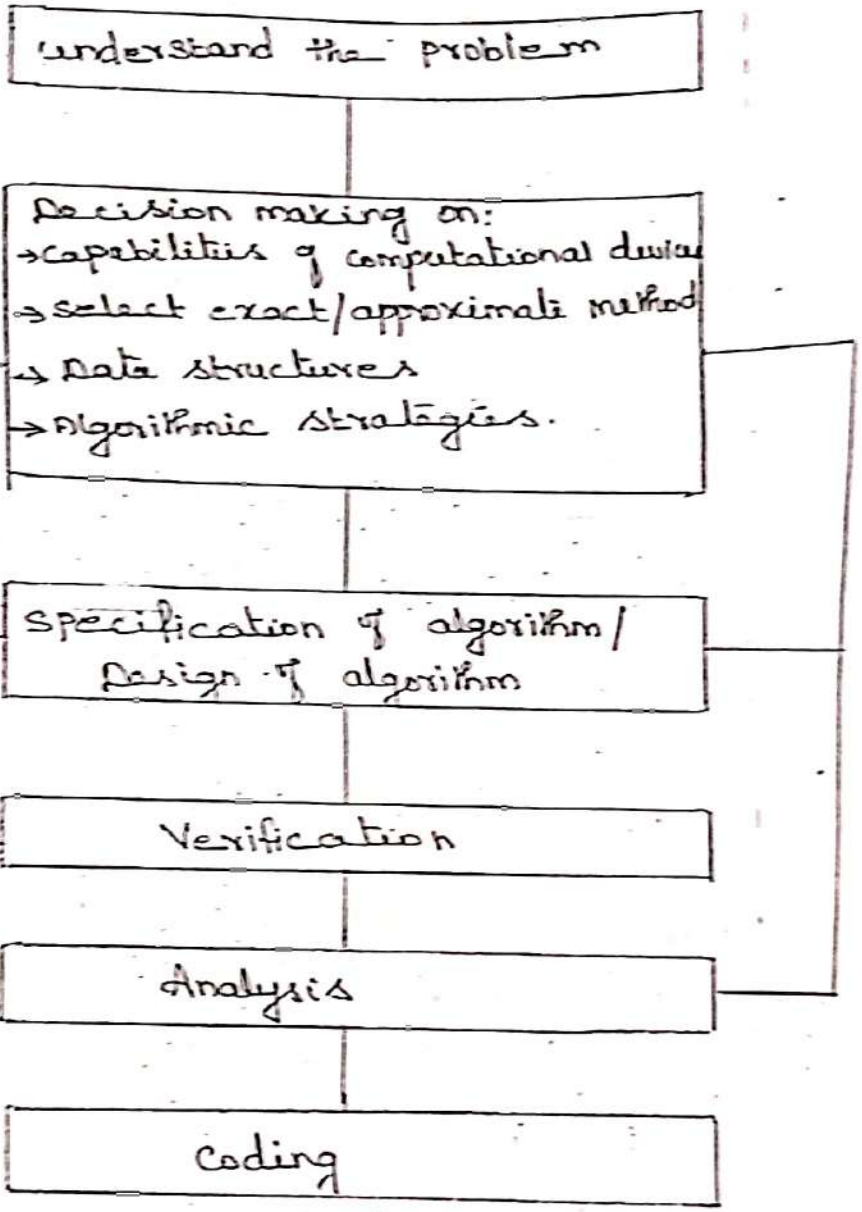


fig: Algorithm design steps.

IV. FUNDAMENTALS OF ANALYSIS OF ALGORITHM: EFFIENCY:

→ The efficiency of an algorithm can be interms of time or space.

→ It checking whether the algorithm is efficient or not means analyzing the algorithm.

→ There is a systematic approach that has to be applied for analyzing any given algorithm.

→ This systematic approach is modelled by a framework called as analysis framework.

IV. ANALYSIS FRAMEWORK:-

→ The efficiency of an algorithm can be decided by measuring the performance of an algorithm.

Two types,

1. Amount of time required by an algorithm to execute.
2. Amount of storage required by an algorithm.
3. This is popularly known as time complexity and space complexity of an algorithm.

I SPACE COMPLEXITY:-

→ It can be defined as amount of memory required by an algorithm to run.

→ To compute the space complexity we use 2 factors.

1. Constant
2. instance characteristics.

The space requirement S(P) can be

$$S(P) = C + S_P$$

→ where 'C' is a constant; ie, fixed part & it denotes the space of input & outputs.

→ The space is an amount of space taken by instruction, variables & identifiers.

→ S_p is a space dependent upon instance 'char' this is a variable part whose space requirement depends on particular problem instance.

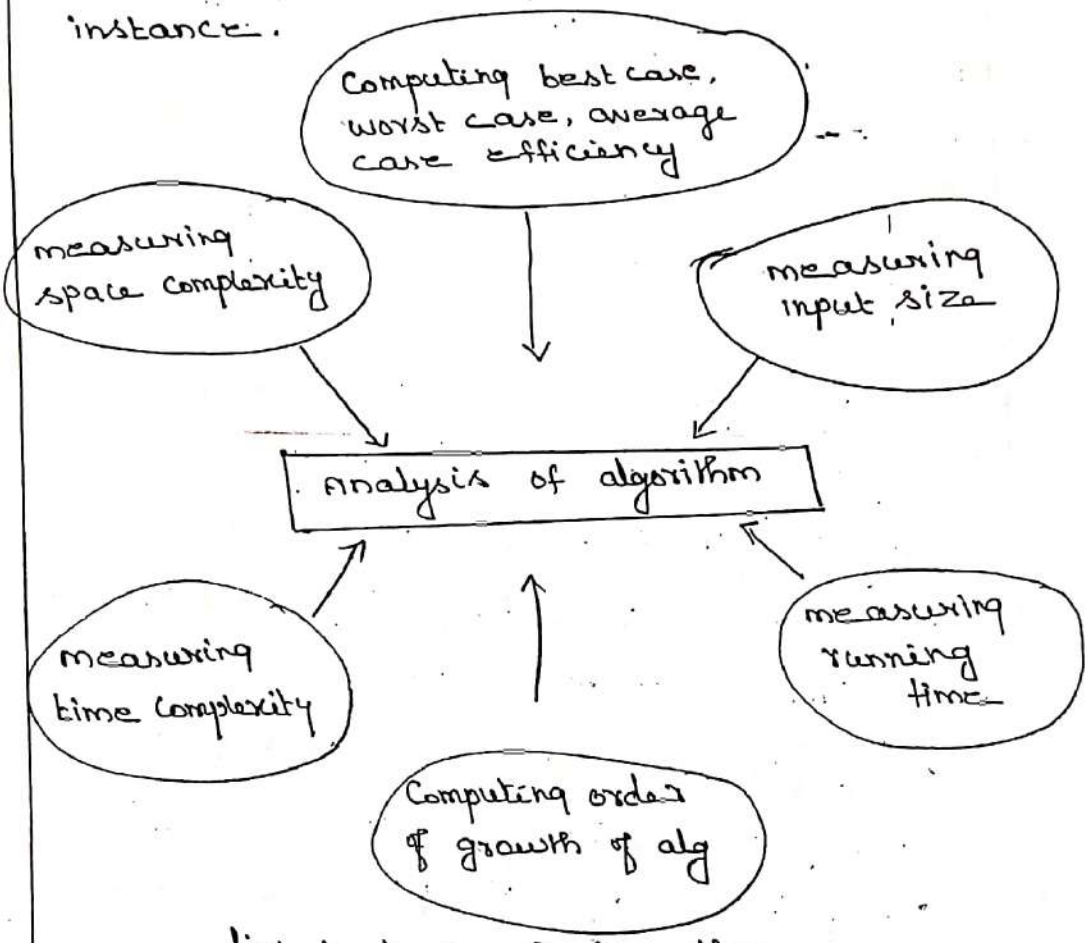


fig. Analysis of Algorithm

2. TIME COMPLEXITY:

→ It is an algorithm is the amount of computer time required by an algorithm to run to completion.

→ It is difficult to compute the time complexity in terms of physically clocked time.

The following factors are,

1. System load
2. Number of other programming running
3. instruction set used.

4. Speed of underlying hardware.

→ It is a terms of frequency count.

→ Frequency count is a count denoting number of times of execution of statement.

ex,

```
for(i=0;i<n;i++)
{
sum = sum + a[i];
}
```

3] MEASURING AN INPUT SIZE:-

→ if the input size is longer than usually algorithm runs for a longer time.

→ we can compute the efficiency of an algorithm as a function to which input size is passed as a parameter.

→ TO implement an algorithm to require prior knowledge of input size.

→ Some times the input size is taken as approximate value.

MEASURING RUNNING TIME

The time complexity is measured in terms of a unit called frequency count.

→ The time which is measured for analyzing an algorithm generally running time from an algorithm.

→ 1st Identify the important operation of an algorithm this operation is called the basic operation.

→ It is not difficult to identify basic operation from an algorithm.

→ Generally the operation which is more time consuming is a basic operation in the algorithm it's basic operation is located in inner loop.

	Problem statement	I/P size	Basic operation
1.	searching a key element from the list of 'n' elements.	list of 'n' elements	comparision of key with every element of list
2	performing matrix multiplication	The two matrices with order nxn	Actual multiplication of the elements in the matrices
3	Computing GCD of two numbers	Two numbers	Division

fig: Basic operations from inputs

→ Then we compute total no. of time taken by this basic operation.

→ TO compute the running time of basic operation by following formula.

$$T(n) = C_{op}(n)$$

- $T(n)$: Running time of basic operation
- C_{op}: Time taken by the basic operation to execute
- (n): No: of times the operation needs to be executed

ORDER OF GROWTH :-

→ To measuring the performance of an algorithm in relation with the input size 'n' is called order of growth.

→ It is clear that the logarithmic function is the slowest growing function.

→ The exponential function 2^n is fastest & grows rapidly with varying i/p size 'n'.

→ The exponential function gives huge values even for small input 'n'.

ASYMPTOTIC NOTATIONS AND ITS PROPERTIES

→ The Asymptotic Notation is defined as to check efficiency of each algorithm.

→ The efficiency can be measured by computing time complexity of each algorithm.

→ It is a shorthand way to represent the time complexity.

→ It can give time complexity as "fastest possible", "slowest possible" or "average time".

The various Notations such as Ω, Θ & O used are called asymptotic Notations.

(i) Big-oh- Notation :-

→ The big-oh- Notation is denoted by 'O'.

→ It is a method of representing the upper bound of algorithm's running time.

→ Using big-oh- Notation, we can give longest amount of time taken by the algorithm to complete

DEFINITION :-

Let F(n) & g(n) be two non-negative function

Let n_0 and constant 'c' are two integers such that n_0 denotes some value of input & $n > n_0$.

→ Similarly 'c' is some constant such that $c > 0$.

→ ~~Similarly 'c' is some constant such that~~ $c \to 0$.

$$F(n) \leq c * g(n)$$

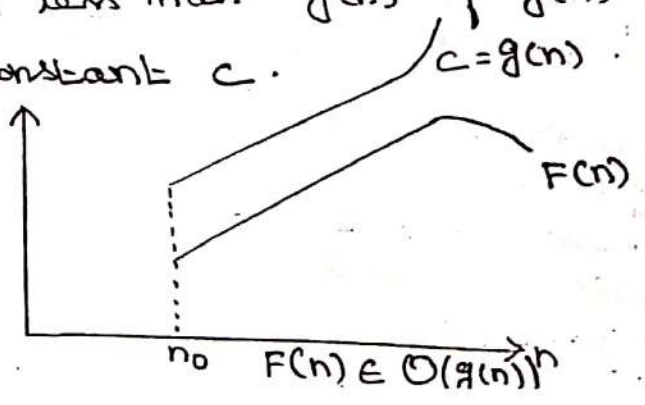
Then,

→ F(n) is big oh of g(n).

→ It is also denoted as $F(n) \in O(g(n))$.

→ In other words,

F(n) is less than g(n) if g(n) is multiple of some constant c.



Example

Consider $F(n) = 2n^2 + 5$ & $g(n) = 7n$

Then if $n=0$

$$F(n) = 2n^2 + 5$$
$$= 2 \times 0^2 + 5$$
$$= 0 + 5$$
$$F(n) = 5$$

$$g(n) = 7n$$
$$= 7 \times 0$$
$$= 0$$

ie, $F(n) > g(n)$

If $n = 1$

$$F(n) = 2n^2 + 5$$
$$= 2 \times 1 + 5$$
$$= 2 + 5 = 7$$
$$F(n) = 7$$

$$g(n) = 7n$$
$$= 7 \times 1 = 7$$

ie, $F(n) = g(n)$

If $n = 2$

$$F(n) = 2n^2 + 5$$
$$= 2 \times 2^2 + 5$$
$$= 2 \times 4 + 5$$
$$= 8 + 5 = 13$$

$$g(n) = 7n$$
$$= 7 \times 2 = 14$$

$F(n) < g(n)$

(i) If n = 3 then

(a) $F(n) = 2(3)^2 + 5$

$= 2 \times 9 + 5$

$= 18 + 5 = 23$

$F(n) = 23$

$g(n) = 7n$

$= 7 \times 3$

$= 21$

ie, $F(n) > g(n)$

Thus for $n > 3$ we get $F(n) > c * g(n)$.

It can be represented as

$2n^2 + 5 \in \Omega(n)$

Similarly any

$n^3 \in \Omega(n)^2$

3. Θ - NOTATION

→ The theta Notation is denoted by Θ.

→ By this method the running time is between upper bound & lower bound.

DEFINITION:-

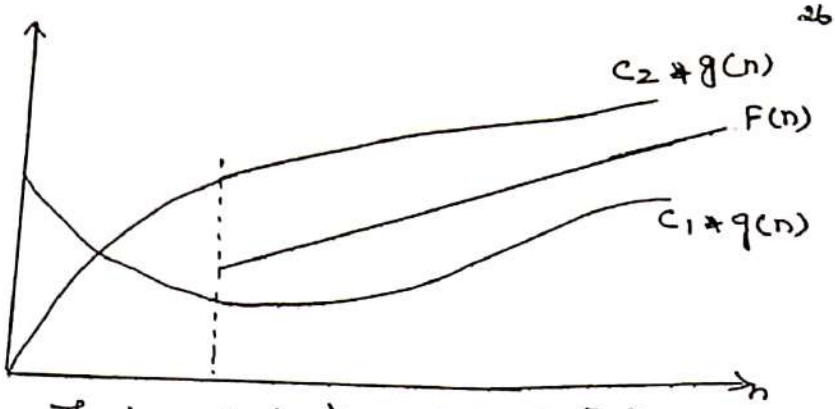
Let F(n) & g(n) be two non negative functions.

→ There are two positive constants, namely c_1 & c_2 such that,

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

Then we can say that

$$F(n) \in \Theta(g(n))$$



Theta notation $F(n) \in \Theta(g(n))$

Example

If $F(n) = 2n+8$ & $g(n) = 7n$.

where $n \geq 2$.

Similarly $F(n) = 2n+8$

$g(n) = 7n$

where $n=2$

$F(n) = 2 \times 2 + 8$

$= 4+8 = 12$

$\boxed{f(n)=12}$

$g(n) = 7n$

$= 7 \times 2 = 14$

$\boxed{g(n)=14}$

$\therefore f(n) < g(n)$

ie, we assume a constant value if

$C_1 = 5$ & $C_2 = 7$ with $n_0 = 2$

Apply formula, where $n \geq 2$

$C_1 * g(n) \leq F(n) \leq C_2 * g(n)$

$5n \leq 2n+8 \leq 7n$

The theta notation is more precise with both O & Ω notations.

PROPERTIES OF BIG-Oh-NOTATIONS :-

1. If there are two function $f_1(n)$, $f_2(n)$.
$f_1(n) = O g_1(n)$ & $f_2(n) = O g_2(n)$. then,
$f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$.

2. If there are two functions,
$f_1(n)$ & $f_2(n)$ such that
$f_1(n) = O[g_1(n) * f_2(n)] = O g_2(n)$. then
$F_1(n) * F_2(n) = O(g_1(n) * g_2(n))$

3. If there exists a function f_1 such that
$f_1 = f_2 * c$. where 'c' is the constant
then f_1 & f_2 are equivalent that means
$O f_1 + f_2 = O f_1 \; O f_2$

4. If $f_n = O g_n$ & g_n & $O h_n$ then $f_n = O h_n$.

5. In a polynomial the highest power terms dominate other term.

6. Any constant value leades to $O \simeq 1$ time complexity that is if $f_n = c$. then it belongs to O time complexities.

7. if $\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$

$f(n) \in O(g(n))$ but $f(n) \in O(g(n))$

(X) MATHEMATICAL ANALYSIS FOR RECURSIVE ALGORITHM

The General plan for analyzing the efficiency of recursive algorithms are,

→ Decide the input size based on parameter n.

→ Identify algorithm's basic operations.

→ Check how many times the basic operation is executed,

1. To find whether the execution of basic operation depend's upon the i/p size n.

2. To determine worst, average, best case for input of size n.

3. If the basic operation depends upon worst case, average case, best case, then that has to be analyzed separately.

4. Set up the recurrence relation with some initial condition & expressing the basic operation.

5. Solve the recurrence or atleast determine the order of growth.

→ while solving the recurrence we will use the forward & backward substitution method.

→ The correctness of formula can be proved with the help of mathematical induction method.

Example:-

Computing factorial of some number n.

The factorial of some number can be obtained by performing repeated multiplication

For instance : if n = 5 then.

step 1 : n! = 5!

step 2 : 4! * 5

step 3 : 3! * 4 * 5

step 4 : 2! * 3 * 4 * 5

step 5 : 1! * 2 * 3 * 4 * 5

step 6 : 0! * 1 * 2 * 3 * 4 * 5

step 7 : 1 * 1 * 2 * 3 * 4 * 5 as 0! = 1

Mathematical Analysis:-

Two types

1. Forward substitution
2. Backward substitution

1. Forward substitution:-

The Recurrence relation is,

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

$M(1) = M(0)+1$

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

$M(3) = M(2)+1 = 2+1 = 3$

Backward substitution:-

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

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

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

From the substitution methods we can

establish a general formula as,

$$\boxed{M(n) = M(n-1)+1}$$

Now let us prove correctness of this formula using mathematical induction as follows,

Prove:

$M(n) = n$ by using mathematical induction

Let $n=0$ then

$M(n) = 0$

ie, $M(0) = 0 = n$

Induction: If we assume

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

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

$= n-1+1$

$= n$

$M(n) = n$

Thus the time complexity of factorial function is $\Theta(n)$.

MATHEMATICAL ANALYSIS OF NON-RECURSIVE ALGORITHM.-

The following steps are,

1. Decide the input size based on parameter n.
2. Identify algorithm's basic operations.
3. check how many times the basic operation is executed.
   ↳ Then find whether the execution of basic operation depends upon the input size n.
   ↳ Determine worst, average & best case for input size n.
   → If the basic operation depends upon worst case, best case, average case then that has to be analyzed separately.
4. set up a sum for the no. of times the basic operation is executed
5. Simplify the sum using standard formula and rules.

SUMMATION FORMULA & RULES USED in EFFICIENCY ANALYSIS:-

1. $\sum_{i=1}^{n} 1 = 1+1+1+\cdots+1 = n \in \Theta(n)$
2. $\sum_{i=1}^{n} i = 1+2+3+\cdots+n = \frac{n(n+1)}{2} \in \Theta(n^2)$
3. $\sum_{i=1}^{n} i^k = 1+2^k+3^k+\cdots+n^k = \frac{n^{k+1}}{k+1} \in \Theta(n^{k+1})$
4. $\sum_{i=1}^{n} a^i = 1+a+\cdots+a^n = \frac{a^{n+1}-1}{a-1} \in \Theta(a^n)$
5. $\sum_{i=1}^{n} (a_i \pm b_i) = \sum_{i=1}^{n} a_i \pm \sum_{i=1}^{n} b_i$
6. $\sum_{i=1}^{n} ca_i = c \sum_{i=1}^{n} a_i$
7. $\sum_{i=k}^{n} 1 = n-k+1$, where n & k are some upper & lower limit.

RECURRENCE EQUATION:

The recurrence equation is an equation that defines a sequence recursively.

The following functions are,

$T(n) = T(n-1) + n$ for $n>0$

$T(0) = 0$

This equation is called recurrence relation

$T(0) = 0$ is called initial condition.

The recurrence equation can have infinite number of sequence.

The general solution to the recursive function specifies some formula.

A Recurrence relation is,

$f(n) = 2f(n-1) + 1$ for $n>1$

$f(1) = 1$

Then by solving this recurrence relation is,

$f(n) = 2^n - 1$

When $n = 1, 2, 3 \& 4$.

SOLVING RECURRENCE EQUATIONS:

The recurrence relation can be solved by following methods.

1. Substitution method
2. master's method.

1. substition method:-

The substitution method is a kind of method in which a guess for the solution is made.

There are two types of substitution.

1. Forward substitution
2. Backward substitution.

1. Forward substitution:-

This method makes use of an initial condition in the initial term & value for the next term is generated.

Example:-

Consider a recurrence relation.

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

with initial condition

$T(0) = 0$.

Let,

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

If $n=1$ then

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

$= T(0) + 1$

$= 0 + 1$

$T(n) = 1$

If $n=2$, then

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

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

$= T(1) + 2$

$= 1 + 2$

$T(2) = 3$

If n=3 then

$T(3) = T(2) + 3$

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

$T(3) = T(3-1) + 3$

$= T(2) + 3$

$= 2 + 3$

$T(3) = 5$

∴ By observing above generated equations we can derive a formula.

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

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

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

We can also denote $T(n)$ in terms of big-oh notation as follows,

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

Backward substitution method:-

In this method backward values are substituted recursively in order to derive some formula.

Example:-

Consider, a recurrence relation

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

Initial condition $T(0) = 0$

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

Putting equation ① & ② we get,

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

$T(n) = T(n-1) + n$ —①

$T(n-1) = T(n-1-1) + (n-1)$ —②

$T(n) = T(n-2) + (n-1) + n$ —③

Let

$T(n-2) = T(n-2-1) + (n-2)$ —④

Putting equation ④ in eq ③ we get

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

$\vdots$

$= T(n-k) + (n-k+1) + (n-k+2) + \cdots + n$

If $k = n$ then

$T(n) = T(0) + 1 + 2 + \cdots n$

$T(n) = 0 + 1 + 2 + \cdots n$

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

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

Again we can denote $T(n)$ in terms of big-oh notation as,

$$\boxed{T(n) \in O(n^2)}$$

VISUALIZATION DEFINITION & IMPLICATION

→ computer based visualization systems provide visual representations of datasets intended to help people carry out some task better.

→ These visualization systems are often but not always interactive.

→ The space of possible visualization system design is huge & full of tradeoff. many of the possibilities are ineffective.

The following steps are,

1. There is a human in the decision making loop.
2. The representation is generated by a computer, not by hand.
3. The human visual perception system is the channel of communication.
4. An external representation is used.
5. The detailed structure of the dataset is important.
6. There is an intended task, whether implicit or explicit
7. There is an operational definition of better
8. Interactivity is on the table.

9. Resource limits for humans, computers and displays all matter.
10. The visualization design space is huge & full of trade-off's.
11. most visualization designs are ineffective.

EMPIRICAL ANALYSIS -

An Empirical analysis of algorithm is also an important idea in its own right.
→ Theoretical analysis does not give much of an idea of how well a given algorithm will perform in a specific situations.

Empirical analysis is considered easy.

The five aspects are,

1. Correctness.
2. work done.
3. space used.
4. Simplicity or clarity.
5. optimality.

Doing Empirical analysis:.

It need to,

1. understand the theoretical analysis.
2. decide on what should be measured
3. decide on appropriate hardware
4. decide on an appropriate implementation language

5. decide on appropriate data structures
6. implement the algorithms.
7. implement some form of timing device.
8. create the input data sets necessary to produce the measurement we need.
9. interpret the results.
10. relate the results to the theoretical analysis.

Examples.

1. Students to work on a design for the empirical analysis of linear search
2. Students submit a design document at the end of that week.
3. Insertion sort & Quick sort
4. work on report document

Empirical Analysis of time efficiency.

1. Select a specific or typical sample of inputs.
2. Use physical unit of time.
3. count actual number of basic operations executions.
4. Analyze the empirical data.

Efficiency

1) Worst case : $C_{worst}(n)$ → max over inputs of size n.

Best case: $C_{best}(n)$ → minimum over inputs of size n.

Average case: $C_{avg}(n)$ → "average" over inputs of size n.

2-mark Questions

1. Define the notation of algorithm? Dec 9, May-13

The algorithm is defined as a collection of unambiguous instructions occurring in some specific sequence & such an algorithm should produce output for given set of input in finite amount of time.

---

2. What are six steps processes in algorithmic Problem solving? Dec-09

The steps are,

1. understanding the problem
2. Decision making on
   - ↳ capabilities of computational devices
   - ↳ choice for either exact or approximate problem solving method.
   - ↳ Data structures
   - ↳ Algorithmic strategies.
3. specification of algorithm
4. Algorithmic Verification
5. Analysis of algorithm
6. Implementation or coding of algorithm

---

3. Write the concept of time & space complexity? Dec-12

Time complexity is the amount of time required by an algorithm to execute.

For computing the time complexity the frequency count of the basic operation in the algorithm is computed.

This frequency count is then denoted in terms of asymptotic notations to express the

time complexity of an algorithm. The space complexity is an amount of space required by an algorithm. The space complexity is denoted in terms of asymptotic notation.

4 Differentiate time complexity from space complexity? May-10

Time complexity is amount of time required by a program to execute.

The space complexity is amount of space required by a program to execute.

5 What is recurrence equation? May-10

The recurrence equation is an equation that defines a sequence recursively. It is normally in following form,

$T(n) = T(n-1) + n$ for $n > 0$.
↳ recurrence relation
$T(0) = 0$ → initial condition.

The Recurrence equation can have infinite number of sequences.

6 Define Big-oh notation? May-12

Let $F(n)$ & $g(n)$ be two non-negative function. Let n_0 and constant c are two integers such that n_0 denotes some value of i/p & $n > n_0$ similarly 'c' is some constant such that $c > 0$. we can write

$$F(n) \leq c * g(n).$$

then $F(n)$ is big-oh of $g(n)$. It is also denoted as $F(n) \in O(g(n))$.

If $F(n)$ is less than $g(n)$ if $g(n)$ is multiple of some constant c.

7 Establish the relation between O & Ω ? Dec-10

The notation O represents the upper bound of algorithms running time.

The omega notation Ω represents the lower bound of algorithm's running time.

if f(n) & g(n) are two function then,

f(n) = O(n) if & only if f(n) = O(g(n)) & f(n) = Ω(g(n)).

8 Write any two properties of big-oh Notation? DEC-11

1. If there are 2 functions, $f_1(n)$ & $f_2(n)$ such that $f_1(n) = O(g_1(n))$ & $f_2(n) = O(g_2(n))$ then $f_1(n) + f_2(n) = \max(O(g_1(n)), O(g_2(n)))$.

2. If there are two functions $f_1(n)$ & $f_2(n)$ such that $f_1(n) = O(g_1(n))$ & $f_2(n) = O(g_2(n))$ then $f_1(n) * f_2(n) = O(g_1(n) * g_2(n))$

9 Define algorithm validation? Dec-12

The process of measuring the effectiveness of the algorithm before actually making programs or code from it. in order to know whether the algorithm is correct for valid input is known as algorithm validation. It can be done with the help of mathematical & empirical method.

10 What is average case analysis? May-14

All the possible inputs are considered & the computing time for all of the I/P is calculated. The sum of all the calculated values is then divided by total no. of inputs. is also called as average case analysis.

11 Define program proving & program verification? May-14

It means proving each & every instruction of the program with the help of mathematical theorems. Program verification means checking the correctness of the program.

12 What do you mean by order of growth? May-12

Measuring the performance of an algorithm in relation with the input size 'n' is called order of growth.

13 What do you understand by the term algorithmic strategy? May-10

It is a general approach by which many problems can be solved algorithmically. These problems may belong to different areas of computing. Algorithmic strategies are also called as algorithmic techniques or algorithmic paradigm.

14 What is time space tradeoff? Dec-14

Time space tradeoff is basically a situation where either a space efficiency can be achieved at the cost of time or a time efficiency can be achieved at the cost of memory.

15 What is conditional asymptotic Notation? Dec-14

many algorithms are easier to analyse if we impose conditions on them initially, imposing such condition, when we specify asymptotic value is called conditional asymptotic Notation.

UNIT - II

BRUTE FORCE & DIVIDE & CONQUER :-

I. BRUTE FORCE

-> Brute force is a straight forward approach of solving the problem

-> It is directly based on the problem statement & definitions of concepts that are directly involved in the problem.

-> It is also known as "just do it" approach.

COMPUTING a^n:-

-> Computing a^n:
For a given number 'a' and a non negative integer n.
Find the exponentiation as follows.

$a^n = a*a*a* \ldots *a$ for n times.

2. computing n! : The n! can be computed as $1*2*3* \ldots *n$.

3. performing multiplication of 2 matrices.

4. searching a key value from given list of elements.

II CLOSEST PAIR AND CONVEX HULL - PROBLEM:-

(i) CLOSEST PAIR PROBLEM:-

-> The closest pair problem is finding the two closest points form the set of 'n' points.

↳ The closet pair problem can be considered to be in two dimensional case.

↳ The point is specified by a pair (x,y). hence P = (x,y) is a point on a two dimensional plane.

→ The distance between two points is denoted by euclidean distance.

It denoted as,

$$d(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

A set of points (finite or infinite) on the plane is called as convex-hull problem.

Algorithm:-

```
Algorithm closest points(P)
{
Returns the indices of closest pair of points
min_dist ← ∞
for i ← 1 to n-1 do
for j ← i+1 to n do
dis ← sqrt((xi - xj)^2 + (yi - yj)^2
if dist < min_dist
    min_dist ← dist
    in1 ← i;
    in2 ← j;
    return in1, in2;
}
```

The basic operation in above algorithm is [illegible] distance b/w 2 points.

↳ The closet pair problem can be considered to be in two dimensional case.

↳ The point is specified by a pair (x,y). hence P = (x,y) is a point on a two dimensional plane.

→ The distance between two points is denoted by Euclidean distance.

It denoted as,

$$d(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

A set of points (finite or infinite) on the plans is called as convex-hull problem.

Algorithm:-

```
Algorithm closest points(P)
{
Returns the indices of closest pair of points
min_dist ← ∞
for i ← 1 to n-1 do
for j ← i+1 to n do
dis ← sqrt((xi - xj)^2 + (yi - yj)^2
if dist < min_dist
    min_dist ← dist
    in1 ← i;
    in2 ← j;
    return in1, in2;
}
```

The basic operation in above algorithm is [illegible] distance b/w 2 points.

CONVEX HULL PROBLEM

Definition

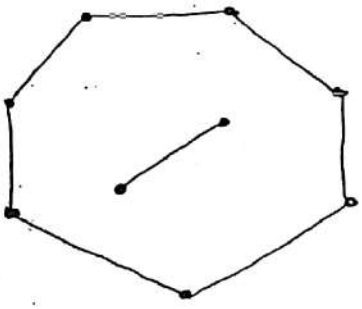
Given a set $S\{P_1, P_2, P_3 \dots P_n\}$ of points in the plane,

→ The convex hull $H(S)$ is the smallest convex polygon in the plane that contains all of the points S.

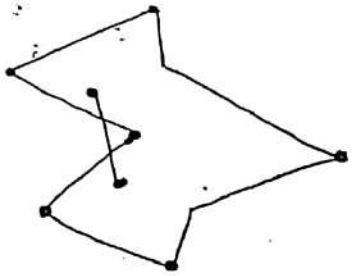
→ The set S is called as convex set.

→ A polygon is convex if and only if any two points from the set forming a line segment with end points entirely within the polygon.

Example :-



Convex



Not a convex

Definition of convex hull:-

A set of points on the plans is called as convex hull.

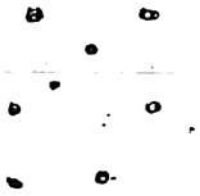
The convex hull of a set S of points is the smallest convex set containing S.

→ If 'S' is a set of two points its convex hull is the line segment connecting these points.

→ If 'S' is a set of 3 points then its convex hull is the triangle.

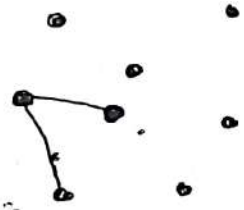
→ Convex hull problem is the problem of constructing the convex hull for a given set 'S' of 'n' points.

Step:1 ⇒

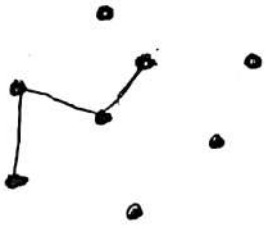


Step:2 ⇒

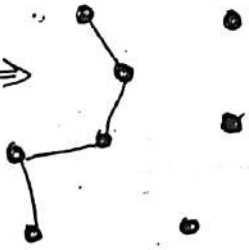
Step:3 ⇒



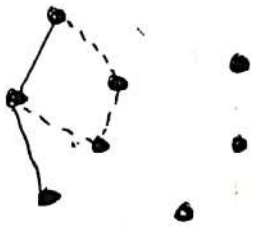
Step:4 ⇒



Step:5 ⇒

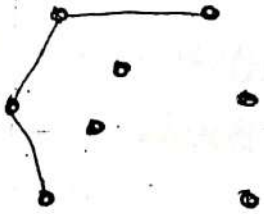


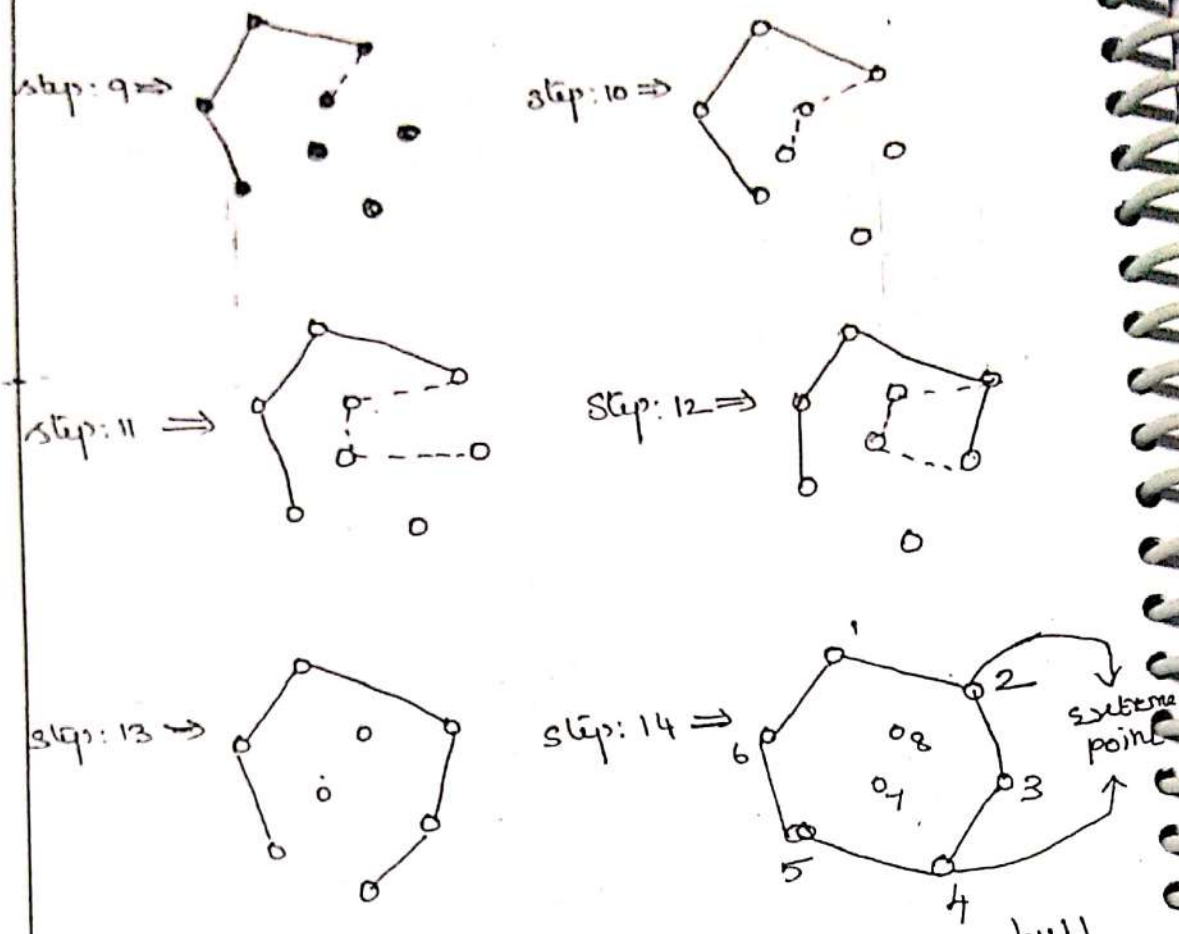
Step:6 ⇒



Step:7 ⇒

Step 8 ⇒





convex hull.

The points {1, 2, 3, 4, 5, 6} are called extreme points.

DEFINITION OF EXTREME POINTS:-

An extreme point of a convex set is a point which is not a middle point of any line segment with end points in the set is also called as extreme points.

Brute force algorithm:-

To find the convex hull using Brute force approach the simple rule is applied.

→ Find out the line segment by connecting 2 points P_i & P_j in such a way all other points

will lie on the same side of the straight line.

→ Repeat it for every pair of points, so that the boundry for convex hull can be formed.

III. EXHAUSTIVE SEARCH:

→ It is a method in which solution is obtained by searching each element of given problem.

→ It makes use of straight forward Brute force's approach.

It contain 3 important problems.

1. Travelling salesman problem.
2. Knapsack problem.
3. Assignment problem.

1. TRAVELLING SALESMAN PROBLEM:-

→ It is a famous problem in the graph theory.

→ There are 'n' cities and travelling salesman to visit each city exactly once and has return to the city from where he has started.

→ This method of problem can be used weighted graph.

→ The vertices of such graph represented as cities.

→ The edges weight specifies the distance between the cities.

→ This problem can also be stated as finding shortest hamiltonian circuit of a graph

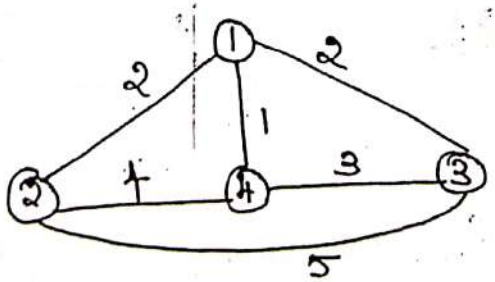
→ The shortest hamiltonian circuit is a cycle in the given graph such that all the vertices of the graph can be visited only one.

I] TRAVELLING SALESMAN PROBLEM :-

Example :-

This is a weighted graph in which weights along the edges represents the distance among the cities.

We have to find hamiltonian circuit the path in which each city is visited exactly once & returning to the city from which it has started initially

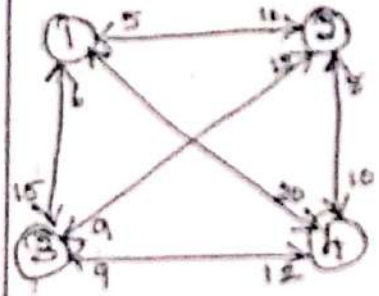


Tour	Path length
1-3-4-2-1	2+3+4+2 = 11 → optimal
1-2-3-4-1	2+5+3+1 = 11 → optimal
1-3-2-4-1	2+5+4+1 = 12
1-2-4-3-1	2+4+3+2 = 11 → optimal
1-4-2-3-1	1+4+5+2 = 12
1-4-3-2-1	1+3+5+2 = 11 → optimal

Thus we have to try each possible path & find the shortest distance which gives optimal tour.

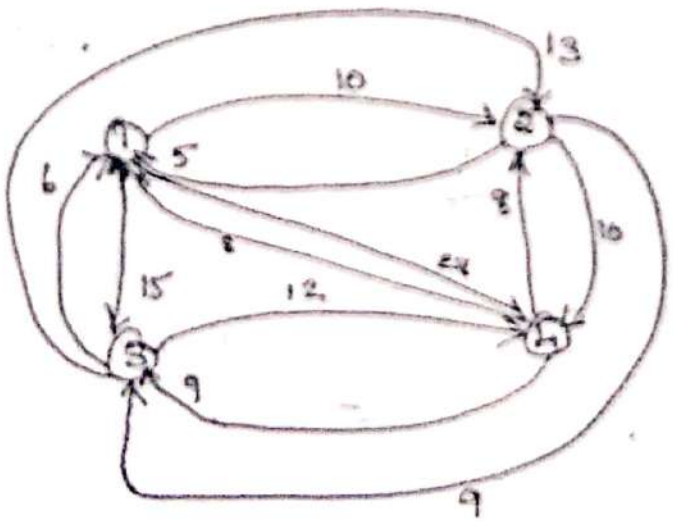
Examples

1)



	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

2)



	1	2	3	4
1	0	10	15	20
2	5	0	9	10
3	6	13	0	12
4	8	8	9	0

A1

5] KNAPSACK PROBLEM :-

This is another popular problem which can be solved using exhaustive search.

It can be stated as follows,

→ There are 'n' objects from i = 1, 2, ... n.

→ Each object 'i' has some weight w_i & values associated with each object is v_i

→ The capacity of knapsack is W.

→ A thief has to pickup the most valuable objects to fill the knapsack to its capacity.

Example :-

Consider a knapsack instance as follows.

i	w_i	v_i
1	2	$1
2	3	$2
3	4	$8
4	5	$6

The capacity W = 8.

Step:1 ⇒ item 1 : $\Sigma w_i = 2$ $\Sigma v_i = \$1$

item 2 : $\Sigma w_i = 3$ $\Sigma v_i = \$2$

item 3 : $\Sigma w_i = 4$ $\Sigma v_i = \$8$

item 4 : $\Sigma w_i = 5$ $\Sigma v_i = \$6$

Step:2

item 1,2 $\Rightarrow \Sigma w_i = 2+3 = 5$ $\Sigma v_i = \$3$

item 1,4 $\Rightarrow \Sigma w_i = 2+5 = 7$ $\Sigma v_i = \$7$

item 1,3 $\Rightarrow \Sigma w_i = 2+4 = 6$ $\Sigma v_i = \$9$

Step:3

item 2,3 $\Rightarrow \Sigma w_i = 3+4 = 7$ $\Sigma v_i = \$10 \rightarrow$ solution

item 1,2,3 $\Rightarrow \Sigma w_i = 2+3+4 = 9$ $\Sigma v_i = 1+2+8 \Rightarrow 11 \rightarrow$ Not feasible

item 1,2,4 $\Rightarrow \Sigma w_i = 2+3+5 = 10$ $\Sigma v_i = 1+2+6 = 9 \rightarrow$ Not feasible

item 2,3,4 $\Rightarrow \Sigma w_i = 3+4+5 = 12$ $\Sigma v_i = 2+8+6 \Rightarrow 16 \rightarrow$ Not feasible

Thus by exhaustive search method, we get the selection of item 2 & 3 for knapsack.

→ The knapsack problem & travelling salesman problem can be inefficiently solved using exhaustive search method.

→ because in this method, each element of the problem domain has to be searched for obtaining solution these problems are also called as NP-hard problem.

3] ASSIGNMEN PROBLEM!-

This is another well known problem for exhaustive search.

→ Consider that there are 'n' people who need to be assigned to execute 'n' jobs.

→ only one person is assigned to execute on one job at a time.

→ Then problem is to find such assignment that gives smallest total cost.

→ The cost can be compuled as cos c[i,j]. ie, ith person assigned to jth job.

Example!-

Consider a smaller instance of problem as follows,

Jobs / persons	Job1	Job2	Job3	Job4
person 1	10	3	8	9
person 2	7	5	4	8
person 3	6	9	2	9
person 4	8	7	10	5

The cost can be obtained by assigning job in various combinations as,

Step:1 ⇒

Job Job Job Job

<1, 2, 3, 4> cost = 10 + 5 + 2 + 5 = 22

1, 2, 4, 3> cost = 10 + 5 + 9 + 10 = 34

1, 3, 4, 2> cost = 10 + 4 + 9 + 7 = 30

1, 3, 2, 4> cost = 10 + 4 + 9 + 5 = 28

1, 4, 2, 3> cost = 10 + 8 + 9 + 10 = 37

1, 4, 3, 2> cost = 10 + 8 + 2 + 7 = 27

Thus by trying 24 permutation (n! = 4! = 24). we can obtain feasible solution.

The feasible solution is.

<2, 1, 3, 4> cost = 3 + 7 + 2 + 5 = 17

Thus we have to generate n! instances to find solution using exhaustive search this also shows how inefficient exhaustive search method is for solving such problems.

Step:2 ⇒

2, 1, 3, 4.> cost ⇒ 3 + 7 + 2 + 5 = 17

2, 1, 4, 3> cost ⇒ [3 + 7 + 9 + 10 = 29

2, 3, 4, 1> cost ⇒ 3 + 4 + 9, 8 = 24

2, 3, 1, 4 > cost ⇒ 3+4+6+5 ⇒ 18

2, 4, 1, 3 > cost ⇒ 3+8+6+10 ⇒ 27

2, 4, 3, 1 > cost ⇒ 3+8+2+8 ⇒ 21

step: 3 ⇒

3, 2, 1, 4 > cost ⇒ 8+5+6+5 ⇒ 24

3, 2, 4, 1 > cost ⇒ 8+5+9+8 ⇒ 29

3, 1, 4, 2 > cost ⇒ 8+7+9+7 ⇒ 31

3, 1, 2, 4 > cost ⇒ 8+7+9+5 ⇒ 28

3, 4, 2, 1 > cost ⇒ 8+8+9+8 ⇒ 33

3, 4, 1, 2 > cost ⇒ 8+8+6+7 ⇒ 29.

step: 4

4, 2, 1, 3 > cost ⇒ 9+5+6+10 ⇒ 30

4, 2, 3, 1 > cost ⇒ 9+5+2+8 ⇒ 24

4, 1, 3, 2 > cost ⇒ 9+7+2+7 ⇒ 25

4, 1, 2, 3 > cost ⇒ 9+7+9+10 ⇒ 35

4, 3, 2, 1 > cost ⇒ 9+4+9+8 ⇒ 30

4, 3, 1, 2 > cost ⇒ 9+4+6+7 ⇒ 26.

Example: 2

PersonJob	Job1	Job2	Job3	Job4
Person1	9	2	7	8
Person2	6	4	3	7
Person3	5	8	1	4
Person4	6	6	9	4

Step: 1 ⇒

1, 2, 3, 4 ⇒ 9+4+1+4 = 18

1, 2, 4, 3 ⇒ 9+4+4+9 = 26

1, 3, 4, 2 ⇒ 9+3+4+6 = 22

1, 3, 2, 4 ⇒ 9+3+8+4 = 24

1, 4, 2, 3 ⇒ 9+7+8+9 = 33

1, 4, 3, 2 ⇒ 9+7+1+9 = 26

Step: 2 ⇒

2, 1, 3, 4 ⇒ 2+6+1+4 = 13

2, 1, 4, 3 ⇒ 2+6+4+9 = 21

2, 3, 4, 1 ⇒ 2+3+4+7 = 16

2, 3, 1, 4 ⇒ 2+3+5+4 = 14

2, 4, 1, 3 ⇒ 2+7+5+9 = 23

2, 4, 3, 1 ⇒ 2+7+1+7 = 17

Step: 3 ⇒

3, 1, 2, 4 ⇒ 7+6+8+4 = 25

3, 1, 4, 2 ⇒ 7+6+4+6 = 23

3, 2, 4, 1 ⇒ 7 + 4 + 4 + 7 ⇒ 22

3, 2, 1, 4 ⇒ 7 + 4 + 5 + 4 ⇒ 20

3, 4, 1, 2 ⇒ 7 + 7 + 5 + 6 ⇒ 25

3, 4, 2, 1 ⇒ 7 + 7 + 8 + 7 ⇒ 29

Step: 4

4, 2, 1, 3 ⇒ 8 + 4 + 8 + 9 = 31

4, 2, 3, 1 ⇒ 8 + 6 + 1 + 6 = 21

4, 1, 3, 2 ⇒ 8 + 4 + 1 + 7 ⇒ 30

4, 1, 2, 3 ⇒ 8 + 4 + 5 + 9 ⇒ 26

4, 3, 2, 1 ⇒ 8 + 3 + 5 + 6 ⇒ 22

4, 3, 1, 2 ⇒ 8 + 3 + 5 + 6 ⇒ 22.