Design and Analysis of Algorithms:-

*Introduction:-

1) To buy a mobile device we have different mobiles Produced by different companies like Samsung, Apple, redmi---, we concentrated on several factors (or) feautures of a mobile like screen size, cost, internal and external memory, camera -- for taking a decision which includes an "Analysis."

2) To solve a problem of an application by using different software products from different software companies like IBM, microsoft, Sun microsystems, mongodb----

Application ⟶ IBM :- 55kB, 50 sec to run

Microsoft :- 25kB, 30 sec

Sun microsystems :- 20kB, 15 sec

*Sunmicrosystems is best among the above 3 companies. as it ~~has~~ occupies low space and executes in less time.

(or)

*Based on time and space.

3) In our syllabus we majorly deals with

note:- Several algorithms with time complexity & Space complexity calculations.

* Design of an Algorithm :-

It deals with minimisation of a cost.

* Analysis of an Algorithm :-

1) It predicts the cost of an Algorithm in terms of resources and performance.

2) Analysis of an Algorithm involves evaluating the following parameters :-

   i, Memory
   ii, Computed time
   iii, Correctness

* 1. Algorithm :- [UNIT-1]

Algorithm :-

1) What is an Algorithm?
2) Main goals of an algorithm
3) Properties of Algorithm
4) Types of Algorithm
5) Uses of Algorithms
6) Study of an Algorithm
7) Process of Design & Analysis of an Algorithm

→ What is an Algorithm?:-

* The term Algorithm was introduced by "Abu Jafar mohammad ib musa" who wrote a textbook on mathematics in the field of Algebra and Numerical Systems in 19th century.

* We adopted the word algorithm from mathematics because it is suitable to our computer operations.

Def:- Algorithm can be defined as "an ordered-Sequence of well defined instructions and effective operations that when executed will always generates the result and terminates in finite amount of time".

* Main goals of an algorithm :-

The Main goal of an algorithm is "always correctness" and "always terminates"

* Properties of an Algorithm:-

i) Finiteness:- Every step of an algorithm must be finite and an algorithm must contain finite number of steps ie, required no. of steps.

ii) Definiteness:- Every step of an algorithm must be preciously stated and there is no ambiguity in the steps.

iii) Effectiveness:- Every step of an algorithm must be effectively stated ie, every operation should be done roughly by pen & paper.

iv) Termination:- Algorithm must be terminated at a particular point if not it is not an algorithm.

v) Input :- (zero (or) more) :-
An algorithm may have zero input (or) 1 input/ 2 inputs - - - that depends on problem statement.

vi) Output:- (one (or) more)
An Algorithm must contain atleast one output and sometimes it may be more than one that depends on the problem statement.

* Types of Algorithm:-

Basically, we have 4 types of algorithms:-

1) Approximation Algorithms
2) Probabilistic Algorithms
3) Infinite Algorithms
4) Heuristic Algorithms

1) Approximation Algorithm:- An algorithm is said to be approximate if it has a repeated value

eg:- π, $\frac{10}{3}$, $\sqrt{2}$

2) Probabilistic Algorithm:- If the solution of an algorithm is uncertain then, we can call it is probabilistic

eg:- 1) Tossing a coin
2) Throwing a die
3) picking a red colour ball from a bag which contains different coloured balls -

3) Infinite Algorithm:- which is not a finite then we can say the Algorithm is infinite

eg:- 1) $\frac{10}{0}$ Divided by zero

4) Heuristic Algorithm:- Giving less input & getting more output

eg:- Any Buisness application

* uses of algorithm:

i, It is very easy to understand

ii, It gives a language independent layout of a Program

iii, By using this basic layout of a program we can design the code in any programming language.

* Study of an algorithm:

→ Generally, any algorithm can be studied in different ways some of them are:-

1) How to devise an algorithm.
2) How to Validate an algorithm.
3) How to Analyse an algorithm.
4) How to test an algorithm.

⇒ How to devise an algorithm?

Devising means "planning", there are several devising techniques to solve a problem like divide & conquer technique, Greedy technique, Dynamic programming technique, Branch and Bound technique —

ex:- 1) For Solving :-

i, quick sort → Divide & conquer
ii, Knapsack → Greedy
iii, Travelling Sales man Problem → Dynamic programming
iv, 8-queens → Back Tracking

⇒ How to validate an algorithm?

* For problem Solving purpose we give some variables, we need to check wether the variables are valid (or) not

eg:- Declaring variables like a,b,c — to solve a problem statement

⇒ How to Analyse an algorithm?

* Analysis of an algorithm is called "analyse" which includes 2 parameters majorly "time-complexity" & "space complexity."

⇒ How to test an algorithm?

i, Any algorithm after completion of analysis then we have to test wether it is valid (or) not by using testing methods

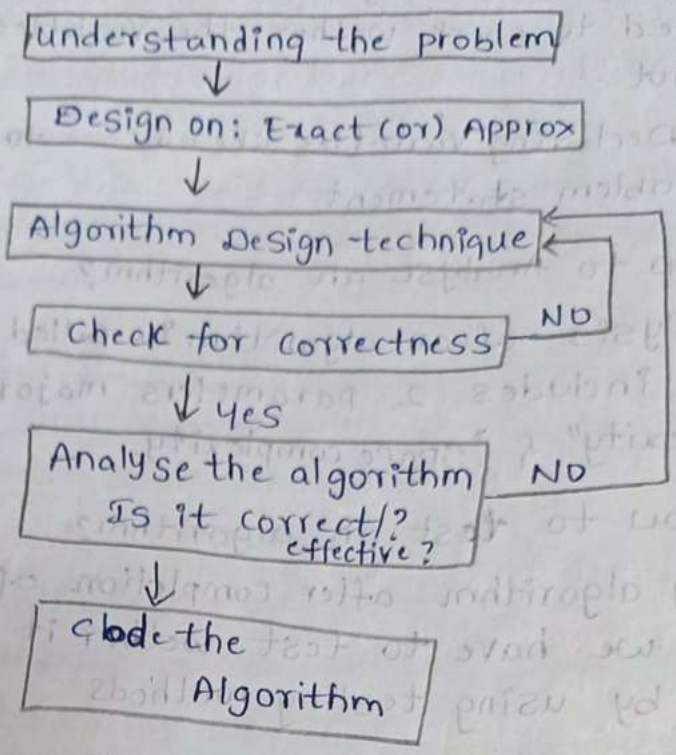
ii, Here, we will discuss 2 things for testing:-

1) Debugging
2) Performance

* Debugging:- Here we can measure errors,

* Performance:- Here we can measure time complexity and space complexity

Process for Designing and analysis of algorithm:



* Psuedocode for expressing an algorithm:-

1) comment:- Comments begins with "//" and continuous until the end of line
2) Blocks are indicated with "{ }" {
3) An Identifier begins with a letter, never starts with a digit, Every statement ends with ';'
4) Assignment of values to a variable can be done as "<variable name>:= <expression>" ie,

eg:- i:=10;

5) Psuedocode supports boolean datatype, the
6) logical operators, Relational operators are also supported ie, &&, ||, !, >, >=, <, <=

6) Elements of multidimensional arrays can be represented with the help of "[ ]"

7) Looping control statements like while, for, (dowhile) repeat-until

a) While :-

```
while(<condition>)
{
// Statements;
}
```

b) for :-

```
for i:= value1 to valuen do
{
   // Statements;
}
```

c) repeat-until :-

```
repeat
{
// Statements;
} until (<condition>)
```

8) Conditional control statements like if, if-else, ~~Goit~~ case

a) if :-

```
if <condition> then <statements>
```

b) if else :-

```
if <condition> then <statements> else <statements>
```

c) case :-

```
case
{
  : <condition 1> : <statement 1>
  |                 |
  : <condition 2> : <statement n>
}
```

9) For input and output we have "read" and "write"

10) For Representing Psuedo code for an algorithm the syntax is :-

Algo Header
Algorithm name (< parameter list >)
{
// Statements;
}

Arguments

Algorithm Body

* Performance Analysis :-
(Evaluating an Algorithm)

(The performance of an algorithm can be analysed)

Performance Analysis, Generally the efficiency of an algorithm can be decided by measuring Performance of an Algorithm in terms of "space complexity" and "time complexity"

Note :- Performance analysis draws a line between what is possible and what is impossib

* Space Complexity :-

- The amount of space required to solve a Program (or) problem is called "space complexity"
- It can be denoted as S(P) (or) S(n)

S(P) = Fixed part + Variable part

Fixed part:- The fixed part is independent of input and output characterstics. This part requires space for the code

ex:- simple variables, constants

Variable part:- This consists of space needed by component variable whose size is dependent on program

note:- Take count=1 for simple variable
Take count=n for an array

Problems:-

ex-1:- Write a psuedo code for sum of three no's and find space complexity.

- Psuedo code:-

```
Alg Sum of three(a,b,c)
{
  Sum:=0;
  Sum:= a+b+c;
  write Sum;
}
```

Space complexity (SP) = F.P + V.P
(a, b, c, sum) + 0
(1+1+1+1) + 0
= 4+0
= 4

ex-2:- write a psuedo code for sum of n numbers & find space complexity

Psuedo code:-

```
Alg Sum of n(a[],n)
{
  Sum:=0;
  for i:=1 to n do
    Sum:= Sum+a[i];
```

return sum;
}

Space complexity $S(P) = F \cdot P + V \cdot P$
$= (i, sum, n) + a[]$
$= (1+1+1) + n$
$= 3 + n$
$= n$

* Time complexity:-
The amount of time required to execute a program is called time complexity. It can be represented as $t(P)$ (or) $t(n)$

$t(P) =$ compile time + Run time

i) Take count = 0 for comments
ii) Take count = 1 for assignment instructions
iii) For loop control statements take count = 'n'
iv)

ex:- Consider the linear search procedure to find best case, average case and worst case of time complexities

Best case:- It is the minimum no of steps to be executed for a given problem (or) program

Average case:- It is average no of steps to be executed for a given problem (or) program

Worst case:- It is maximum no of steps to be executed for a given problem (or) program

i) If the search element is 2 then it requires only one comparision as it is the first element in the array. The time complexity is 'O(1)'. It is considered as 'Best case'.

ii) If the search element is 13 then it requires 5 comparisions. The time complexity is $O(n/2) = O(n)$. It is 'average case'

iii) If the search element is 21 then it requires 9 comparisions as it is the last element in array. The time complexity is 'O(n)'. It is considerd as "worst case".

* Asymptotic notations :-

→ Basically we have 3 types of notations :-

1) Big-oh notation (O)
2) omega notation (Big-omega) (Ω)
3) Theta notation (Big-Theta) (Θ)

Note :- Omega and Theta notations are truly called big-omega notation and big-Theta notation because their exists small omega and small theta notations.

1) Big-oh notation:-

$f(n) = O(g(n))$ [$f(n)$ is Bigoh of $g(n)$] if there exists two +ve constants c and no and $f(n) \leq c \cdot g(n)$ $\forall n, n \geq no$

Let $f(n) = 3n+2$ [$f(n) \leq c \cdot g(n)$]

$g(n) = 4n$

$\therefore$ **c=4**

finding no:-

Take n=1

$3n+2 \leq 4n$

$3(1)+2 \leq 4(1)$

$5 \leq 4$

False

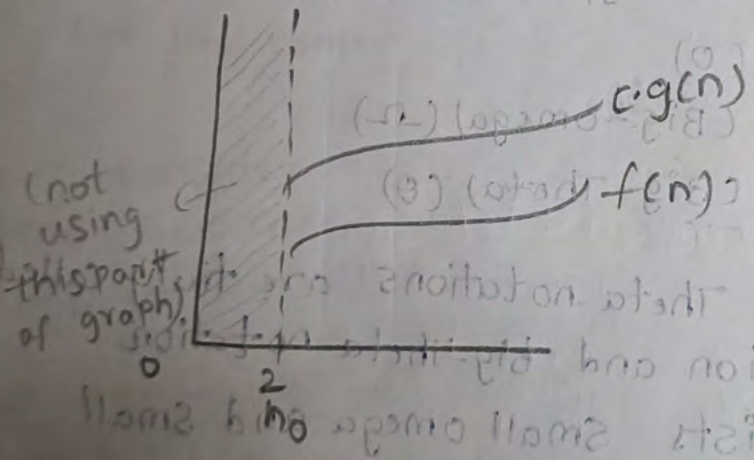
n=2

$3(2)+2 \leq 4(2)$

$8 \leq 8$

True

$\therefore$ **no=2**



2) $f(n) = 10n^2 + 5n + 1$

$c \cdot g(n) = 11n^2$

$c = 11, g(n) = n^2$

n=1 $\quad 10(1)^2 + 5(1) + 1 \leq 11n^2$

$\Rightarrow 10+5+1 \leq 11(1)^2$

] if there
$n) \leq c \cdot g(n)$

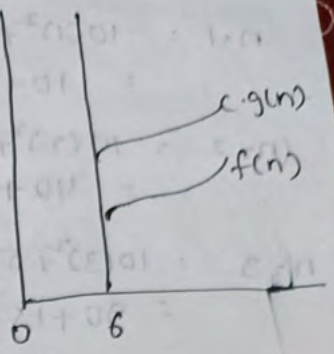
$n=2$: $10(2)^2+5(2)+1 \leq 11(2)^2$
$= 40+10+1 \leq 44$ (F)

$n=3$: $10(3)^2+5(3)+1 \leq 11(3)^2$
$= 90+15+1 \leq 99$ (F)

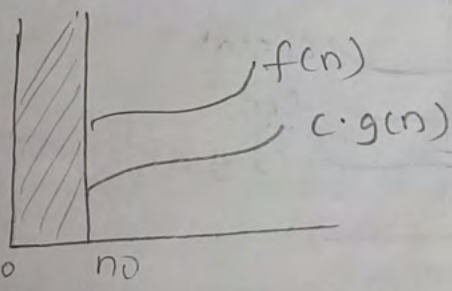
$n=4$: $10(4)^2+5(4)+1 \leq 11(4)^2$
$= 160+20+1 \leq 176$ (F)

$n=5$: $10(5)^2+5(5)+1 \leq 11(5)^2$
$= 250+25+1 \leq 275$ (F)

$n=6$: $10(6)^2+5(6)+1 \leq 11(6)^2$
$= 360+30+1 \leq 396$
$= 391 \leq 396$ (True)



2) Big omega :- $f(n) = \Omega(g(n))$ if and only if there exists 2 positive constants c, n_0 such that $f(n) \geq c \cdot g(n)$ $\forall n$, $n \geq n_0$



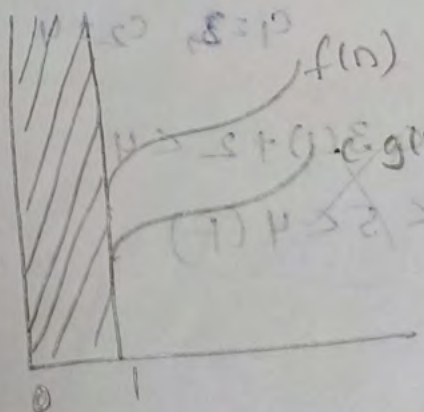
Ex :- 1) Let $f(n) = 3n+2$, $g(n) = 4n$

$f(n) = \Omega(g(n))$

$\therefore c = 4$

$n_0 = 1$ $3n+2 \geq 4n$
$= 3(1)+2 \geq 4(1)$
$= 5 > 4$ (True)

$\therefore \boxed{n_0 = 1}$



* write the psuedo code for matrix multiplication & find the time complexity.

Psuedo code:

```
Alg matrixmul (a[][], b[][])
{
    c[][], i, j, k;
    for i:=1 to n do
      for j:=1 to n do
        c[i][j]:=0;
        for k:=1 to n do
      c[i][j] := c[i][j]+ (a[i][k] * b[k][j]);
      write c[i][j];
}
```

Time complexity

$$\sum_{i=1}^{n} \cdot \sum_{j=1}^{n} \cdot \sum_{k=1}^{n} 1 = \sum_{i=1}^{n} \cdot \sum_{j=1}^{n} n \quad \because \left[\sum_{k=1}^{n} = n\right]$$

$$= \sum_{i=1}^{n} \frac{n(n+1)}{2} \quad \left[\sum_{j=1}^{n} \frac{n(n+1)}{2}\right]$$

Sum of 'n' natural no's

$$= \sum_{i=1}^{n} \frac{n^2+n}{2}$$

$$= \sum_{i=1}^{n} \frac{n^2}{2} + \sum_{i=1}^{n} \frac{n}{2} \quad \left[\sum_{i=1}^{n} \cdot n^2 = \frac{n(n+1)(2n+1)}{6}\right]$$

Sum of squares of natural no's

$$= \frac{n(n+1)(2n+1)}{6\times 2} + \frac{n(n+1)}{2\times 2}$$

$$= \frac{n(n+1)(2n+1)}{12} + \frac{n(n+1)}{4}$$

note:

i) is us

ii) We wi for fin

iii) Amon Count

iv) The tir

for matrix multiplication

$= \frac{n(2n^2+n+2n+1)}{12} + \frac{n^2+n}{4}$

$= \frac{2n^3+n^2+2n^2+n}{12} + \frac{n^2+n}{4}$

$= \frac{(2n^3+n^2+2n^2+n+3n^2+3n}{12}$

$= 2n^3+1)$

$\frac{2n^3+3n^2+n}{12} + \frac{n^2+n}{4}$

$\equiv \frac{2n^3+3n^2+n+3n^2+3n}{12}$

$= \frac{2n^3+6n^2+4n}{12}$

$\Rightarrow$ Highest degree = 3

(Time complexity)$_{mm} = O(n^3)$

note:- The no. of times a statement is executed is usually referred as "frequency count".

ii) We will take the frequency count of every operation for finding the time complexity

iii) Among all frequency counts, the max frequency count will be treated as order of algorithm

iv) The time complexity for the following expressions:-

$[\sum_{K=1}^{n} = n]$

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

Sum of 'n' natural no's

$\frac{n(n+1)(2n+1)}{6}$

n of squares of tural no's

1) $n^3 + n\log_2 n + n^3\log_2 n$

Tec T.C = $O(n^3 \log_2 n)$

2) $n^3 + n^3\log_2 n + 2^n + n^2\log_2 n$

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

v) $O(1)$ = constant time
$O(n)$ = Linear time
$O(n^2)$ = Quadratic time
$O(n3)$ = cubic time
$O(2n)$ = exponential time

vi) $1 < n < \log_2 n < n\log_2 n < n^2\log_2 n < 2^n < 3^n < \cdots$

Part-2

n A
i, A
ii,

e

*Th
is

*

Part-2 * Sets and disjoint sets:-

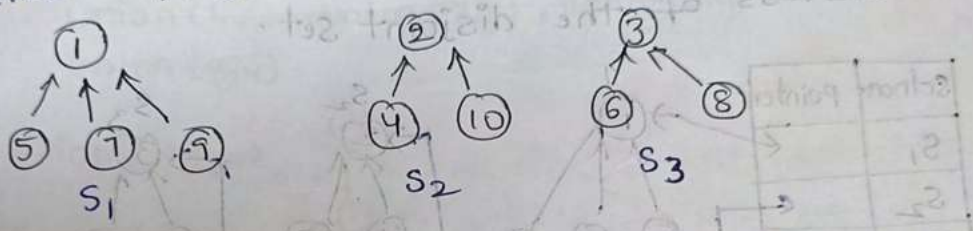
i, A set of elements

i, A collection of elements is called a "set"

ii, If S_i and S_j are 2 sets where $i \neq j$ then we can say S_i & S_j are disjoint sets if no 2 elements that are common in S_i and S_j

ex:- Let us consider the elements 1 to 10 and $S_1 = \{1, 5, 7, 9\}$, $S_2 = \{2, 4, 10\}$, $S_3 = \{3, 6, 8\}$ are disjoint sets

* The tree representation of the above disjoint sets is



* Note:- Majorly, we are going to perform 2 operations on disjoint sets :- 1) union
2) Find

Union :- If S_i and S_j are 2 disjoint sets then $S_i \cup S_j$ gives all elements (x) where x is in S_i (or) S_j

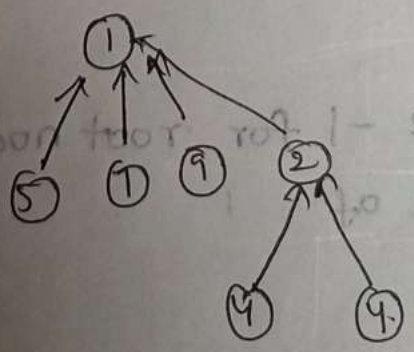
ex :- $S_1 \cup S_2$

$S_1 = \{1, 5, 7, 9\}$, $S_2 = \{2, 4, 10\}$

$S_1 \cup S_2 = \{1, 5, 7, 9, 2, 4, 10\}$

and its resultant tree is :-

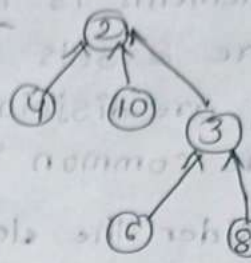
* Tree



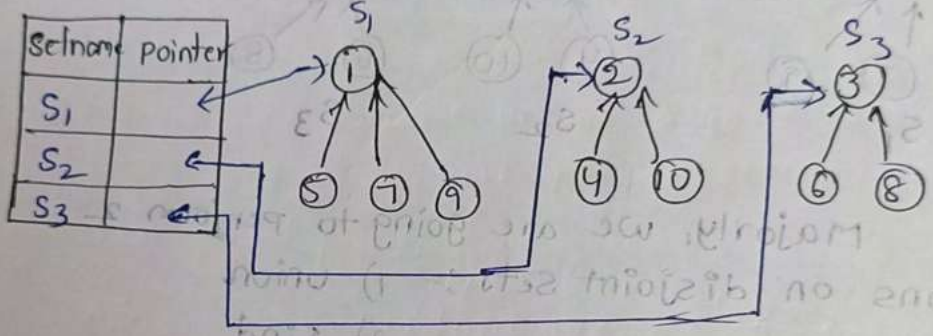
2) $S_2 = \{2, 4, 10\}$, $S_3 = \{3, 6, 8\}$

$S_2 \cup S_3$ $\{2, 4, 10, 3, 6, 8\}$

* "Resultant tree is :-



→ To access the disjoint sets easily we can use set name, keep a pointer to the root of the tree which represents the base address of the disjoint set.



The set of elements which are numbered from 1-10, we represent the tree nodes using an array P[1:10] and its memory Representation (or) array representation of S_1, S_2, S_3 is

i	1	2	3	4	5	6	7	8	9	10
P	-1	-1	-1	2	1	3	1	3	1	2

Note:

1) Take the value -1 for root node of a tree

2) // Take the value of : 1

2) For other elements of an array take its root value.

```
Alg SimpleUnion(i,j)
{  P[i] := j
}
```

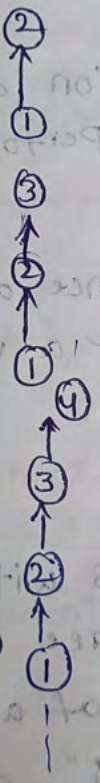
* We pass 2 trees with roots i,j, the simple union operation(i,j) Gives first tree becomes a subtree of second tree.

ex:- find the union operation for the following disjoint sets

union(1,2), union(2,3), union(3,4) --- union(n-1, n).

union(3,4)

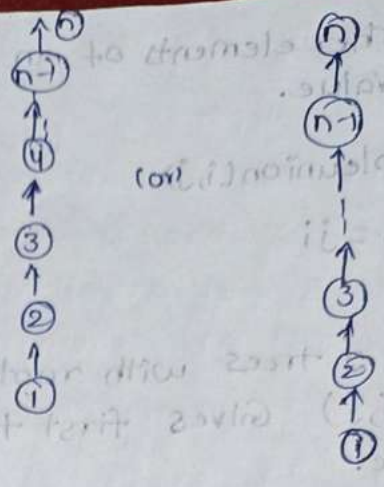
Sol union(1,2) ⇒



union(2,3) ⇒

union(3,4) ⇒

union(n-1, n) =



note:- The time complexity for a simple union operation of n, no.of disjoint sets is "O(n)"

$$\therefore \boxed{t(n) = O(n)}$$

- Although, the simple union algorithm are very easy to stated, their performance characteristics are not good
- To Improve the performance of union operation of disjoint sets we can use a weighted rule for union operation

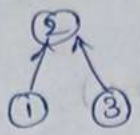
Weighted rule:-

1) Let us consider 2 trees with roots i & j, if the number of nodes of a tree with root i is greater than number of nodes of a tree with root j then make 'j' as subtree to 'i'. otherwise, make "i" as subtree to "j"

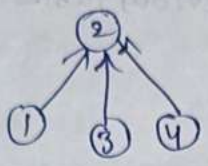
ex:- union(1,2), union(2,3), union(3,4)- union(n-1,n)

union(1,2)

union(2,3)

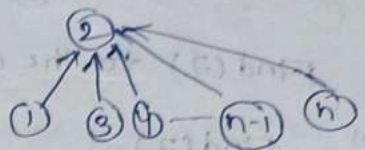


union(3,4)



Since 3, is not root node
So, assigning the value of 4 to node 2

∴ Depth of tree is at level 1 so, time complexity is O(1)



∴ Time complexity = O(1)

T(C) = O(1)

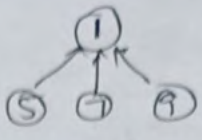
ii) FIND:-

1) Given an element i, find the set containing i.
2) If the node (element) is found in the given tree then it returns its root
3) If the search element is root itself then, it returns '-1'

Alg:- Alg find(i)

```
{
  while (P[i] ≥ 0)
  {
    i = P[i];
    return i;
  }
}
```

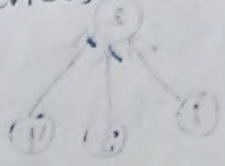
ex:- Take a set size {1,5,7,9} its tree PS



Find (7) to the above set

Sol)

```
Find (7)                                  (P[7] = 1)
while (P[7] ≥ 0)   // from previous table
{                  i > 0  // True
    i = P[i];
}
if i = 1;
  return i;
}
```



P[1] ≥ 0 · (1 is root node)

-1 ≥ 0 (false)

Time complexity for find operation:-

$T(C) = \sum_{i=1}^{n} ti$ // only one loop is in the algorithm (main role in alg)

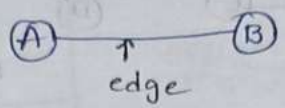
$= \frac{n(n+1)}{2}$

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

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

* connected components:-

1) A Graph is said to be connected if there exists a path between any 2 vertices. ie,



2) If "G" is a connected undirected graph then all the vertices of a graph "G" will be visited on the first call of DFS (or) BFS

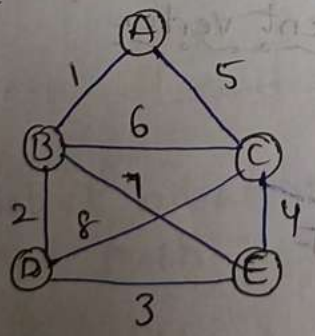
3) If "G" is not connected then we can go for second call of a vertex of BFS (or) DFS.

* Spanning tree:-

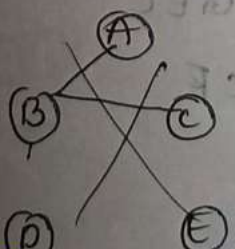
i) A spanning tree is a subgraph of a graph G which is basically a tree which contains all the vertices of G with no ~~rule~~ loops (or) circuits

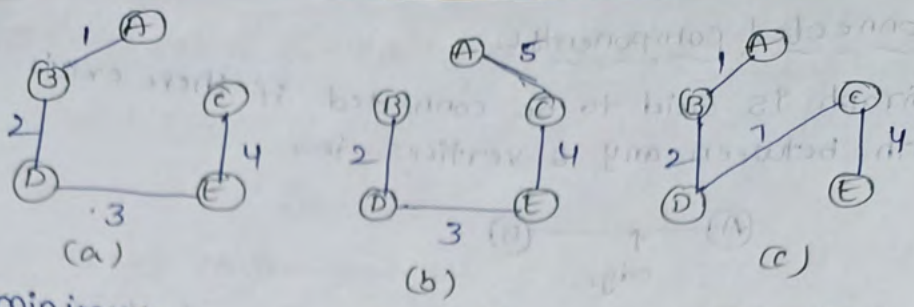
ii) A minimum spanning tree of a weighted connected graph "G" is a spanning tree with minimum cost (or) weight.

ex:-



∴ The possible spanning trees for the above Graph is :-





(a) (b) (c)

minimum cost = 10

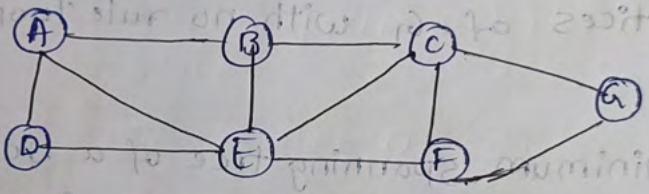
m.c = 14

m.c = 14

* The minimum cost spanning tree is "a" ie "b"

* Tree Traversal techniques
* Basically, we have 2 techniques
1) DFS (Depth first search) [stack]
2) BFS (Breadth first search) [queue]

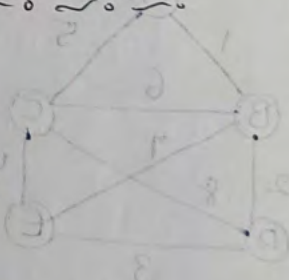
1) DFS:-

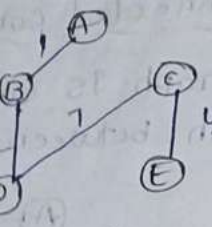


Construct spanning tree by using DFS.

Sol

Vertex	adjacent Vertex
A	B D E
B	A C E
C	G E B F
D	A E
E	A B C D F
F	G E C
G	C F



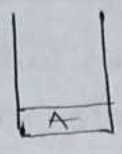


(c)

mc = 14

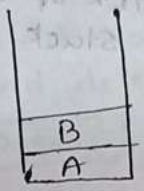
is "a" ie 16

1) moving in alphabetical order
2) consider the empty stack
3) Select the vertex A and push it into stack



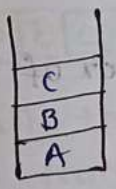
o/p :- (A)

4) Select the adjacent vertex of A ie, consider B and push it into stack



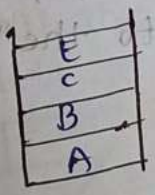
o/p :- (A) → (B)

5) Select the adjacent vertex of B ie, consider C and push it into stack

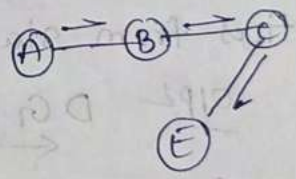


o/p :- (A) → (B) → (C)

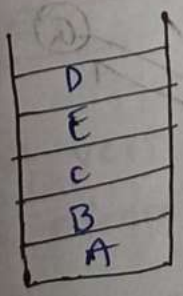
6) Select Adjacent vertex of C ie, consider D E and push it into stack

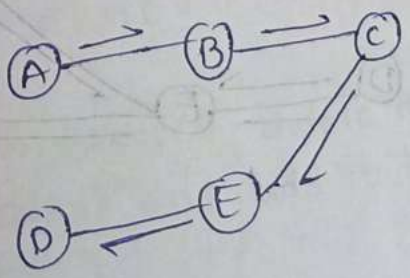


o/p :-

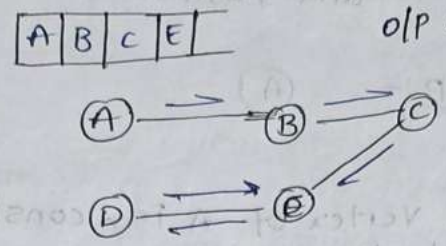


7) Select adjacent vertex of E ie, consider D and push it into stack

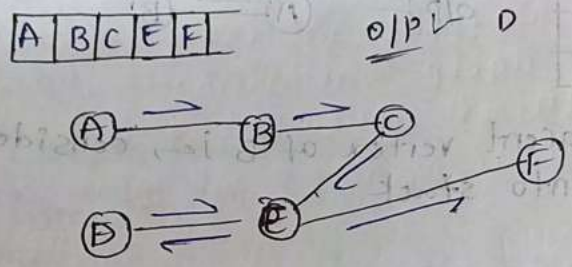




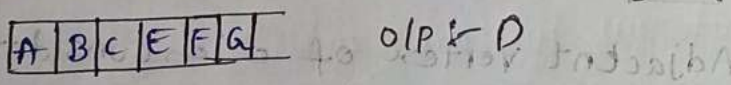
8) Now consider the adjacent vertex of D as all the vertices of D are visited so, roll back to E from D and pop D from stack



9) Now consider the adjacent vertex of E i.e consider F and push it into stack



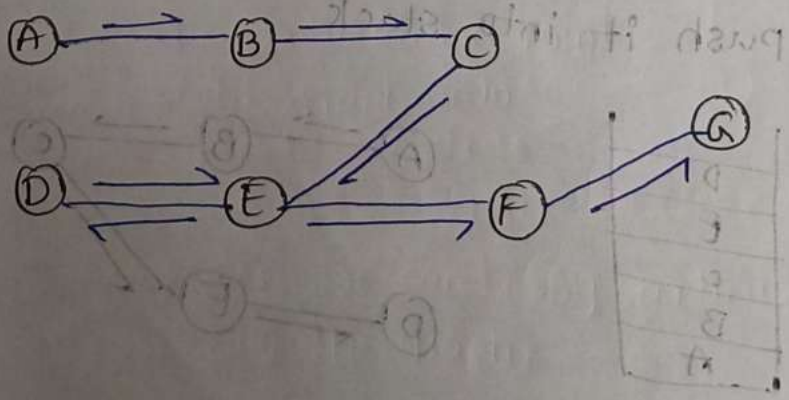
10) Now, consider the adjacent vertex of F and push it into stack



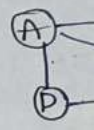
* As, all the vertices are reached pop all the vertices from stack and add to the "O/P"

O/P :- D G F E C B A

∴ Resultant DFS Diagram

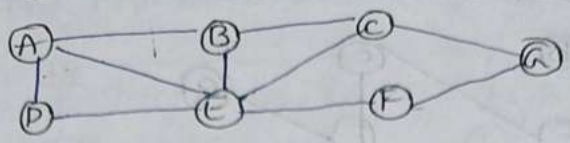


2) Breadth



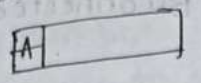
1) consider th
2) Select th queue
A
3) Insert a queue
B
4) Insert queue
5) Insert delete
6) Inser and
7) Inser queu
AS, verti

2) Breadth first search:-



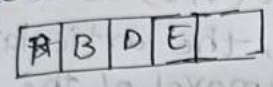
1) consider the empty queue

2) Select the starting vertex A and insert it into queue.



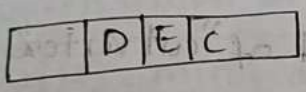
O/P :-

3) Insert all the adjacent vertices of A ie, BDE into queue and delete A from queue & add it to O/P



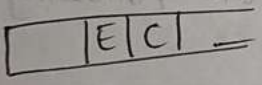
O/P :- A

4) Insert all the adjacent vertices of B ie, CEA into queue and delete B from queue (non visted vertices)



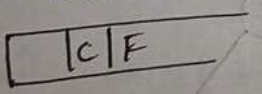
O/P :- AB

5) Insert all the adjacent vertices of D ie- AE and delete D from queue



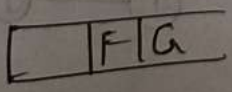
O/P :- A B D

6) Insert all the adjacent vertices of E ie- A B C F G and delete E from queue



O/P :- A B D E

7) Insert all adjacent vertices of C ie- B, G, E into queue & delete C from queue

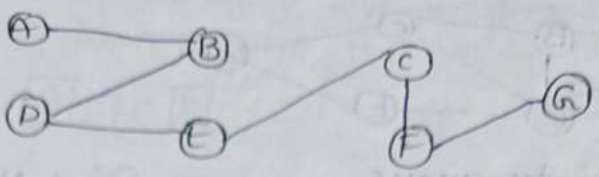


O/P :- A B D E C

AS, all vertices are covered. Delete the all vertices from the queue and add it to O/P

O/P :- A B D E C F G

Resultant BFS Diagram:-



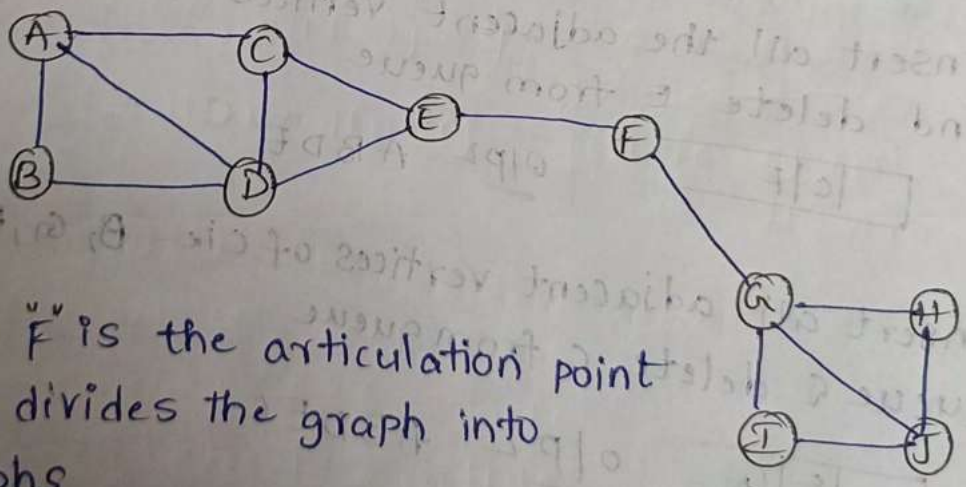
* Biconnected components:-
Here, majorly going to learn two important things ie, "Articulation point" and "Biconnected components"

* Articulation point:-

1) Let us consider the Graph "G" = (V, E) is a connected undirectional graph, the articulation point is a point where, the removal of the verter from the Graph "G" makes a graph into two graphs.

2) Articulation Point is one kind of "cutverter".
[cut verter = Articulation point]

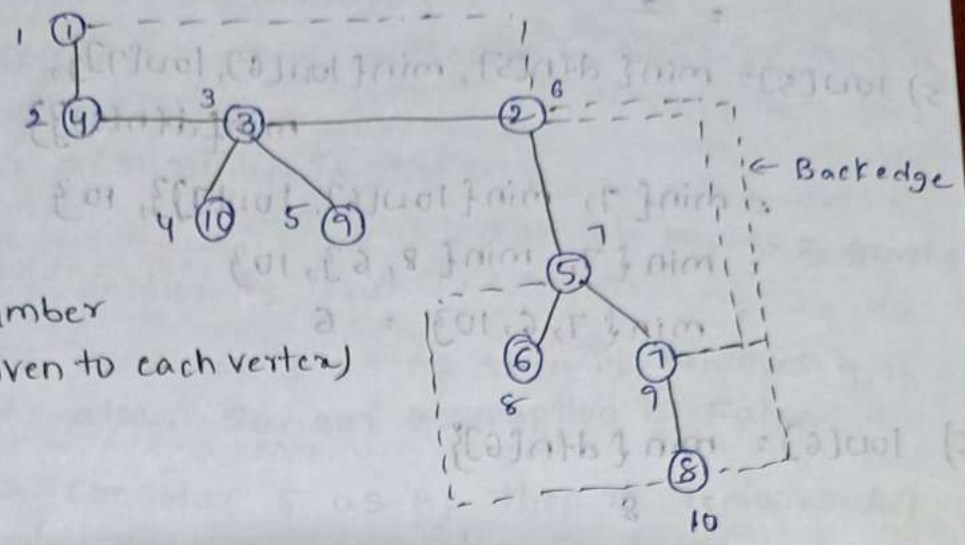
⇒ Find the articulation point of the following graph



* Here, "F" is the articulation point which divides the graph into 2 graphs

(In some examples, we can't say articulation point just by observing the graph)

→ Find the articulation point to the following graph with the help of DFS



Dfn number
(Given to each vertex)

- To find the articulation point to the given graph G we need to find the low values of all the vertices of Given graph G by using the following formula

$$low[u] = \min\{dfn[u], \min\{low[v] \mid v \text{ is child of } u\}, \min\{dfn[w] \mid v \text{ is back edge}\}\}$$

→ Where, "u" is vertex

1) $low[1] = \min\{dfn[1], \min\{low[4]\}, \min\{dfn[2]\}\}$
$= \min\{1, \min\{low[4]\}, 6\} = 1$

2) $low[2] = \min\{dfn[2], \min\{low[5]\}, \min\{dfn[7], dfn[8], dfn[1]\}\}$
$= \min\{6, \min\{low[5]\}, \min\{9, 10, 1\}\}$
$\min\{6, \min\{low[5]\}, 1\}$
$= 1$

3) $low[3] = \min\{dfn[3], \min\{low[10], low[9]\}\}$
$= \min\{3, \min\{4, 5\}\}$
$= \min\{3, 4\}$

4) low(4) = min{ dfn[4] } for
= 2

5) low[5] = min{ dfn[5], min{ low[6], low[7] },
min{dfn[8]}}
= min{ 7, min{ low[6], low[7]}, 10 }
= min{ 7, min{ 8, 6 }, 10 }
= min{ 7, 6, 10 } = 6

6) low[6] = min{ dfn[6]}
= 8

7) low[7] = min{ dfn[7]; min{ dfn[8]}} min{ dfn[2]}}
= min{ 9, 10} = 6

8) low[8] = min{ dfn[8], min{ dfn[2], dfn[5]}}
= min{ 10, min{ 6, 7}}
= min{10,6} = 6

9) low[9] = min{ dfn[9]} = 5

10) low[10] = min{ dfn[10]} = 4

low[v] ≥ dfn[w] (where w = child
v = vertex)

low[4] ≥ dfn[1]
1 ≥ 1 True

low[5] ≥ dfn[2] = 6 ≥ 6 True
low[10] ≥ dfn[3] = 4 ≥ 3 True
low[9] ≥ dfn[3] = 5 ≥ 3 True
low[1] ≥ dfn[4] = 1 ≥ 2 false
low[6] ≥ dfn[5] = 8 ≥ 7 True
low[7] ≥ dfn[5] = 6 ≥ 7 False

* From the

i, If u con
got only
So, our

ii, If we
our a

iii, If u
vertic

iv, If u
verti

* T

* From the above vertex 1,2,3,5 got True values for low[w] ≥ dfn[a]

i, If u consider 1 as a articulation point then we got only 1 graph
so, our assumption is False

ii, If we consider 2 as Ap then it makes 2 graphs our assumption is True

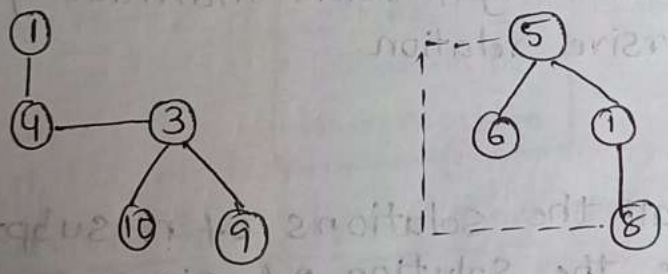
iii, If we consider 3 as Ap then it removes 9,10 vertices also, so, our assumption is False

iv) If we consider 5 as Ap then it removes 6,7 vertices, so our assumption is False

* The articulation point to Given Graph 'G' is 2'

∴ ~~AG~~ = 2

A·P = 2



2: Divide and conquer techniques:-

1) General method:-

- If the problem is large, then divide the problem into n no of sub problems and solve each & every sub problem individually and then combine all the solutions of each problems then we will get the solution of given problem is known as "divide and conquer technique" (DCT)

* Divide and conquer technique describes the following :-

i, Divide:-

Divides the problem into n no of subproblems with size n

ii, conquer:-

Solves each and every sub problem individually by using recursive relation.

iii, Combine:-

It combines all the solutions of n-subproblem which will give the solution of given problem

hniques:-

divide the Problem
l solve each
nd then
ch problems then
ven problem
technique" (DCT)

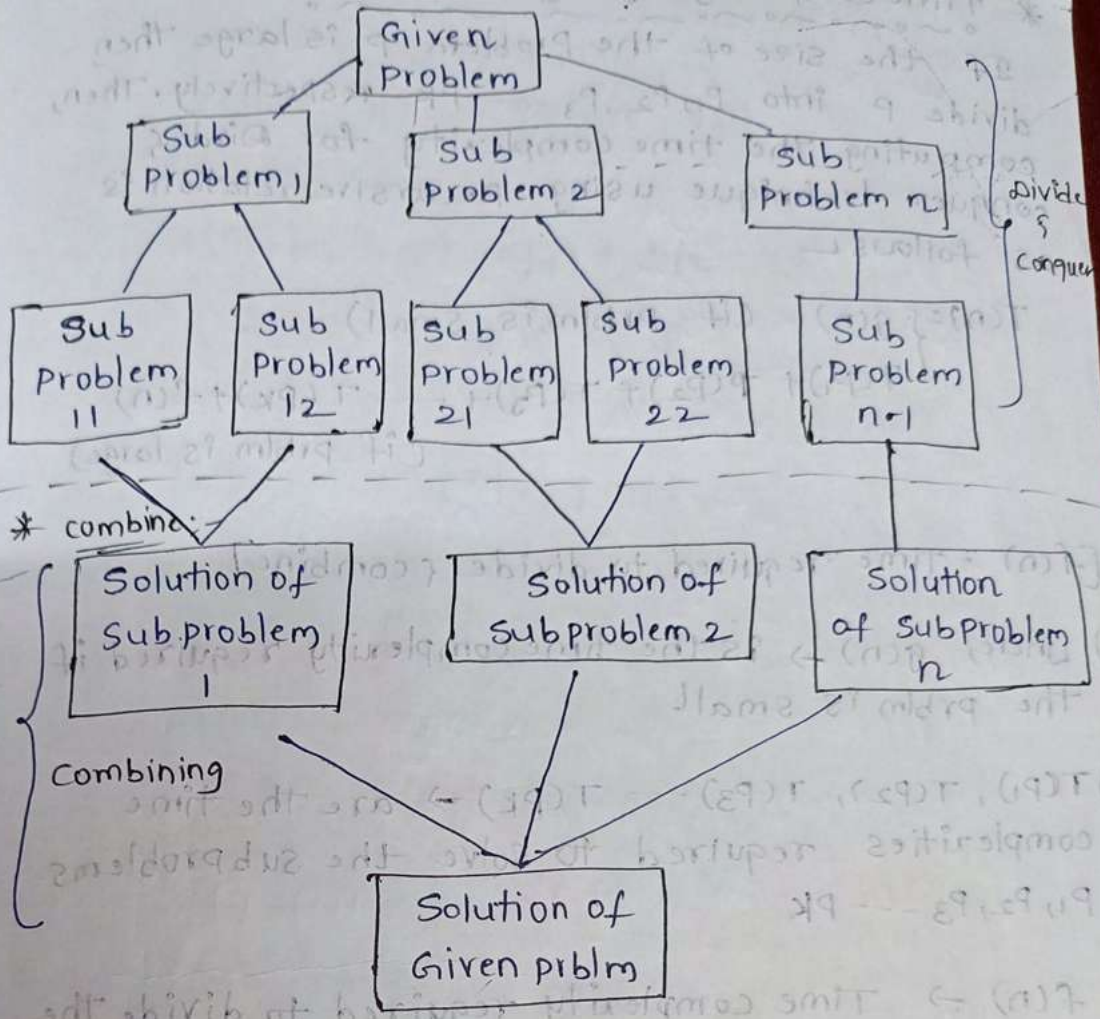
escribes the

of subproblems

ndividually

f n-subproblems
given problem

* Divide and conquer:-



* Algorithm for divide & conquer:-

```
Alg   DandC(P)
{
  if small(P) then
    return solution of P

  else
  {
    Divide the problem(P) into
      P1, P2 -- Pk

    return( DandC(P1), DandC(P2)---
                DandC(Pk))

  }
}
```

* Time complexity of Divide & conquer:-

If the size of the problem P is large then divide P into $P_1, P_2, P_3 \dots P_K$ respectively. Then, computing the time complexity for Divide & conquer technique using Recursive relation is as follows:-

$$T(n) = \begin{cases} g(n) & \text{(if prblm is small)} \\ T(P_1) + T(P_2) + T(P_3) + \dots T(P_K) + f(n) & \text{(if prblm is large)} \end{cases}$$

[$f(n)$: time required to divide & combine]

1) Where, $g(n)$ → is the time complexity required if the prblm is small.

2) $T(P_1), T(P_2), T(P_3) \dots T(P_K)$ → are the time complexities required to solve the subproblems $P_1, P_2, P_3 \dots P_K$

3) $f(n)$ → Time complexity required to divide the Problem into subproblems and, to combine all the solutions of subproblems.

* General method Time complexity:-

$$T(n) = \begin{cases} T(1) & \text{if } n=1 \\ aT(n/b) + f(n) & \text{if } n>1 \end{cases}$$

Where, a, b are constants

* Let

* Let us assume $a=2, b=2, T(1)=1, f(n)=n$

$T(n) = aT(n/b) + f(n)$

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

Put $n = n/2$

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

// Put eq (2) in eq (1)

put $n = n/4$ in (1)

$T(n/4) = 2T(n/8) + n/4 \rightarrow (3)$

Sub eq (3) in eq (1)

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

$T(n) = 4T(n/4) + n + n$

$\boxed{T(n) = 4T(n/4) + 2n} \rightarrow (4)$

Sub eq (3) in eq (4)

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

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

$T(n) = 8T(n/8) + 3n$

$T(n) = 2^3 T\left(n/2^3\right) + 3n \quad (T(n) = aT(n/b) + f(n))$

$\boxed{T(n) = 2^k T\left(n/2^k\right) + nk} \rightarrow (5)$

Let us assume $2^k = n$

Apply "log" on b.s

$$\log_2 2^k = \log n$$

$$\therefore k \log_2 2 = \log n$$

$$k = \frac{\log n}{\log 2}$$

$$\boxed{k = \log_2 n}$$

Sub k value in 5

$$T(n) = 2^k T(n/2^k) + nk$$

$$T(n) = 2^{\log_2 n} T\left(n/2^{\log_2 n}\right) + n \cdot \log_2 n$$

$$T(n) = n\, T(n/n) + n \cdot \log_2 n$$

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

$$T(n) = n + n \log_2 n \qquad \because [T(1) = 1]$$

$$\boxed{T(n) = O(n \log_2 n)}$$

* Merge Sort:-

* sorting
sortin
order

Merge So
*Merge S
1) 2 Way
2) Recu

1) 2 Way
1) 2 Way
where

ex:-

Sort

Sort

* Sorting:-

Sorting the unordered list either in ascending order (or) Descending order is called sorting

Merge Sort:-

*Merge Sort can be done in 2 ways:-

1) 2 way (or) k way merge sort
2) Recursive method

1) 2 way method:-

1) 2 way mergesort is also known as k-way mergesort where k=2

ex:- 26 5 7 15 33 8 1 52 99 17 (Initially take 2 no's as a pair)

(26,5) (7,15) (33,8) (1,52) (99,17)

Sort (5,26) (7,15) (8,33) (1,52) (17,99)

Merge Merge

(5,26,7,15) (8,33,1,52) (17,99)

Sort (5,7,15,26) (1,8,33,52) (17,99)

Merge

(5,7,15,26) (1,8,33,52,17,99)

Sort (5,7,15,26) (1,8,17,33,52,99)

Merge

(5,7,15,26,1,8,17,33,52,99) Sort

(1,5,7,8,15,17,26,33,52,99)

*final sorted order

2) Recursive method :-

26	5	7	15	33	8	1	52	99	17
0	1	2	3	4	5	6	7	8	9

mid = $\frac{0+9}{2}$ = 4.5 = 4 consider the 1st sublist

26	5	7	15	33
0	1	2	3	4

=> mid = $\frac{0+4}{2}$ = 2

26	5	7	15	33
0	1	2		

mid = $\frac{0+2}{2}$ = 1

26 | 5 | 7 | 15 | 33

1st 2nd

Now, combine the 2 elements

(26, 5) (7) | (15, 33)

~~Merge~~ Sort

(~~26~~ 5, 26) (7) | (15, 33)

Merge

(5, 26, 7) | (15, 33)

Sort

(5, 7, 26) (15, 33)

Merge

(5, 7, 26, 15, 33)

Sort

(5, 7, 15, 26, 33)

Similarly, perform the same procedure for second sublist:

∴ Sorted list : (1, 8, 17, 52, 99)

1st Sublist = (5, 7, 15, 26, 33)
2nd Sublist = (1, 8, 17, 52, 99) } merge

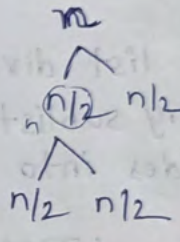
(5, 7, 15, 26, 33, 1, 8, 17, 52, 99)
sort

∴ Final sorted Sublist = (1, 5, 7, 8, 15, 17, 26, 33, 52, 99)

* Algorithm for mergesort:

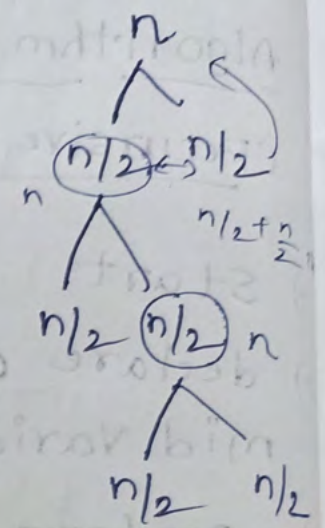
(Recursive method)

1) Start.
2) declare array & left, right, mid variable
3) perform merge function
   mergesort(array, left, right)
   if left > right (or) low > high
   return
   mid = (left + right)/2
   mergesort(array, left, mid)
   mergesort(array, mid+1, right)
   mergesort(array, left, mid, right)
4) stop



* Time complexity for merge sort:-

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



* as list divides in 2 sublists, Then every sublist becomes n then again divides into n/2.

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

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

Put $n = n/2$

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

Put $n = n/4$

$$T(n/4) = 2T\left(\frac{n}{8}\right) + \frac{n}{4} \rightarrow (3)$$

Sub eq (2) in eq (1)

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

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

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

Sub eq (3) in eq (4)

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

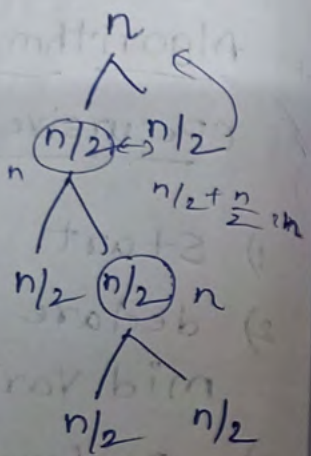
$= 8T(\frac{n}{8}) + n + 2n$

$\boxed{T(n) = 8T(n/8) + 3n}$

* It can be written as :-

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

$\boxed{T(n) = 2^k T(n/2^k) + 3nk} \rightarrow (5)$



→ let us take $2^k = n$

Apply log on b.s

$= \log 2^k = \log n$

$k \log 2 = \log n$

$k = \frac{\log n}{\log 2}$

$\boxed{k = \log_2 n}$

Sub k in eq (5)

$T(n) = 2^{\log_2 n} T(n/2^{\log_2 n}) + 3nk$

$T(n) = 2^{\log_2 n} T(n/2^{\log_2 n}) + n \log_2 n$

$T(n) = n \cdot T(n/n) + n \log_2 n$

$T(n) = n \cdot T(1) + n \log_2 n$ $\boxed{T(1) = 1}$

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

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

$\therefore$ Time complexity for merge sort (Recursive method) $= O(n \log_2 n)$

* Quick sort:-

We can impliment the quick sort by using divide & conquer technique in 2 ways.

i) Without condition.

ii) With condition.

(before pivot → lesser, after pivot → greater)

1) Quicksort without condition:-

(26) 5 37 1 61 11 59 15 48 19

↑ pivot

(5 1 11 15 19) 26 (37 61 59 48)

↑ pivot — 1st sublist — 2nd sublist

(1 (5) 11 15 19) 26 (37 61 59 48)

pivot — pivot

final sorted list:

1 5 11 15 19 26 37 48 59 61

* Quick sort with condition:-

We can implement Quick sort with condition by using the following conditions.

```
if  a[i] < pivot → i++
if  a[j] > pivot → j--
if (i<j) then
  swap i with j
else
  swap i with pivot
```

1) Repeat condition 1 until we get false (i starts with 1 and remember the value of i
(the value of i is where the condition failed)

2) After step 1, Repeat condition 2 until we get false and remember the value of j

3) Now, compare the values of i & j and then perform the swap according to condition 3

eg:- 26 5

Note:- While implementing the Quick sort using condition method track the variable i in forward direction and j in Backward direction

* Time complexity for quicksort:-

* Algorithm:-

1) Choose the pivot element
2) Take two variables to point left & right of the list excluding pivot
3) Left points to the low index
4) Right points to the high
5) While value at left is less than pivot move right
6) While value at right is greater than pivot move left
7) Reapeat the step 5 & 6
8) if left ≥ right, the point where they met is new pivot

* Time complexity for quicksort:-

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

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

Put $n = n/2$

$T(n/2) = 2T(n/4) + \frac{n}{2} \rightarrow (2)$

Put $n = n/4$

$T(n/4) = 2T(\frac{n}{8}) + \frac{n}{4} \rightarrow (3)$

Sub eq (2) in (1)

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

$\boxed{T(n) = 4T(n/4) + 2n} \rightarrow (4)$

sort

t left & right of

en pivot move

than pivot move

they met is

Sub ③ in ④

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

$T(n) = 8T(n/8) + 3n$

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

$\boxed{T(n) = 2^k T(n/2^k) + 3k} \rightarrow ⑤$

Take $n = 2^k$

Apply log on b.s

$\log n = \log_2 2^k$

$\log n = k \log 2$

$k = \frac{\log n}{\log 2}$

$\boxed{k = \log_2 n}$

Sub k value in eq ⑤

$T(n) = 2^{\log_2 n} T\left(n/2^{\log_2 n}\right) + \log_2 n \cdot n$

$= n \cdot T(n/n) + \log_2 n \cdot n$

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

$= n + n \log_2 n$

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

$\therefore$ Time complexity for $T(n) = n \log_2 n$ Quick sort

* Algorithm for min & max

1) Start
2) Consider the array with some numbers
3) if (n==1)
   then min & max = is that particular number
4) else if (n==2)
   then compare the two no's
5) else
6) apply divide and conquer technique & continue until 2 elements occurs in the arrays.
7) Then find min & max from each subarray by combining all subarrays we get the min & max element in the array
8) Stop

* Time complexity for min & max:-

$$T(n) = \begin{cases} 0 & n=1 \\ 1 & n=2 \\ 2T(n/2)+2 & n>2 \end{cases}$$

NOTE:- 2 denote the comparisions min with min, max with max (1+1=2)

$$T(n) = 2T(n/2) + 2 \quad \text{---(1)}$$

put $n = n/2$

$$T(\tfrac{n}{2}) = 2T(n/4) + 2 \quad \text{---(2)}$$

put $n = n/4$

$$T(n/4) = 2T(n/8) + 2 \quad \text{---(3)}$$

Sub eqn (2) in (1)

* Defective

1. we can
$2^k \times 2^k$ wh

2. If k=1

* If k=

$T(n) = 2T[2T(n/4)+2]2$

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

$= 4T(n/4)+6$

$= 2^2 T(n/2^2)+6$ — (4)

sub (3) in (4)

$T(n) = 2T[4T(n/4)+6]+2$

$= 8T(n/8)+12+2$

$= 8T(n/8)+14$

$= 2^3 T(n/2^3)+14$

$T(n) = 2^k T(n/2^k)+2^{k^2}-2$

$T(n) = 2^k T(n/2^k)+2^{k-1}-2$ — (5)

Assume $2^k = n$

$\therefore k = \log_2 n$

$T(n) = 2^{\log_2 n} T(n/2^{\log_2 n})+[2 \cdot 2^{\log_2 n}-2]$

$= nT(n/n)+2 \cdot n-2$

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

$= n+2n-2$

$= 3n$

* Defective chessBoard:-

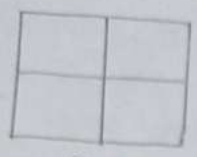
1. we can create n/n chessboard where $n=2$ i.e., $2^k \times 2^k$ where k is always $>=1$

2. If $k=1$, then $(n=2^k)$, $n=2^1=2$

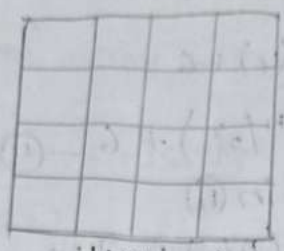
$=2\times2$

* If $k=2$, $n=2^2=4 \Rightarrow 4\times4$ chessBoard.

* If $k=3$, $n=2^3=8 \Rightarrow 8\times 8$ chessBoard.
* If $k=4$, $n=2^4=16 \Rightarrow 16\times 16$ chessBoard.



2×2



4×4

8×8

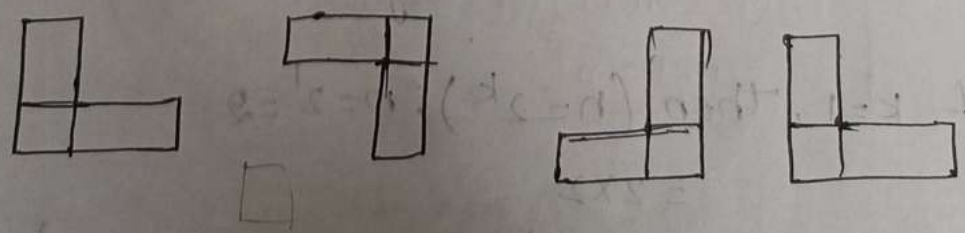
3. In a given $n\times n$ chessboard, we have only 1 check defective.

4) For a given chess Board of size $n\times n$ the no. of defective checks is "1" (one and only one)

5) The no. of non defective checks for a given chess Board of size $n\times n$ is $\boxed{2^{2k}-1}$ (or) n^2-1

6) The no. of non defective checks is always divisible by "3".

7) To Identify the defective check we have to place any one of the following "L-Shape tyre":-



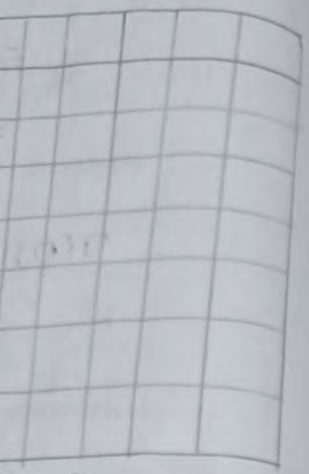
* Let us co
I check d

* Let us c
check de

*) consider
3) Before
4 2×2
"tile" B

4) Apply

oard.
uBoard.



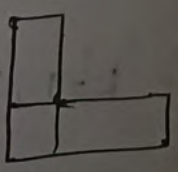
8×8
have only

xn the no. of
y one)

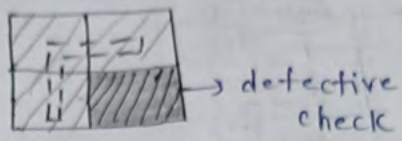
for a given
k-1 (or)

always

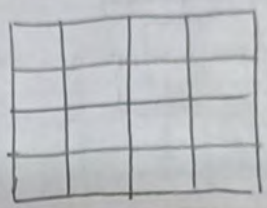
: we have
ing



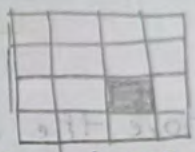
* Let us consider a 2×2 chessBoard with 1 check defective



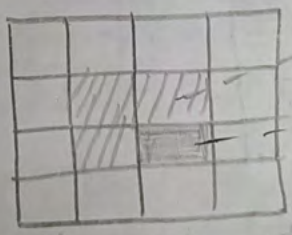
* Let us consider 4×4 chessBoard with 1 check defective



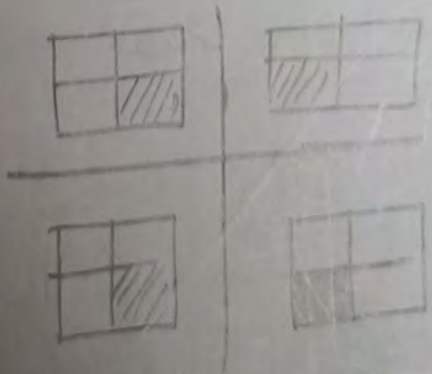
*) consider the defective cheque



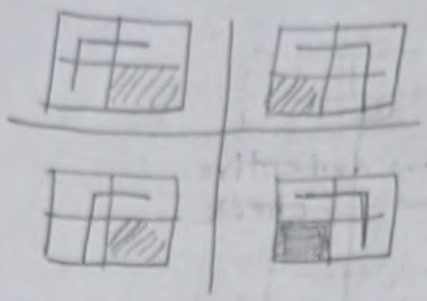
3) Before going to divide the 4×4 chessBoard into 4 2×2's place a duplicate defective "L-shape "tile" Based on origin



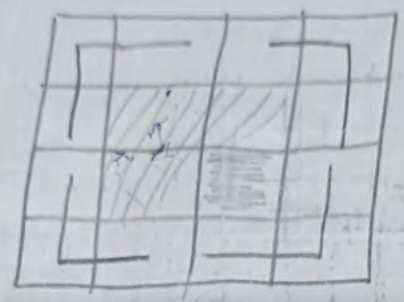
4) Apply divide & conquer



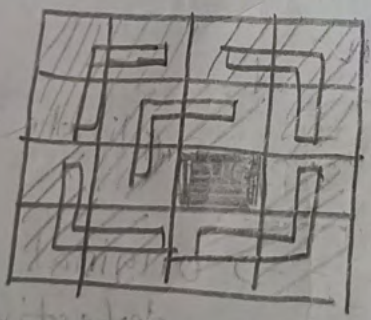
5) Place all the duplicate L shape tyre



6) Now, combine all the 2×2 chessBoards



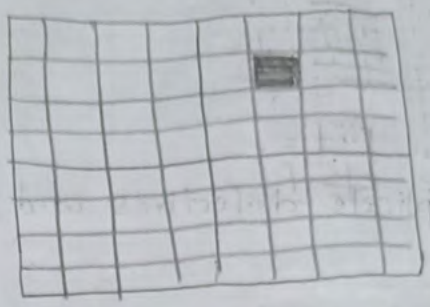
7) Remove the duplicate defective checks and place the L shaped tile



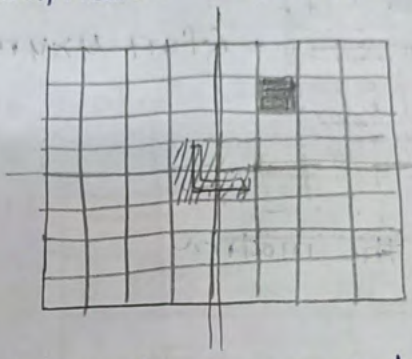
8) The shaded check in the above diagram is the original defective check.

* 8×8 chessBoard

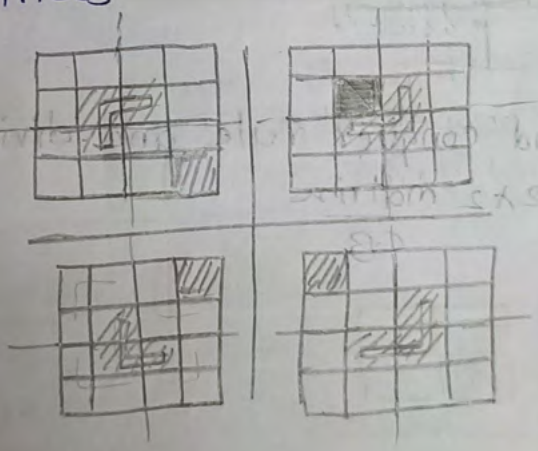
*Let us consider the 8x8 chess Board with the defective check



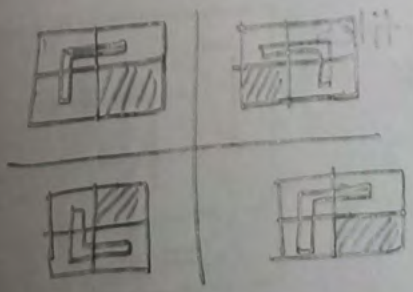
2) Place the duplicate defective L-shaped tile Based on origin



3) Apply divide and conquer rule and divide it into 4x4 matrices

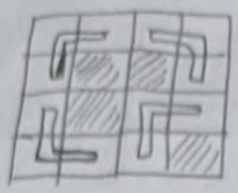


4) NOW, divide each 4x4 chessboard into 2x2 matrices

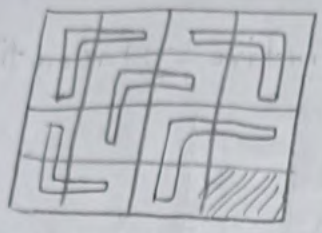


5) Place L-shaped tiles as above

6) Combine the above matrix



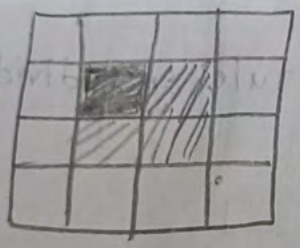
7) Remove the duplicate defectives and place the L-shaped tile



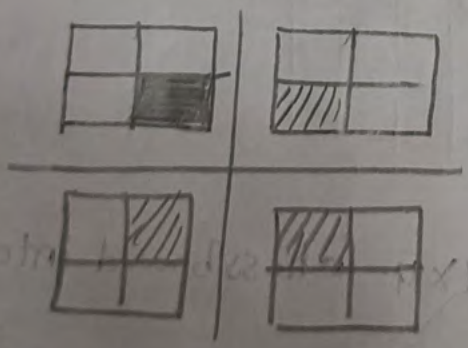
(for 4×4 matrix)

*2nd 4×4 matrix:

1) Let us consider the matrix



2) Apply divide and conquer rule and divide the 4×4 matrix into 2×2 matrix
CB CB

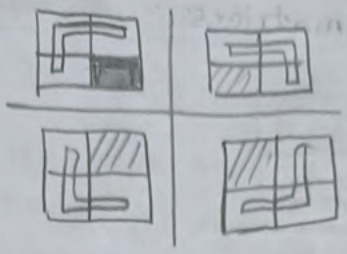


3) Place the L shaped tiles

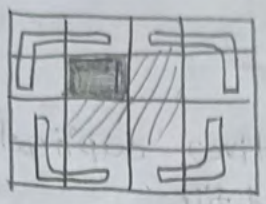
4) Combine

5) Remove
L-shaped

* Simila
2 4×

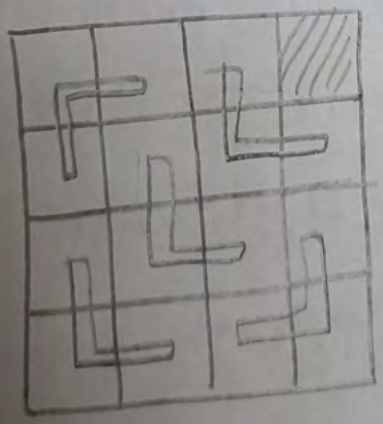


4) Combine all the 4x2 chessboards

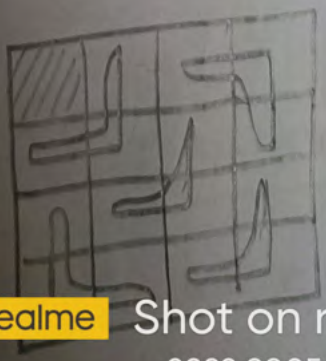


5) Remove the duplicate defective and place the L-shaped tile

* Similarly, perform the same for the remaining 2 4x4 chessBoards we get

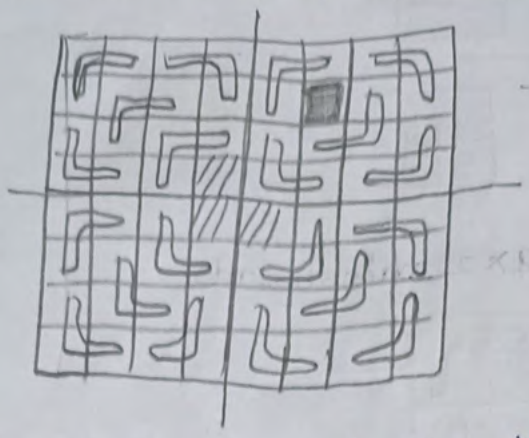


→ 3rd 4x4 matrix

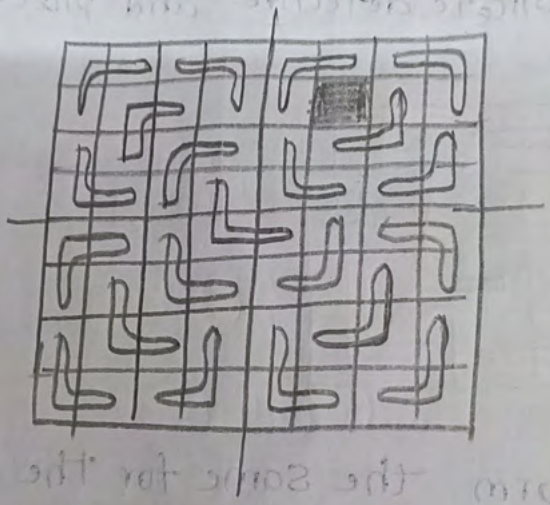


→ 4th 4x4 matrix

i, NOW, combine all 4x4 matrices



ii, NOW, remove the (defective) duplicate defective and place the L-shaped tile



iii, The shaded check in the above diagram is the Resultant defective check.

* Time complexity for Binary search

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

[Because one side of tree we had not touched so $T(\frac{n}{2}) + 1$ is just comparision i.e with mid/root value]

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

→ Sub $n = n/2$

$T(n/2) = T(\frac{n}{4}) + 1 \rightarrow (2)$

→ Sub $n = n/4$

$T(n/4) = T(n/8) + 1 \rightarrow (3)$

→ Sub eq (2) in (1)

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

$\boxed{T(n) = T(\frac{n}{4}) + 2} \rightarrow (4)$

~~T(n)~~ = Sub (3) in (4)

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

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

$T(n) = T(n/2^3) + k3$

$\boxed{T(n) = T(n/2^k) + k} \rightarrow (5)$

assume $n = 2^k$

Apply "log" on b.s

$$\log n = \log_2 2^k$$

$$\log n = k \log 2$$

$$\Rightarrow k = \frac{\log n}{\log 2}$$

$$\boxed{k = \log_2 n}$$

- Sub k in eq (5)

$$T(n) = T(n/\log_2 n) + \log_2 n$$

$$T(n) = T(n/n) + \log n$$

$$T(n) = T(1) + \log n$$

$$T(n) = 1 + \log n$$

$$\boxed{T(n) = O(\log n)}$$

* Time complexity ~~for~~ Binary Search = $O(\log n)$

* Time Complexity for Defective chess Board:-

$T(n) = 4T(n/2) + n \rightarrow (1)$

(Every time we are dividing into 4 parts)

Put $n = n/2$

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

Put $n = n/4$

$T(n/4) = 4T\left(\frac{n}{8}\right) + \frac{n}{4} \rightarrow (3)$

Sub (2) in (1)

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

$T(n) = 16T\left[\frac{n}{4}\right] + \frac{4n}{2} + n$

$T(n) = 16T\left(\frac{n}{4}\right) + 2n + n$

$\boxed{T(n) = 16T\left(\frac{n}{4}\right) + 3n} \rightarrow (4)$

Sub (3) in (4)

$T(n) = 16\left[4T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 3n$

$= 64T\left(\frac{n}{8}\right) + 4n + 3n$

$\boxed{T(n) = 64T\left(\frac{n}{8}\right) + 7n} \rightarrow (5)$

→ This equation is written as

$T(n) = 64T\left(\frac{n}{8}\right) + 4n + 2n + n.$

$T(n) = (2^3)^2 T\left(\frac{n}{2^3}\right) + 2^2 n + 2^1 n + 2^0 n$

$T(n/2^k) + n \cdot 2^k$ → (5)

Let $n = 2^k$

apply "log" on b.s

$\log n = k \log 2$

$\boxed{k = \log_2 n}$

Sub k value in eq (5)

$T = (2^{\log_2 n})^2 T(n/2^{\log_2 n}) + n \cdot 2^{(\log_2 n - 1)}$

$= n^2 \cdot T(n/n) + n(2^{\log_2 n} \cdot \log_2^2)$

$= n^2 \cdot T(1) + n(n-1) \cdot 2^{\log_2 (n-2)}$

$= n^2 + n \cdot (n-2)$

$= n^2 + n^2 - 2n$

$= 2n^2 - 2n$

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

∴ Time complexity for defective chess board is $O(n^2)$.

*// Binary Search tree

* Greedy technique (Part-2)

1) Greedy technique can be applied to "optimisation problems".

2) It constructs the solution through a sequence of steps

3) In each and every step it makes a choice that always looks for the best at that moment

4) The choice made must be :-

i, Feasible :- Represents all possible solutions, it has to satisfy the problems constraints

ii, Locally optimal :- It has to be the best local choice for all possible feasible solutions

iii, Irrevocable :- once the choice is made it cannot be changed

Note :- Greedy technique may not be always leads to an optimal solution

i, Container loading problem :-

1) Loading as many containers as possible into the cargo without sinking.

2) The cargo is loaded with containers where the capacity of cargo is "C"

3) There are 'm' no of containers that are available for loading into cargo

4) The weight of a container is 'w' ie, the weight of ith container is "wi"

5) The weight of each container is always a positive number

Note:- 1) The major constraint of loading containers into cargo is :

Sum of containers weight < c

2) Let us consider $x_i \in \{0, 1\}$ then,

i, $x_i = 0$ stands for the container is not loaded in cargo

ii, if $x_i = 1$ stands for container is loaded in cargo

3) loading the containers into the cargo can be done in the incremental order of weights of containers.

1) The capacity of cargo is 500, the no. of containers is 7 and their weights are shown in the below table

w_1	w_2	w_3	w_4	w_5	w_6	w_7
110	30	29	98	70	210	60

Find the optimal solution for loading the containers into the cargo using greedy technique

Sol) Given

capacity of cargo (c) = 500

no. of containers (m) = 7

& weights are $w_1 = 110$, $w_2 = 30$, $w_3 = 20$, $w_4 = 90$, $w_5 = 70$, $w_6 = 210$, $w_7 = 60$

Step-1:- Initially select the container 3 i.e w_3

$20 < 500$ (True)

Load container 3 into cargo

Step-2:- Next Select the container 2 (W=30)

20+30 < 500

50 < 500 (True)

*Load the container 2 into cargo

Step-3:- Select the container 7 (W=60)

50+60 < 500

= 110 < 500 (True)

*Load the container 7 into cargo

Step-4:- Select the container 5 (W=70)

110+70 < 500

= 180 < 500 (True)

*Load the container 5 into cargo

Step-5:- Select the container 4 (W=90)

180+90 < 500

= 270 < 500 (True)

*Load the container 4 into cargo

Step-6:- Select the container 1 (W=110)

270+110 < 500

380 < 500 (True)

*Load the container 1 into cargo

Step-7:- Select the container 6 (W=210)

380+210 < 500

590 < 500 (False) *Donot load the container

The optimal solution for loading the given containers into the cargo is

w_1	w_2	w_3	w_4	w_5	w_6	w_7
1	1	1	1	1	0	1

* Job Sequence With Deadline:-

1) Let n=4. The profits of Jobs (P_1, P_2, P_3, P_4) = (100, 10, 15, 25) and the deadlines are (d_1, d_2, d_3, d_4) = (2, 1, 2, 1) then find the feasible solution for a given Jobsequence with deadline?

Sol

Sno	Feasible Solution	max Profit
1	–	–
2	J_1	100
3	J_2	10
4	J_3	15
5	J_4	25
6	J_1, J_2	100+10=110
7	J_1, J_3	100+15=115
8	J_1, J_4	100+25=125
9	J_2, J_3	10+15=25
10	J_2, J_4	10+25=35
11	J_3, J_4	15+25=40

max Profit

- The Job Sequence J_1 & J_4 will be the max Profit ies 125 give

2) n=5, (
$(d_1, d_2,$

S.no
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
2
2
2
2
2
2
2

2) $n=5$, $(P_1, P_2, P_3, P_4, P_5) = (20, 10, 5, 15, 1)$
$(d_1, d_2, d_3, d_4, d_5) = (2, 2, 1, 3, 3)$ max deadline = 3.

S.no	Tobsible Solution	max profit
1	-	-
2.	J_1	20
3.	J_2	10
4.	J_3	5
5.	J_4	15
6.	J_5	1
7	(J_1, J_2)	20+10=30
8.	$(J_1\ J_3)$	20+5=25
9.	$(J_1\ J_4)$	20+15=35
10.	$(J_1\ J_5)$	20+1=21
11	(J_2, J_3)	10+5=15
12.	(J_2, J_4)	10+15=25
13.	(J_2, J_5)	10+1=11
14.	(J_3, J_4)	5+15=20
15.	(J_3, J_5)	5+1=6
16.	(J_4, J_5)	15+1=16
17.	(J_1, J_2, J_3)	20+10+5=35
18.	(J_1, J_3, J_4)	20+5+15=40
19)	(J_1, J_4, J_5)	20+15+1=36
20	(J_1, J_3, J_5)	20+5+15=40
21	(J_1, J_2, J_4)	20+10+15=45 max Profit
22.	(J_2, J_4, J_5)	10+15+1=26
23.	(J_2, J_3, J_4)	10+5+15=30
24.	(J_3, J_4, J_5)	5+15+1=21
25.	(J_2, J_3, J_5)	10+5+1=16
26		