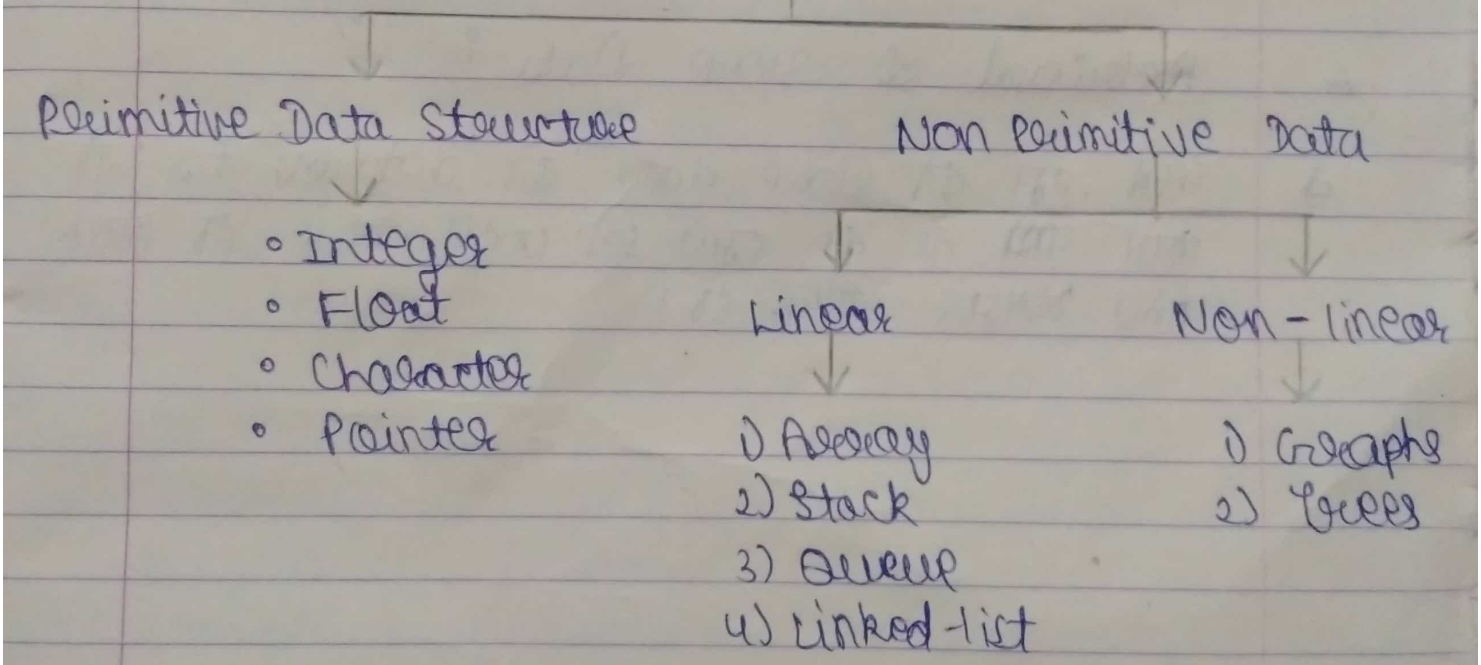
Name = Yash Hokte

Sub = Data Structure

Data structure is a representation of logical relationship b/w individual elements of data

Classification of Data Structure:



Primitive data → It is a data type which is provided by a programming language as a basic building block.

Non primitive data structures are the type which is created or derived by using primitive data types.

Sub → Data Structure

☆ Data Structure :-

⇒ A data structure is a particular way of organizing data in a computer so that it can be used effectively.

two main words are there for Data structure which is organizing & efficiency.

- Data can be defined as an elementary value or the collection of values, for example, student's name & its id are the data about the student.

⇒ There are various operations that can be performed on Data Structures. These are:

① Creation = Creating a data structure according to the requirement.

② Insertion = Inserting values into data structure. There can be three ways to insert elements into data structure: at the beginning, at the end & at the desired location.

simply Insertion means to add the new values into data structures.

③ Traversing = Visiting each element of the data structure at least once.

(4) Search : searching of an element in the given no. of element. It can be searched in two ways:

a) Linear search : Simplest way of searching an element.

b) Binary search : It works on divide & conquer rule.

(5) Sorting : कोई भी data को arrange करना एक particular order like ascending to descending.
-to arrange the data in a particular order

(6) merging : combine the data items of two sorted file into a single so file is known as merging.

Simply किसी भी data की 2 files को एक ही file में combine करना या mix करना उसे हम merging कहेंगे।

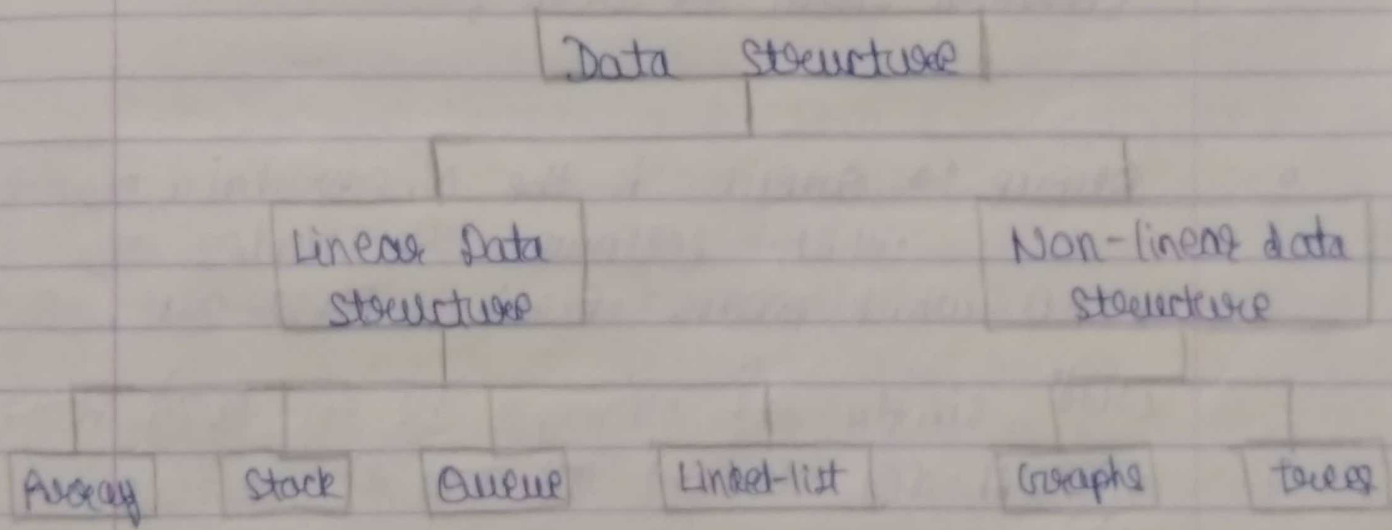
(7) Updation :- कोई भी old data को new data के साथ replace करना या फिर to update (add the new things to that data.

(8) Deleting : Deleting the undesired value from the data structure

* Characteristics of Data Structures:

1) Linear or non-linear = It describes whether the data items are arranged in sequential order, such as with an array, such as with a graph.
Non-linear is in multiple order.

2) Homogeneous = It describes whether all the data items are in a given repository are of the same type. One eg. is a collection of elements in an array, or of various types, such as an abstract data type.



* Stack:

→ The data structure follows the rule of LIFO (Last in first out) where the data last added element is removed first. Push operation is used for adding an element of data on a stack & the pop operation is used for deleting the data from the stack & from the same end.

- Stack में एक ही जगह से elements को push (insert) किया & pop (deletion) किया जाता है simply it means stack has only one end for deletion & insertion both.

imp ★ और जब empty हो stack तो अगर हम उसमे से elements को remove करने की try करेंगे तो तो underflowed कहलाएगा।

★ same way में stack की maximum limit से ज्यादा अगर हम elements को push (insert) करवाएंगे तो इस time Overflow की situation foem हो जाएगी।

- Queue :→ Queue is the linear data structure which follows the rules of FIFO which means First in first out. इसमें simply जो elements को हम पहले insert करवाएंगे वही element भी पहले ही remove होंगे।

Queue ~~Stack~~ में 2 ends होते हैं एक जगह से element (insert) enter होता है जिसे enqueue कहते हैं और एक जगह से element exit (remove) होता है जिसे dequeue कहते हैं।

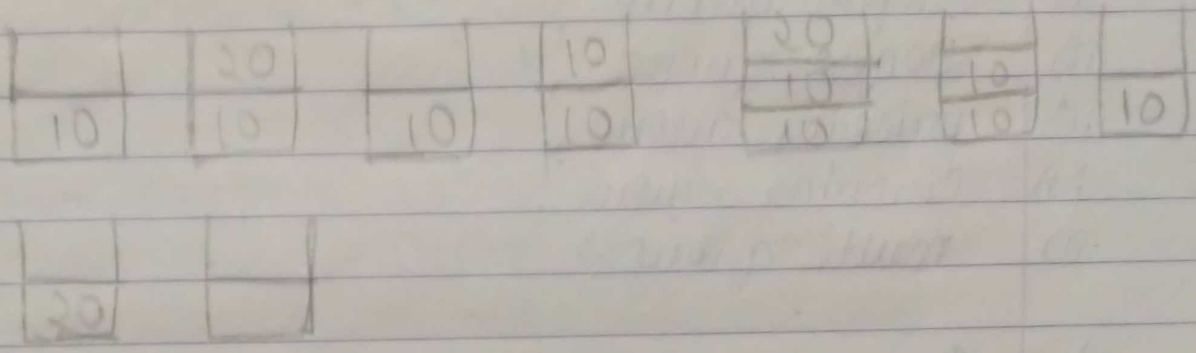
(Stack based question)

Ques The following sequence of operation is performed on a stack:

Push(10), Push(20), Pop, Push(10), Push(20), Pop, Pop, Pop, Push(20), Pop.

What will the sequence of popped value?

Soln



Thus the sequence of Pop is 20, 20, 10, 10, 20

☆ (i) Queue = Queue is the linear data structure which follows the rules of FIFO which simply means First in First Out. इसे simply समझ सा रहा है कि जो element पहले enter होगा वही remove भी पहले ही होगा।

Here the element which insert firstly it remove firstly also.

Queue में 2 अलग ends होते है एक end से insertion होता है means element add होता है जिसे हम Enqueue कहते है और एक दूसरे end से deletion (removing) होता है जो element उसे हम Dequeue कहते है।

Also, जहां से element insert होते हैं we called it [rear] end & जहां से the end from where the element get remove that end we called as [front] end.

☆ there are 5 types of Queue :-

i) Simple queue,
ii) Deque queue,
iii) Priority queue,
iv) Circular queue,
v) multi queue.

→ Deque queue :→

- It means double ended queue, here in this we can insertion & deletion from both the ends (include rear & front).

Now in this there are two types of Deque queue :

i) Input restriction → In this, insertion can be done from only one end & deletion can be both the ends.

ii) Output restriction → In this, insertion can be done from both the sides but deletion can be done from only one end.

Page No.
Date

Early way:

★ Operations on Data structure:

1) Insertion = To add (insert) the new element in the Data structure.

2) Deletion = To remove an element from the data structure.

3) Searching = to find/search an element in the Data structure.

4) Traversing = To visit the data element atleast one time.

5) Sorting = To arrange the element in some logical order like from ascending to descending or from descending to ascending.

6) Merging = Combining two data structure of same type to form a new data structure of same type.

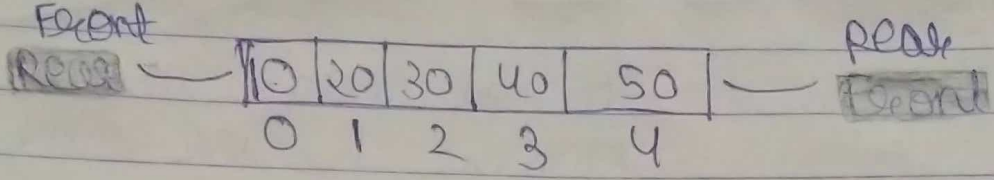
★ Simp

★ Types of Queue:

① Simple Queue = In simple queue, we can perform enque (insertion) at rear end & deque (deletion) at front end which follows the rule FIFO (first in first out). ~~FIFO~~ elements full होने का बाद element को add नहीं कर सकते।

Page No
Date

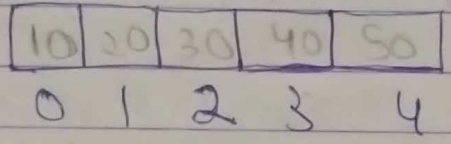
Simple Eg like :→



Simple जिस element को हम पहले enter करवाएंगे वही पहले remove होगा।

In this simple queue, insertion can be done from rear & deletion can be done from front.

→ जब एक queue के structure में कोई भी element ना हो null की situation हो तो उस time हम Front = 0 Front = -1 & rear = -1 लेंगे :



in case of no element present, here we can take Front = rear = -1

Now if i have to enqueue the element :

$$\begin{aligned} rear &= rear + 1 \\ &= -1 + 1 \\ &= 0 \end{aligned}$$

Now, another if I have to enqueue the another element :

rear = rear + 1
= 0 + 1
= 1

Next element 50 will be in 1st position

∴ same concept Front का है :
जो first element insert हुआ होगा वही first remove (deque) होगा :

Front = Front + 1
= -1 + 1
= 0

जब null या तो Front की value -1 थी
that's why -1 लिया फिर deletion कर रहे है।

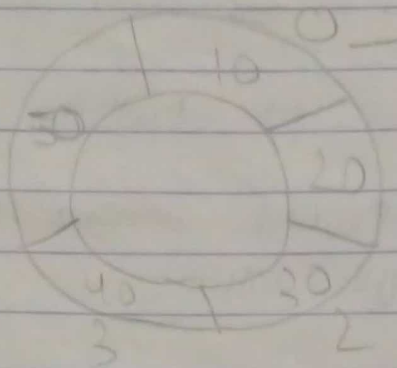
(ii) Circular Queue :- In circular ~~position~~ queue, last position is connected to first position to make circular & the main advantage of circular queue is the utilization of memory.

→ As, circular queue जो है वो circular form में होता है means हम यह कह सकते कि उसका Last point जो है वो first point से connect होगा जैसे circle में सभी interlinked होते है वैसे ही वो obvious सी बात है Last position पर element insert करने के बाद वापस starting point से element insert कर सकते है जिससे memory waste नही जाएगी साथ साथ memory

For null queue :
Front = -1
Rear = -1

Page No.
Date

empty state से जब element insert होता है तब
So we can say that in circular queue
there is utilization of memory means
it can't be waste.

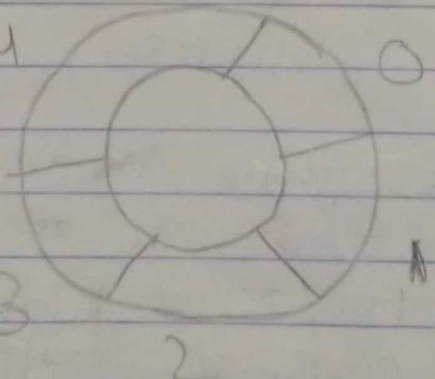


For Enqueue (Insertion) : From rear end

Rear = (Rear + 1) % Maximum size

Que ⇒ Like we will take one example as the circular queue is empty & we have to enqueue (insert) one element which is 10 & its maximum size is 5

Soln ⇒



As we have to insert 10 so before this it is null queue & we know that at the time of empty queue the

Value of Front = Rear = -1

Formula for enqueue the element from rear end for circular queue is:

$$\text{Rear} = (\text{Rear} + 1)\ \%\ \text{Maximum size}$$
$$= -1 + 1\ \%\ 5$$
$$= 0\ \%\ 5$$
$$= 0$$

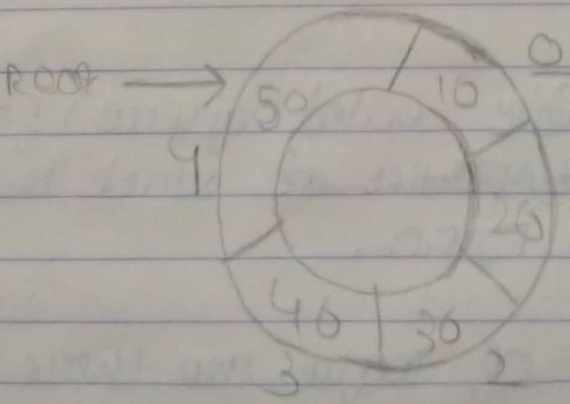
Thus, the element 10 will be insert in 0^{th} position.

⇒ Now, same for Deque (deleting) the element from Front end:

Front = (Front + 1) % Maximum size

Ques As in the circular queue there are 5 elements where size is full namely 10, 20, 30, 40, 50 & we have to deque the element.

Solⁿ



→ सबसे पहले जो element insert होगा वही remove होगा (FIFO) rule लगेगा।

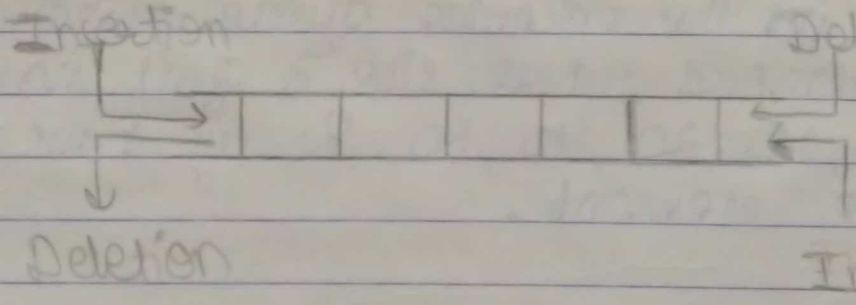
formula for Front is:

$$\text{Front} = (\text{Front} + 1) \% \text{ maximum size}$$
$$= -1 + 1 \% 5$$
$$= 0 \% 5$$
$$= 0^{th}$$

Thus, 0^{th} position का 10 element सबसे पहले remove होगा।

(iii) Deque (Double ended Queue):

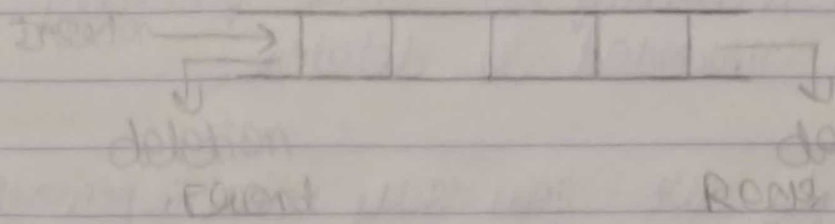
→ In Double ended queue, insertion (enqueue) & deletion (dequeue) are allowed from both the rear & front end.



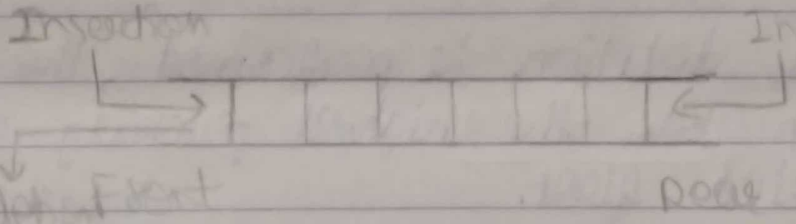
Deque (Double ended queue) follows both the properties of Stack & Queue which is LIFO & FIFO.

Two types of Deque are there:

① Input restriction = In input restriction, insertion can be done from only one end & deletion deletion can be done from both the ends.



② Output restriction = In output restriction, insertion can be done from both the ends but deletion (dequeue) can be done from only one end.



* Operations which we perform on deque:

- insert at front.
- delete from front.
- insert at rear.
- delete from rear.

☆ 3) Priority Queue = A priority queue is a linear data structure in which every element has some priority & according to their priority that element get inserted & deleted.

Simply we can say that in priority queue elements जो हैं वो priority के हिसाब से insert होंगे और priority के हिसाब से ही delete होंगे :

- It processed by the following rule :-

⇒ When deletion is performed, the element which has the highest priority get deleted first.

⇒ When insertion is performed, the new element is placed or enter on the basis of its property.

⇒ When the elements are of same level priority then the element which enter first that get inserted or deleted first

- Priority queue are two types :

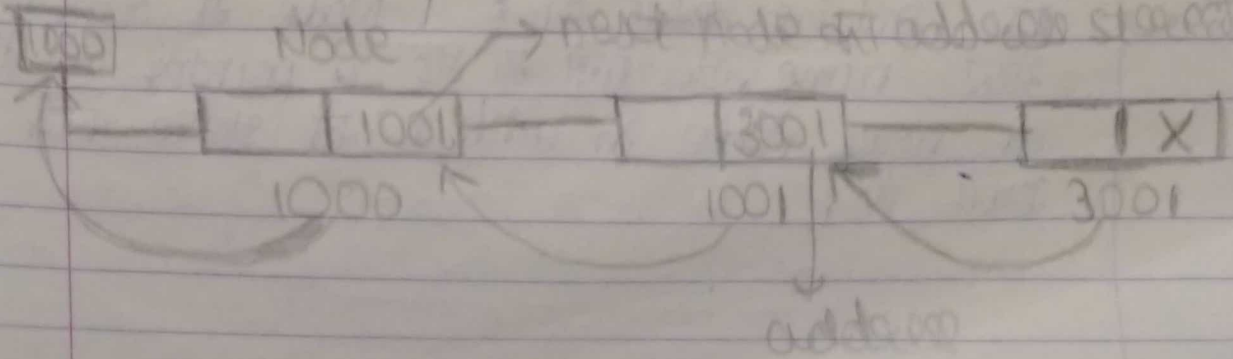
1) Ascendings = Insertion is random but deletion can be done from smallest element which has lower priority.

(2) Descending = Insertion is random but deletion can be done from greater element which has higher priority.

Linked List

* Linked List = A linked list is a linear data structure. It is a collection of data elements called node. Every Node is the place where we store data items. Every node has two part one is information & second is pointer or address where we store the address of next elements or next node which link next element in the list.

* Null means कोई भी invalid address.
* Every node has two part, first one is information which includes names, address & so on.
* Pointer field में इसमें next node का address रहेगा।
Pointer field में हम next node के address को maintain means रखते हैं।



⇒ In this pointer stores the address of the next node.

- One most important thing about linked list is that:

> In linked list the elements are not stored at contigous memory locations. The elements in a linked list are linked using pointers.

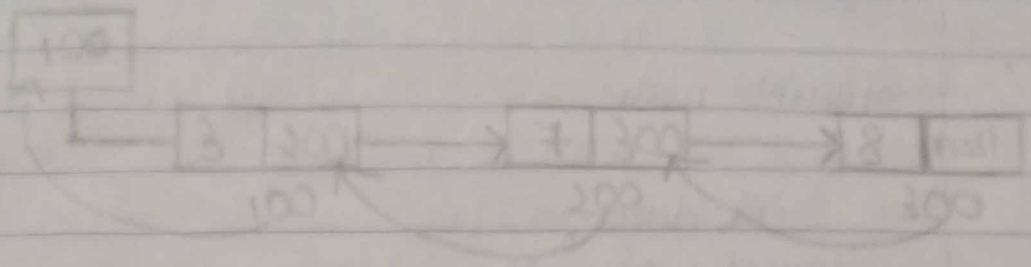
- Types of Linked List:

* - Singly Linked list:

⇒ In singly linked list, every node contains some data & a pointer to the next node of the same data type. Here the node contains a pointer which stores the address of the next node in the sequence. A single linked list allows traversal of data only in one way.

- हर एक node अपने अंदर data & pointer को contain करता ही है। Node में data के अंदर name, etc. चीजे आ गई & pointer जो है वो next node के address को store करके रखता है।

e.g:→

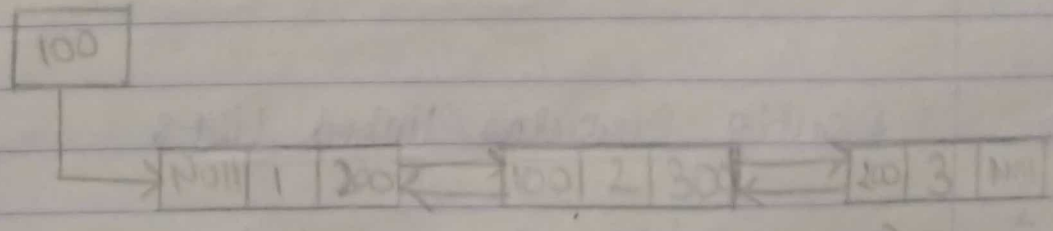


Implementation:

```
struct node
  {
   int data;
   struct node * next;
    };
```

* Doubly linked list:

→ In double linked list, every node contains data & pointer one & pointer two or 2 pointers in which left side pointer stores the address of previous nodes while right side pointer stores the address of next node. this would enable us to traverse the list in the backward direction & upward direction.

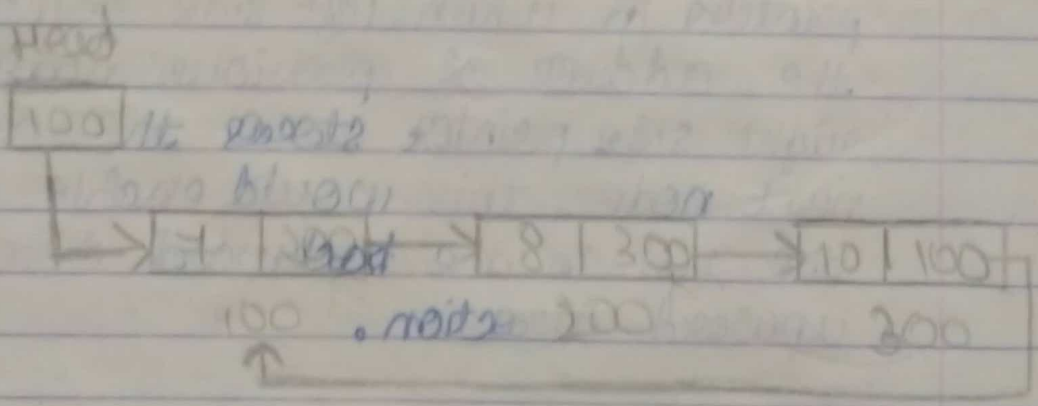


Algorithm:

```
struct node
 {
    int data;
 struct node * next;
 struct node * previous;
  }
```

☆ Circular linked list :-

⇒ In circular linked list, the last node is connected to the first node means in this the pointer of the last node contains or stores the address of the first node. Simply we can say that a circular linked list has no beginning & no end.

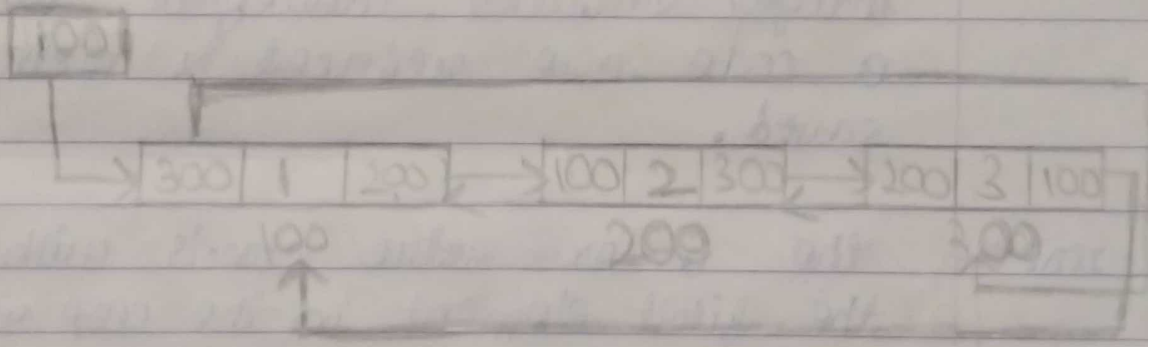


☆ Double Circular linked list:

⇒ In double circular linked list same here the last node is connected to the first node & here every node

contains or stores the data & two pointers in which left side or we can say that previous node, so left side pointer stores the address of previous node & right side pointer stores the address of next node as here the nodes are in circular form so last node is directly connected to the first node in the same way, the pointer of the last node stores the address of first node.

e.g. 8 :



* Array :

⇉ Array is the collection of similar kinds of data types. ये एक तरह का collection होता है same तरह के data types का जैसे हम कह सकते हैं अगर int में है तो int में ही रहेगी & अगर Float में है तो Float में है।

- the array is a type of data structure that stores elements of the same data type.

Page No
Date

Array is the most basic & fundamental data structures. Data stored in each position of an array is given a positive value called the index of the element. The index helps in identifying the locations of the elements in an array.

→ If supposedly we have to store some data i.e. the price of ten cars, then we can create a structure of an array & store all the integers together. This doesn't need creating ten separate integer variables. Therefore the lines in a code are reduced & memory is saved.

Imp:) The index value starts with 0 for the first element in the case of an array

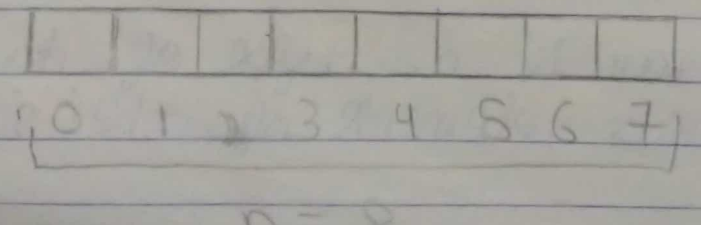
Array में कोई भी elements के लिए index values हमेशा 0 से start होती।

- Declaration of an array:

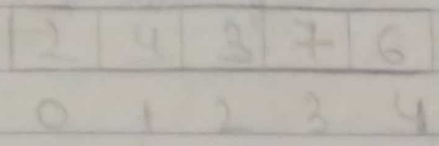
Data Type arrayName [array size];

e.g.:

```
int num [8];
```



⇒ An array is given:



⇒ Here upperbound = 4 (which is highest index)
& lowerbound = 0 (which is lower index)

Size of array = (upperbound - lowerbound) + 1
= (4 - 0) + 1
⇒ 4 + 1 = 5

* In array, we can store many values in one variable.

* Array is a collection of more than one elements & the condition is that the data type of those elements should be similar.

Ques How the elements of the data is stored in the memory in array?

Ans All the elements are stored in consecutive/continuous manner, one after another.

* Simple & Point to Point definition of an Array:

→ Array is a fixed size sequenced collection of data items of same data type.

→ How to declare an array :

data type array name [array size];

Simple Flowchart of defining an Array

→ What is Array ?

↓

Array is the collection of similar kind of data types (like int-int, float-float, char-char).

↓

How to declare an Array ?

data type array name [array size];

e.g → int a [5]

↓ ↓ ↓

data type | array name | array size

↓

memory allocations → contiguous form में memory allocate होती है means एक के बाद एक form में।

→ Array के size को subscript के अंदर लिखा जाता है।

subscript [ ] ~~को~~ को subscript कहलाता है।

Array का जो size है उसे subscript [ ] symbol के अंदर लिखा जाता है।

Initialization of an Array

First type = int a[5] = {10, 20, 30, 40, 50};

Second type = int a[ ] = {10, 20, 30};

↓

इसका simply मतलब ये है कि जो भी हमने array को initialize किया है मतलब जितने भी elements दे रहे वही उतने elements array के size के रूप में आ जाएंगे, जैसे:

int a[ ] = {10, 20, 30};

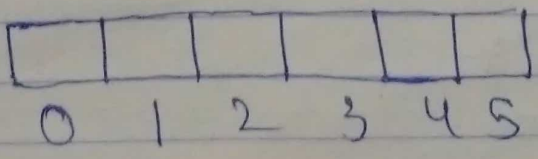
↓

मतलब इसमे 3 values initialize करायी है तो ये array का size भी 3 आ जाएगा।

- Some common points of array which will define array in every language

1) An array is the collection of similar kind of data type.

2) The first element in the array is numbered 0 so the last element will be 1 less than the size of Array. Lets take ex.: if the size of an array is 6 then its index value starts from 0.



Thus the last element is 5, 1 less than 6.

3) Array is declare with subscript [ ] so it is also called subscripted variable.

Subscript = [ ]

4) Array is declared with : ~~subscript~~
data type array name [array size];

5) Array is always stored in contiguous memory allocation. (एक के बाद एक)

6) Array initialization is done with { } brackets.

One Dimensional Array

⇒ One-Dimensional array is one in which only one-subscript [ ] specification is needed to specify a particular element of the array.

Simply one-dimensional array में हम एक ही subscript [ ] use करते हैं, ~~एक से~~ ~~element~~ के

Declaration of an Array in 1 Dimensional

Data Type Array name [Array size];

- Now Here :-

⇒ data type = data type is the type of element to be stored in the array.

1) One dimensional array :→ In this array only one subscript, either in form of rows or columns used to define the elements of Arrays.

- Initializing / assigning one-dimensional Array :→

⇒ data type array name [size] = { values }

Eg. for int :→

int num [10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
char char-acter [5] = {'A', 'B', 'C', 'D', 'E'}

2) Multi-Dimensional array :→ Multi-Dimensional arrays use more than one subscript to describe the array elements. Its types :

i) Two Dimensional arrays :→ It uses two subscript, one subscript to represent row value & second subscript to represent column value. It mainly use for matrix representating.

Declaration of two Dimensional Arrays :→

Data type array name [row][column];

[size]

Eg: int num[3][2];

* Initialization of 2-D Array:

Data type array name[rows][columns] = {values}

Ex: int num[3][2] = {1,2,3,4,5,6}

or

case 2: int num[][] = {1,2,3,4,5,6}

case 2: में हम ये कह सकते है कि अगर size नहीं दिया हो but values initialize की गई हो तो वो values के हिसाब से rows & column में store हो जाती है।

a) Ex:

	0	1
0	1	2
1	3	4
2	5	6

(0,0) = 1
(0,1) = 2
(1,0) = 3
(1,1) = 4
(2,0) = 5
(2,1) = 6

* Now we will see one-one program each for 1-D array & 2-D array.

Page No .....
Date .....

⇒ some basics point:

In C language :

is defined as pree processor
<stdio.h> is the header file & stdio is stands for standard input output & it includes the function printf & scanf.

& <conio.h> is also the header file where conio is stands for console input output & it includes the function clrscr(), getch().

Que⇒ Write a C program to make the program of one dimensional array.

⇒
```
#include <stdio.h>
int main(){
 int a[6]; int i;
printf("Entered the Array elements");
 for(i=0; i<6; i++){
  scanf("%d\n", &a[i]);
  }
 printf("The entered Array is");
 for (i=0; i<6; i++){
  printf("%d\n", a[i]);
  }
 return 0;
  }
```

Tree is a non linear DS which shows parent child relationship among the nodes it organizes data in hierarchical manner.

Trees → Non-linear DS

☆ Trees :

→ A tree is a non-linear hierarchical data structure that consist of nodes connected by edges.

→ A tree is a hierarchical data structure defined as a collection of nodes which are linked or connected through edges.

For e.g. :→

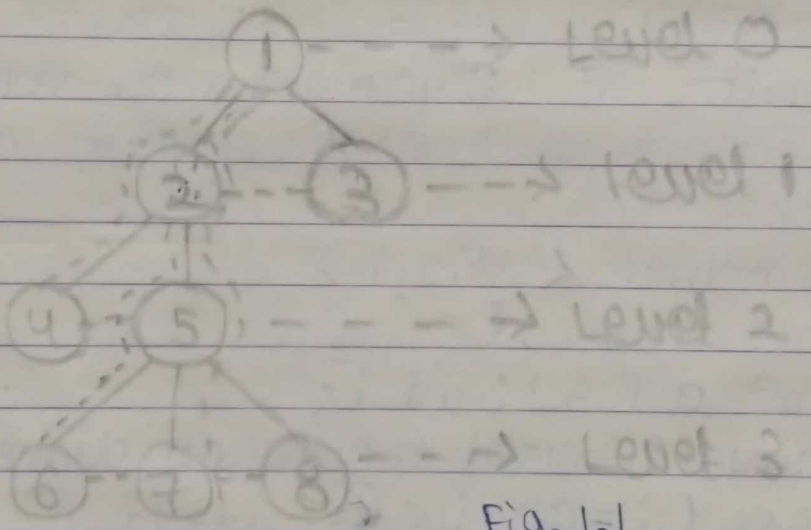


Fig. 1.1

↑ this is a tree

- Root node = the highest or topmost node of a tree & the node which don't have any parent node called as root node.

- Now, what is node ?

Ans: In the tree, each element consists or stores some information in it which is referred to as nodes.

- Sibling :→ Siblings are the childerens of the same parent or we can say that if the parent has 2 or more than 2 nodes then these nodes are the siblings of each other.

- Degree of node :→ degree of node is the no. of childeren of that node.

- Degree of tree :→ Maximum degree among all the nodes. (जिसकी सबसे ज्यादा degree होगी. जिस भी node की Highest / maximum degree होगी वही degree of tree रहेगी।

★ Leaf node or Terminal node :-

⇒ The node which do not have any child node or the degree of that node is 0 is called as leaf node or Terminal node.

★ Non-Terminal nodes or internal nodes :→

⇒ The node which has atleast one child node & the degree of any node be atleast 1 or more than 0, that node is called as Non terminal / internal nodes.

☆ Ancestor : Any predecessor node on the path from root of that node.

For e.g in the Fig. 1.1
ancestor of 7 is = 5, 2, 1
ancestor of 6 is = 5, 2, 1
ancestor of 4 is = 2, 1

☆ Descendant : Any successor node on the path from that node to leaf or terminal node is called descendant

For e.g in the Fig 1.1 =
descendant of 2 is = 5, 6, 7, 8, 4
descendant of 5 is = 6, 7, 8

☆ Depth of node : length of path from root to that node.

☆ Height of node : no. of edges in the longest path from that node to a leaf node.

☆ Level of a node : It is the no. of edges connected to the nodes.

From fig. 1.1, level of 2 is 1 & height of 2 is 1

Height of node = Level of node

- Path = The way from source node to destination node through consecutive edges.
- Forest = set of disjoint trees
- Parent node = Predecessor node of the any node.
- Child node = successor node / descendant node of any node.

Traversing = Visiting the each element.

Page No
Date

Graphs

⇒ Graph is the collection of vertices (node) & edges where edges works to link the or to connect the vertices / two vertices. (Edges is the set of vertices)

Simply we will take one example :

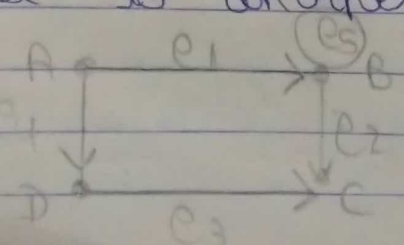
अगर हमे एक city से किसी दूसरे city में जाना है तो इन cities के बीच की बहुत सारे cities आएंगे तो वो city हमारी nodes/ vertices हो जाएंगे & उनके बीच का distance और path हमारी edges हो जाएगी।

Like if we are going from one city to another city then in b/w them there will lot of cities so here the cities are the vertices or we can say that nodes & the edge distance b/w them or the path are the edges.

- Types of Graphs :

(Digraph)

i) Directed Graph = Simply जिसमे हम direction को show करते है with the help of arrow In this all the edges are directed from one vertex to another.



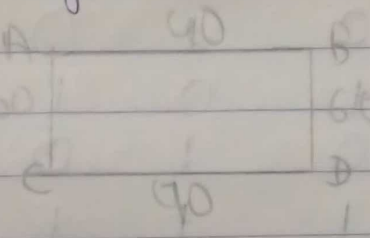
For example हम ABCD कोई graph लेते है तो इसमे A to B & B to A दोनो अलग directions से जाएगी।

→ Degree = No. of edges connect to that vertices.

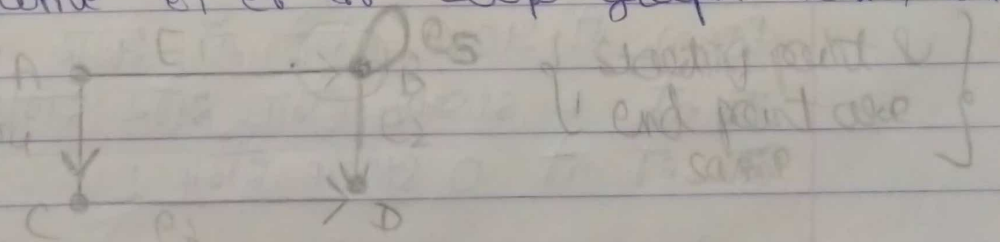
→ Adjacent = which are connected by same edge

(ii) Undirected graph = जिसमें edges तो होती है but कोई directions shown नहीं की होती।
Undirected graphs have edges but they do not have any directions but all the edges are bidirectional.

(iii) Weighted graph = Simply it means जब हम हर edge को weighted value assign कर दें।
It refers to where weights are assigned to each edge.

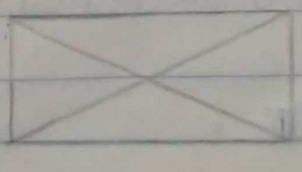


(iv) Loops = जिसका starting point & end point same ही हो तो Loops graph कहलाता है।



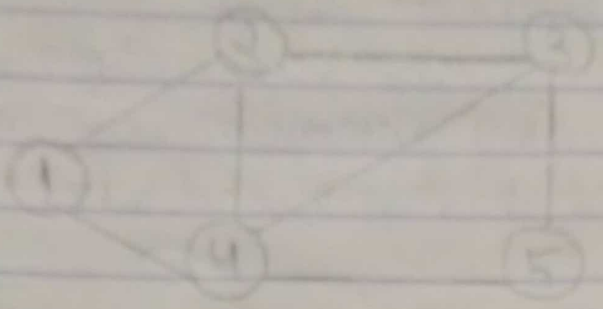
* Graph & its Representation:

• Complete Graph: In complete graph every (vertices) node is adjacent to every other node (vertices)



Here n no. of vertices & edges = $\frac{n(n-1)}{2}$

① Adjacency matrix :-



→ Here n is used for rows & column like :

i ↓ \ j →	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	1	0
3	0	1	0	1	1
4	1	1	1	0	1
5	0	0	1	1	0

5x5

यहा पर i & j में value bits (binary) के form में store होती है अगर i, j दोनो adjacent हो तो 1 store होगा और अगर i, j दो adjacent न हो तो 0 store होगा।

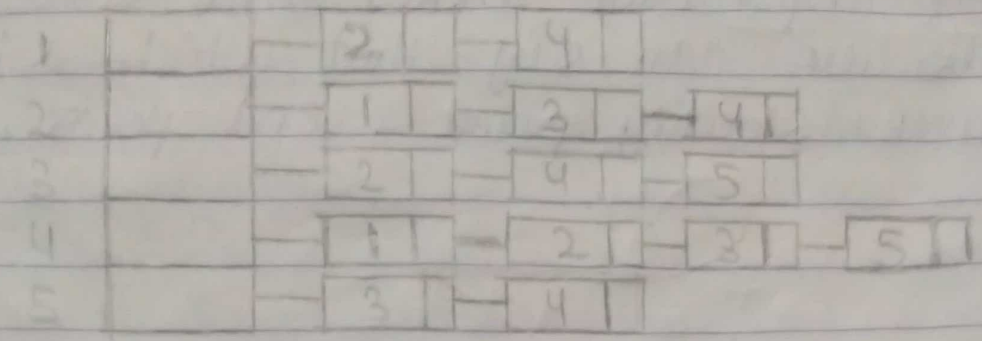
It is a matrix A[n][n] where n is no. of vertices &:

$$\begin{cases} a[i][j] = 1 & \text{if } i \text{ \& } j \text{ are adjacent} \\ = 0 & \text{otherwise not adjacent} \end{cases}$$

In this matrix :

$$\boxed{\Theta(n^2)}$$

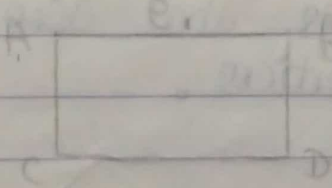
(2) Adjacency list :-



By this :

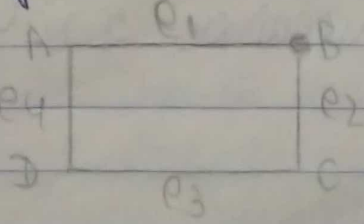
$$O(n+2e)$$

☆ i) Adjacent vertex = If two vertex are joined by the same edge they are called adjacent vertex.



So, here two vertex A & B are joined by the same edge e_1.

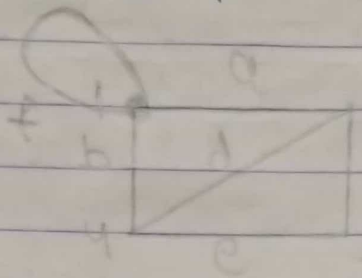
☆ ii) Adjacent edge = If two edges are meet / lie on the same vertices, they are called adjacent edge.



Here two edges e_1 & e_2 are incident

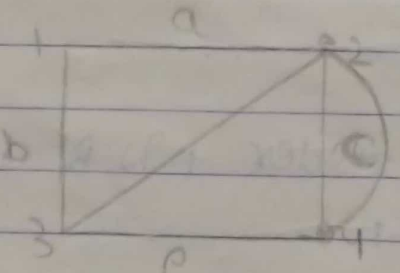
on the same vertex B.

iii) Self loop :- Edge having same vertex (Vi, Vi) also we can say that which is having same starting point & end point.



Here, f is starting from 1 & end on 1 which means start & end from the same point.

iv) Parallel edges = These are also called multiple edges, in an undirected graph, two or more edges that are incident to the same two vertices.

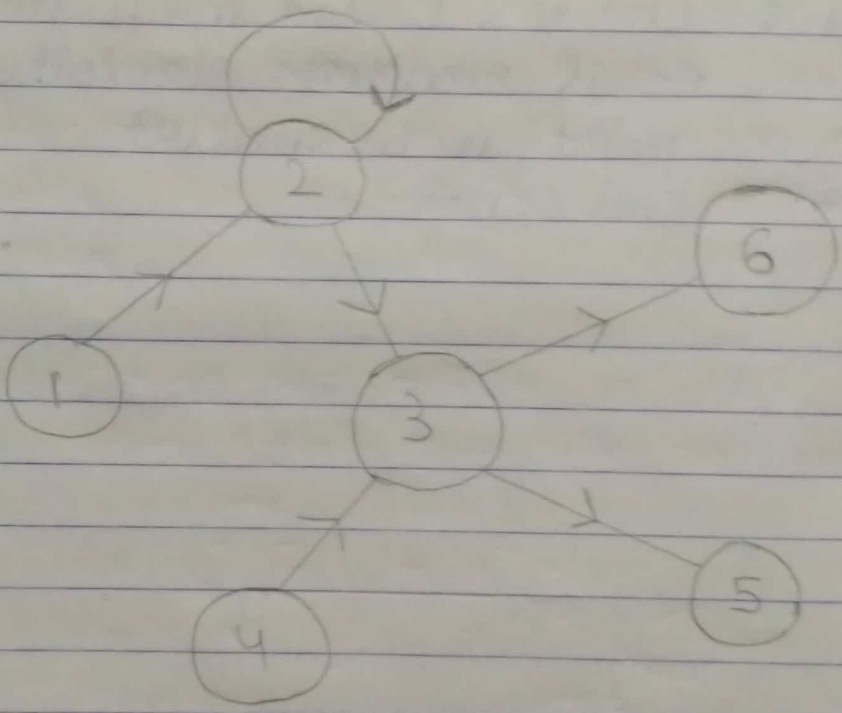


So, here, two (2) &
So, here c & d two edges are incident to the same vertices 2 & 4.

Remaining Graphs

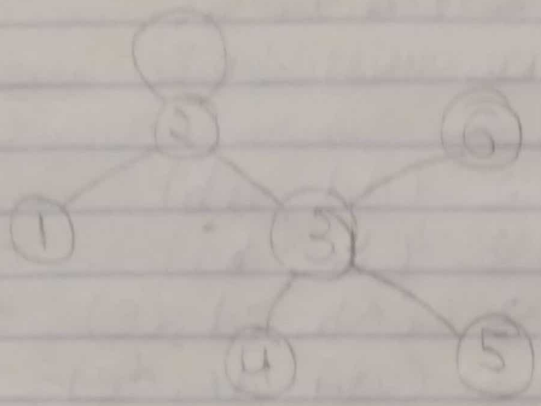
- Adjacency matrix of given graph for directed / digraph.

Here we will store the value in the form of bits (0 & 1) if i & j are adjacent then 1 will be stored otherwise 0 if not adjacent:



i → / j ↓	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	1	1	0	0	0
3	0	0	0	0	1	1
4	0	0	1	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0

Now, adjacency matrix of a given graph for undirected graph.

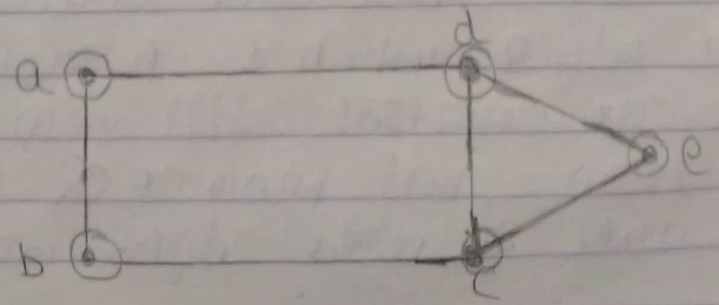


Now here there were no directions so we can take bidirectional graph:

i →	1	2	3	4	5	6
1	0	1	0	0	0	0
2	1	1	1	0	0	0
3	0	1	0	1	1	1
4	0	0	1	0	0	0
5	0	0	1	0	0	0
6	0	0	1	0	0	0

Now, we will see the degree of the graph:

* Degree → No. of edges which are connect / incident to that vertex.

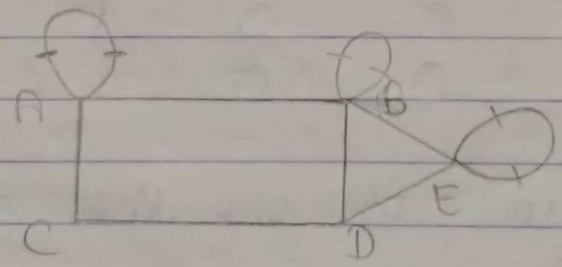


→ Now, here from the vertices a there were two edges are connected which is from a to d & a to b.
So, the degree will be:

degree (a) = 2 (ad, ab)
degree (b) = 2 (bc, ba)
degree (c) = 3 (cb, cd, ce)
degree (d) = 3 (da, dc, de)
degree (e) = 2 (ed, ec)

Note → If there were no edges connected to that vertex then the degree of that vertex be 0 which is called as isolated vertex.

Now, if there were a self loop in the graph then the degree will change:



Here the self loop contains 2 degree (2 edges) in itself:

We can say that the vertex already has two edges which is A to B & B to C so we can say that till now the degree of A is 2, but because of self loop there will 2 more edges increased.

∴ degree (A) = 4
degree (B) = 5
degree (C) = 2
degree (D) = 3
degree (E) = 4

Now, we will studied the indegree & outdegree in the Directed graph:

IMP → Note that: in degree & outdegree is only in the directed graph because of the directed graph showing directions:

Firstly we will define indegree as:

1) Indegree = No. of edges which are are connected / are incident on that vertex which are only coming towards in the direction of that vertex, which we called as indegree.

2) Outdegree = No. of edges which are connected or incident on that vertex which are going away from that which, we called as outdegree.

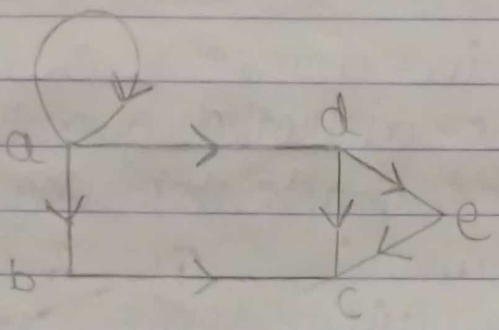
- Now, here we will define these degrees with the help of example:

Directed graph:-

Page No.
Date

Here one more important thing is that if the self loop is made on any vector then it should considered as 1 edge is coming towards that vertex & 1 edge is going going away from that vertex:

Directed graph :



Here :

indegree (a) = 1
outdegree (a) = 3

indegree (b) = 1
outdegree (b) = 1

indegree (c) = 3
outdegree (c) = 0

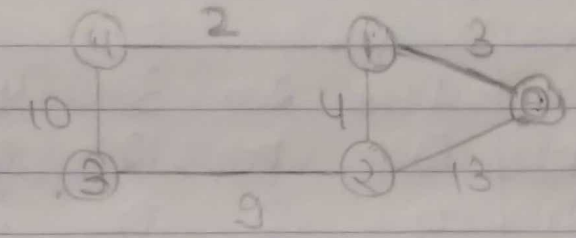
indegree (d) = 1
outdegree (d) = 2

indegree (e) = 1
outdegree (e) = 1

New topic :=
Undirectional Weighted graph
/ Bi-directional graph weighted graph in Adjacent list

* Undirectional weighted graph in Adjacency list :: In this we will assign the weight :

First we will see undirectional weighed graph in adjacency matrix :



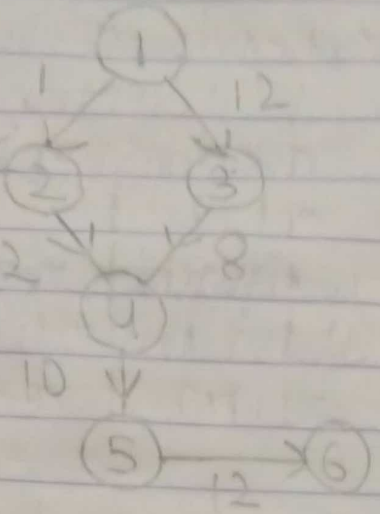
⇒ We will see its representation :

$$\begin{array}{c} j \rightarrow \\ i \downarrow \end{array} \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 0 & 3 & 13 & 0 & 0 \\ 1 & 0 & 4 & 9 & 2 \\ 2 & 4 & 0 & 9 & 0 \\ 3 & 0 & 9 & 0 & 10 \\ 4 & 2 & 0 & 10 & 0 \end{bmatrix}$$

(5×5)

Here we will assign the values of weight according to their adjacent edges.

* Now, we will see the digraph weighted graph in adjacency matrix.

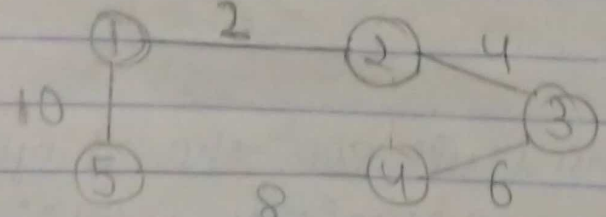


there are 6 vertex so matrix will seemed of 6x6 (6 row & 6 column)

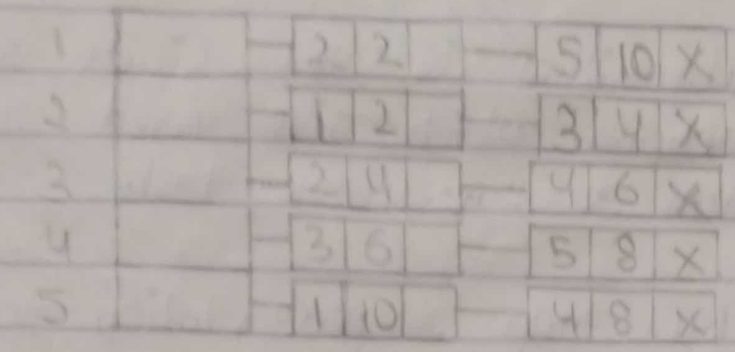
	1	2	3	4	5	6
1	0	1	12	0	0	0
2	0	0	0	2	0	0
3	0	0	0	8	0	0
4	0	0	0	0	10	0
5	0	0	0	0	0	12
6	0	0	0	0	0	0

(6x6)

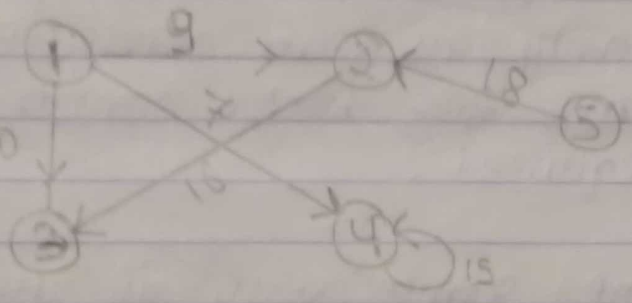
(ii) Undirected weighted graph in adjacency list : In the first pointer we will write the vertex & in second block we will write the value of weight we assign it



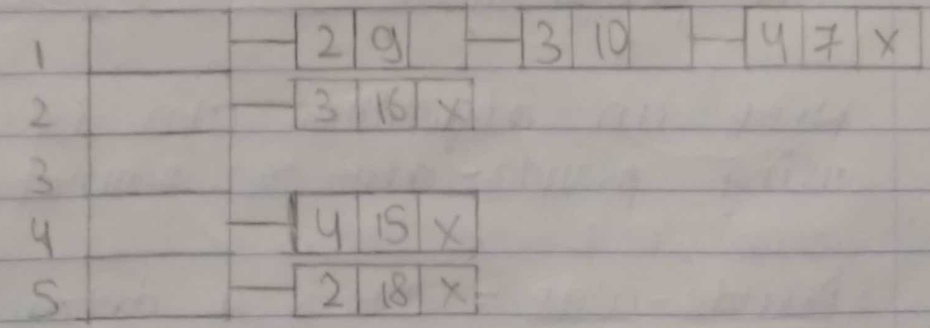
we will store in the form of linked list :-



☆ Now for directional weighted graph in adjacency list form :



Here also we will store in the form of adjacency list :



Step by step representation of any program

Page No
Date

Algorithm

⇒ Algorithm = It is the finite sequence of instructions which is used to solve a problem & step by step representation

⇒ Easiest definition :

- An algorithm is the sequence of the process / instruction or it is the collection of steps which shows the logic of the program with input & output.

- It is a set of steps to accomplish a task.

Que⇒ Why we need the algorithm?

Ans⇒ We will understand why we need the algorithm by using an example : as if we have to construct the building firstly we have to design the map of that building means for better construction of building map is required.

In the same way if we have to make the program we have to design its algorithm (or set of instruction which we need to design the programs) so we can say that " to design the better program algorithm is required."

☆ Difference b/w Algorithm & Program

→☆ Algorithm :

i) In algorithm we have to design the phase not to implement, we just have to design the phase in the page, notepad, wordpad etc. but we don't have to implement it on any ide's.

ii) We have to write these algorithm in our natural language like English, we don't have to maintain any syntax in it.

iii) Algorithm can be created by those person who should have the domain knowledge of that particular algorithm.

iv) We have to analyze that algorithm means we have to check it either we did any mistake or not then we can correct it.

☆ Program :

i) In program, we have to implement the phase in the ides.

ii) We have to write the program by using proper sytax of that program & includes all the rules of that programming

Flowcharts in algorithm
divide conquer & combine

Page No
Date

language.

iii) We have to compile & run the programs or testing the program.

Que) How to write the algorithm:

- We have to do sum of two integers:

Algorithm:

Step 1 : Start
Step 2 : Input of a & b where a is 45 & b is 68
Step 3 : Sum = a+b
Step 4 : print the sum
Step 5 : Stop / End

- Now if we have to check that which no. is greater:

Algorithm:

Step 1 : Start
Step 2 : Enter a & b
Step 3 : if a > b then print that a is greater else print b is greater.
Step 4 : end
/ stop

* Another way if we have to write the algorithm for checking which no. is greater.

Algorithm =

Step 1 : Start
Step 2 : Read the a & b
Step 3 : if a > b then go to step 4 else goto step 5.
Step 4 : print a is greater.
Step 5 : print b is greater.
Step 6 : stop / end.

* Now algorithm to find the largest number in b/w them :

Algorithm =

Step 1 : Start
Step 2 : Read a, b & c
Step 3 : if a > b and a > c then print that a is largest.
Step 4 : else if b > a and b > c then print that b is largest otherwise / else print c is largest.
Step 5 : stop / end.

Que 7 We have to write the algorithm for finding the smallest number among them.

Algorithm:

step 1 : Start
step 2 : Read l, m & n
step 3 : if l<m and l<n then print l is smallest
step 4 : else m<l and m<n then print m is smallest else / otherwise print n is smallest.
step 5 : End / Stop

Algorithm

⇒ An algorithm is the finite set of instructions which are used to solve the problem or the given task.

→ Step by step representation to solve the particular task.

- As, if we want to construct the building, firstly we have to design its map means with the proper planning then only we can construct any building it means for better construction map is required.

In the same way if we want to design the better program, algorithm is required, which is the infinite set of instructions which are eas. used to solve the program.

☆ Here an algorithm can be specified using pseudo-code or flowcharts:

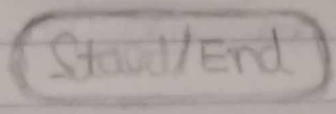
1) Pseudo-code = It is a detailed description of an algorithm which is easier to read & is expressed in the natural language (English).

e.g. ⇒ Algorithm for sum of two numbers in the pseudo code form :

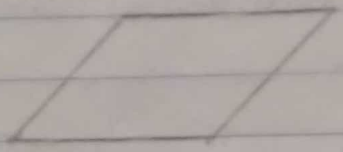
⇒
step 1 : Start
step 2 : Input a & b
step 3 : c = a + b
step 4 : Print c
step 5 : Stop / end

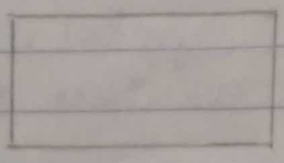
★ Flowcharts = It is the type of algorithm which is the pictorial or graphical representation with the help of graphical symbols which are used to solve the problem.

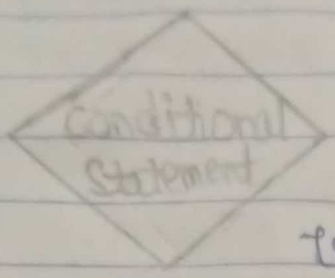
Some basic terminology which we have to understand are :

(i) 

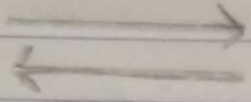
= Here oval shape is used to start & end the problems. It is also called as terminals.

(ii)  = Parallelogram symbol is used for input and output statements.

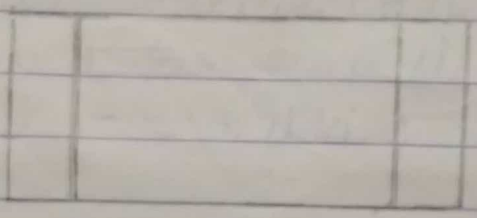
(iii)  = Rectangle is used to perform process which includes calculations, addition, subtraction etc. / arithmetic

(iv) 

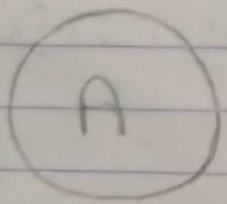
= Here diamond / rhombus represents a decision or conditional statements. It contains a true / false statements. Here it includes two arrows, one is for true condition & second is for false condition.

(v)  = Here Arrows represent the flow of control.

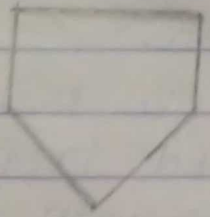
(vi) Pre define process symbol :



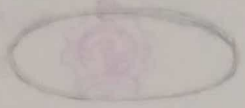
Predefined process symbol is used to show the operation or process that has been already define by another flowchart.

(vii) 

This is on-page reference.
= This circle is also known as connector which indicates that the next (or previous) step is somewhere else situated.

(viii) 
This is off-page reference. It
= indicates that the flow continues on a matching symbol containing the same letter somewhere else on a different page.

start & end can be :

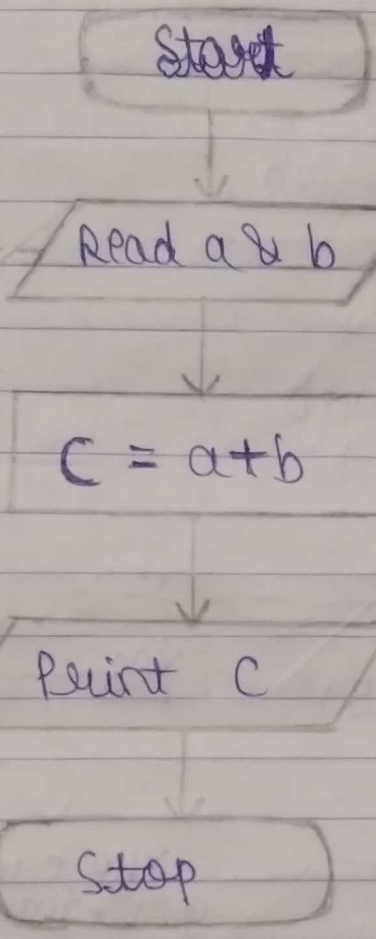


both are allowed

Page No
Date

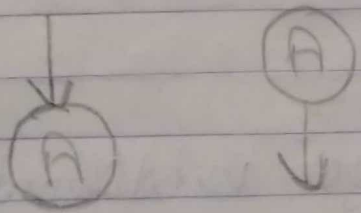
Representation of Flow Chart

☆ For addition of two number :

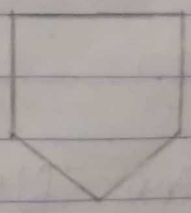


☆ One more important symbol is ;

☆ Connectors ;;

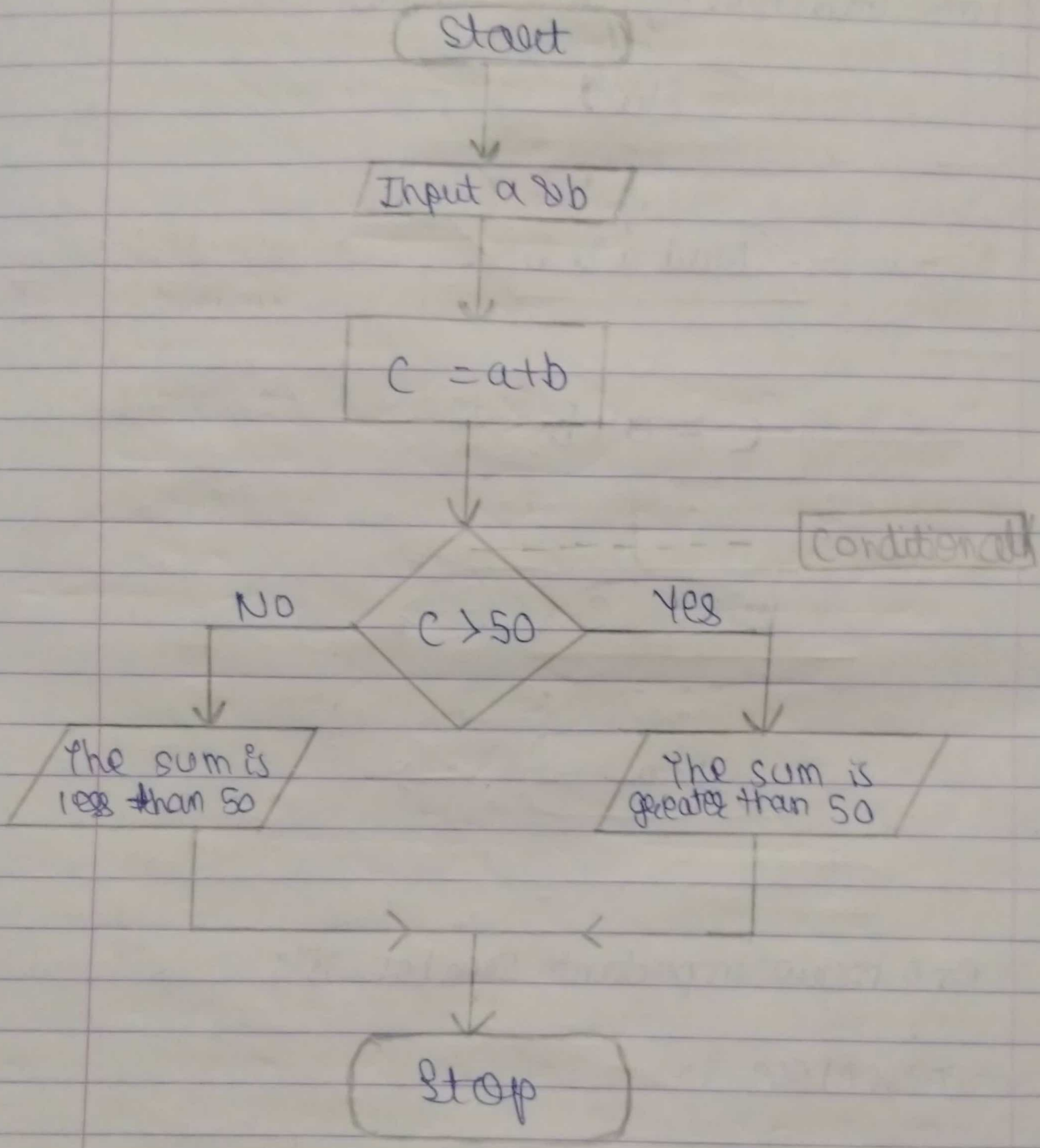


On page reference

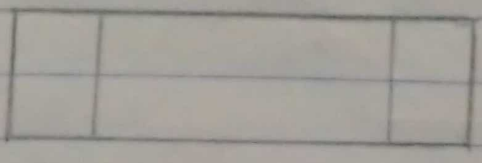


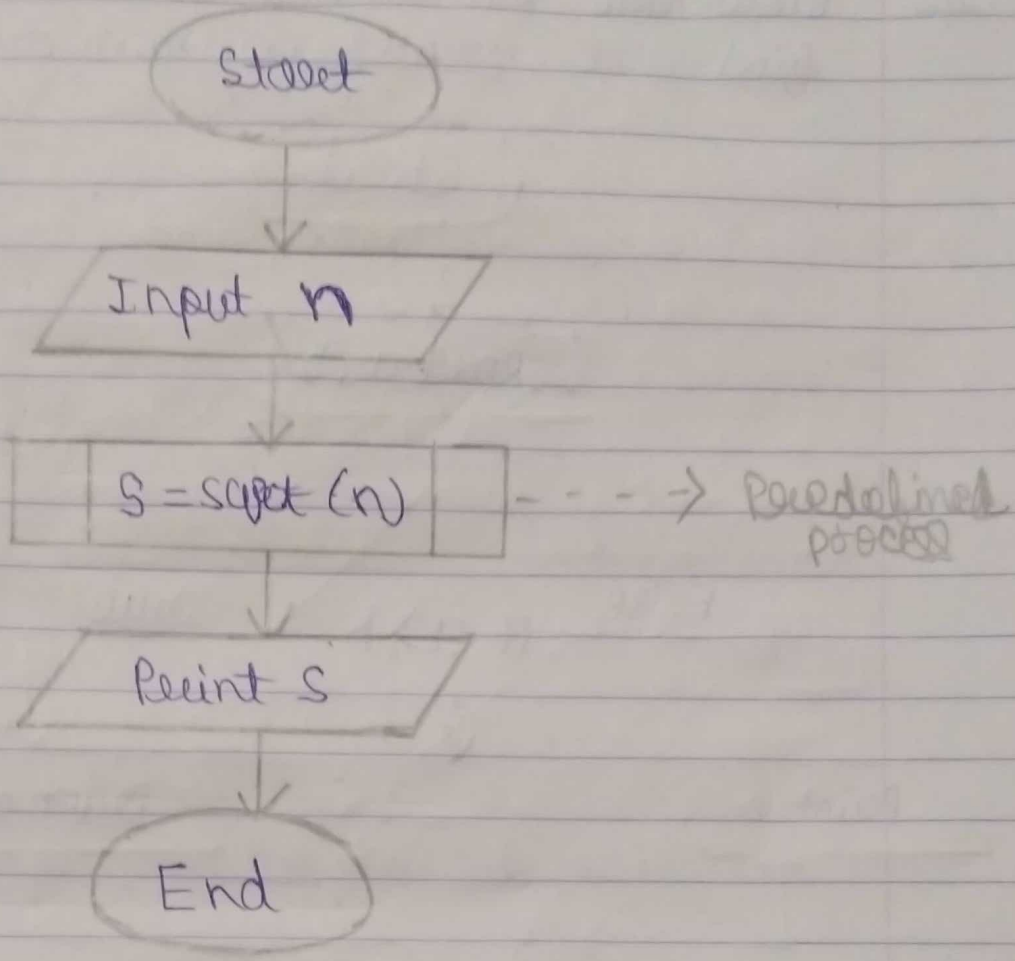
Off Page reference

Draw the flowchart to check whether the sum is greater than 50 or not:



One flow chart for Predefined process symbol:





- Key points:

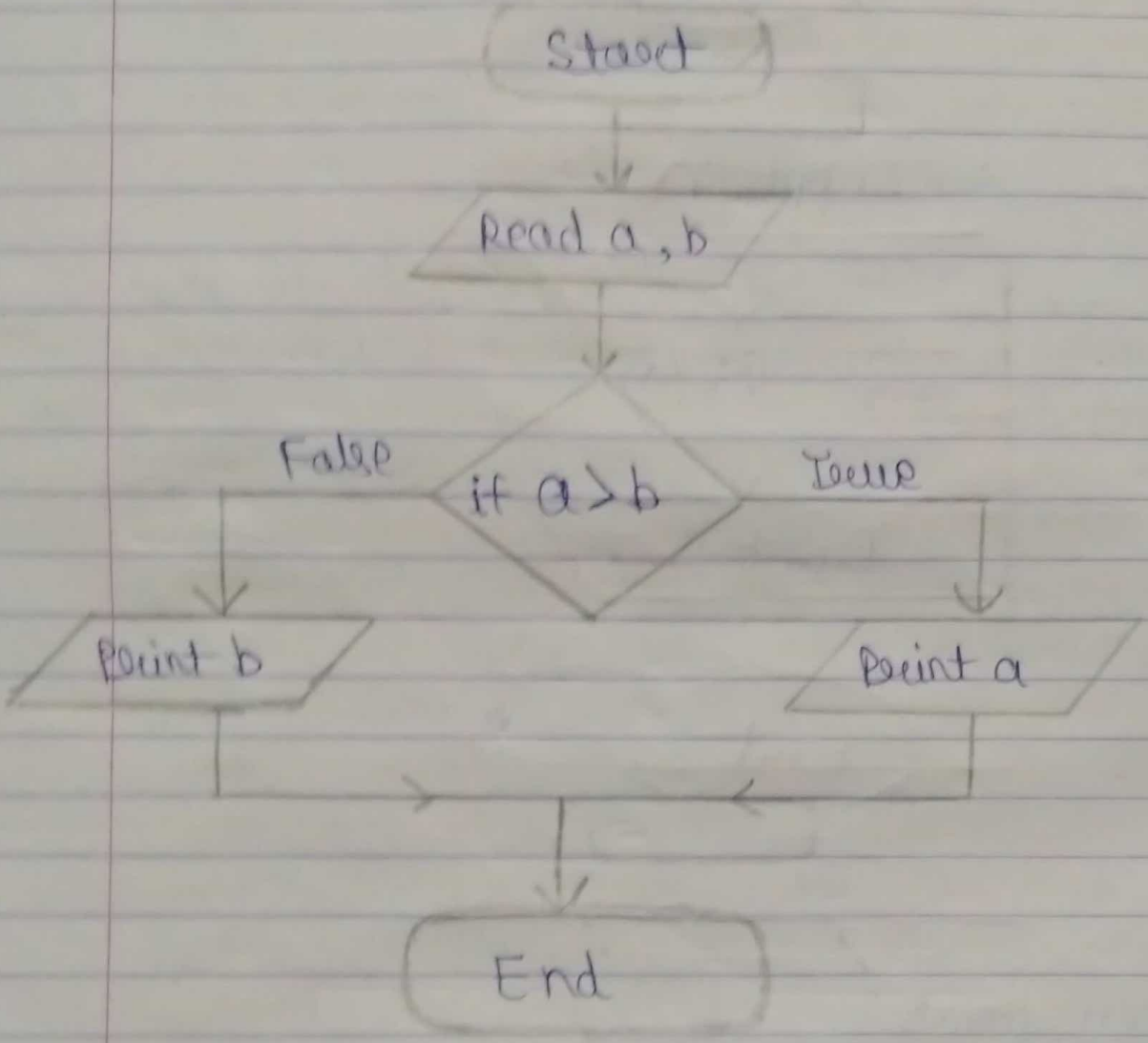
Input & output के लिए parallelogram shape use होता है।

Start & end के लिए oval shape use होता है

Decision making या conditional के लिए rhombus / diamond shape use होता है।

For looping Flowchart

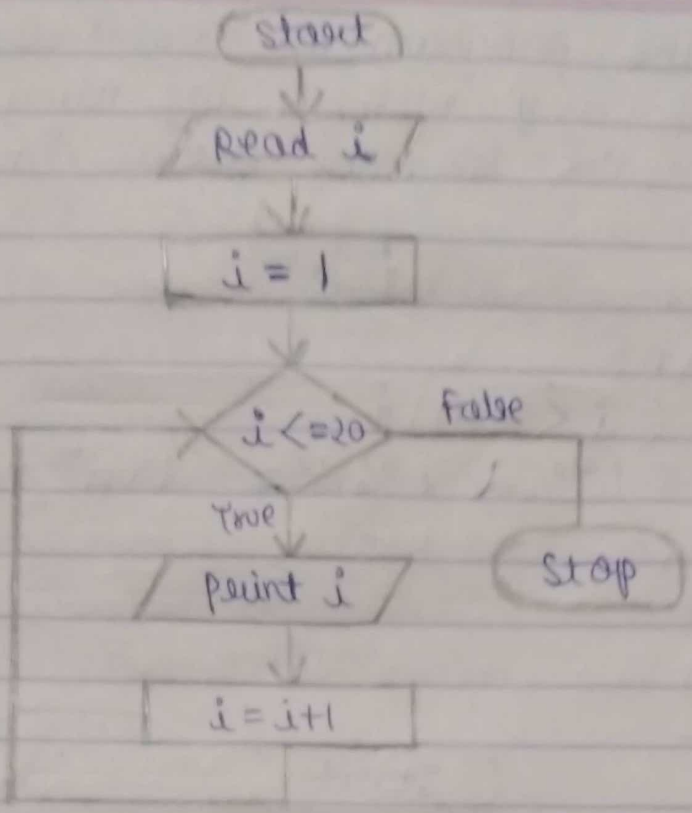
// Flowchart for making the program to find the greater no. b/w two numbers:



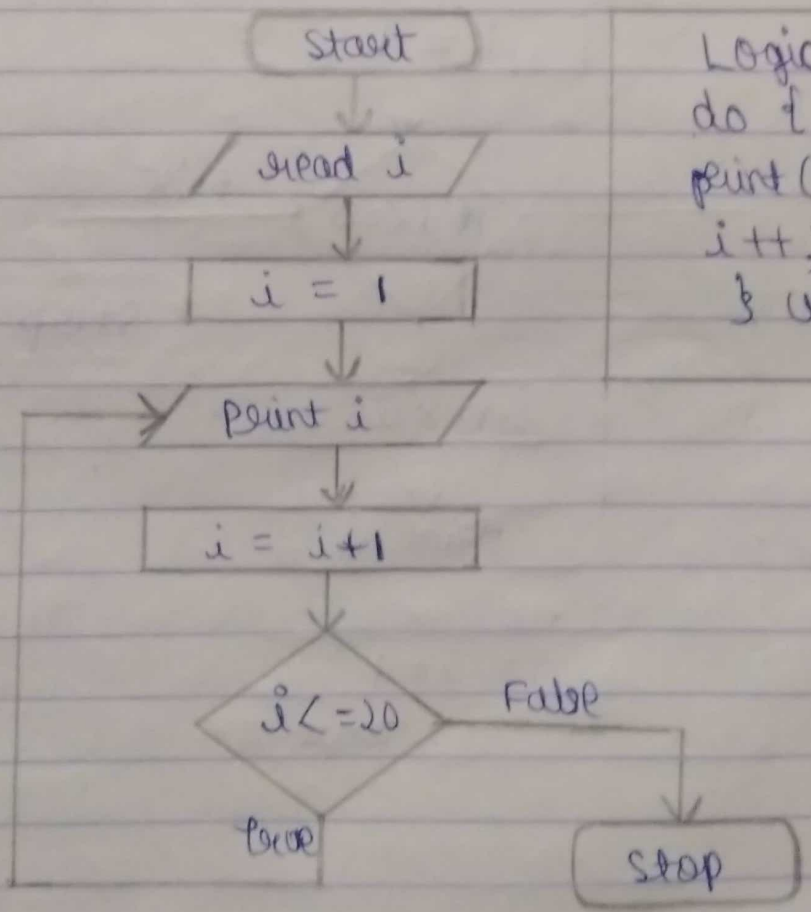
Here,

* Now we will make the program to print the numbers from 1-20 by using while loop with the help of flowchart:

Logic =
```
int i = 1
while (i<=20) {
printf("%d\n", i++);
}
```



Now same program in do while loop :



Logic :

```
do {
print("%d\n", i);
i++;
} while (i<=20)
```

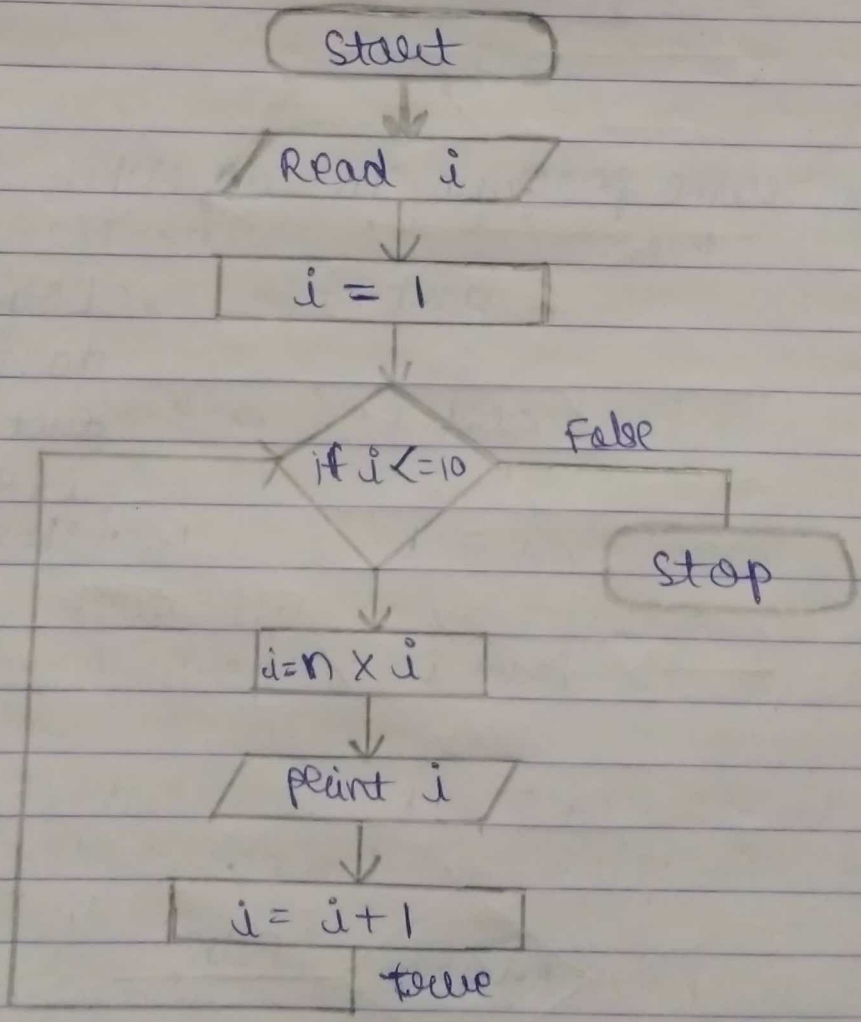
Page No.
Date

Que → Now, the program to write the multiplication table of any number by using while / for loop in the form of flow chart:

⇒ Logic :

```
int i=1,n;
while (i<=10){
printf("%d x %d = %d\n",n, i, n*i);
i++;
}
```

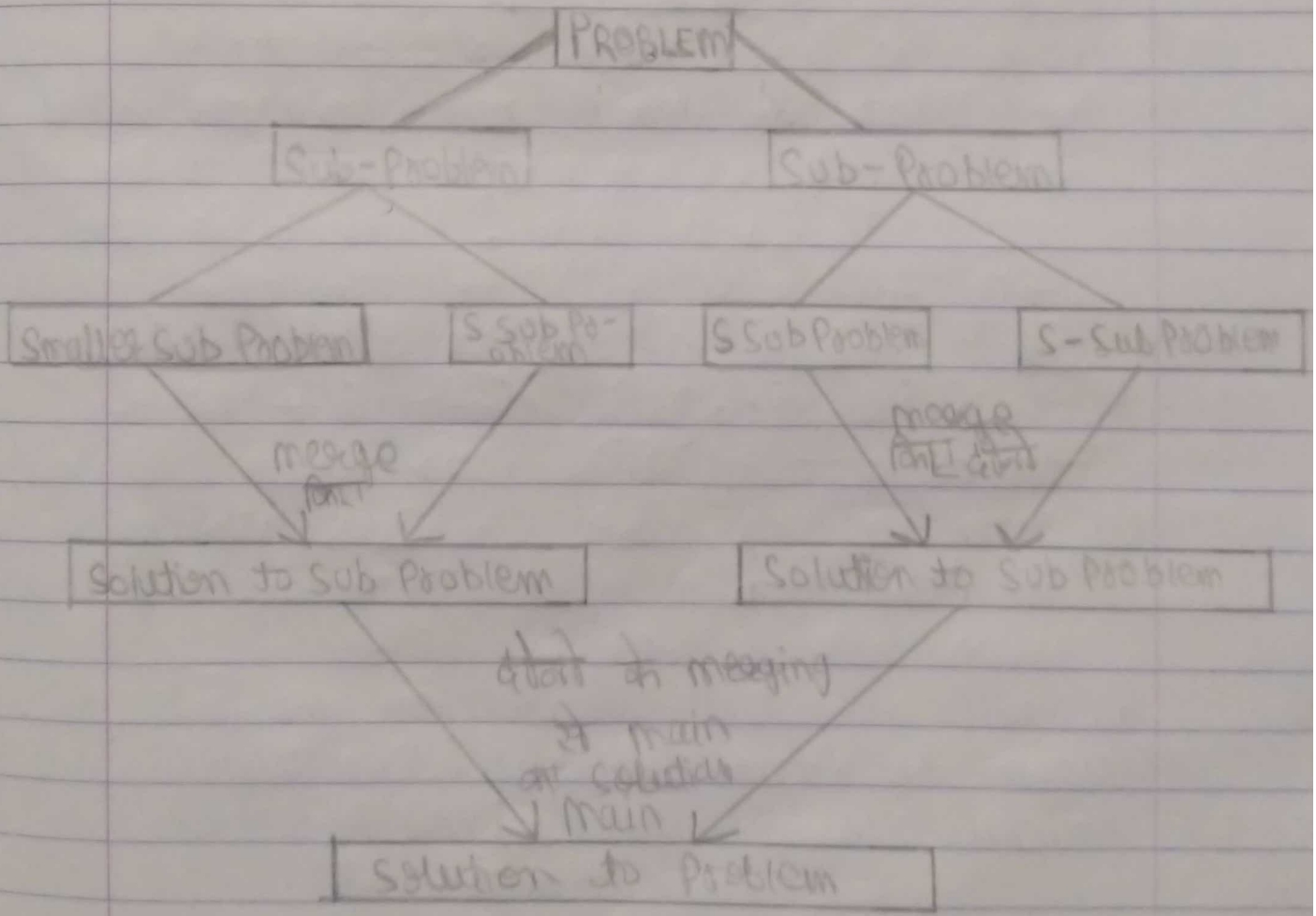
Start =



✱ Divide & Conquer :

→ इसमें simply हम एक बड़े problem को छोटे-छोटे subproblems में divide करते हैं ताकि हम उस problem को easy way में और directly solve कर सकें।

→ Divide & conquer is an algorithm design pattern which works by recursively breaking down a problem into sub-problem of similar type until these become simple enough to solve directly.



✱ Used in divide & conquer :

Page No
Date

1) Firstly we divide the problem into sub problems.
2) Then we solve the sub problems.
3) Atlast we merge the solutions to getting the main problem's solution.

Asymptotic notation

* Design & analysis of an algorithm :

Here there are many solutions of same program so we will implement only that one which takes least time & least memory. So here it is all depends upon time & memory. If that solution takes least time & least memory or s- than that solution only get implemented than any other solution.

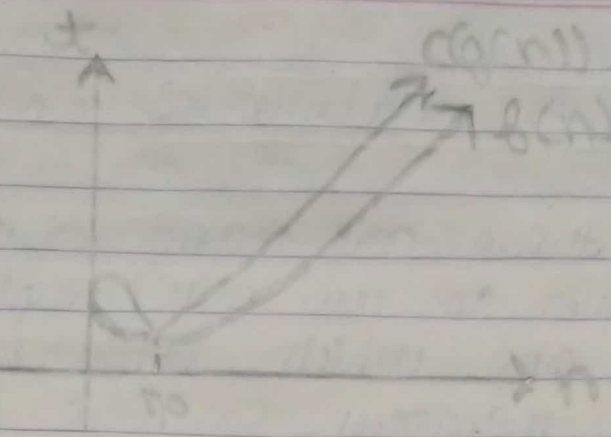
So simply here analysis means to read or abscreb that in how much time the problem will solve & how much it take the memory to store.

* Asymptotic notation : Basically there were three asymptotic notation, first one is :

1) **Big Oh notation (O)** = Here we are representing Big Oh as Capital (O).

⇒ Big Oh notation is used to describe asymptotic upper bound, where upper bound means a curve representing the limit of a function.

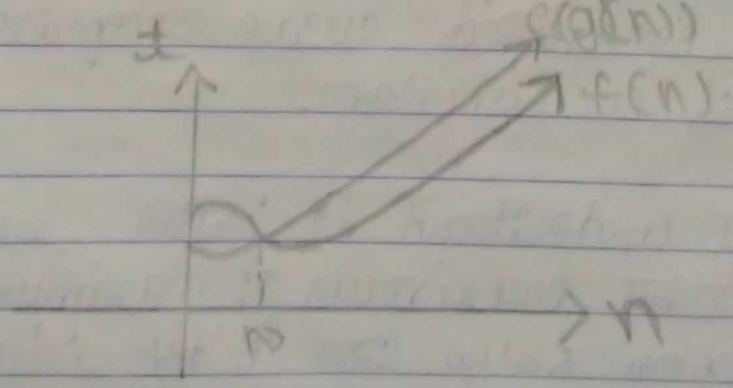
We will understand it better by diagram:
maximum maximum time
problem solve time taking
That's why worst case time taking



Here, g(n) is the upper bound for f(n) so we can say that f(n) is upper bounded by g(n) & where c is constant.

- Here, n is the input size & t is the time growth. So, here the time complexity is there or the time increases which we named as f(n) where f is function & n is input size. Now here we will define some other function which we named as c(g(n)) where when the f(n) is bounded with the other function c(g(n)) & after some limit no, then in this case (g(n) is always greater than the f(n) or f(n) is less than the c(g(n)).

Now all about its condition:



$f(n) \leq cg(n)$

$cg(n) \geq f(n)$

and $n \geq n_0$, $c > 0$, $n \geq 1$

Here why we take n as greater than or equal to one because an input should be atleast one.

So, here, $f(n) = O\, g(n)$.

also we can say that $f(n)$ is smaller than $g(n)$.

Lets take one question:

Q. $f(n) = 3n+2$, $g(n) = n$

Soln

$f(n) = O\, g(n)$

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

$3n+2 \leq cn$ & as $g(n) = n$]

Now, we put the value of c as more than 3 when comparing with LHS like $c = 4$:

$3n+2 \leq 4n$

$4n \geq 3n+2$

$n \geq 2$

for $f(n) = 3n+2$, $g(n) = n$, so we prove the condition or satisfy the condition that:

$f(n) = O\, g(n)$

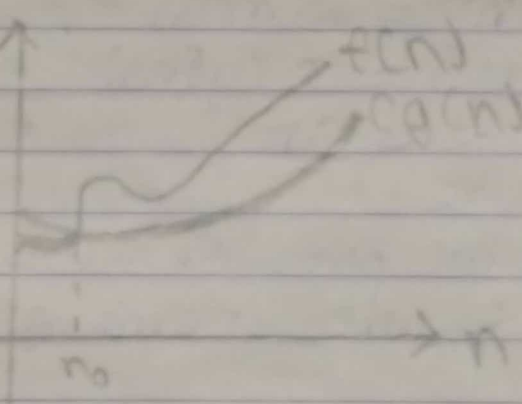
Here it means that $g(n)$ is bounded the function $f(n)$ so we can definitely say that any value greater than n like $n^2, n^3 \ldots$ are definitely going to bound $f(n)$ in the case of upper bound.

② **Big Omega** = It is represented as the symbol Ω

⇒ Big omega Ω is used to describe the asymptotic lower bound.

when we describe it in graph it will be look like:

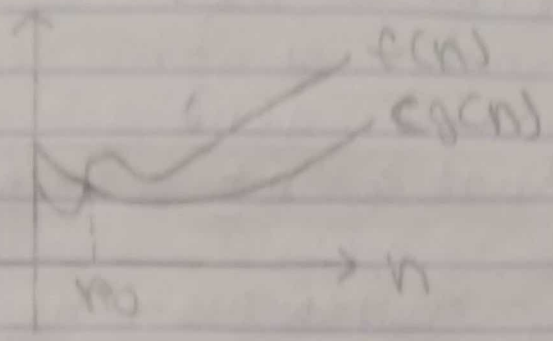
Big Omega में कितना least से least time लेता है, कम से कम time में problem को solve करने के लिए that's why it best case कहलाता है।



Here, $g(n)$ is the lower bound for $f(n)$ or we can say that $f(n)$ is lower bounded by $g(n)$.

Here we can say that the time complexity we named as $f(n)$ then we have to define the lower bound for it then we will define some other function

which we named as $cg(n)$.



Hence, $f(n) \geq c(g(n))$ where,
$n \geq n_0$ & $c > 0$ & $n_0 \geq 1$

Same here why we take $n_0 \geq 1$ because as it is input & input should have atleast one.

We will take one example:

Ex $f(n) = 3n+2$, $g(n) = n$

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

$f(n) \geq c(g(n))$

$3n+2 \geq cn$

And, $c > 0$ & $n_0 \geq 1$ we will take $c = 1$

$3n+2 \geq n$

or $3n+2 = \Omega n$

It simply means that $f(n)$ is lower bounded by $g(n)$, let's take an another example for better understanding:

Q. $\Rightarrow f(n) = 3n+2$, $g(n) = n^2$

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

$f(n) = \Omega n^2$

$3n+2 = \Omega n^2$

$3n+2 \geq cg(n)$

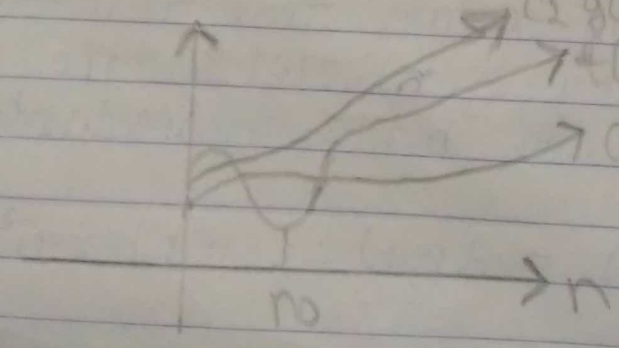
$3n+2 \geq cn^2$ & Hence, $g(n) = n^2$

Hence, $3n+2$ norms $f(n)$ can't able to solve in this situation so we clearly say that any value more than n in the case of lower bound are not going to bound.

⇒ We can say that's for any value more than n are not going to bound but the value less than n like $\log n$, $\log(\log(n))$ can bound easily in the case of lower bound.

(3) **Big thetha** : Big thetha we can represented as Θ.

- In Big thetha asymptotic notation we will define the function of both upper bound & lower bound. Now we will describe it in graph:



Big thetha (Θ) is used when both the Big (Oh) & Big Omega (Ω) are same)

we can easily identified it by seeing the diagram, as f(n) is lower bounded by $c_1 g(n)$ & f(n) is upper bounded by $c_2 g(n)$, so the situation will be

$\therefore \quad f(n) = \Theta(g(n))$

where, $c_1 g(n) \leq f(n) \leq c_2 g(n)$,
$n \geq n_0$ & $n_0 \geq 1$

$\therefore$ when $f(n) = 3n+2$ & $g(n) = n$

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

$3n+2 \leq 4n$, & $c = 4$}

$4n \geq 3n+2$

$n \geq 2$

which satisfies the condition for all the values of $n_0 \geq 1$ in the case of upper bound

and when, $f(n) \geq c_1 g(n)$

$3n+2 \geq 1n$ & $c = 1$}

$3n+2 \geq n$

which satisfies the condition for all the values of $n_0 \geq 1$ in the case of lower bound.

★ Short description:

- Big Oh (O) is the worst case algorithm.
- Big Omega (Ω) is the best case algorithm.

- And Θ is the average case where Θ is used when both the O & Ω are same.

☆ In divide & conquer we can say that, the given list is split into sublist to solve the problem.

Merge sort by divide & conquer

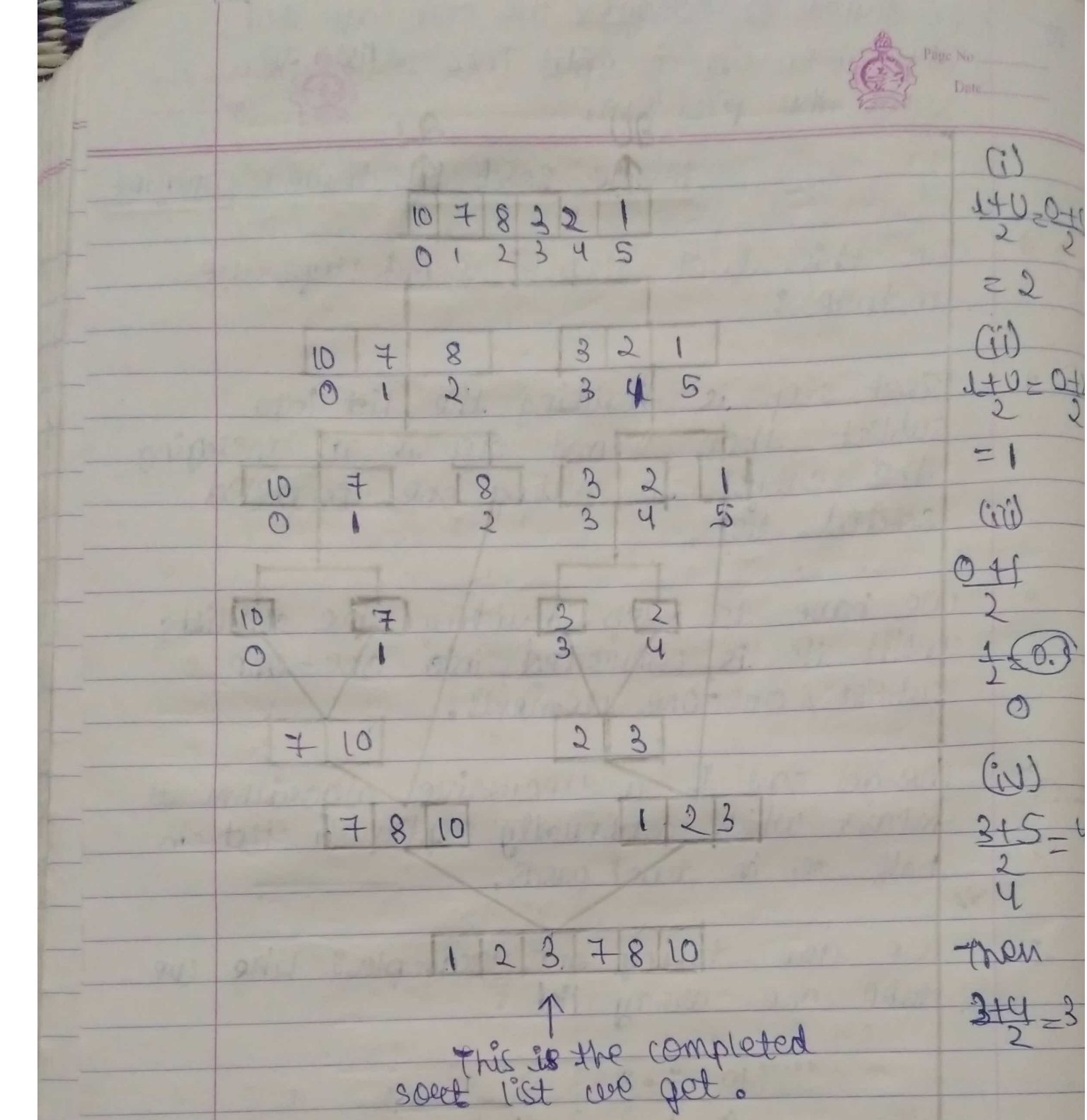
→ In this first step & second steps we include :

- First step is dividing the list into sublist then second step is to merging the sublist to getting one completed sorted list.
- we have to keep dividing the sublists untill it is converted into one-one sublist / one-one elements.
- Merge sort is a recursive algorithm it means which continually splits a list in half or in two parts.
- we are taking one example : Like we take one array list :

10	7	8	3	2	1
0	1	2	3	4	5

Here we have to divide the list untill lower bound ≤ upper bound.
Here is the formula to finding the midpoint as if we have to divide the list into sublist we have to directly find its mid point which divides the given list.

$$= \frac{\text{lower bound} + \text{upper bound}}{2}$$



(i)
$\frac{1+6}{2} = \frac{0+6}{2}$
$= 2$

(ii)
$\frac{1+6}{2} = \frac{0+6}{2}$
$= 1$

(iii)
$\frac{0+1}{2}$
$\frac{1}{2} = 0.5$
0

(iv)
$\frac{3+5}{2} = 4$
4
then
$\frac{3+4}{2} = 3$

merge sort II e.g^{g-1}
Like here one sublist array list is given, we have to solve this by using merge sort with the help of divide & conquer.

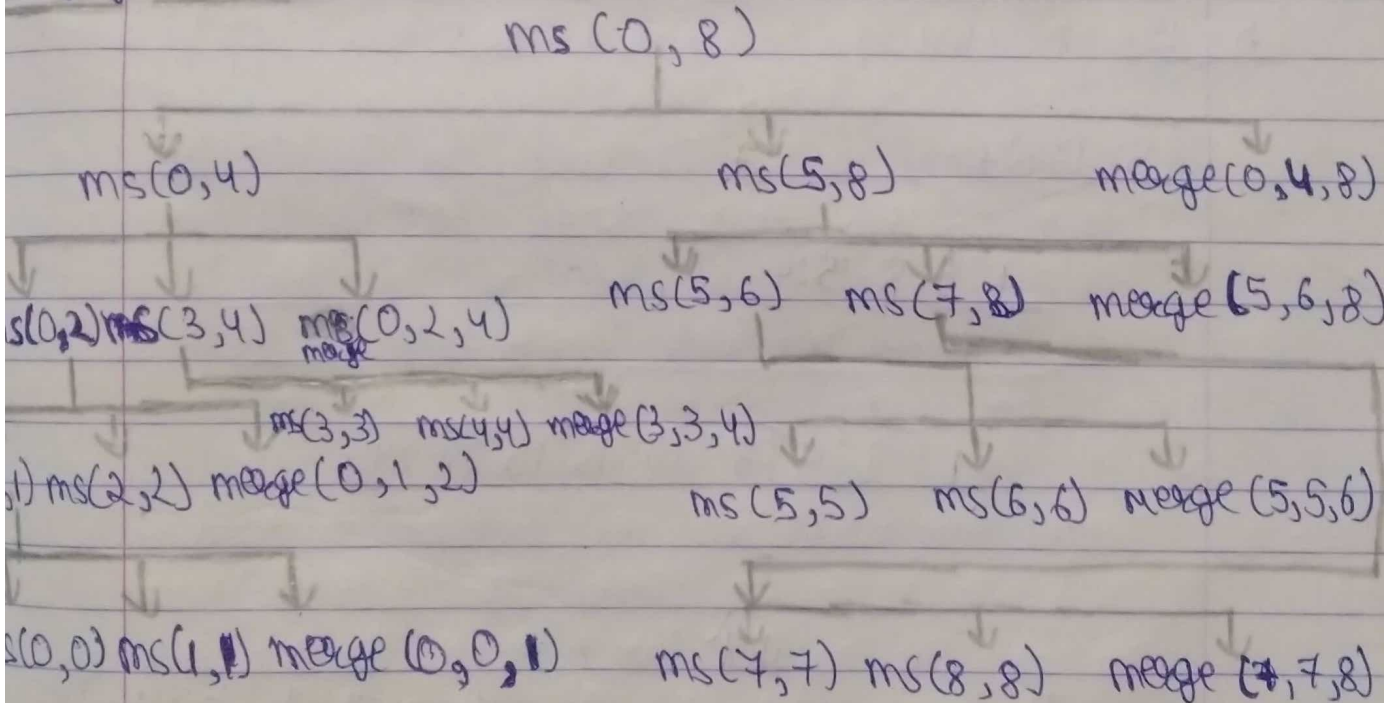
15 5 24 8 1 3 16 10 20
0 1 2 3 4 5 6 7 8

Merge sort के Program को लिखने का तरीका?
How to write merge sort program:

```
mergesort (A, lb, ub)
{
  if (lb < ub)
  { mid = (lb + ub)/2
  mergesort (A, lb, mid)
  mergesort (A, mid+1, ub)
   merge (A, lb, mid, ub);
   }
```

Recursive tree for how we merge
Now, how can we solve the fig.1 problem, we will give short summary type:

merging method



Here control will flow from (0,0,1) to the end & it will merge all the sublists into sorted form, result will be:

1	3	5	8	10	15	16	20	24
0	1	2	3	4	5	6	7	8

$\frac{lb+up}{2} \Rightarrow \frac{0+8}{2} = 4$, (i) $\frac{0+4}{2} = 2$

$\frac{0+1}{2} = 0$ $lb \leq up$, $\frac{0+2}{2} = 1$ $\frac{3+4}{2} = \frac{7}{2}$

$5+8 = 6$ 3

Now we will see the full process :

0	1	2	3	4	5	6	7	8
15	5	24	8	1	3	16	10	20

0	1	2	3	4
15	5	24	8	1

5	6	7	8
3	16	10	20

0	1	2	3	4
15	5	24	8	1

3	16	10	20

3	16	10	20

0	1	2	8	1
15	5	24		

3	16	10	20

1	8

0	1
15	5

3	10	16	20

5	15

5	15	24

1	5	8	15	24

1	3	5	8	10	15	16	20	24

↑

This is our final completed sorted list which we getting by dividing into sub the given list into sublist then by sort & merging we get final completed sorted list.

Page No.
Date

IMP topic

Quick Sort

- Quick sort is a divide & conquer algorithm. It works by selecting a 'pivot' element here pivot means the element which is chosen randomly by a chance from the given set, so here quick sort works by selecting a pivot element from the array & partitioning the other elements into two sub-arrays, according to if the it, the elements which are less than the pivot element should be shifted to left side of the pivot element & the elements which greater than the pivot element should be shifted to right side of the pivot element.

- Quick sort is based on the idea that if an element is in sorted form if all the element which is lesser than that element is on the ~~be~~ left side & all the element which is greater than that element is after that on the right side of that element & that element is in sorted form

---

★ Now here we will solve some problems by Quick sort method with the help of divide & conquer.

i)

0	1	2	3	4	5	6	7
60	30	90	80	20	70	50	∞
lb						ub	
i							j

Let we take pivot = 60 so simple all the elements which is less than the pivot will be shifted to left side of the pivot & all the elements which is greater than the pivot will be shifted right side of the pivot.

Step 1) First of all from the left side we have to increment till we get the value greater than pivot so here we will increment to 30 which is not greater than 60 so we will increment again now on 90 which is greater than 60 then we will stop there, then from the right side we will decrease till the value get less than 60, first is 50 which is less than 60 so we will swap or interchange the digits 90 to 50.

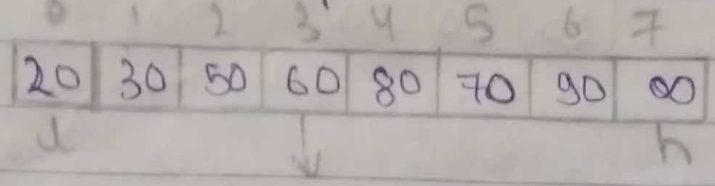
0	1	2	3	4	5	6	7
60	30	50	80	20	70	90	∞
l			i	j			h

Step 2) Now we will move to 80 which is greater than 60 from the left than from the right we will decrement to 70 but it is not less than 60 so again we will did decrement & move to 20, now stop their & swap both the 80 and 20.

0	1	2	3	4	5	6	7
60	30	50	20	80	70	90	∞
l			j	i			h

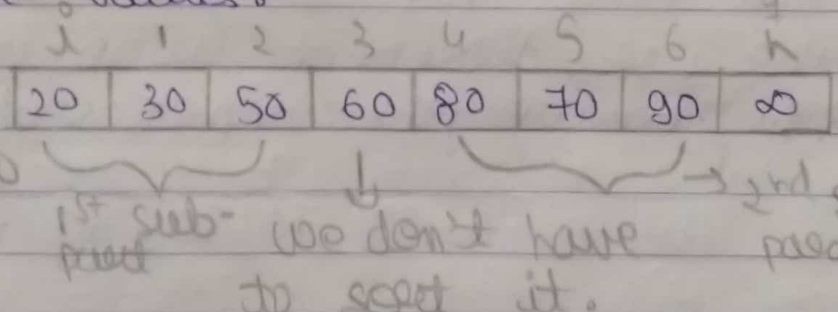
Page No
Date

step3) Now here i is crossed to j, i is greater than j so now we will not increment & decrement, now we will swap j with pivot element which is 60.



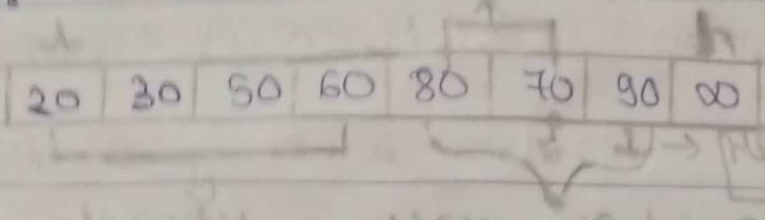
as before this all elements are lesser than it & after it all elements are greater than it.

Now we will divide it into two subparts first is of from 20, 30, 50 & next part is 80, 70 & 90, why we don't take 60 because it is already sorted now we don't know either than this other values are sort or not so we have to sort all other values.

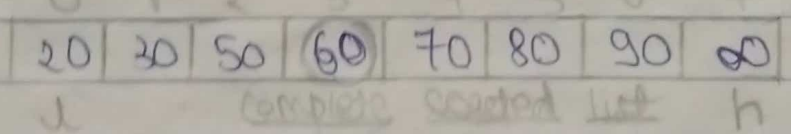


step4 Now we have to solve 1st subpart as we will take pivot is 20 then we will increment to 30 as 30 is greater than 20 so we will stop there than from the right we will decrement to 50 as 50 is not less than 20 so we move to 30 which is also not less than 20 at last we

one to 20 which is pivot so hence it is classified that 1st subpart is also sorted.



Steps) Now we will take pivot = 80 so we will increment to 70 which is not greater than 80 so we move towards to 90 which is greater than 80 so we will stop there then from the right we will decrement to 70 which is less than 80 but here we don't swap both of these because these are cross each other so we will swap the j or decrement value with pivot 80 then we get complete sorted list.



Ans

Here, infinity is used as a end marker. So it is just used to end the list.

So, we get the completed sorted list by using quick sort which is based on divide & conquer.

simple algorithm for it:

```
Quick sort (l, h)
{
   if (l < h)
  {
      j = partition (l, h);
      Quick sort (l, j);
      Quick sort (j+1, h);
  }
}
```

2) Now with the help of another example we will understand it better.

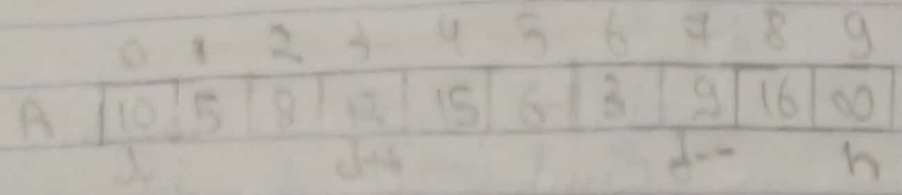
	0	1	2	3	4	5	6	7	8	9
A	10	16	8	12	15	6	3	9	5	∞
	l	i+1							j-1	h

Pivot = 10

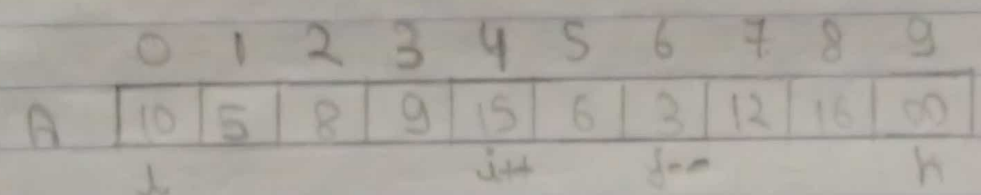
Solving this problem step by step:

Step1) As we take the pivot be 10, now we have to firstly sort this pivot as this pivot will be sort only when all the element which is greater than the pivot should be its right side & all the element which is less than pivot should be its left side. So, here first of all from left side we will increment till we get the element greater than that pivot so we will increment to 16 which is greater than pivot so we

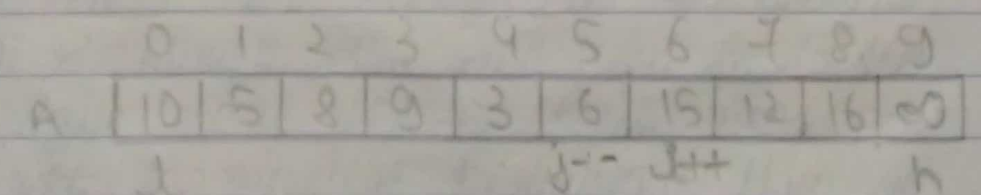
stop there then after that from left side we will decrement to 5 which is smaller than 10 so swap both of these or interchange both of these.



step 2) Now we will move to 12 or index value 3 A[3] so here 12 is greater than 10 so we stop there than from right we will decrement to 9 which is less than 10 so we will swap / interchange both of these :

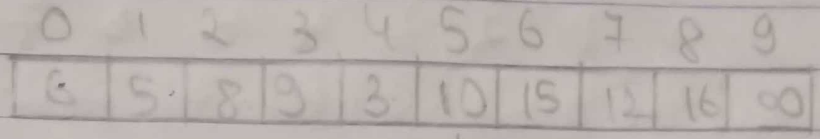


step 3) Now from left we will move to increment from 9 to 15 as 15 is greater than 10 so we stop there now then from right we will decrement to 3 which is less than 10 so we swap / interchange both of these :



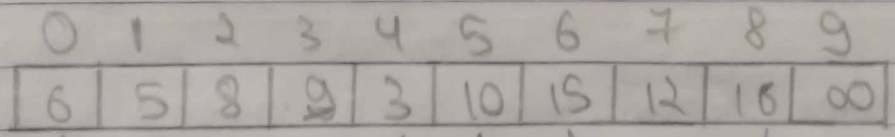
step 4) Here now, i is crossing to j which

means i is greater than j & then no increement & decrement which this type of case appears but after that i will be interchanged its position with the pivot value so in this case only pivot value will be sorted:



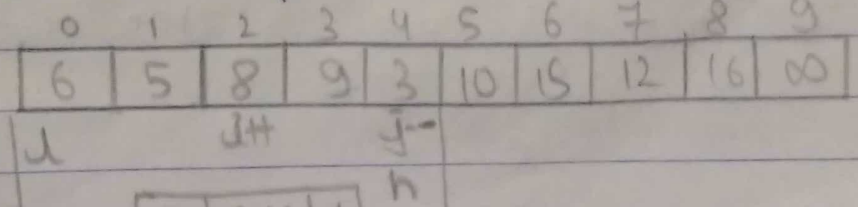
10 is sorted

Here how can we say that 10 is sorted as the values before pivot (10) is less than it & after pivot all the elements is greater than pivot so 10 is sorted. Now we will sort all the elements by dividing into subpart 1 & subpart 2 :

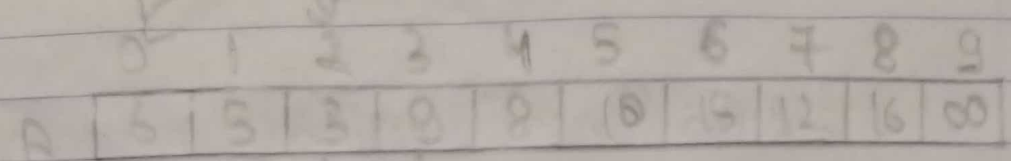


As, 10 is already sorted so don't need to sort it again

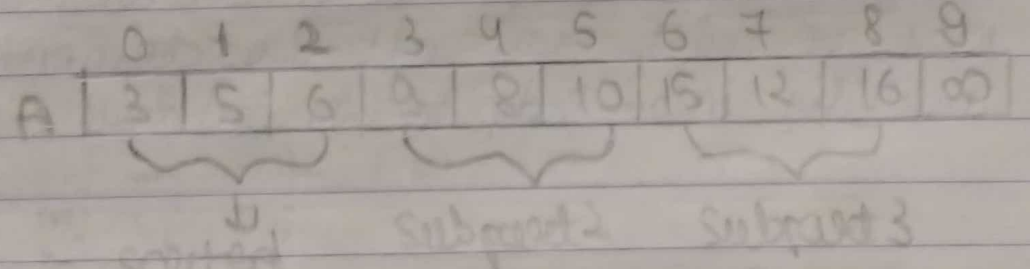
Steps) Here now when solving subpart 1 we will take 6 as pivot :



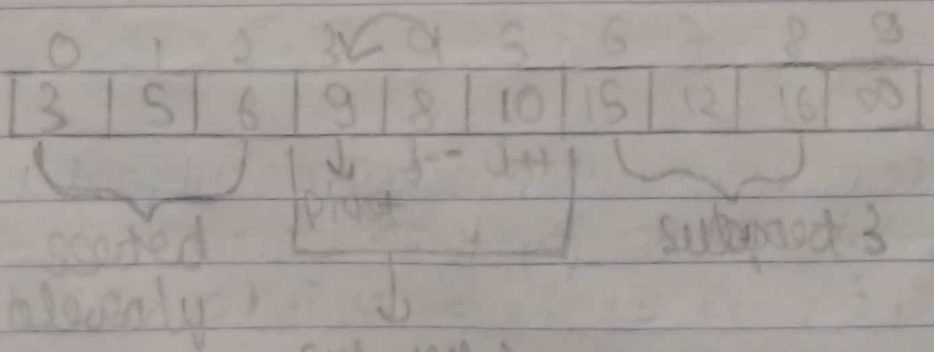
Step6) ∴ In the subpart 1, from the left we will increment to 5 which is not greater than 6 so we move towards to 8 which is greater than 6 so we stop there, now from right we will decrement to 3 so swap both of these:



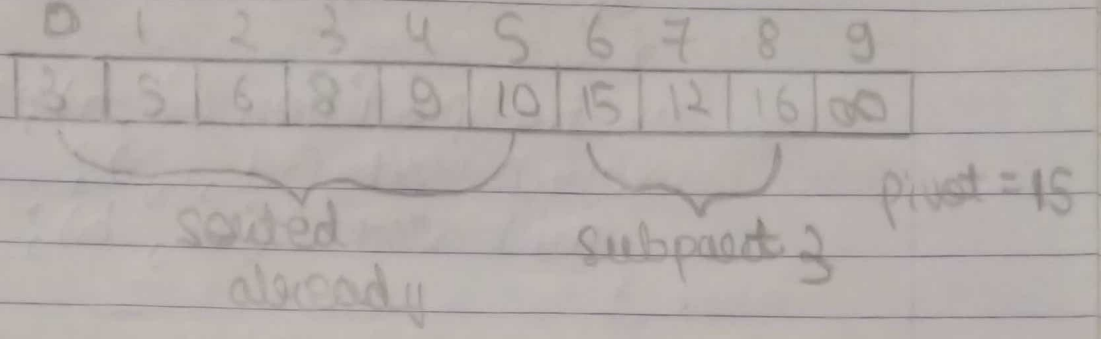
Step7) ∴ Now, we will swap i with pivot as we can't increment & decrement more cuz they both cross intersect each other so, i will interchanged with pivot which



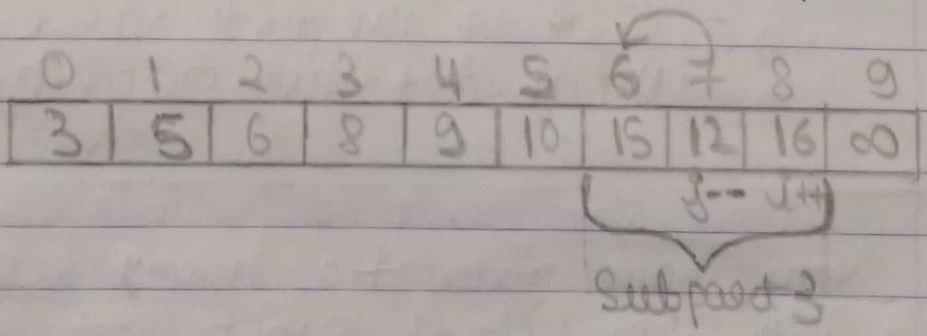
As, 10 is already sorted but with the help of 10 we will sort 9 & 8 that's why we will take help of it:



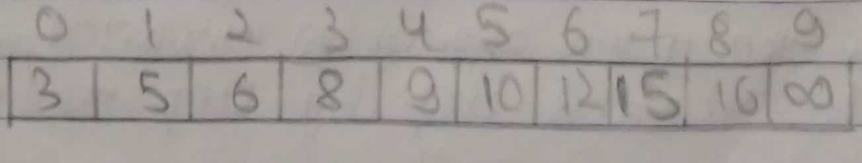
step8) Here in subpart 2 now we will increment to 8 as we take pivot 9 so 8 is not greater than 9 so we move toward to 10 & we also stop then from right we will decrement to 8 which less than 9 then we swap both 9 & 8 :



Now here from index 0 to 5, all the elements are sorted then we will solve the subpart 2 here we taking pivot = 15 :



step9) As, in subpart 3 i is crossed to j so we don't increment or decrement it, but we will swap j with pivot value is so we get completed sort list :



Ans

Thus :: here all the elements are

completely get sorted.

∴ Same algorithm for it:

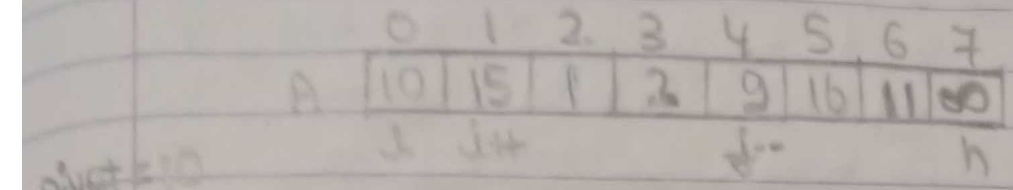
```
quick sort (l, h)
{
  if (l < h)
  {
    j = partition (l, h);
    quick sort (l, j);
    quick sort (j+1, h);
  }
}
```

∴ l = lower,
h = higher,
j = pivot or the point where from we divided two subparts

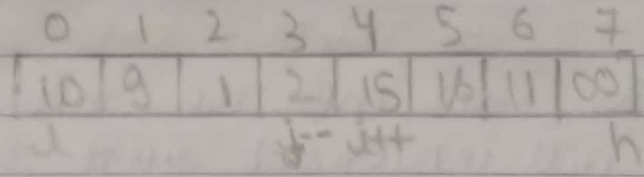
3) You have given the array sort this with the help of quick sort by using divide & conquer:

	0	1	2	3	4	5	6	7
A	10	15	1	2	9	16	11	∞
	l							h

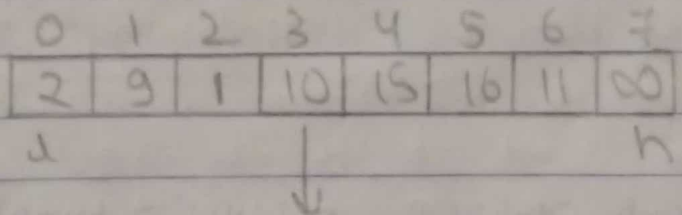
Here the given array is unsorted so we will sort this step by step method:



step1 Here we are taking pivot = 10 so first of all from the left we will increment till the element greater than pivot so here 15 is greater than 10 so we will stop there then from right we will decrement as 11 & 16 both are not less than 10 so we will decrement again, now 9 is less than 10 so we stop here & swap both the 15 & 9;



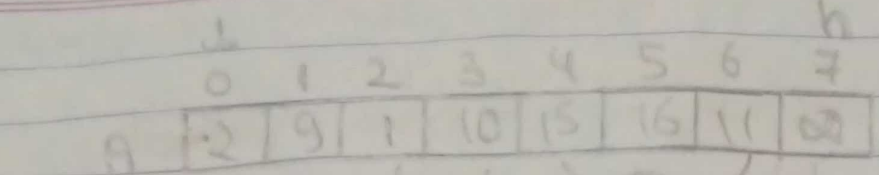
step2 Now here i is crossed the j so we cant increment or decrement anymore in this case we will swap j with the pivot value



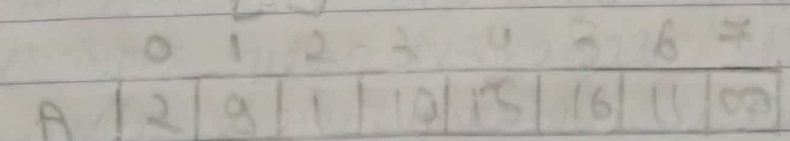
10 is sorted here;

Now we know that 10 is sorted?

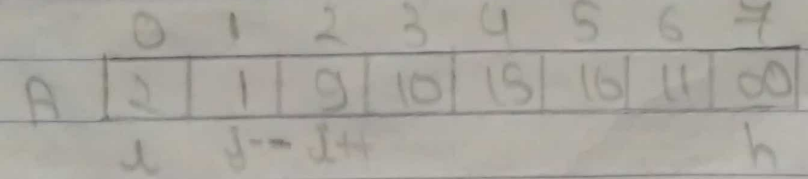
→ As all the elements which is less than 10 is on its left side & all the elements which is greater than 10 is on its right side;



Now we will solve subpart 1 firstly as how it works :

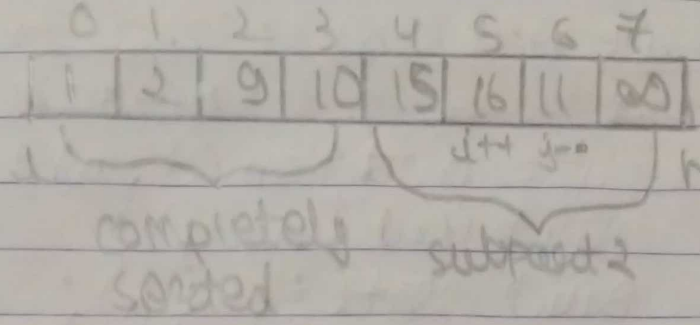


Steps As, here we are taking pivot = 2 so we will increment first to 9 here 9 is greater than 2 so it stops there then from left from index A[2] so it will decrement to 1 & then swap 9 & 1 :

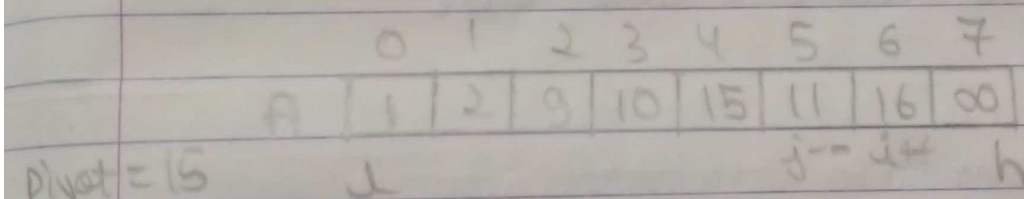


Step 4 Now control move towards from left that increment will to 9 whose 9 is greater than 2 so we stop there than from right decrement to 1 but here i is crossed with j so no increment & decrement will possible, so in this

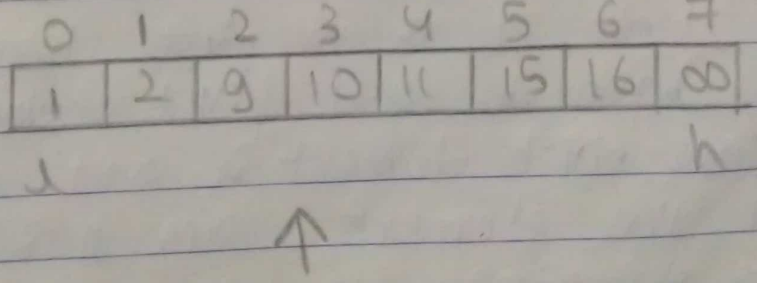
case we will swap j with pivot value & we get:



steps) Here now in subpart 2, from left we will increment to 16 as 16 is greater than the pivot value 15 so we stop here then from right 11 is less than 15 so we swap both 16 & 11 :



steps) Here i is crossed j so increment & decrement is not possible so in this case we will swap j with pivot value & then we will get complete sorted list:



Ans

Here is the complete sorted list.

Page No
Date

Algorithm: (same as before):

```
Quick sort (l, h) {
if (l < h)
 {
  j = partition (l, h);
Quick sort (l, j);
Quick sort (j+1, h);
 }
}
```

where, l = lower size
h = higher size
j = pivot or the point where we divide 2 subparts.

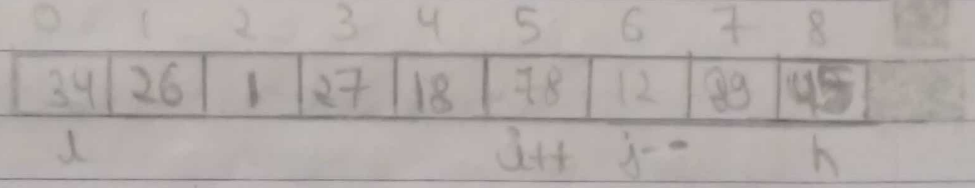
* Here another example:

0	1	2	3	4	5	6	7	8
34	26	1	45	18	78	12	89	27
Pivot = 34	l							h

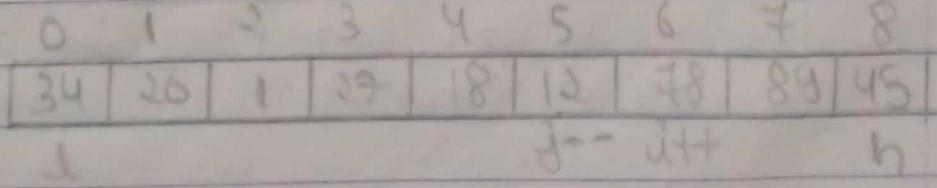
step 1) Here is the given array list in which we took 34 as an pivot element & firstly we have to sort pivot element which means pivot element is sort only when all the elements lesser than pivot should be shifted on its left & all elements greater than pivot should be shifted on its right.

0	1	2	3	4	5	6	7	8
34	26	1	45	18	78	12	89	27

step2) Here firstly we will increment i till we get the element greater than pivot so here first from the left we will increment to 26 but 26 is not greater that pivot element again we will increment to 1 also 1 is not greater than pivot element so again we will increment to 45 which is greater than 34 so we will stop there than from the right we decrement to 27 which is less than the pivot element so we swap both of the elements 45 & 27:

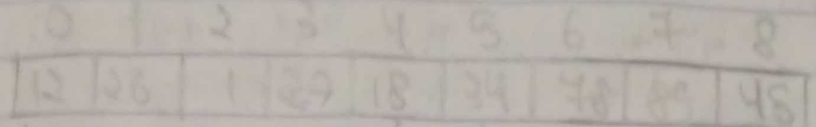


step3) Here, now from the we will move to 18 which is not greater than 34 so again we will increment to 78 which is greater than 34 so we stop there, now from the right we will deg decrement to 89 but 89 is not less than 34 so again decrement to 12 which is less than 34, so now we will swap both of the element 78 & 12:

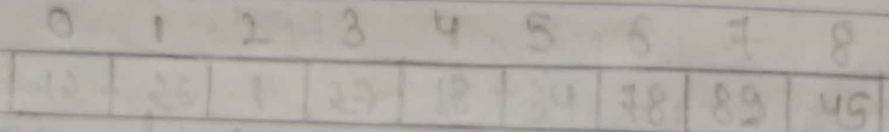


step4) As i crossed j so we stop increment & decrement as the value of i is greater than j so in this case we will swap j with

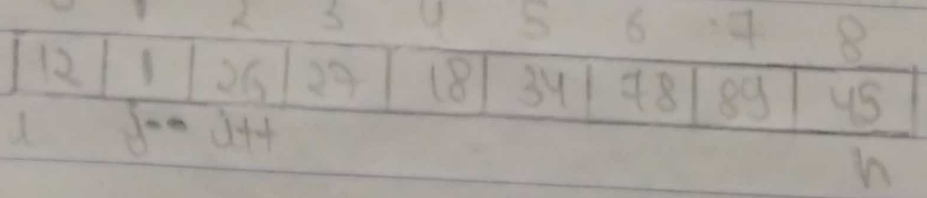
the pivot value:



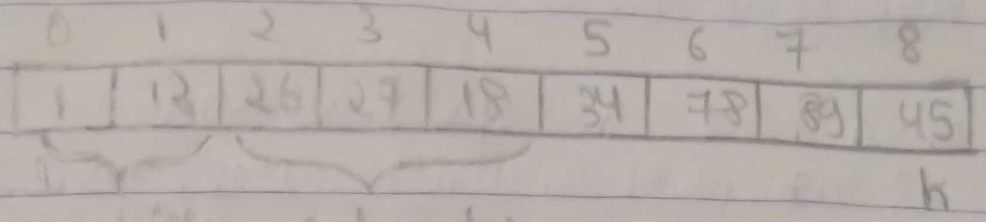
Step 5) Now we will solve firstly subpart 1 & here we took pivot as 12 :



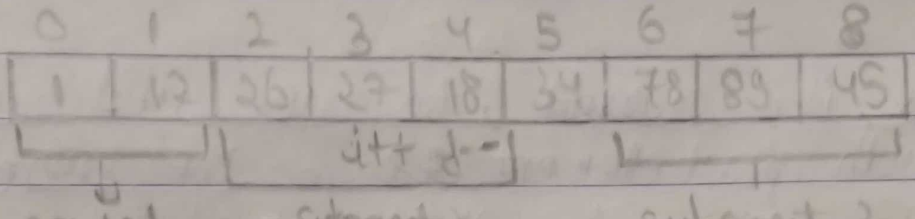
Step 6) As from the left we will increment to 26 which is greater than 12 so we stop there, now from the right we will decrement to 18 which is not less than 12 so again move to 27 which is also not less than 27 again we will decrement to 1 which is less than 12 so we swap these



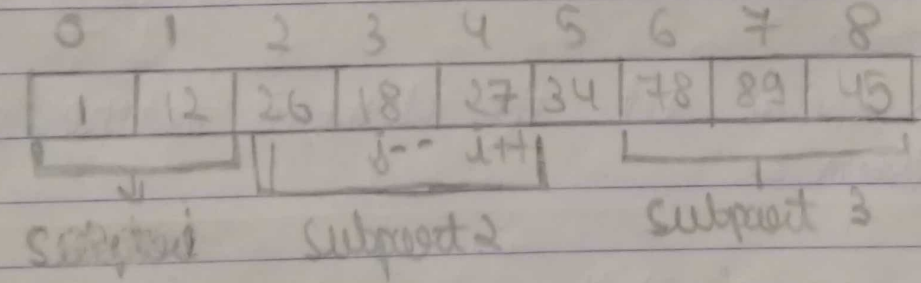
Step 7) Then, i crossed j so in this case we will swap j with the pivot element & we get :



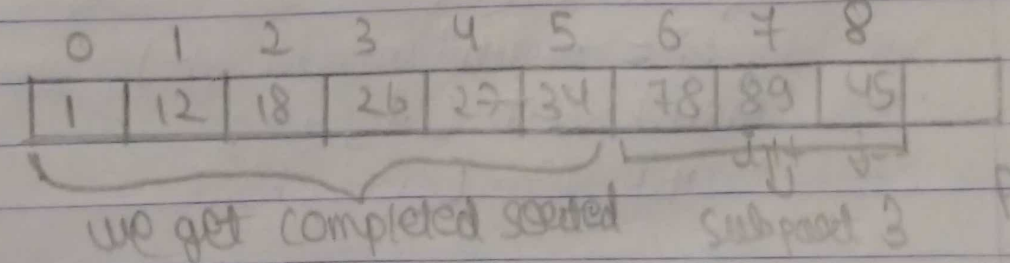
Step8: Now we will solve subpart 2 from index value 2 to 4 as 34 is already sorted so we doesn't involve this:



Step9: Here in subpart 2, we will increment to 27 from the left & from the right decrement to 18 then swap both of these:

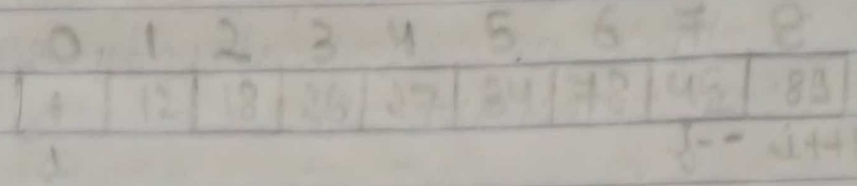


Step10: Here i crossed j so we will swap j with the pivot element:

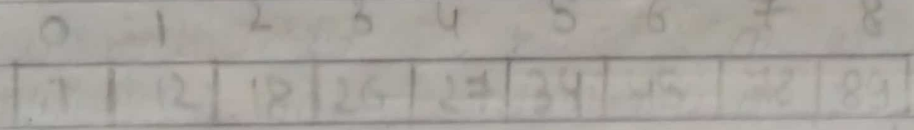


Step

Step(1) Now here in the subpart 3, we take 78 as pivot & will increment to 89 & stop there now from the right side decrement to 45 which is less than pivot so swap both of these:



Step(2) As i is crossed j so we will swap j with pivot element 78 in the subpart 3 so we will get completely sorted list.



As, here all the elements in the sorted form:

- Now algorithm:

```
Quick sort (l, h) {
if (l < h) {
   j = partition (l, h);
Quick sort (l, j);
Quick sort (j+1, h);
}
}
```

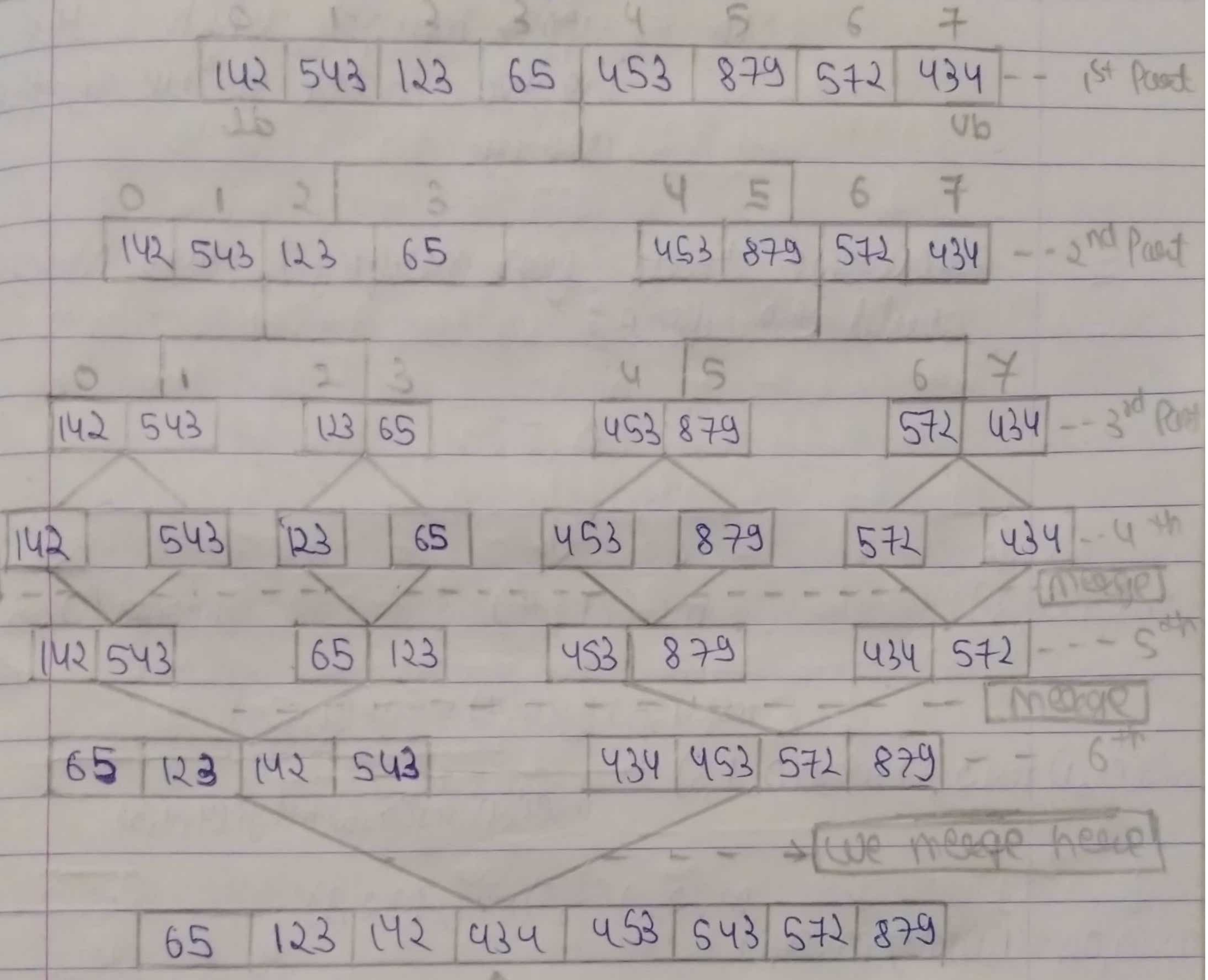
mid = $\frac{lb+ub}{2}$ → (i) $\frac{0+7}{2} = 3$, (ii) $\frac{0+3}{2} = 1$

(iii) $\frac{0+1}{2} = 0$ (iv) $\frac{4+7}{2} = 5$

Here are some questions : IMP RGPV

[H.W] [solved]

You have given the array, solve this array by using merge sort:



Algorithm :

Here we get the completed sorted list : Algorithm for merge sort :

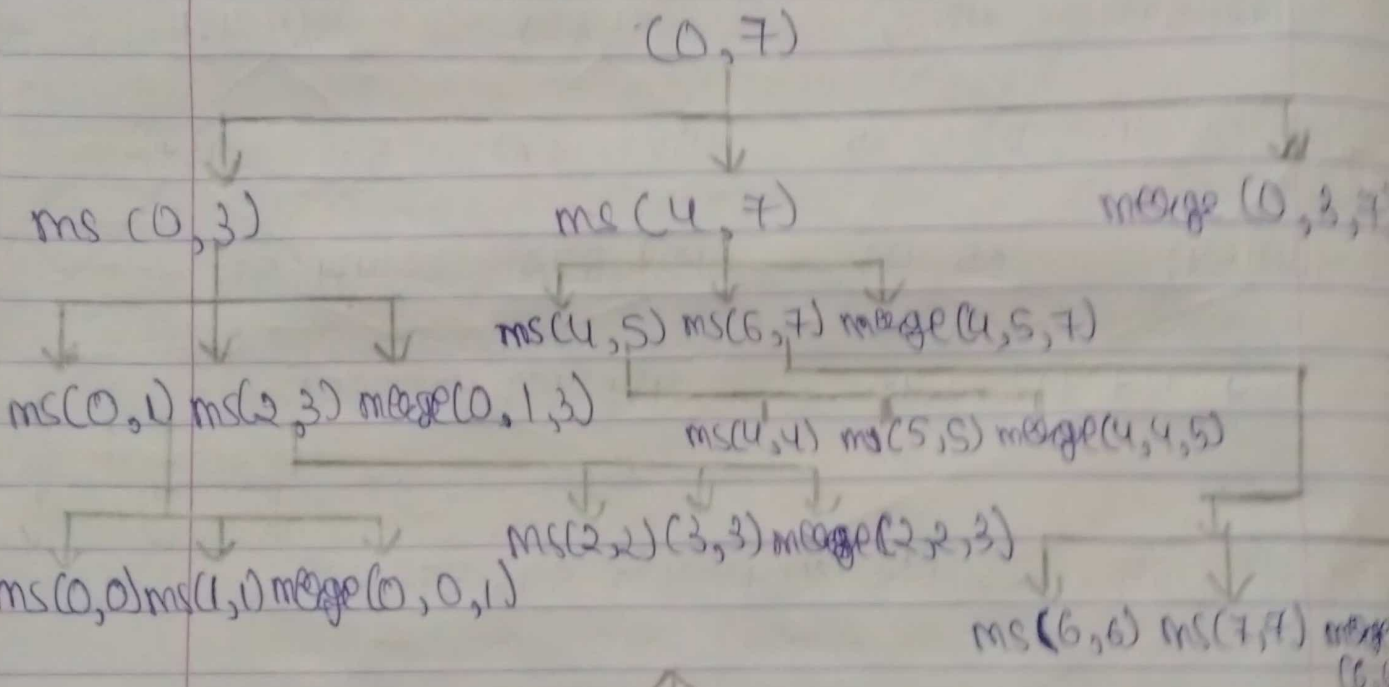
```
merge sort (A, lb, ub) {
if (lb < ub) {
mid point = lb + ub / 2
merge sort (A, lb, midpoint)
merge sort (A, midpoint+1, ub)
 merge (A, lb, mid point, ub)
```

(IMP) After merging the elements we get that element in the sorted form:

where, lb = lower bound which is always starts from 0, means index value be 0.

ub = upper bound which is the largest index value of the array.

* Recursive tree for merging / how merging could be done:



this is the full recursive tree for how to merge the elements after dividing the element.

Simply it is the method to merge the subparts

Page No
Date

ans2) You have given the array, solve the problem by using Quick sort:

0	1	2	3	4	5	6	7	8	9	10	11
44	33	11	55	77	90	40	60	99	22	88	66
l											h

pivot = 44

step1) Here we take pivot = 44 so what we have to do is that we have to sort 44 as the elements which is less than pivot should be shifted to its left & the elements which is greater than pivot should be shifted to its right. Now firstly we will increment towards 55 as this element only after 44 is greater so we will stop there & from the right we will decrement towards 22 which is less than 44:

0	1	2	3	4	5	6	7	8	9	10	11
44	33	11	55	77	90	40	60	99	22	88	66
l			i++						j--		h

step2) Here when comparing with pivot, 33 & 11 both are not greater from 44 so we will increment to 55 from the left then from the right as 66 & 88 are not less than 44 so we move to 22 then swap both of the element:

0	1	2	3	4	5	6	7	8	9	10	11
44	33	11	22	77	90	40	60	99	55	88	66
l				i++		j--					h

Step 3) Now we move to 77 which is greater than pivot so we stop there now from the right we decrement to 40 which is less than pivot so we swap both i & j

0	1	2	3	4	5	6	7	8	9	10	11
44	33	11	22	40	90	77	60	99	55	88	66
				j--	i++						

Step 4) As i crossed j so we don't increment & decrement now, in this case we swap the value of j with the pivot value:

0	1	2	3	4	5	6	7	8	9	10	11
40	33	11	22	44	90	77	60	99	55	88	66

Subpart 1 | Subpart 2

∴ As we can see that 44 is only get sorted as the values greater than 44 is on its right side & the values less than 44 is on its left side, so we can clearly say that 44 is get sorted.

∴ Now take subpart 1 & here we take its pivot value is 40 then 33 where we increment to 40 from right & decrement to 22 from left:

0	1	2	3	4	5	6	7	8	9	10	11
40	33	11	22	44	90	77	60	99	55	88	66
	i++		j--								

steps) Now swap both i & j :

0	1	2	3	4	5	6	7	8	9	10	11
22	33	11	40	44	90	77	60	99	55	88	66
		j--	i++								

steps) As we increment i to 40 which is greater than 33 (pivot value). as i crossed j so in this case j is swap with pivot value :

0	1	2	3	4	5	6	7	8	9	10	11
22	11	33	40	44	90	77	60	99	55	88	66
	j--	i++									

Again let we take pivot = 22, here in this case i crossed j so, j is swap with pivot value (here 11 replaced with 22) :

0	1	2	3	4	5	6	7	8	9	10	11
11	22	33	40	44	90	77	60	99	55	88	66

this part is get sorted | subpart 2

(ep7) ∴ In subpart 2 we will take 90 as pivot & we will increment at 99 which is greater than 90 from left & from right we will decrement to 66 : which is less than 90 :

0	1	2	3	4	5	6	7	8	9	10	11
11	22	33	40	44	90	77	60	99	55	88	66
								i++			j--

Step8): Now we will swap the values of i & j:

0	1	2	3	4	5	6	7	8	9	10	11
11	22	33	40	44	90	77	60	66	55	88	99
										j--	i++

Step9): As, i crossed j so in this case increment & decrement is not possible as i is larger than j, so in this case we swap the value of j with pivot:

0	1	2	3	4	5	6	7	8	9	10	11
11	22	33	40	44	88	77	60	66	55	90	99
									j--	i++	

Step10): Now we take pivot as 88 then here i is crossed j so in this case we will swap j with pivot value.

0	1	2	3	4	5	6	7	8	9	10	11
11	22	33	40	44	55	77	60	66	88	90	99
										j--	i++

Step11): Here, we took pivot 90 & then increment towards 99 & decrement towards 90, then swap j with pivot 90 so we get same result it means 90, 99 both are sort:

0	1	2	3	4	5	6	7	8	9	10	11
11	22	33	40	44	55	77	60	66	88	90	99
										j--	i++

step(2): In Subpart 4, pivot = 55 then increment to 77 & decrement to 55 so in this case i crossed j so we swap pivot value with j means 55 is placed with 55, it means 55 is sorted:

0	1	2	3	4	5	6	7	8	9	10	11
11	22	33	40	44	55	77	60	66	88	90	99

i++ j--

this is completely sorted — pivot = 60 — sorted

step(3) Here we take index 6 & index 7 so we took pivot 60 so we increment to 77 & decrement to 60 then swap both the values of i & j:

0	1	2	3	4	5	6	7	8	9	10	11
11	22	33	40	44	55	60	77	66	88	90	99

i++ j--

sorted — 60 also got sorted — pivot = 77 subpart 4 — sorted

step(4) Here in subpart 4 we took pivot there are two elements only so increment to 77 & decrement to 66 then swap both of these & we get completed sorted list:

0	1	2	3	4	5	6	7	8	9	10	11
11	22	33	40	44	55	60	66	77	88	90	99

completed sorted list

Ans

Page No.
Date

Algorithm :-

```
quicksort (l, h) {
if (l < h) {
  j = partition (l, h)
quick sort (l, j)
quick sort (j+1, h)
    }
    }
```

Recursion

→ In this a function called itself & in recursion a single prob is divided into subprobs to solve the problem easily.

One of the most imp. topic of recursion is Towers of Hanoi.

Here are some rules for solving the problems of Towers of Hannoi:

i) Only one disk can be moved at a time. (एक time पर एक ही disk move हो सकती है)

ii) In simple language, you can move the upper disk from the any pillar & also only place on the top of the pillar.

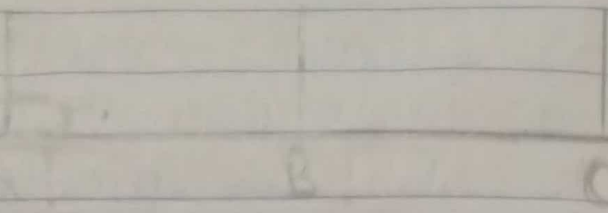
iii) Each move consists of taking the upper disk from one of the stacks & placing it on top of another stack. In other words, a disk can only be moved if it is the uppermost disk on on a stack.

iv) No largeer disk can be placed on top of the smaller dick. (बड़ी नीचे रहेगी & फिर उसके नीचे उससे small रहेगी)

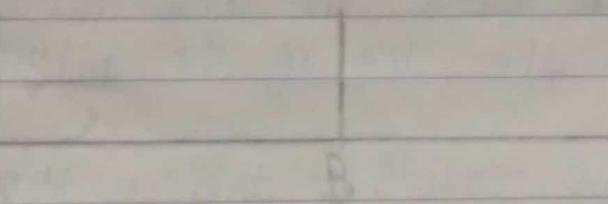
Solution for dicks :

Page No.
Date

* Solution for single disc :

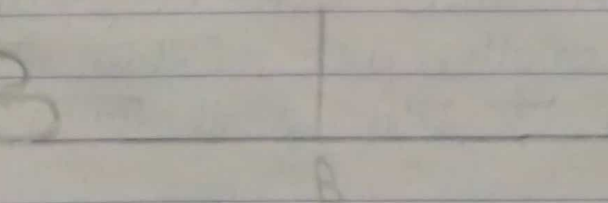


∴ There were three pillars, A, B & C here A is source pillar, B is auxiliary (helping) pillar & C is destination pillar. When we talk about single disk, we can directly move a disk from A to C but if we have without taking help from the B pillar but if A had multiple no. of disc then we can't move directly from A to C disc that time we will take help from pillar B.



Here, we completely moved the disc from A to C.

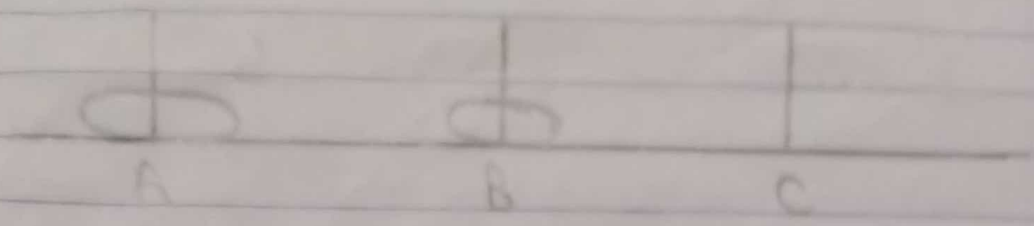
* Now, solution for 2 disc :



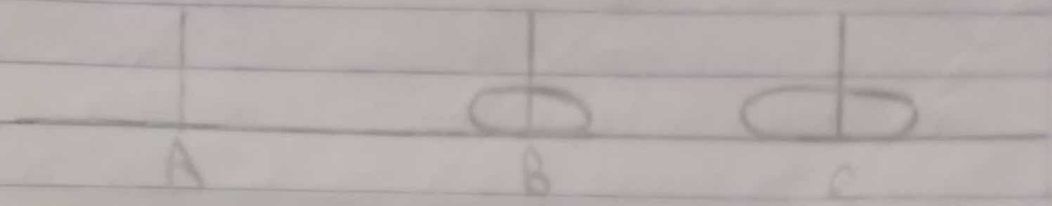
∴ • Move a disc from A to B using C
• Then move a disc from A to C using B

∴ At last move a disc from B to C :

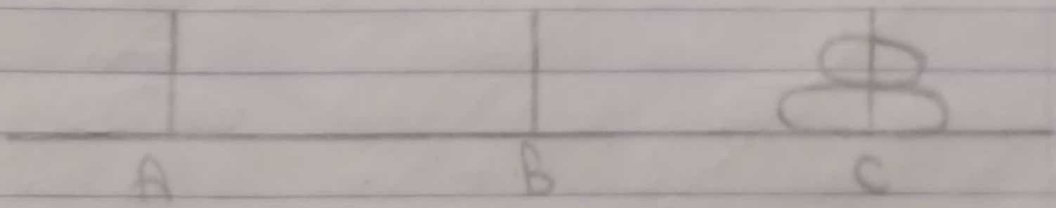
∴ i) move a disc from A to B:



ii) move a disc from A to C:

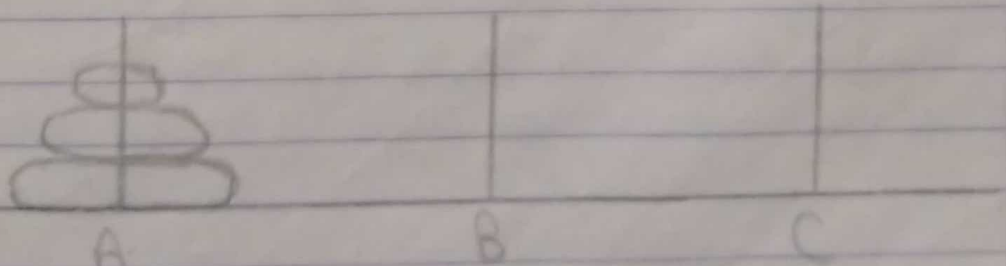


iii) move a disc from B to C for getting the final result:

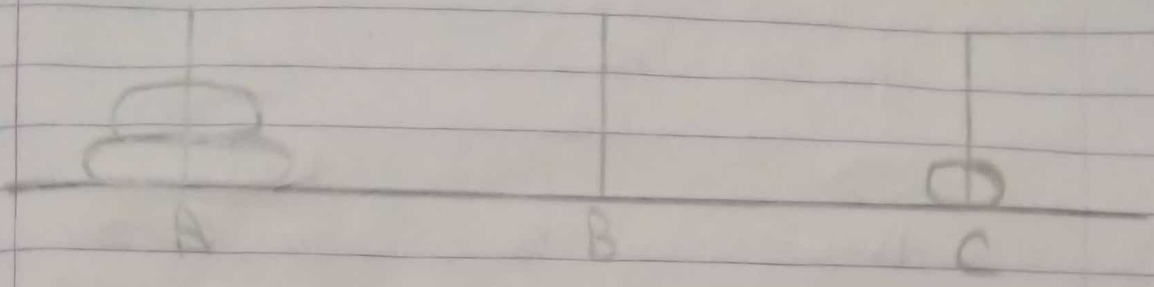


Here we completely moved all the disc from source to destination pillar.

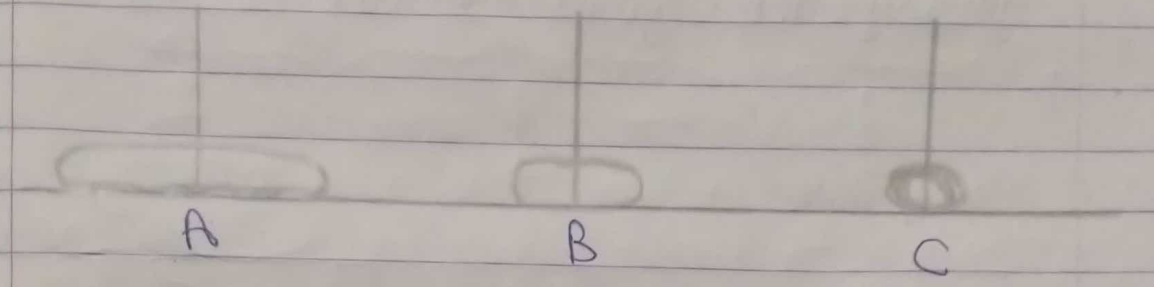
* Now solution for 3 disc:



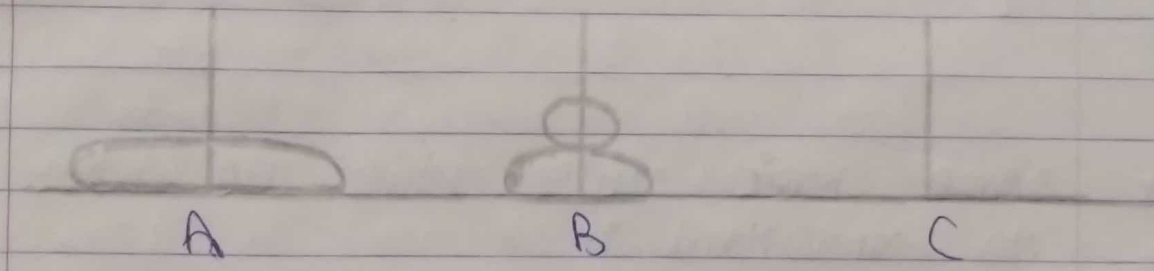
step1) Now move a disc from A to C:



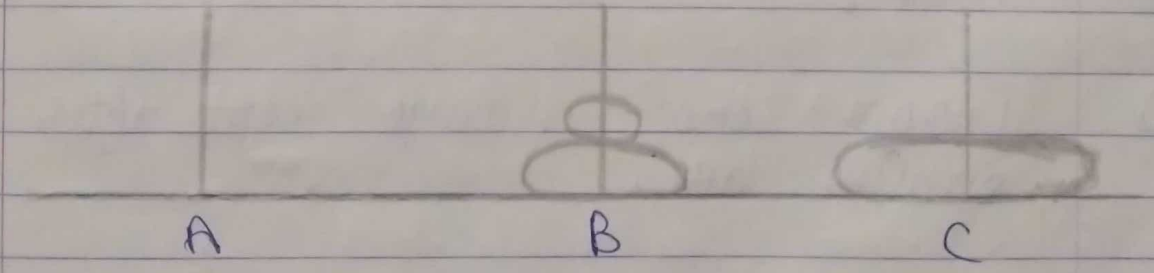
step2) move a disc from A to B:



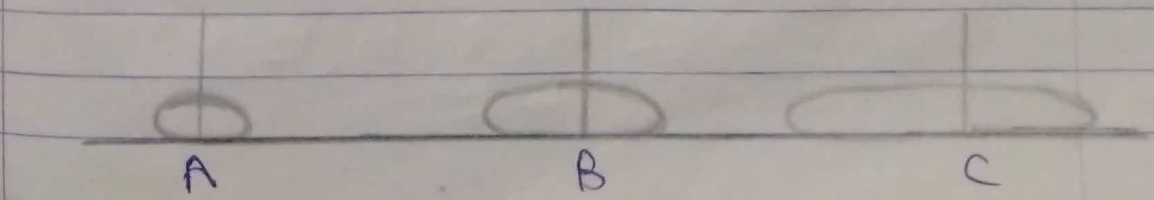
step3) move a disc from C to B:



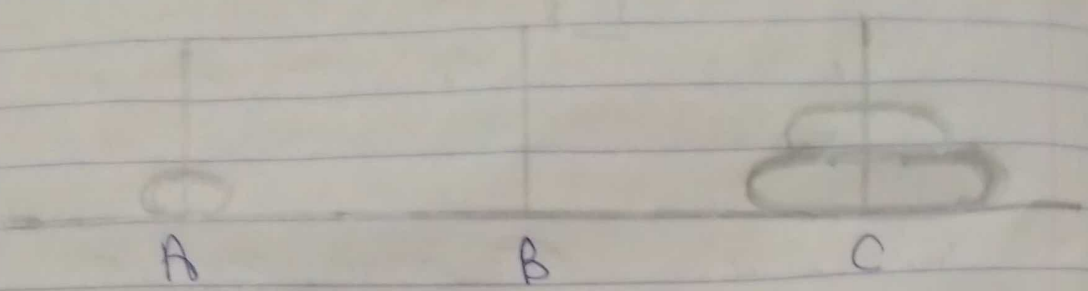
step4) Now, move a disc from A to C:



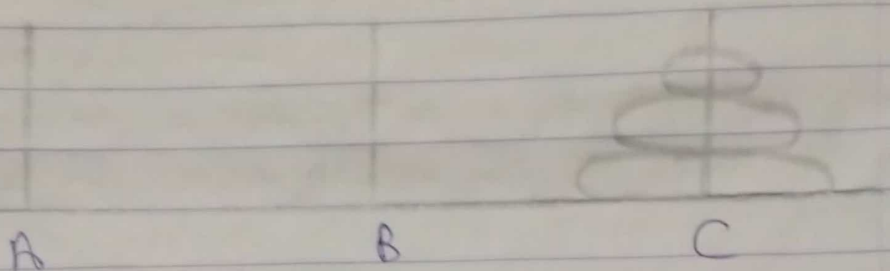
step5) Now, move a disc from B to A:



step6) Move a disc from B to C



step7) Move a disc from A to C for getting the final result:



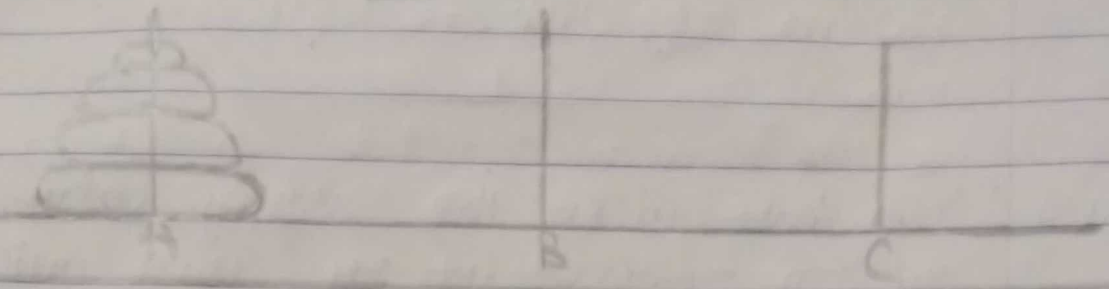
* some basic rules which we have to understand :

i) We can move only one disc at a time.

ii) Larger disc always kept after smaller disc.

iii) The disc which is on the topmost can only move & placed.

* solution for n disk :



steps:
1) move n-1 disk from A to B using C.
2) move a disk from A to C.
3) move n-1 disk from B to C using A.

↳ Here if n = 4, by using upper diagram if n=4 then firstly we will move 3 (4-1) disk to B then we move A disk from A to C, atlast we move that 3 (4-1) disk to C for getting completely final result :

- some rules for n no. of disks :

i) move the top n-1 disks from source to destination auxiliary pillar.

ii) move the n^{th} disk from source to destination pillar.

iii) move the n-1 disk from auxiliary pillar to destination pillar.

iv) Recursively the top n-1 disks from source to auxiliary pillar can again be thought as a fresh problem & can be solved in the same manner.

* steps will be $\boxed{2^n - 1}$ where n is no. of disk.

Que → You have given the array, solve the problem & sort it by using quick sort

0	1	2	3	4	5	6	7	8
10	4	12	2	14	8	16	6	18

Step1) We have taken the pivot = 10 so we have to sort the pivot as like the elements which is less than the pivot should be shifted to its left & the elements which is greater than the pivot should be shifted to its right. So, we will increment towards 12 from the left as it is greater than the pivot & from the right we will decrement to 6 as it is less than 10 so we will point it as j;

0	1	2	3	4	5	6	7	8
10	4	12	2	14	8	16	6	18

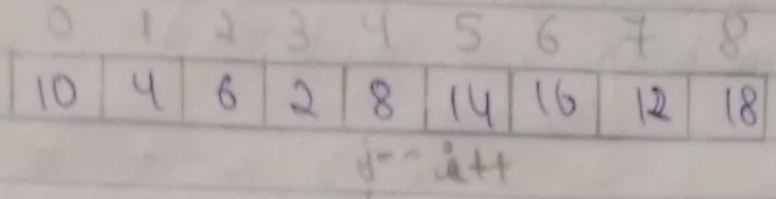
i++ j--

Step2) Now swap both i & j;

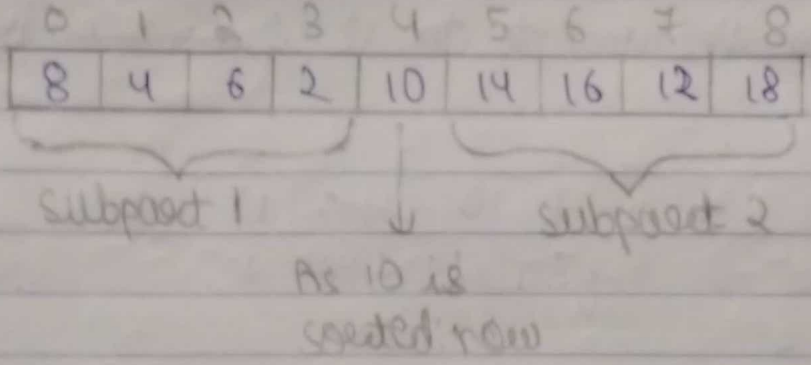
0	1	2	3	4	5	6	7	8
10	4	6	2	14	8	16	12	18

i++ j--

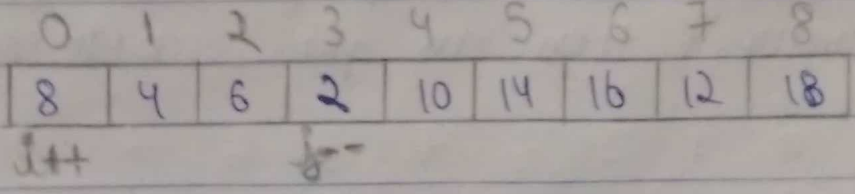
Step3: Now increment(i) to 14 & decrement(j) to 8 then swap both of i & j which we incre. & decre.



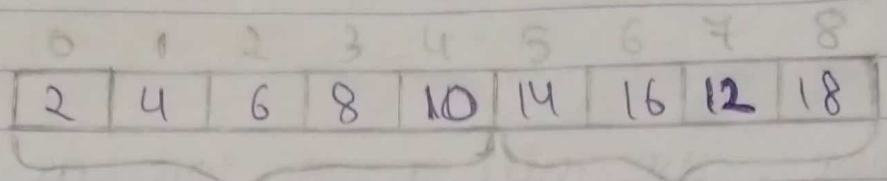
Step4: Nowe, we will move towards for finding next i & j, so we will find here that i crossed j so in this case we swap j with the pivot value:



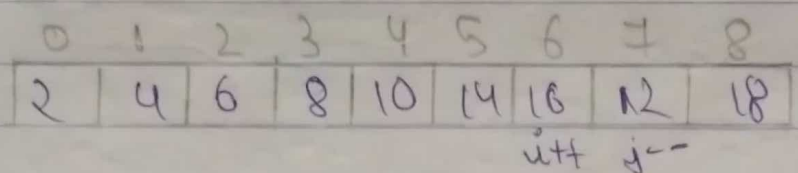
∴ Here, as 10 is sorted we all the elements which is less than it is on its left side & all the elements which is greater than it is on its right side. Now we will solve subpart 1 where we take 4 as pivot:



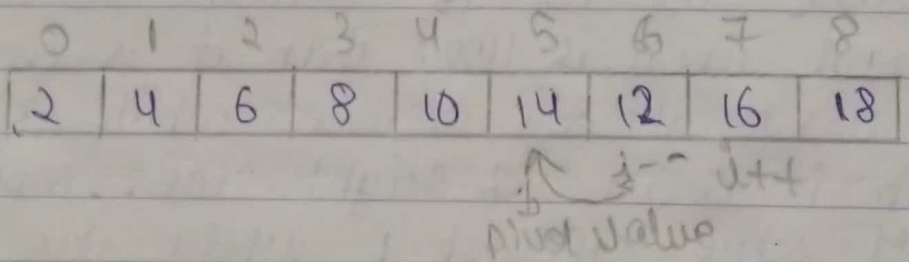
Step5: As we have taken pivot = 4 so we will increment towards 8 which is greater than 4 from left so we will stop there & then from right decrement to 2 which is less than pivot value, so we will swap both i & j (decre.):



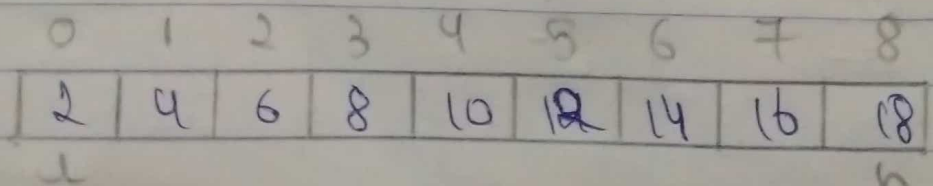
∴ Now in subpart 2 we will take pivot = 1 & will increment to 16 from left & then decrement to 12 from right:



Step6: Now swap both i & j:



Step6: As here now i crossed j so in this case we will swap j with its pivot value & then we get complete sorted list



Ans

- Algorithm :

```
Quick sort (l, h) {
 if (l < h) {
   y = pivot/partition (l, h)
 Quick sort (l, j)
 Quick sort (j+1, l)
   }
   }
```

Remaining tower of Hanoi

- Tracing for 3 discs

- As for n no. of disk :

- If we have three towers A is source tower, B is auxiliary tower & C is destination tower. So for n no. of tower :

1) Move n-1 disk from A to B.
2) Move a [illegible] (nth) disk from A to C.
3) Move n-1 disk from B to C.

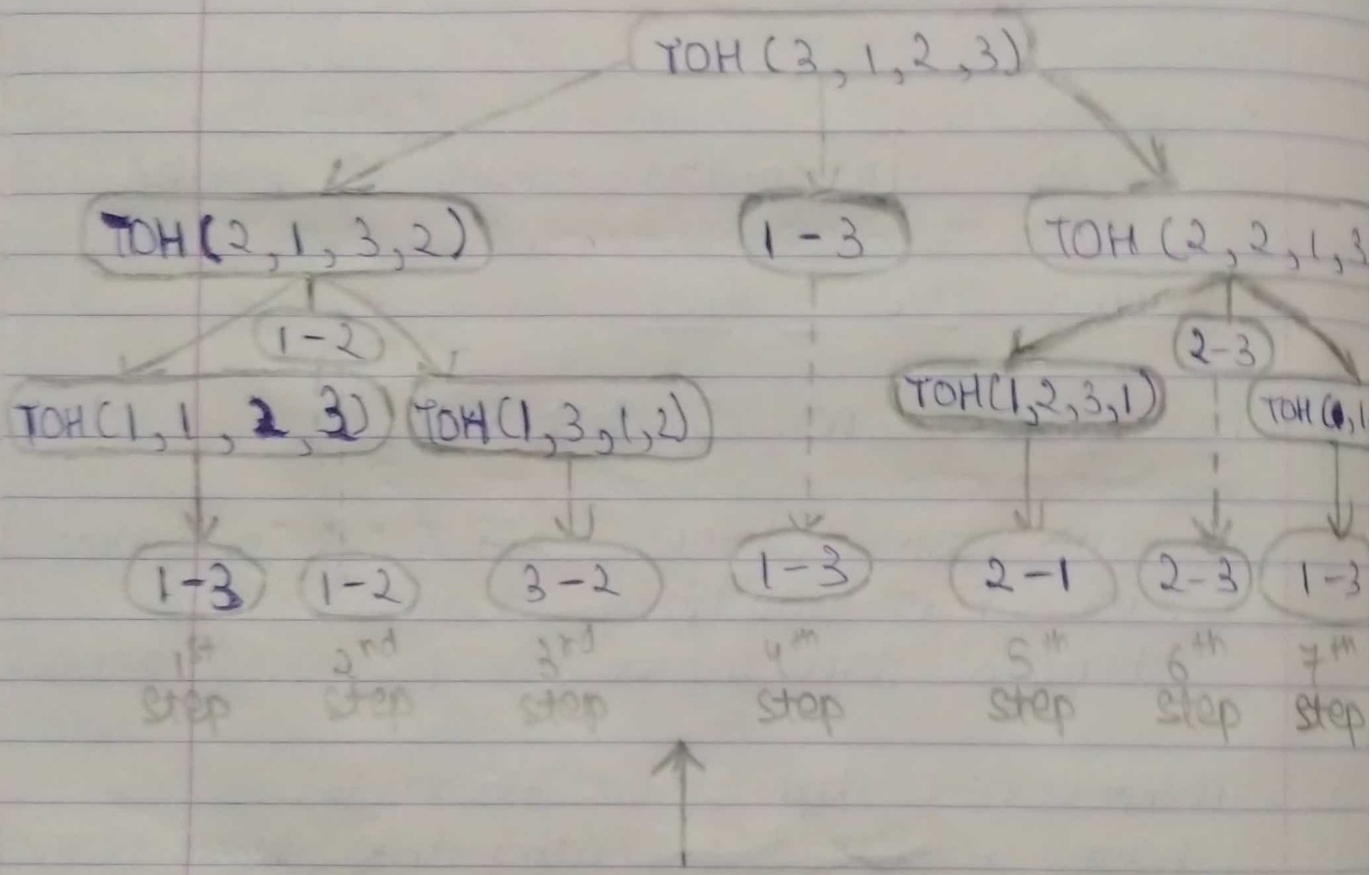
- The program for n disk :

```
void TOH (int n, int A, int B, int C)
{
   if (n > 0)
   {
      TOH (n-1, A, C, B);
```

move n-1 disc from A to B
move nth disc from A to C
move n-1 disc from B to C

printf("move a disc from %d to %d", A, C);
TOH(n-1, B, A, C);
}
}

Tracing for 3 Discs



This is how we can done transferring of disc from source to destination.

∴ For 4, 5 & any disc we can transfer easily with this method.

Fibonacci Series

- Another example of Recursion is Fibinacci series :

- Fibinacci series :- इसमे पहले के 2 no. का sum हमे third term में मिलता है तो आगे ये series आगे बढ़ती है तो second term first term में convert हो जाती है & previous 2 term का sum second term में आ जाता है।

Like there is a series :
0, 1, 1, 2, 3, 5, 8, ...

next term = first + second
first term = second
second term = next term

Write a program to print fibinacci series :

```
#include <stdio.h>
int main(){
int first = 0, second = 1, num, i, next;

printf("Enter the number: ");
scanf("%d", &num);

printf("The %d nombers in fibinacci series
        is: ", num);

for(i = 0; i<num; i++){
 if ( i <= 1) {
    next = i;
  }
```

```
else {
    next = first + second;
    first = second;
    second = next;
    }
printf("%d", next);
    }
return 0;
    }
```

Input:

Enter the number : 5

Output:

The 5 numbers in fibinacci series is:
0 1 1 2 3