Paper

DSA

(IMPORT TOPIC)

PAGE NO.:
DATE: / /

Data structure, types (Big O, Big theta, Big omega)
Algorithm, Analysis of Algorithm & Time complexity.
Array, stack, queue, sparse matrix.
linear search & Binary search.
Quick sort, merge sort, Heap sort & its time best & worst complexity.
Tree, BST, BFS, AVL, DFS B+ tree & B tree.

DSA UNIT-1

PAGE NO.:
DATE: / /

Data structure and algorithm

Data structure → arrangement of data

algorithm → step by step solution of a problem

In computer when we write algorithm of any problem then at that time data arrangement should be these

Properties of algorithm →

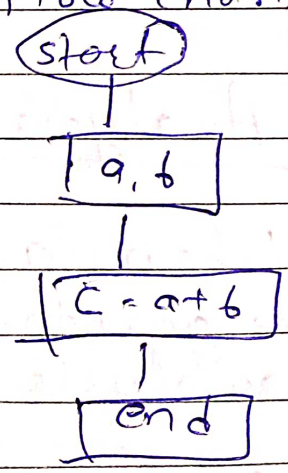
(I) finite steps
(II) precise & meaningful
(III) language free

Algorithm

Pseudo code

Begin
Take two no a, b
c ← a + b ;
print c
End

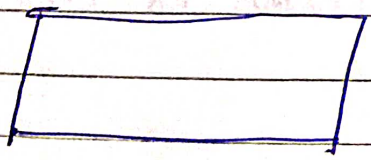
Flow chart



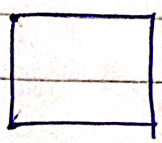
#

start / stop

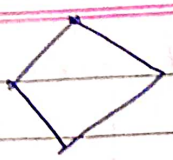
#



input / output

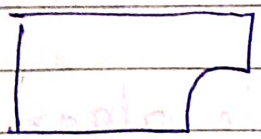


Process



Branching / Decision / Diamond

→ Flow



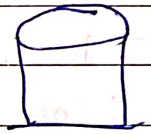
Document

page connector

Multi input mixer

} comments

HDD



DBMS

Analysis of algorithm -

When a problem have more than one algorithm is best is decided by the analysis.

An analysis of algorithm is called complexity.

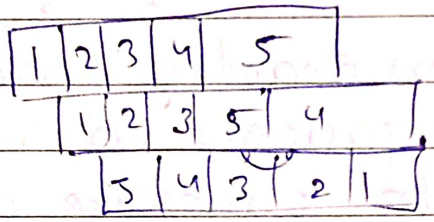
PAGE NO.:
DATE: / /

Complexity

space complexity
It tells us how much space is occupied by the program

Time Complexity
Time taken to run a program by central processing unit

#cases
1. Best case
2. Average case
3. worst case



Arrange in ascending order

Data structure Type

Primitive DS
INT
Float
Double
Boolean
char

Non primitive DS
linear
stack
queue
linked list
Array
string

Non linear
1. tree
2. graph

DATE: / /

operation on data structures

(1) Traversing
(2) searching
(3) Insertion
(4) merging
(5) Delete
(6) sorting

Traversing =>

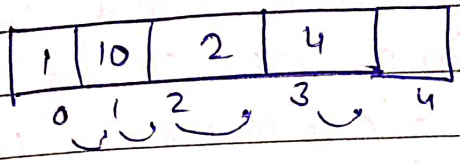
It is used to process each data item exactly once.
It is a process in which each element of a data structure is accessed.
It is performed to display every element of data structure.
It simply means that "visiting" or "touching" the element of the structure & doing some thing with the data.

Searching =>

It is used to find out the location of the data item.
The process of finding the desired information from the set of items stored in the form of the element in the computer memory is called searching in data structure.
These sets of item are in various form such as array, tree, graph

Insertion =

It is used to add the new data item. Insert operation is to insert one or more data element into an array.



Pos = 4 new element

new Pos = 5

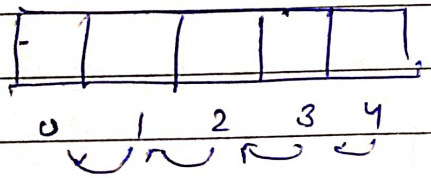
Best case = Empty case

worst case = [0]

Deleting =

It is used to delet an existing data item from the given collection of data structure. Item.

It refers to removing an existing element from the array and re-organizing all element of an array.



Best case = last index

worst case : [0] index

Sorting =

It is used to arrange data item is some order either ascending or descending.

The arrangment of data in a preferred order is called sorting in data structure By sorting data. It is easier to search through it quickly & easily.

Example dictonary.

Saturday 04/02/23

Date

linked list =>

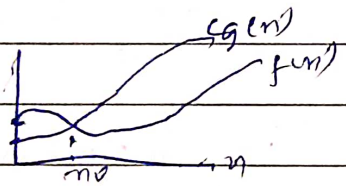
It is the most sought after data structure when it comes to handling dynamic data element

A linked list consists of a data element known as node. And each node consist of two fields one field has data & in second field, the node has an address that keep a refrence to next node

It is sequence of data structure which are connected together via links.

It is often used because of their efficient insertion & deletion.

Big O :



f(n)=O(g(n))

It is define as upper bound and upper bound on an algorithm. is the most amount of time required. (Waste case performance)

It is used to express algorithmic complexity using algebraic terms.

It describe the upper bound of an algorithm runtime & calculate the time & amount of memory needed to execute the algorithm for an input value.

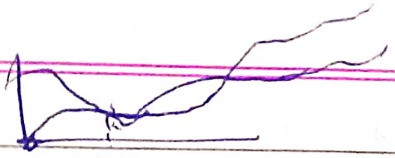
It is a way to measure an algorithm efficiency.

O(f(n))

order of magnitude is called Big O notation.

Date

(2) Big Omega (Ω) =



It is defined as lower bound & lower bound on an algorithm is the least amount of time required (best case & more efficient).

It provide asymptotic lowes bound.

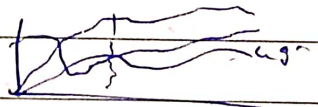
It is also used in computer science to describe the performance or complexity of an algorithm

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

It is written as

It is also measure an algorithm efficiency.

(3) Big theta Θ -



It is defined as tighest bound & tighest bound is the best of all worst case time that the algorithm take place

It specifies asymptotic bound (both upper & low) for a function and provide average time complexity of an algorithm

$f = \Theta(g)$ & $g = \Theta(f(n))$ It is written as

It measure execution of an algorithm.

UNIT - 2

PAGE NO. :
DATE : / /

Sparse matrix

Sparse matrix is a 2-D array in which nodes of the elements are null. In sparse matrix, memory is wasted because null values are stored and hence to overcome wastage of memory a mechanism is followed.
(1) Linked list (2) Array.

	0	1	2	3	4	5	6	7
0	0	0	0	0	5	0	0	0
1	0	0	4	0	0	0	6	0
2	1	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	8	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	10	0	7	0	0	0

7 x 8 = 56.
56 x 2 = 112
Here 2 is number stored in integer way.

7 x 8

Sparse matrix

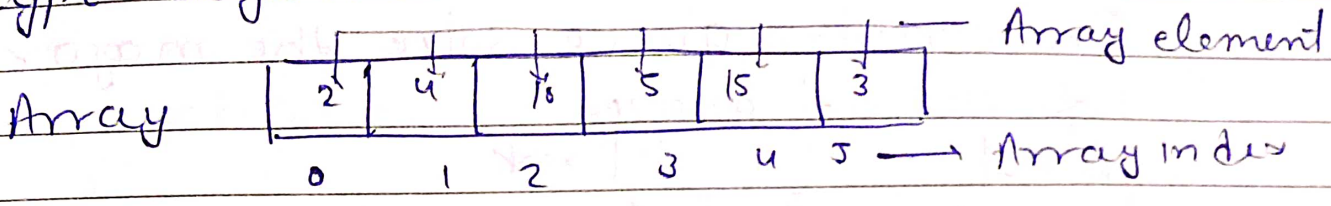
	0	1	2	3	4	5	6	7
row	0	1	1	2	4	4	6	6
colm	4	2	6	0	1	5	3	4
value	5	4	1	1	8	3	10	7

Assignment : (1) What is stack, queue & linked list.
(2) Write program of queue & array
(3) Write operation of data structure
(4) Why we use analysis of algo
(5) What is sparse matrix & type of sparse matrix
(6) Explain 0, Θ, Ω

DATE : / /

Array ⇒

An array is a collection of item stored at contiguous memory locations. The idea is to store multiple item of the same type together.



We can say that array is a linear data structure that is collection of similar data types.
It is static data structure with a fixed size.

Application - (1) storing & accessing data
(2) Sorting, searching
(3) Matrices
(4) Dynamic programming
(5) Graph.
(6) Data mining
(7) Financial analysis

Advantage - (1) Efficient access to element
(2) Fast data retrieval
(3) memory efficiency
(4) Easy to implement
(5) capability with hardware
(6) Versatility

Types - (1) Single dimensional array
(2) multiple dimensional array.

stack =>

A stack is a container of a object that are inserted & removed according to last in first out (LIFO). Principle.

Object can be inserted at any time. Inserting an item known as "pushing".

It allows insertion & deletion operation from one end of the stack data structure, that is top.

A stack is a logical concept that consist of a set of similar element.

It is an abstract data type that hold an ordered linear sequence of items.

It is used for evaluting expression with operands & operation (Infix to postfix conversion)

Queues =>

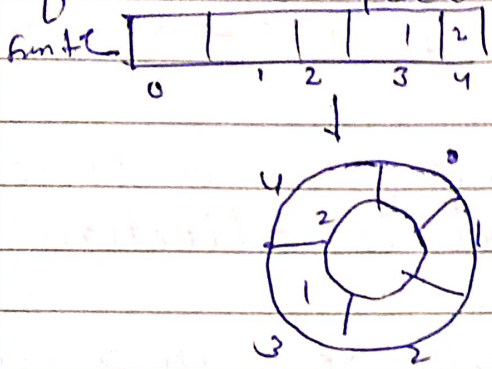
A queue is defter from a stack that its insertion & removal operation according to first in first out (FIFO) principle.
Elements may be inserted at any time. But only element which has been in the queue the longest may be remove.

Element are inserted at the rear (enqueued) & removed from the from (dequeued)

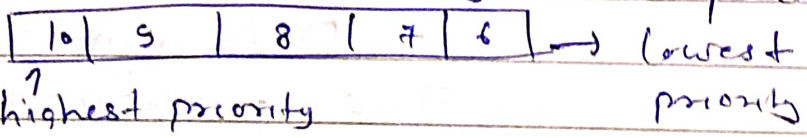
It is a linear data structure that open at both end
It is used to manage threads in multithreading & implementing priority queue

Circular queue:

- It is the extended version of a regular queue where last element is connected to first element.
- Thus forming a circle like structure
The circular queue solve the major limitation of normal queue

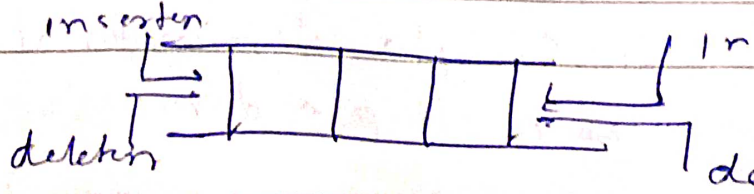


Priority queue: It is a type of queue that arranges element based on their priority value. It is an abstract data type that behaves similarity to normal queue except that each element has some priority.



Double Ended queue

It is a type of queue in which insertion & removal of element can either be performed from front or rear. In deque, you can perform both insertion and deletion operation at both of its ends.



PAGE NO.:
DATE: / /

Evolution of expanantion :

operator	precidence	Associativity
↑	1	Right to left
* /	2	left to Right
+ -	3	left to Right

Infix

<operand1><operator><operand2>

Postfix :

<operand1><operand2><operator>

Prefix :

<operator><operand1><operand2>

Q.1 $2 \uparrow 3 \uparrow 2$

Ans $2 \uparrow 3^2$

$2 \uparrow 9$

$2^9 = 512$

Q.2 $2+3*4/6-2\uparrow 1\uparrow 2*2/2$

Ans $2+3*4/6-2\uparrow 1^2*2/2$

$2+3*4/6-2*2/2$

$2+2-4/2$

$2+2-2$

2.

#17/12/22

PAGE NO.:
DATE: / /

Data structure & Algorithm :- (DSA)

(i) POSTFIX :- (left to right)

(1) 234*6/+2|2↑↑2*2/-

⇒ 2|26/+2|2↑↑2*2/-
⇒ 22+2|2↑↑2*2/-
⇒ 42|2↑↑2*2/-
⇒ 421↑2*2/-
⇒ 422*2/-
⇒ 442/-
⇒ 42-
⇒ 2

(ii) Prefix :- (right to left)

(1) -+2/*346/*↑2↑122

⇒ -+2/*346/*↑212

⇒ -+2/*346/*22
⇒ -+2/*346/42
⇒ -+2/*3462
⇒ -+2/1262
⇒ -+222
⇒ -42
⇒ 2

Conversion of Infix into postfix, prefix :-

Q.1 # Infix ⇒ a + b * c

s.1? ⇒ a + (* b c)

postfix ⇒ + a * b c

prefix ⇒ a + b * c

⇒ a b * c +

⇒ a + [b c *]

⇒ a [b c *] +

⇒ a b c * +

Q.2 # (A + B) * c / D convert into prefix.

s.1? Prefix -

(A + B) * / c D

(+ A B) * c / D

* (+ A B) / c D

/ * (+ A B) c D

[+ A B) * c / D
left to

[* + A B c] / D

/ * + A B c D)

Postfix ⇒

(A B +) * c / D

[(A B +) c *] / D

(A B +) c * D /

Ans ⇒ (−+a*/*bc↑d↑ef a/bc)

<operator><operand><operand>

+ a b

PAGE NO.:
DATE:

Q.3 ## a+b*c/D↑e↑f*a−b/c Convert from infix into prefix.

Sol)

a+b*c/D↑(↑ef)*a−b/c

⇒ a+b*c/(↑D)(↑ef)*a−b/c

⇒ a+(*bc)/(↑D)(↑ef)*a−b/c

⇒ a+[/(*bc)(↑D)(↑ef)]*a−b/c

⇒ a+(/*bc)(↑D)[*↑ef(a)]−b/c

a+[/*bc)(↑D)[*↑ef(a)]−/bc)

+a(/*bc)(↑D)(*↑ef(a)]−/bc)

⇒ a+*[/(*bc)(↑D↑ef)(a)]−b/c

⇒ a+*[/(*bc)[↑D↑ef)a]−(/bc)

⇒ [+(a*/*bc↑d↑ef a)]−[(/bc)]

⇒ −[+(a*/*bc↑d↑ef a/bc)] Ans

⇒ (−+a*/*bc↑d↑ef a/bc) Ans

PAGE NO. :
DATE : / /

Ans:- (ab+)

Postfix ⇒ <operand 1><operand 2><operator>

a + b * c / D ↑ e ↑ f * a - b / c

⇒ a + b * c / D ↑ (e f ↑) * a - b / c

⇒ a + b * c / (D e f ↑ ↑) * a - b / c

⇒ a + (b c *) / (D e f ↑ ↑) * a - b / c

⇒ a + (b c * D e f ↑ ↑ /) * a - b / c

⇒ a + (b c * D e f ↑ ↑ / a *) - b / c

⇒ a + (b c * D e f ↑ ↑ / a *) - (b c /)

⇒ [a (b c * D e f ↑ ↑ / a *)] + - (b c /)

⇒ (a b c * D e f ↑ ↑ / a * b c / -)

⇒ a b c * D e f ↑ ↑ / a * + b c / - [Answer]

Homeworks ⇒

Q1. (A + B) * (C + D)

Q2. A * B + C * D

Q3. A + B + C + D

Q4. K + L - M * N + (O ↑ P) * W / U / V * T + Q

(Postfix)

PAGE NO.:
DATE: / /

① $(A+B) * (C+D)$

⇒ Prefix →

$* (A+B)(C+D)$

$* (+AB)(C+D)$

$* (+AB)(+CD)$

$\Rightarrow * + AB + CD$

$(AB+) * (CD+)$

$(AB+)(CD+)*$

$AB+CD+*$

② $A * B + C * D$

Prefix

$(*AB) + C * D$

$(*AB) + (*CD)$

$+ (*AB)(*CD)$

$+ *AB*CD$

Postfix

$(AB*) + C * D$

$(AB*) + (CD*)$

$(AB*)(CD*) +$

$AB*CD*+$

③ $A + B + C + D$

Prefix

$(+AB) + C + D$

$[+(+AB)(C)] + D$

$[+(+(+AB)(C)D]$

$+ + + ABCD$

Postfix

$= (AB+) + C + D$

$\Rightarrow (\underset{op1}{AB+C+}) + \underset{op2}{D}$

$\Rightarrow (AB+C+D)+$

$AB+C+D+$

(4) Post'

K + L − M * N + (O ↑ P) * W / U / V * T + Q

K + L − M * N + (O P ↑) * W / U / V * T + Q

K + L − (M N *) + (O P ↑) * W / U / V * T + Q

K + L − (M N *) + (O P ↑ W *) / U / V * T + Q

K + L − (M N *) + (O P ↑ W * U /) / V * T + Q

K + L − (M N *) + (O P ↑ W * U / V /) * T + Q

K + L − (M N *) + (O P ↑ W * U / V / T *) + Q

(K L +) − (M N *) + (O P ↑ W * U / V / T *) + Q

(K L + M N * −) + (O P ↑ W * U / V / T *) + Q

(K L + M N * − O P ↑ W * U / V / T * +) + Q

K L + M N * − O P ↑ W * U / V / T * + Q + Ans

PAGE NO.:
DATE: / /

(5) $a * (b + c)$

(i) Prefix:

$a * (+bc)$

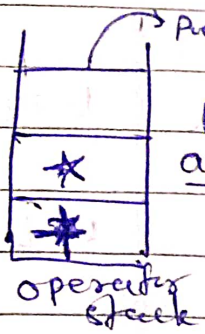
$(*a+bc)$

(ii) Postfix

$a * (bc+))$

$(abc+) *$

$abc+*$



Algorithm in form of table

I/P	TOS	
H	L	PUSH
L	H	POP
E	E	PUSH(R-L)
E	E	POP(L-R)
C	op+	PUSH
op+	C	PUSH
C	C	PUSH

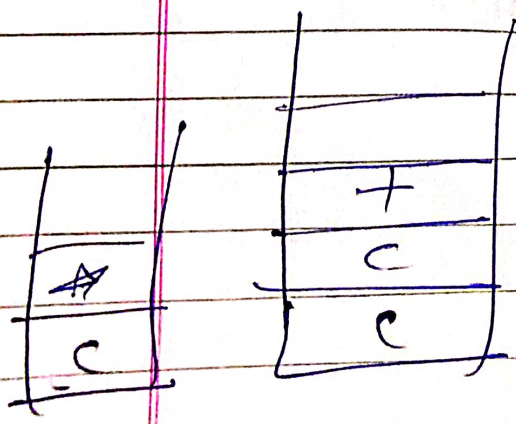
operand A, B, C, X, Y, P, Q ...
operator +, -, *, /, ^

Q.1 $((a+b) * c)$

Sol: Postfix

$((ab+) * c)$

$= (ab+c*)$



$ab+c*$

Q.2 # $a+b*(c/Def\uparrow)*(a-b)/c$

Post:-

$\Rightarrow a+b*(c/Def\uparrow)*(a-b)/c$

$\Rightarrow a+b*(c/(Def\uparrow))*(a-b)/c$

$\Rightarrow a+b*(c/(Def\uparrow))*(a-b)/c$

$\Rightarrow a+b*(cDef\uparrow/)*(a-b)/c$

PAGE NO. :
DATE : / /

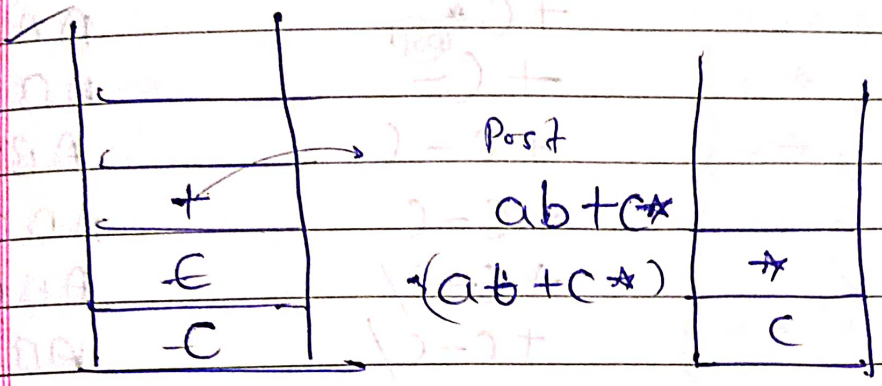
(6) (1) a + b - c

(2) a * b + c

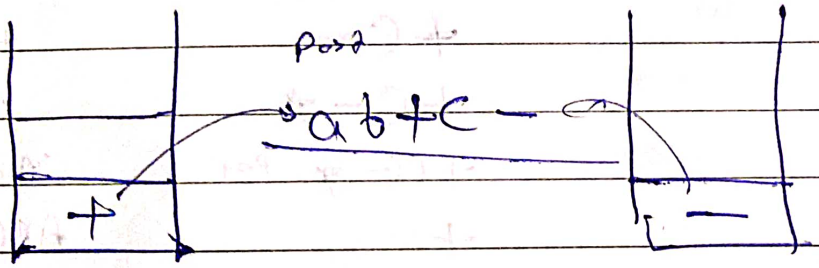
(3) a ↑ b ↑ c

(4) a * (b + c

(5) ((a+b) * c)



(1) a + b - c



Q.1 A + (B * C - (D / E ↑ F) * G) * H

2nd (A+B/C*(D+E)-F)

PAGE NO.
DATE

Q-2

SYMBOL	STACK	Postfix
(	(	
A	(	A
+	(+	A
B	(+	AB
/	(+/	AB
C	(+/ (pop)	ABC
*	(+*	ABC/
(	(+*(	ABC/
D	(+*(	ABC/D
+	(+*(+	ABC/D
E	(+*(+	ABC/DE
)	(+* (pop	ABC/DE+
-	(+ (pop	ABC/DE+*
F	(-	ABC/DE+*F
)	((pop	ABC/DE+*F-

A+(/B

Q (A+B/C*(D+E)-F)

Prefix ⇒

⇒ (A+B/C*(+DE)-F)

⇒ (A+(/BC)*(+DE)-F)

⇒ (A+[*(/BC+DE)]-F)

⇒ (+A*/BC+DE)-F

⇒ -+A*/BC+DEF

(A+B/C*(+DE)-F)

(A+B/*(C+DE)-F)

(A+(/B*(C+DE)-F)

A+(-/B*C+DEF)

+A-/B*C+DEF

(1) $(A+B/C*(D+E)-F)$

Prefix ⇒ $(F-(E+D)*C/B+A)$ ⇒

PAGE NO.:
DATE: / /

SYMBOL	STACK	POSTFIX
C	C	
F	C	F
−	C−	F
(	(−(	F
E	(−(	FE
+	(−(+	FE
D	(−(+ pop	FED
)	(−	FED+
*	(−*	FED+
C	(−* pop	FED+C
/	(−/	FED+C*
B	(−/ pop	FED+C*B
+	(+	FED+C*B/−
A	(+	FED+C*B/−A
)	pop	FED+C*B/−A+

⇒ FED+C*B/−A+

Reverse:

+A−/B*C+DEF

⇒ A+(B*C−(D/E↑F)*G)*H

A+(B*C−(D/(↑Ef))*G)*H

⇒ A+(*BC)−(D/(↑Ef)*G)*H

⇒ A+[(*BC)−(/D↑Ef)*G]*H

⇒ A+*[−*BC*/D↑EfG]*H]

⇒ A+(*−*BC*/D↑EfGH)

⇒ +A*−*BC*/D↑EfGH Proved that

Prefix to Postfix using stack operation

~~A + (B * C - (D / E ↑ F) * G) * H~~

H * (G * (F ↑ E / D) - C * B) + A

PAGE NO.:
DATE: / /

H	-	H
*	*	H
C	* (	H
G	* (	HG
*	* (*	HG
(	* (* (	HG
F	* (* (	HGF
↑	* (* (↑	HGF
E	* (* (↑	HGFE
/	* (* (/	HGFE↑
D	* (* (/	HGFE↑D
)	* (*	HGFE↑D/
-	* (-	HGFE↑D/*
C	* (-	HGFE↑D/*C
*	* (- *	HGFE↑D/*C
B	* (- *	HGFE↑D/*CB
)	*	HGFE↑D/*CB**
+	+	HGFE↑D/*CB**+
A	+	HGFE↑D/*CB**A+
.		

~~+ A * - B C * / D ↑ E F G H~~

+ A * - * B C * / D ↑ E F G H

a+b*c/D↑e↑f*a-b/c

c/b-a*f↑e↑D/C*b+a

PAGE NO.:
DATE: / /

c	-	c
/	/	c
b	/	eb
-	+	cb/
a	-	cb/a
*	-*	cb/a
f	-*	cb/af
↑	-*↑	cb/af
e	-*↑	cb/afe
↑	-*↑↑	cb/afe↑
D	-*↑↑	cb/afe↑D
/	-*	cb/afe↑D↑*
C	-/	cb/afe↑D↑*C
*	-*	cb/afe↑D↑*C/
b	-*	cb/afe↑D↑*C/b
+	-+	cb/afe↑D↑*C/b*-
a	+	cb/afe↑D↑*C/b*-a
-		cb/afe↑D↑*C/b*-a+

⇒ +a-*b/C*↑↑Defa/bc

⇒ +a-*b/C*↑↑Defa/bc

Key = 6?

0	1	2	3	4	5	6
4	8	3	6	11	15	10

Linear search :-

linear search is a very simple search algorithm in this type of search a sequential search is made over all item one by one.

every item is checked & if a match is found then that particular item is returned otherwise search continue till the end of data collection.

① best case O(1)
② worst case O(n)

Binary search :-

0	1	2	3	4	5	6	7	8	9
10	15	18	25	29	32	37	40	54	60

l — Target 37 — r

(sorted assending order)

element are arranged in sorted form.

Time complexity O(log₂n)

$$mid = \frac{l+r}{2}$$

$$mid = \frac{0+9}{2} = 4.5 = 4$$

condetions occurs

① A[mid] == Target
return;

② A[mid] < Target
l = mid + 1;

③ A[mid] > Target
r = mid - 1;

	l	r	mid
①	0	9	4
②	5	9	7
③	5	6	5
④	6	6	6

⇒

0	1	2	3	4	5	6	7	8	9
10	15	18	25	29	32	37	40	54	60

l — mid — r

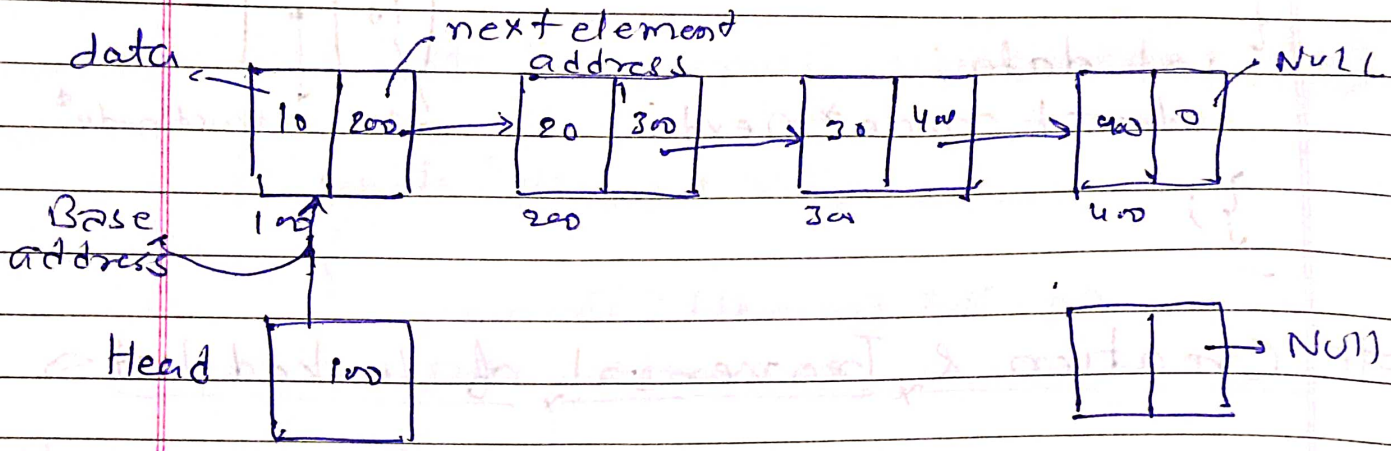
0	1	2	3	4	5	6	7	8	9
10	15	18	25	29	32	37	40	54	60

l — mid — r

(UNIT-3)
DSA

PAGE NO.:
DATE: / /

Linked List ⇒ (linear data structure)



Linked list is a linear data structure used to store data element seqvently and successive elements connected by pointer and last nodes contain null which mean end of linked list.

Main advantage of linked list is we can increase the size dynamically until the memory is full.

Major disadvantage of linked list is support only sequential access not random.

Implementation of linked list.

Linked list can be implemented using structure in C language

UNIT-3

PAGE NO.:
DATE: / /

Self referential structures of linked list.

It have one or more pointers which points to the same type of structure, as their member

```
struct node {
    int data1;
    char data2;
    struct node *link;
};
```

Dynamic memory allocation of linked list:

memory is allocated at compile time, memory is allocated at run time.
Linked list is a data structure that is based on dynamic memory allocation using calloc() malloc(), realloc() & free().

malloc(): allocate single block of requested memory

calloc(): allocate multiple block of requested memory

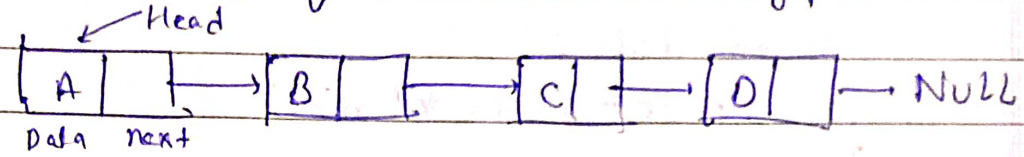
realloc(): reallocate the memory occupied by malloc() & calloc() function

free() → free the dynamically allocated new

Syntax = pointer_variable = (type cast*) malloc (sizeof(struct node));

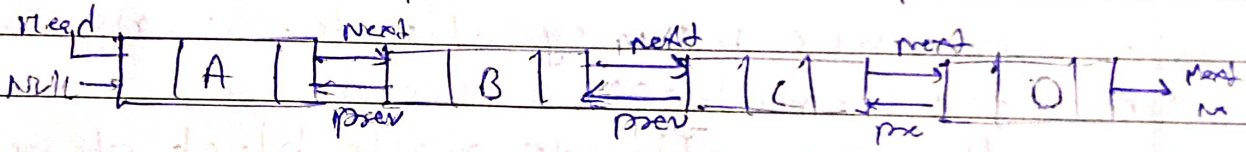
Type of linked list :-

(1) Singly linked list ⇒ It is simplest type of linked list in which type every node contain some data & a pointer to next nodes of the same data type



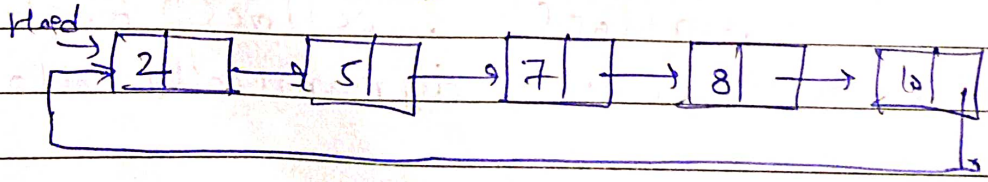
Time complexity O(n)

(2) Doubly linked list :- It is two-way linked list is more complex type of linked list that contain a pointer to the next as well as previous node in sequence.



(3) Circular linked list ⇒

A circular linked list is that in which last node contain the pointer to first node of the list



operation on linked list.

(1) Insertion ⇒ Add an element at beginning
(2) Deletion ⇒ Delete an element at beginning
(3) Display ⇒ Display the complete list.
(4) Search ⇒ Searches an element using given key
(5) Delet - Delete an element using given

UNIT-4
DSA

UNIT-4

Bubble sort :-

In bubble sort, each element is compared with it adjacent element. If first element is larger than the second element than the position of the element are interchanged otherwise it is not changed.

Path(1)
10 14 2 11 6
10 14 2 11 6
10 14 2 11 6
10 2 14 11 6
10 2 14 6 14

Path(2)
10 2 11 6 14
2 10 11 6 14
2 10 11 6 14
2 10 6 11 14

Path(3) → 2 10 6 11 14
2 10 6 11 14
2 6 10 11 14 (exit)
2 6 10 11 14
2 6 10 11 14

Path-1

Ex (2) 4 1 5 6 2

(4) 1 5 6 2

1 (4) 5 6 2

1 4 (5) 6 2

1 4 (5) 6 2

1 4 5 (6) 2

1 4 5 2 6

Path (2)

(1) 4 5 2 6

1 (4) 5 2 6

1 4 (5) 2 6

1 4 2 (5) 6

1 4 2 5 6

Path (3)

(1) 4 2 5 6

1 (4) 2 5 6

1 2 4 5 6

Ex (3) 32, 51, 27, 85, 66, 23, 13, 15, 57

Path (32) 51, 27, 85, 66, 23, 13, 15, 17

32 (51) 27 85 66 23 13 75, 57

32 27 (51) 85 66 23 13 85, 57

32 27 51 (85) 66 23 13 15 57

32 27 51 66 (85) 23, 13 15 57

32 27, 51, 66 23 (85) 13 15, 57

32 27 51 66 23 13 (85) 15 57

32 27 51 66 23 13 15 (85), 57

32 27 51 66 23 13 15 57 85

DATE: / /

Path (2) =

(32) 27, 51, 66, 23, 13, 15, 57, 85

27 (32) 51 66 23 13 15 57 85

27 32 (51) 66 23 13 15 57 85
27 32 51 (66) 23 13 15 57 85
27 32 51 23 (66) 13 15 57 85
27 32 51 23 13 (66) 15 57 85
27 32 51 23 13 15 (66) 57 85
27 32 51 23 13 15 57 (66) 85

Path (3)

(27) 32 51 23 13 15 57 66 85
27 (32) (51) 23 13 15 57 66 85
27 32 (51) 23 13 15 57 66 85
27 32 23 (51) 13 15 57 66 85
27 32 23 13 (51) 15 57 66 85
27 32 23 13 15 (51) 57 66 85

Path (4)

(27) 32 23 13 15 51 57 66 85
27 (32) 23 13 15 51 57 66 85
27 23 (32) 13 15 51 57 66 85
27 23 13 (32) 15 51 57 66 85
27 23 13 15 (32) 51 57 66 85

Path (5)

(27) 23 13 15 32 51 57 66 85
23 (27) 13 15 32 51 57 66 85
23 13 (27) 15 32 51 57 66 85
23 13 15 27 32 51 57 66 85

PAGE NO. :
DATE : / /

Path 4 (23), 13 15 27 32 51 57 66 85

13 (23), 15 27 32 51 57 66 85
13 15 23 27 32 51 57 66 85

11, (77), 33, 44, 11, 88, 22, 66, 55

Pass 1

33 (77), 44 11 88 22 66 55
33 44 (77), 11 88 22 66 55
33 44 11 (77), 88 22 66 55
33 44 11 88 77 22 66 55
33 44 11 22 77 66 55
33 44 11 88 22 66 77 55
33 44 11 88 22 66 55 77

Pass 2

(33) 44 11 88 22 66 55 77
33 (44), 11 88 22 66 55 77
33 11 (44), 88 22 66 55 77
33 11 44 (88), 22 66 55 77
33 11 44 22 (88), 66 55 77
33 11 44 22 66 (88) 55 77
33 11 44 22 66 55 (88) 77
33 11 44 22 66 55 77 88

Pass 3 :- (33) 11 44 22 66 55 77 88
11 (33), 44 22 66 55 77 88
11 33 (44), 22 66 55 77 88
11 33 22 (44), 66 55 77 88
11 33 22 (44), (66) 55 77 88
11 33 22 44 55 66 77 88

Pass (1)

(11) 33 22 44 55 66 77 88

11 (33) 22 44 55 66 77 88

11 22 (33) 44 55 66 77 88

Selection sort ⇒ $O(n^2)$

(1) 77 33 44 11 88 22 66 55

(k, min, sort)

11 33 44 77 88 22 66 55

11 22 44 77 88 33 66 55

11 22 33 77 88 44 66 55

11 22 33 44 88 77 66 55

11 22 33 44 55 77 66 88

11 22 33 44 55 66 77 88

(2) 66 33 40 20 55 88 31

20 33 40 66 55 88 31

20 33 40 66 55 88 33

20 31 33 66 55 88 40

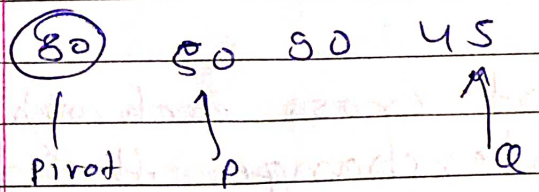
20 31 33 40 55 88 66

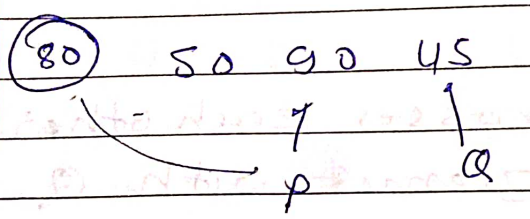
20 31 33 40 66 88 88

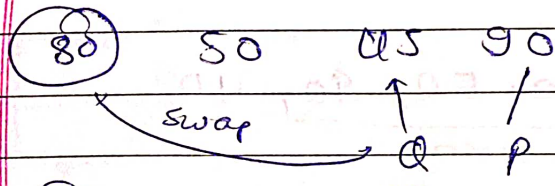
20 31 33 40 66 88 88

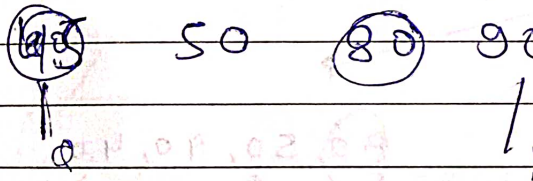
90 31 38 40 61 55 88
L

Quick short;









45 50 80 90

Q: # 35 50 15 25 80 20 90 45
Pivot
V:35
P
Q

Quick short works on devide and conquer root. In this we devide bigger problem into smaller subproblems P increases or move toward R++ & stop only R/Res it will get the element greater than the pivot element

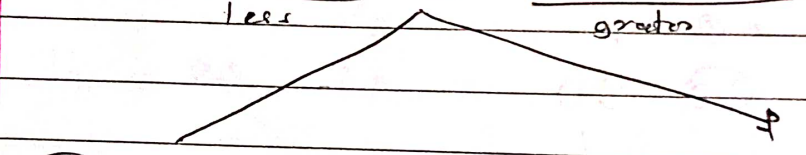
equals to 35 (pivot).
q decreases or move towards L.H.S and stops only when it will get the element lesser the pivot elemet equal to 35 (pivot)

If P & Q, are not cross each other then swAp or interchange their values.

check(if) P & Q (crosses each other) then replace pivot element with Q.

25, 20, 15 (35) 80, 50, 90, 40

less — greater



(25), 20, 15 -- 40 — (80) 50, 90, 40

pivot p q p — pivot p q

(80) 50 90 40 — p q

15 20 25

(80) 50 40 90 — q p

(80)

=> 40 50 80 90

combined as

15 20 25 40 50 80 90

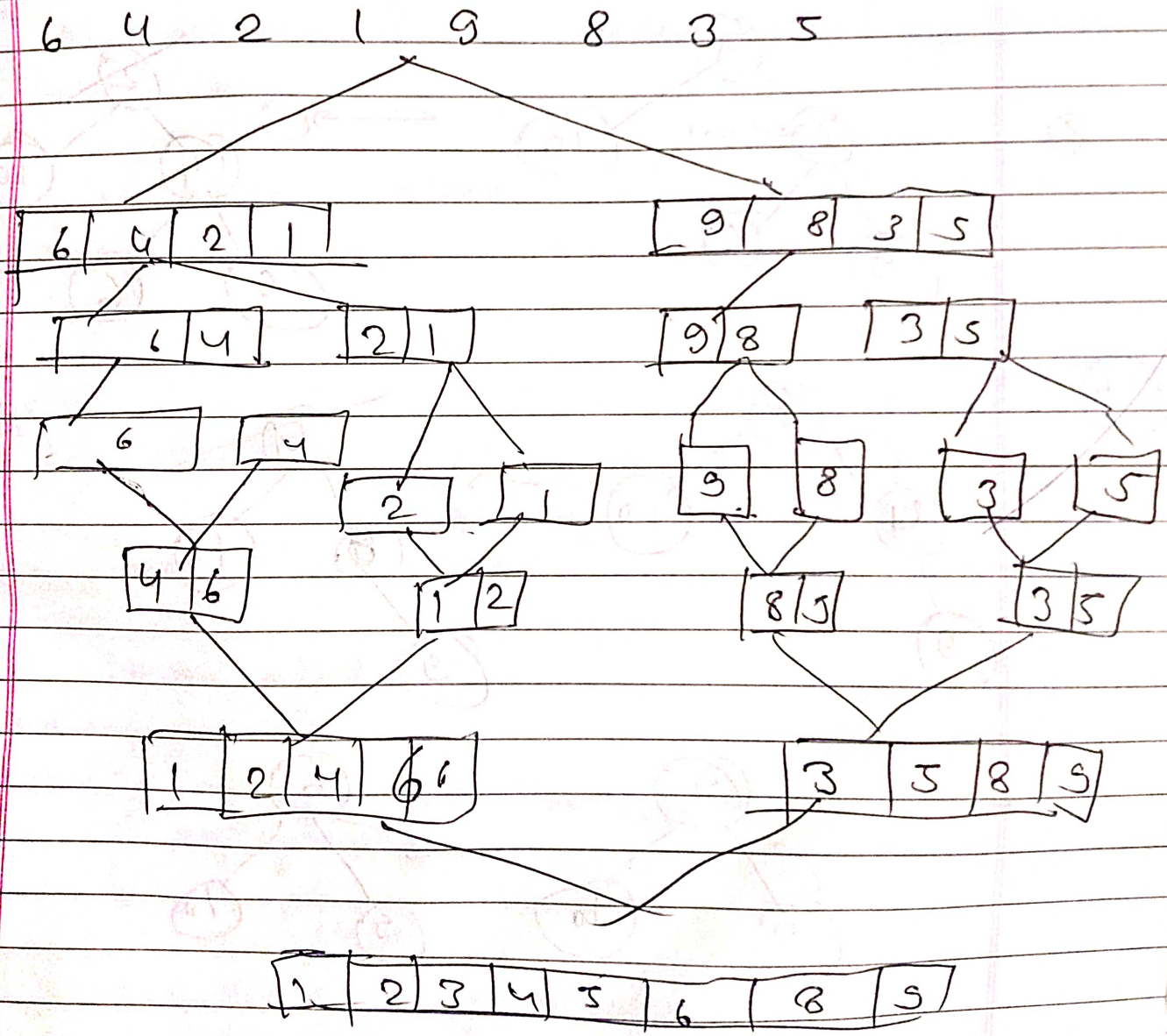
Best case n log n

worst case O(n²)

(Time complexity nlog(n))

Merge sort :-

It is a sorting algorithm that works by dividing any array into smalles subarrays.
Sorting each subarray then merge the sorted sub array back together to form the final sorted array.



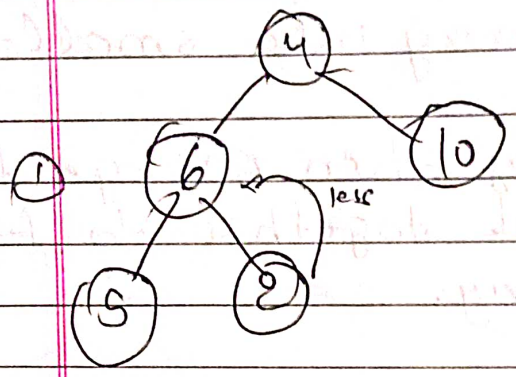
Ex ① 42, 84, 75, 20, 60, 10, 5, 30

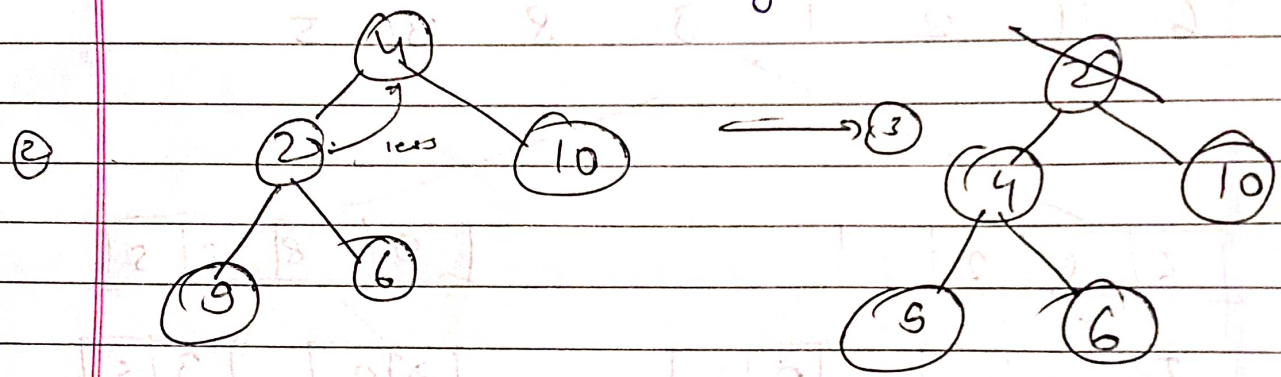
PAGE NO.:
DATE: / /

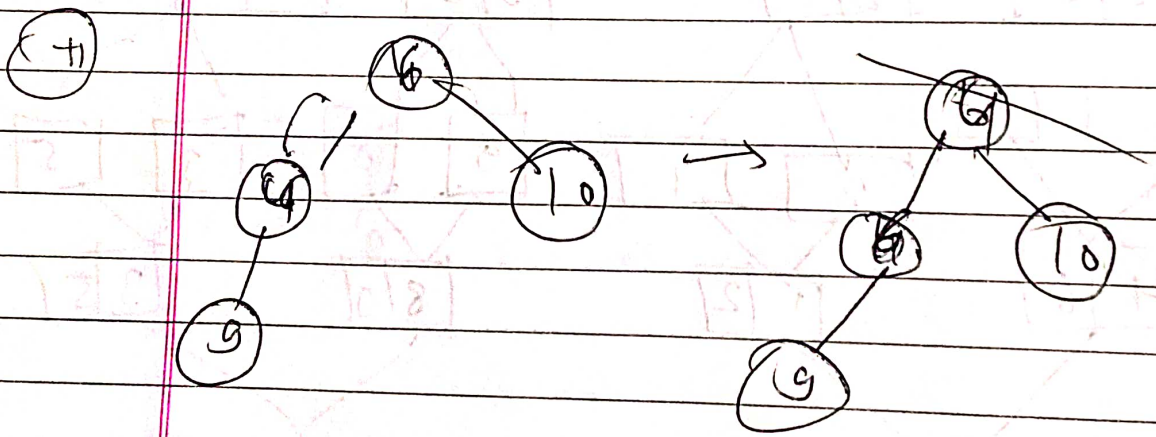
Heap sort ⇒ [Complexity $n \log(n)$]

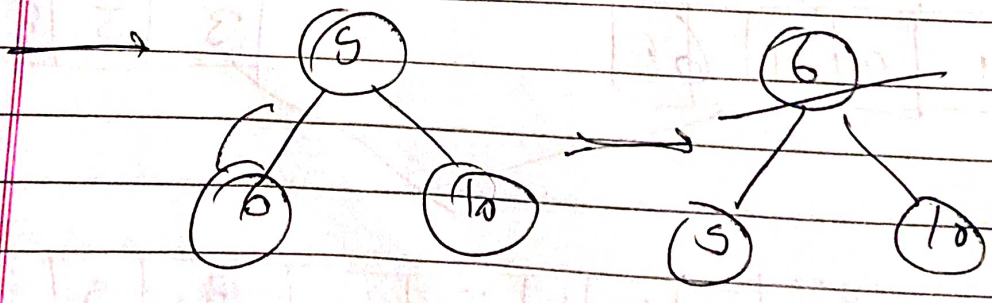
It is a comparison based technique based on binary heap structure.
It is similar to the selection sort but Heap sort is an in-place algorithm.

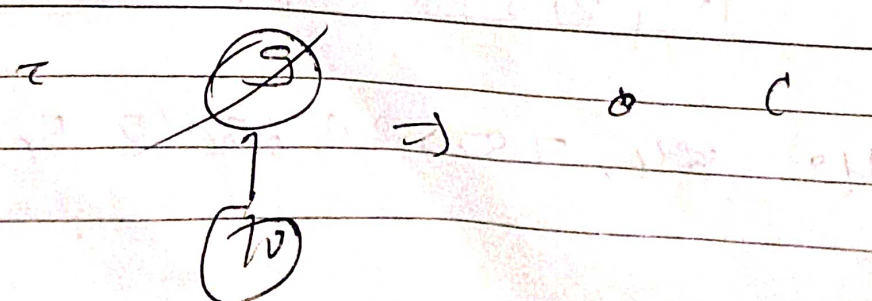
4 6 10 9 2











DATE: / /

Hashing techniques ⇒

Hashing in data structure is a technique of a mapping a large chunk of data into small table using a hashing function. It is also known as message digest function. (1) static hashing (2) dynamic hashing

hashing function ⇒ It is a function that take a set of input of any arbitary size & fit them into a table or other structure that contain fixed-size element

(1) Mid square (2) Divison (3) folding.

675248 (1) Mid square ⇒ It is hasing technique in which
4552159861 unique key are generated. in this
959861 technique a seed value is taken & it squared

(2) Division ⇒ We divide the element with the size of the hash table and use the remainder as index of element in hash table
h(k) = k mod(n)

(3) folding ⇒ It break up a key value into precises segments that are added to form a hash value. (1) fold shift (2) fold boundary

Collusion resolution ⇒

When two or more key have the same hash value, a collosion happens. To handle this collosion, we use collosion resolution techniques.

It is used when two or more item should be kept in same location, especially in hash table

(1) linear (2) double (3) quardatic

DATE: / /

Overflow handling techniques ⇒

An overflow occurs when home bucket for a new pair (key, element) is full. We may handle overflow by: f search the hash table in some systematic fashion for a bucket that is not full

① linear probing ② Quardic ③ Random

Digit analysis:-

A static file is one in which all identifier are known in advance

Using this method, we first transform the identifier into number using some radix.

We then examine the digits of each identifier deleting those digit that have the most skewed distributions

It is simply means that used to analyse the distribution of digit in hashes.

UNIT 5 — BFS & DFS

① Binary tree
② full binary (Strict binary tree)
③ Complete binary & AVL (Almost complete binary tree)
④ BST & AVL Tree
UNIT 5 non

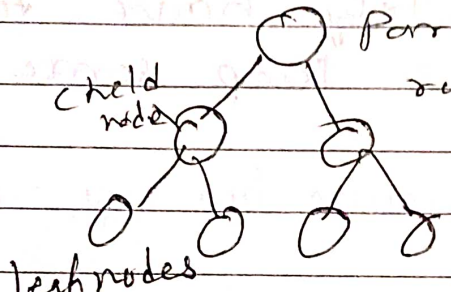
PAGE NO.
DATE: / /

Tree ⇒ Tree is a linear data structure use to store data element.

Types ⇒ ① Binary tree
② An Array tree

① Binary tree –

A tree in which every node can contain almost two child node, is called a Binary tree.

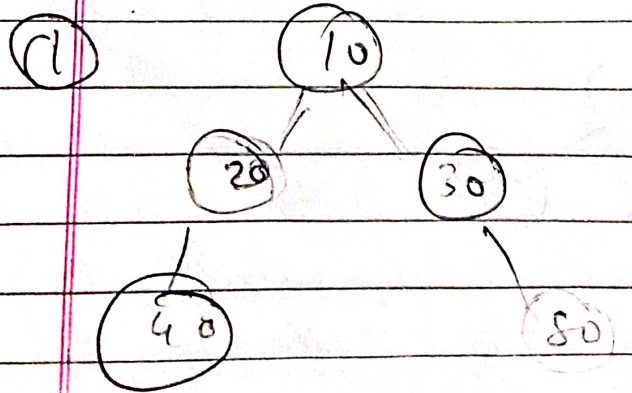


level	height
1	0
2	1
3	2

height – The longest path from the root node to leaf node is called height of tree.

height = # level − 1

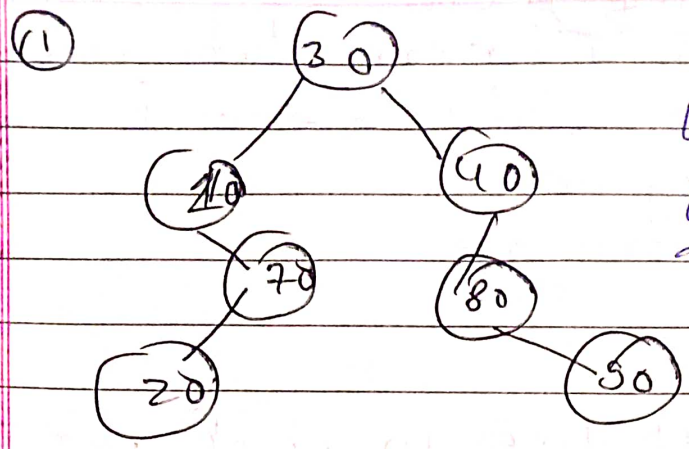
Tree Traversal ⇒ ① Preorder
② Inorder
③ postorder



Inorder = 10, 20, 40, 30, 50

Preorder = 40, 20, 10, 30, 50

post = 40, 20, 50, 30, 10



(1st visit)
Preorder = 30, 10, 70, 20, 40, 80, 50

(2nd visit)
Inorder = 10, 20, 70, 30, 80, 50, 40

(3rd visit)
Postorder = 20, 70, 10, 50, 80, 40, 30

To apply the above technics every node should definitely have two child node otherwise keep some dumy node

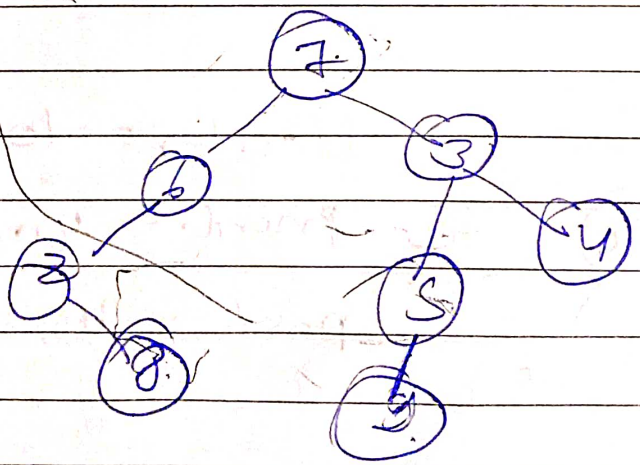
Constructing unique binary tree :

A unique binary tree can be constructed if any two traversal are given and one of them should in order.

Eg If any two traversal are given

postorder : 8, 2, 6, 5, 5, 4, 3, 7

inorder : 2, 8, 6, 7, 5, 5, 3, 4



PAGE NO.:
DATE: / /

Construct the unique binary with the two tree

Post: 10, 8, 23, 22, 27, 25, 15, 50, 95, 60, 40, 29

In: 8, 10, 15, 22, 23, 25, 27, 29, 40, 50, 60, 95

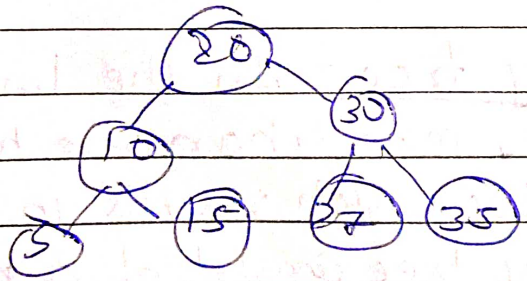
Binary search tree (BST):-

BST is also a binary tree but with the restriction of all the element of left subtree should be smaller than the root node.
All the element of right subtree should greater than root node.

All the element in BST should be unique

BST is used for searching an element.

Q) Construct the BST with the given element
20, 10, 15, 30, 35, 27, 5



Preorder - 20, 10, 5, 15, 30, 27, 35
Inorder ⇒ 5, 10, 15, 20, 27, 30, 35
Postorder ⇒ 5, 15, 10, 27, 35, 30, 20

PAGE NO.:
DATE: / /

NOTE :-

Inorder Traversal of a BST is always sorted order

last element in the preorder and last element in the inorder are same. (This is true for only complete binary tree)

Time complexity for a BST =>

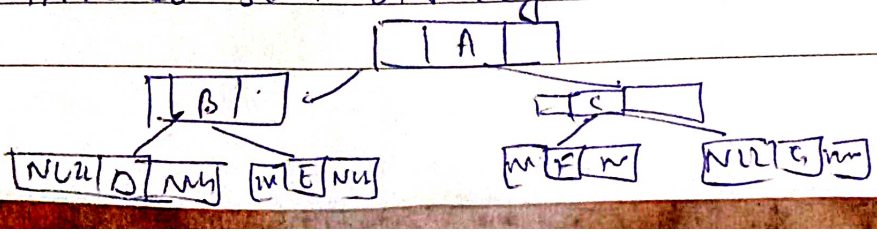
Best case O(log n)

worst → O(h)
height

Stack data structure.

It is linear data structure
It work on LIFO
insertion & deletion operations from one end of stack data structure.
1. register stack
2. memory stack

Threaded binary tree = In the linked representation
of binary tree, more than one half of link field contain NULL value which results in wastage of storage space. If binary tree consist of n node then n+1 link field contain NULL values. a method was devised in which NULL links are replaced with special link called threads. such binary tree with threads.



Difference between B Tree & B+ Tree

B Tree	B+ Tree
① data is stored in leaf & as well as node. (internal node)	data is stored in leaf node.
② Searching is slower deletion is complex.	② Searching is faster & deletion is easy.
③ No redentant search key	Redentant key present
④ leaf is not linked together	linked together like link list

AVL Tree - It is self-balancing binary data structure in which each node maintain extra information.

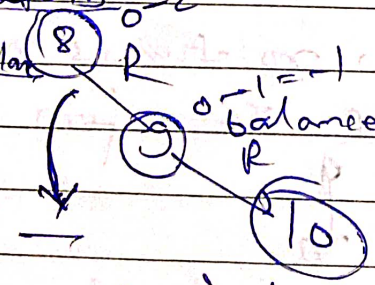
Balance factor = (height of LST - height of RST)

BST will not use then we use AVL

Range: -1, 0, 1

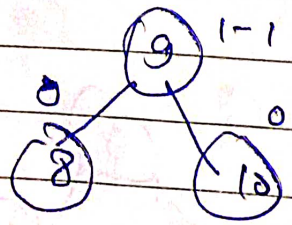
Endtons = 0 - 2 = -2

Unbalanced

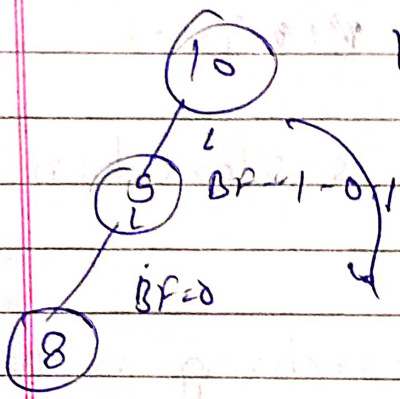


RR Transform

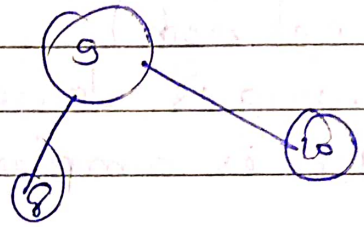
1 Anticlockwise Rotation



(2) 10, 9, 8

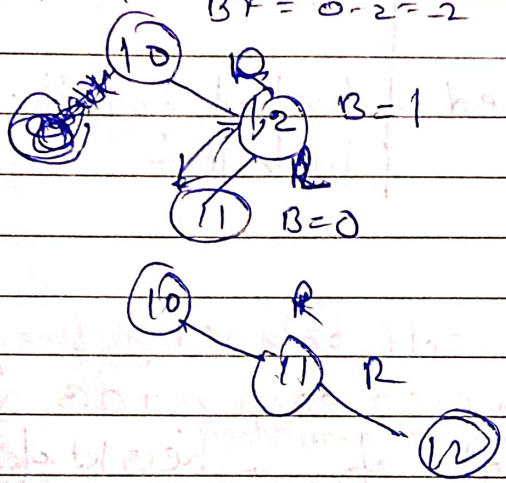


LL Transform



Clockwise Rotation

(3) 10, 12, 11 BF = 0-2=-2

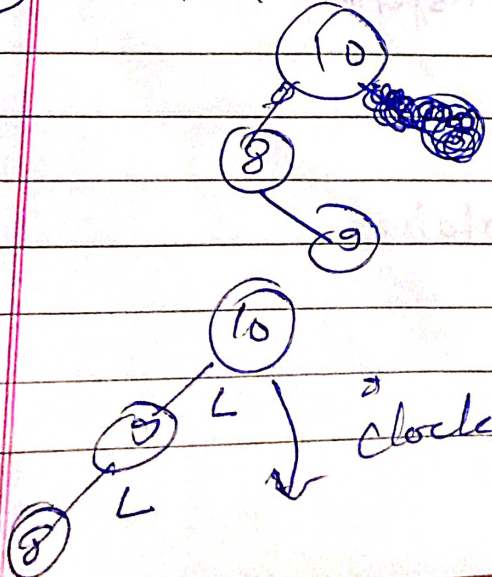


RL Transform

exchange in RR Transform

no of rotation is 2

(2) 10, 8, 9



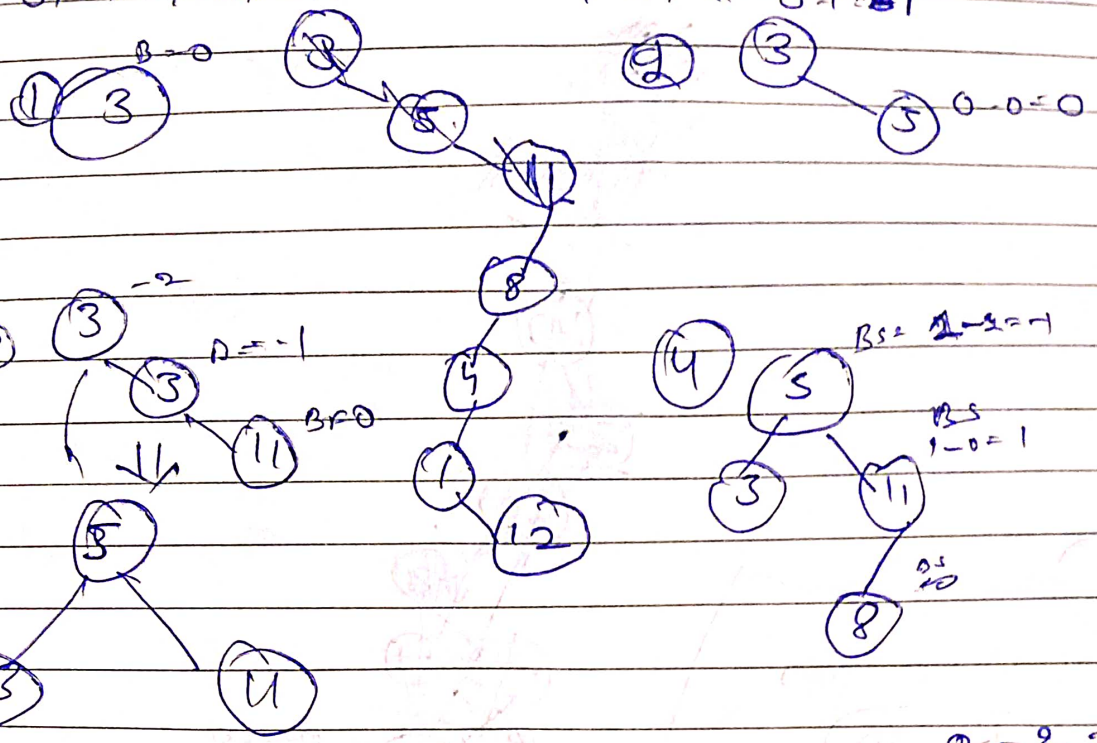
(2 rotation is required)

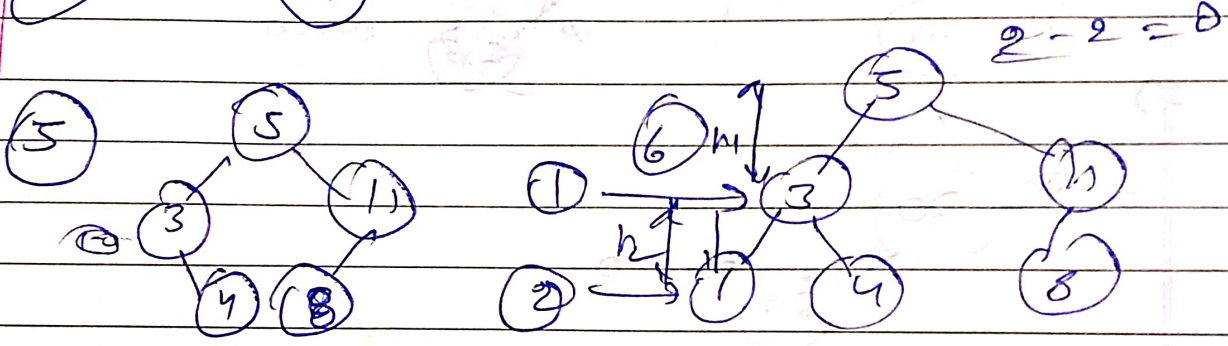
LR Transform

LL Transform

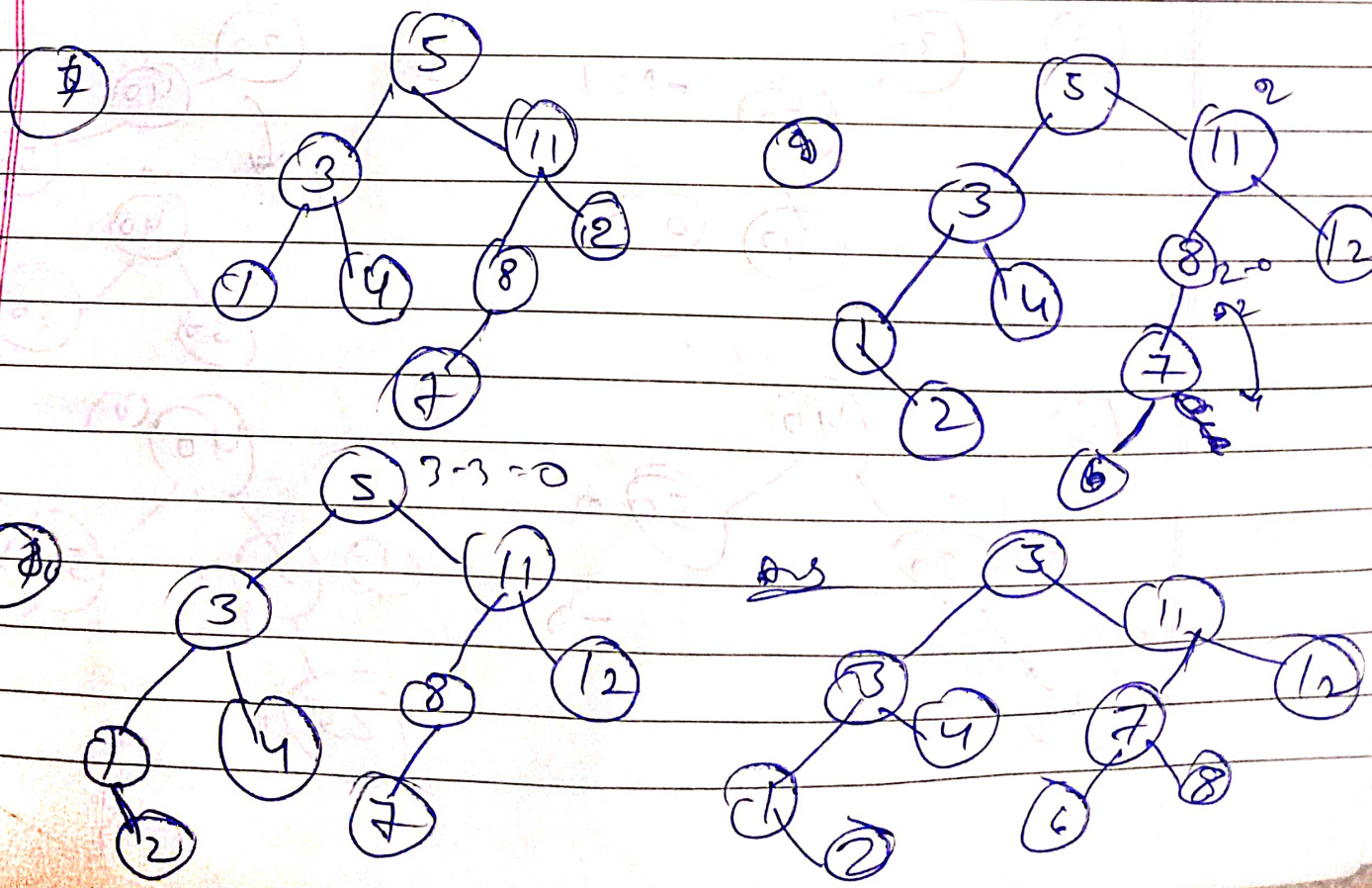
AMP

3, 5, 11, 8, 4, 1, 12, 7, 2, 6



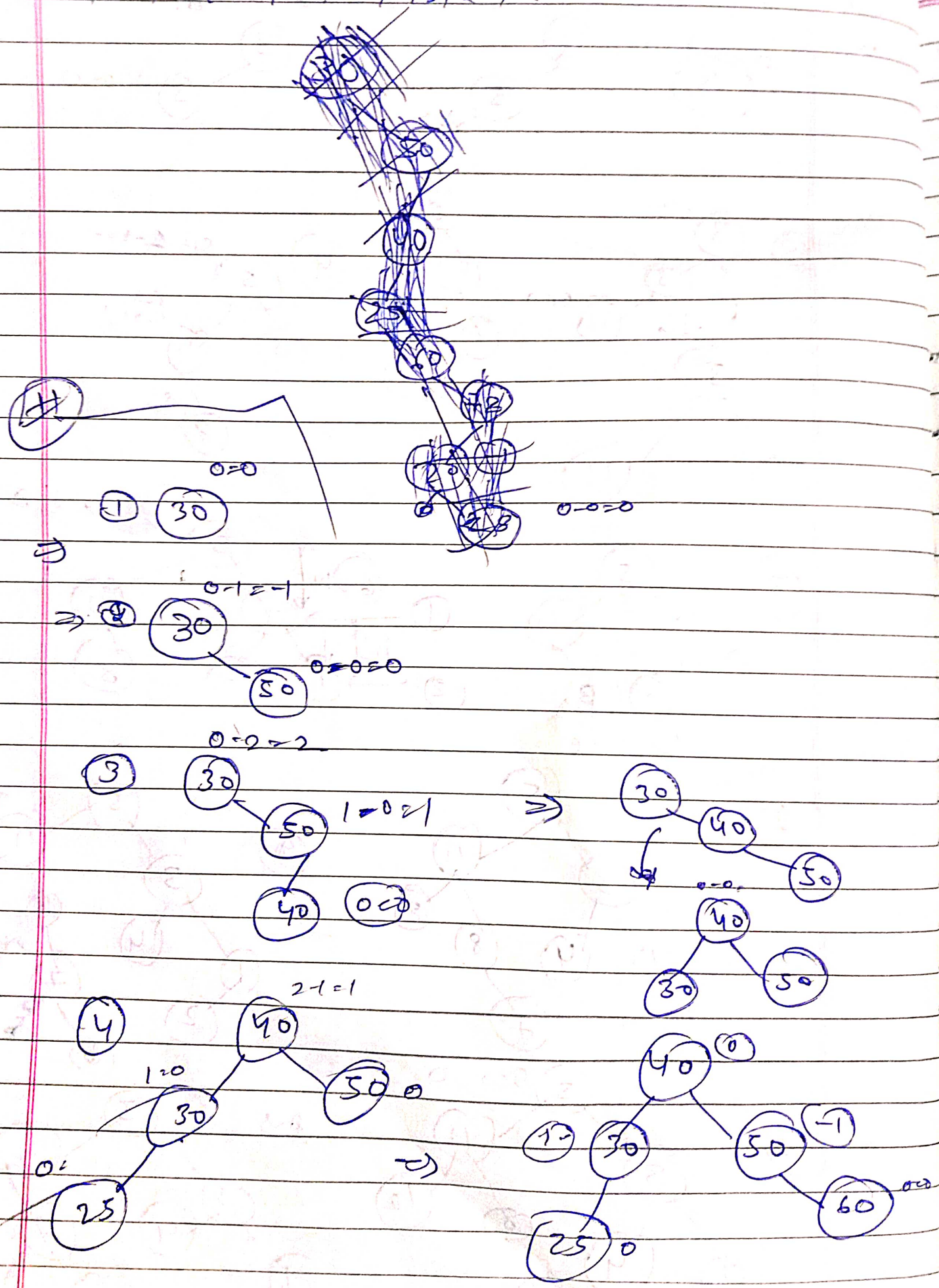


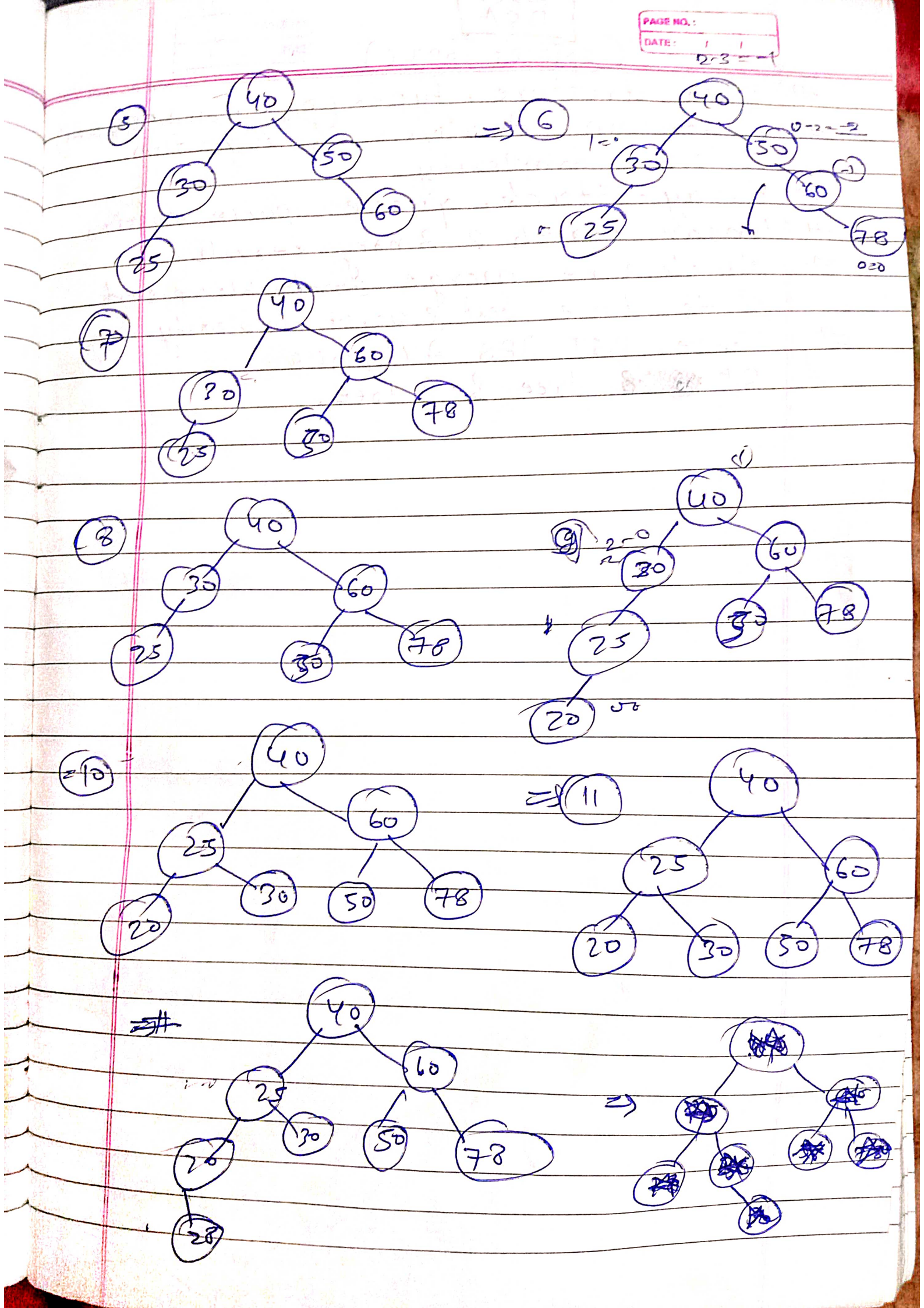
Balance



DATE: / /

30, 50, 40, 25, [illegible], 60, 78, 20, 28




PAGE NO.:
DATE: / /
2-3 = -1
5
40
30
50
25
60
6
40
30
50
25
60
78
7
40
30
60
25
70
78
8
40
30
60
25
30
78
9
40
30
60
25
5
78
20
10
40
25
60
20
30
50
78
11
40
25
60
20
30
50
78

Graphs :-

A graphs is a non-linear kind of data structure made-up of nodes or vertices & edges
(1) A finite set of vertices also called nodes
(2) A finite set of ordered pair of form (V, V) called as edges.

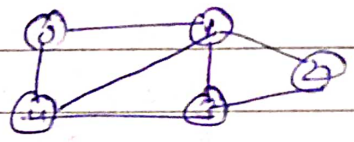
Representation of graph:- two commonly represented are used

(1) Adjacency matrix
(2) Adjacency list
some other like incidence matrix & incidence list.

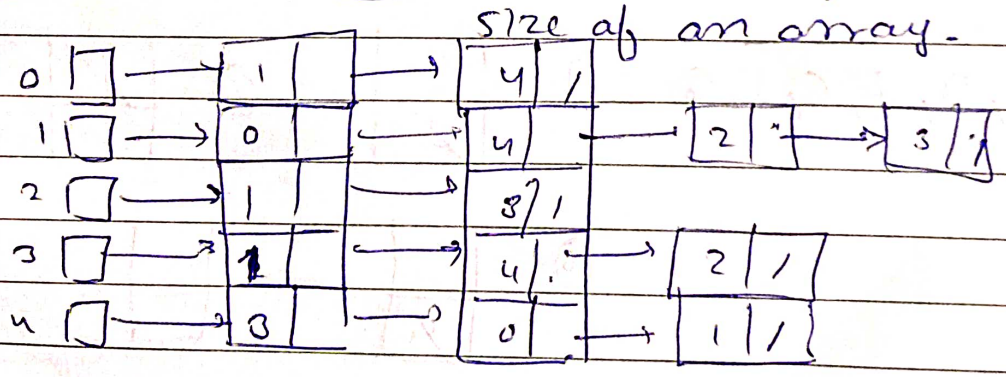
(1) Adjacency matrix :-

=>

	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0



(2) Adjacency list :- The number of vertices equal to size of an array.



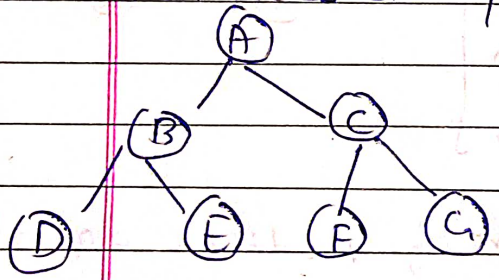
Application of graph :- (1) help to define flows of computation of software program
(2) Used in google map for building transportation system
(3) Social media marking
(4) operating system, WWW

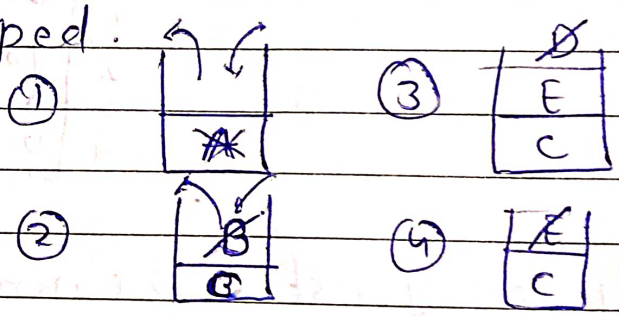
DATE: / /

IMP

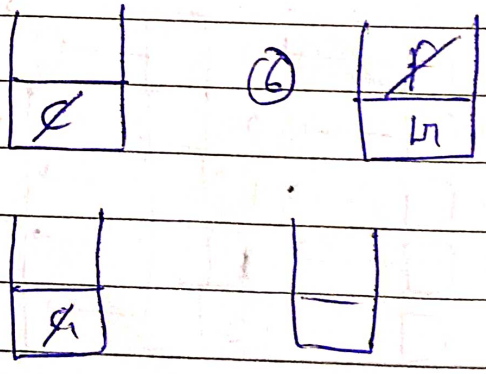
Depth first search:

- It is uniformed search tree.
- It uses the stack (LIFO) stack data structure and performs vertically.
- It doesn't guarentee a solution always.
- It is Blind search.
- Time complexity os = O(V+E)
  nz = O(b^d)
  b = branch factor, d = depth
- It performs two steges. first visited vertices are pushed into the stack, and second if there are no vertices then visited vertices are popped.





- A, B, D, E, C, F, G



- Backtracking is possible.

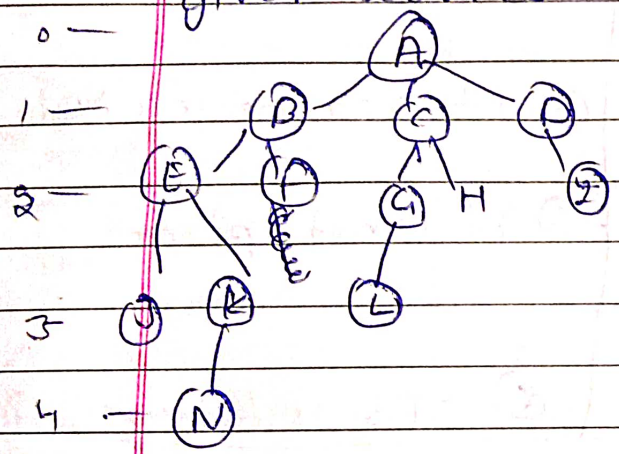
PAGE NO.:
DATE: / /

Breath first search:

It is a vertex-based technique for finding the shortest path in graph. It uses a queue data structure (FIFO)

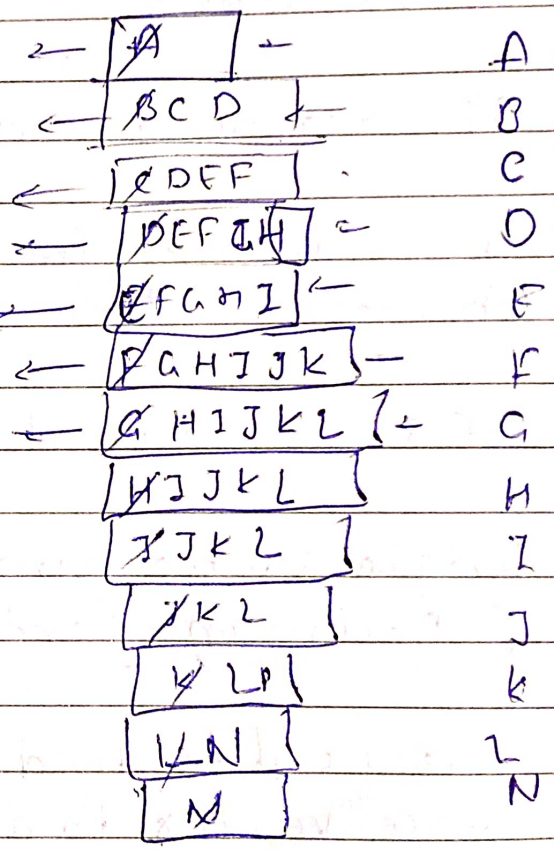
+ It gives the optimal result
+ Time complexity (1) $O(V+E)$
(2) $O(b^d)$

+ BFS works better when a user searches for the vertices that stay closer to any given sources.



A B C D E F G H I J K L

shortest path



Depth first (DFS) search	Breath first (BFS) search
(1) It uses queue (FIFO) data structure	(1) It uses stack (LIFO) data structure)
(2) BFS build by tree level by level	(2) DFS builds the tree sub tree by sub tree
(3) There is no concept of backtracking	(3) It uses idea of backtracking
(4) DFS require more memory	(4) It require less Mem
(5) It is optimal for finding shortest path	(5) It is not optimal.
(6) DFS is slow than DFS	(6) DFS is faster

Single source Single destination algorithm
(Bellman - Fort algorithm)

↓ It is used to find minimum distance from source vertex to any other vertex.

It means that it find shortest path for single destination from a single source vertex to all other vertices in a graph

It works when there is negative weight edges.

* Time complexity O(VE)

V - vertices
E - Edges