Important Notes

#-symbol
include-keyword
<> - relational operator
stdio.h - Header file (printf(), scanf())
#include - preprocessor directive, tells the program to include another file.
conio.h - console input/output, not standard clibrary
clrscr(); clear console screen
getch(); wait for key press
getche(); wait for key press and display pressed key on console

size = 2
idx = -1
idx = 0, 1

Insert_first (head, data)
Step1: (check for overflow)
If ptr = NULL then
print overflow
exit
else
ptr = (node*) malloc (sizeof (node))
Step 2: Set ptr -> data = data
Step 3: Set ptr -> next = head
Step 4: Set head = ptr

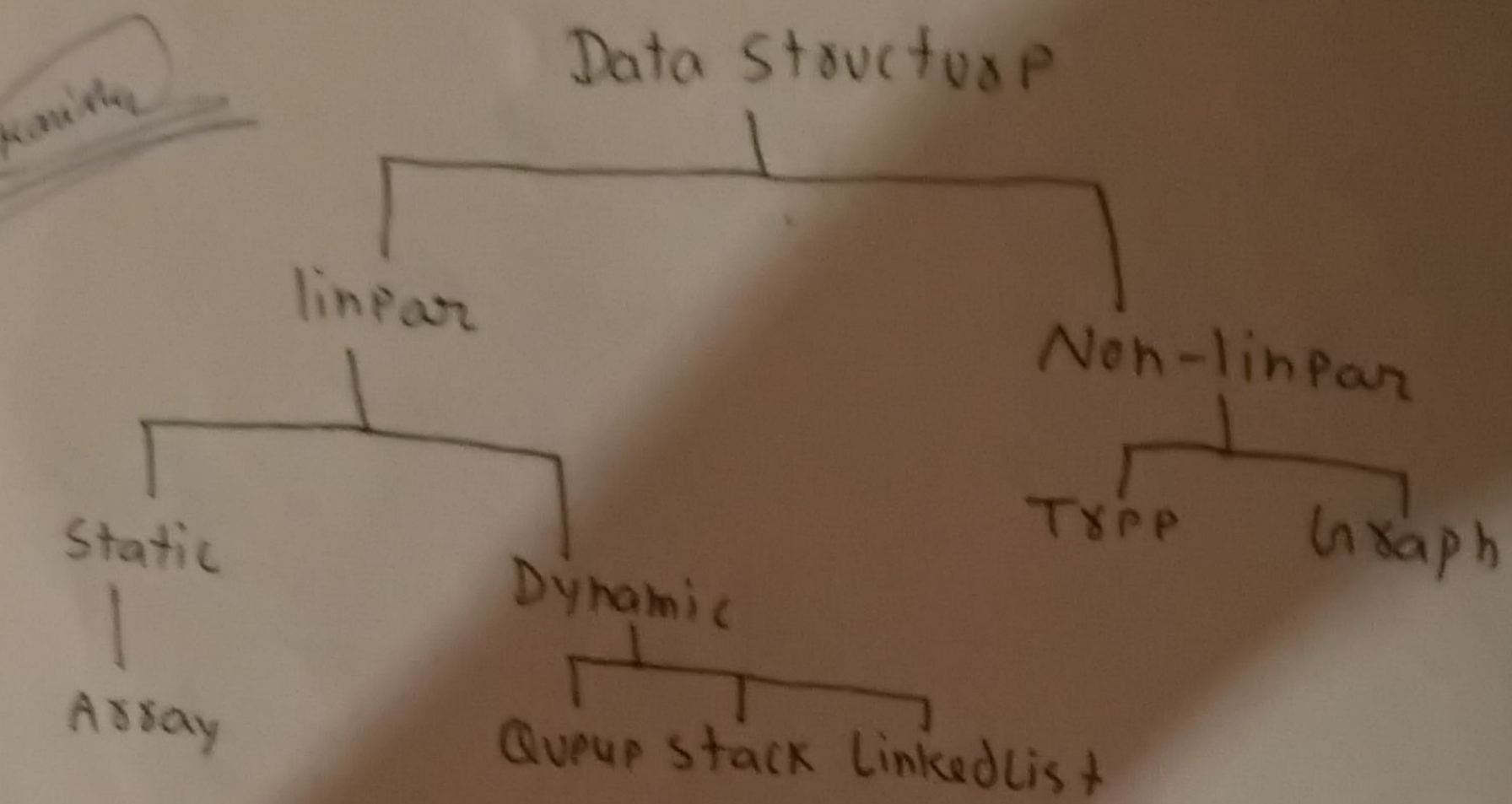
pointer
set
start
end
print
exit

If a > b then
else
No data type, pointer & void.
int x, int *p ✗
x, p ✓

ABHISHEK @csabhishek

C is medium level language by Dennis Ritchie in 1972

#



Array
function
Pointer
Structure
Dynamic allocation

Array
1-D (Single, double, circular)
Stack
Queue
Searching and Sorting

Decimal (0-9)
Binary (0-1)
octal (0-7)
Hexa Decimal (A-F, 0-9)

0+0 = 0
1+1 = 10
0+1 = 1
1+0 = 1

0-0 = 0
1-1 = 0
0-1 = 1 (0-1 = 10-1 = 2-
1-0 = 1

$$\begin{array}{r} 5 \\ -3 \\ \hline 2 \end{array} \qquad \begin{array}{r} 010 \\ 101 \\ -11 \\ \hline 010 \end{array}$$

DATA STRUCTURE :-

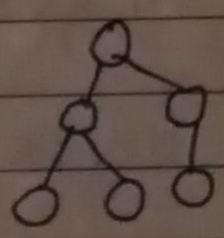
DATA Structure is a specific way to store and organize data in a computer's memory so that these data can be used efficiently later.

LINEAR DATA STRUCTURE :-

arrangement of Data in sequential manner is known as Linear Data Structure.

NON LINEAR DATA STRUCTURE :-

When one element is connected to the 'n' number is know as non-linear Data structure.



Array.

It is a collection of similar data elements stored at contiguous memory locations.

1D Array 1st way.

```
int arr[4];
arr[0]=1;
arr[1]=31;
arr[2]=21;
arr[3]=44;
```

2nd way.

```
int arr[]={1,2,3,4,5};
```

3rd way

```
int arr[5]={1,2,3,4,5};
```

4th way.

```
int arr4[5];
scanf("%d", &arr4[0]
```

2D Array 1st Way

```
int arr[2][2];
arr[0][0]=1
arr[0][1]=31;
arr[1][0]=21;
```

2nd Way

```
int arr[5][2]={{1,23},{3,43},{5,63},{7,83},{9,2}};
```

3rd way

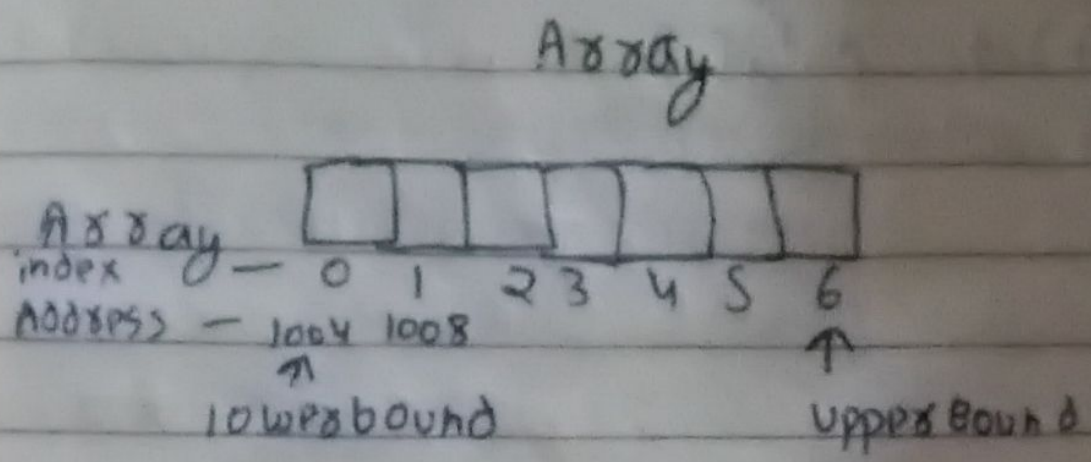
```
int arr[5][3];
```

#include <math.h>	
floor(x);	ceil(x);
4.5 → 4	4.5 → 5
$\lfloor 4.5 \rfloor = 4$	$\lceil 4.5 \rceil = 5$
$\lfloor 2.6 \rfloor = 2$	$\lceil 2.6 \rceil = 3$

- **Static Memory Allocation :-**
  Memory Allocated during compile time is called Static Memory.
  - The memory allocated is fixed and cannot be increased or decreased during runtime

- **Problem faced in static memory allocation :-**

: - If you are allocating memories for an Array during compile time then you have to fix the size at the time declaration. Size is fixed and user cannot increase or decrease the size of the Array at run time.
- If the value stored ~~[illegible]~~ in the array at run time is less than the size specified then there will be wastage of memory



An array is a linear DS that collects elements of the same data type and stores them in contiguous and adjacent-memory locations.

Types of Arrays
1) 1d Array
2) Multi Dimensional Array

Operations perform on an Array:
1) Traversal
2) Insertion
3) Deletion
4) Searching
5) Sorting

1) Traversal: Traversing an array involves visiting each element of the array involves visiting each element of the array in a specified order, typically from the to end or from the end to the beginning

Advantages of Array

- Array store multiple elements of the same type with the same name.
- You can randomly access elements in the array using an index number.
- Array memory is predefined, so there is no extra memory loss.
- Array avoid memory overflow
- 2D Arrays can efficiently represent the tabular data

Disadvantages of Array

- The no. of elements in an array should be predefined
- An array is static. It cannot alter size after declaratio
- Insertion and deletion operation in an array is quite tricky as the array stores elements in continuo form.
- Allocating excess memory than required may lead to memory wastage.

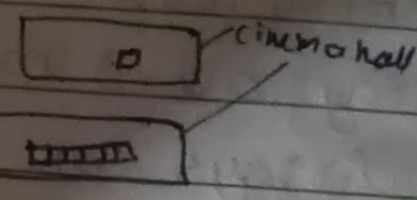
unsigned int - +ve whole numbers.

Date: 29/7/24
Page No. 8

Dynamic Memory Allocation
The process of allocating memory at the time of execution is called dynamic memory Allocation.

NOTE: Allocating memory can only be accessed through pointer
Builtin function
i) malloc ( )
ii) calloc ( )

Analgy
// single
// family



i) malloc( ) is a builtin ~~function~~ declared in header file <stdlib.h> - exit(1) present in this header file
standard library

- malloc is a short name for "memory allocation" and is used to dynamically allocatting single large block of contiguous memory acc^n to size specified.

- Syntax
(void*) malloc (size_t size)

- malloc() function simply allocates a memory block acc^n to the size specified in the heap and on success it returns a pointer pointing to the first byte of the allocated memory else it return null. ~~size_t~~.

- size_t is defined in <stdlib.h> as unsigned integer.

(void *) : void pointer :- malloc doesn't has idea of what it is pointing to.
- It merely allocates memory requested by the user without knowing the type of the data to be stored inside the memory.

NOTE :- The void pointer can be typecasted to an appropriate type

Example :-

```
int *ptr = (int *) malloc (4)
                          ↳ 4 byte of memory
```

malloc :- It allocates 4 byte of memory in the heap and the address of the first byte is stored in the pointer *ptr.

```
int *ptr = (int *) malloc (n * sizeof (int));

if (ptr == NULL) {
    printf (" memory not available");
    exist (1);
          ↳ exist failure status.
}
```

Calloc :- calloc function is used to dynamically allocates multiple blocks of memory.
It is different from malloc in two ways
i) calloc need two argument instead of one

Syntax :-

void * calloc (size_t, size_t size);
↳ no. of blocks

Ex. int *ptr = (int *) calloc (10 * sizeof (int));

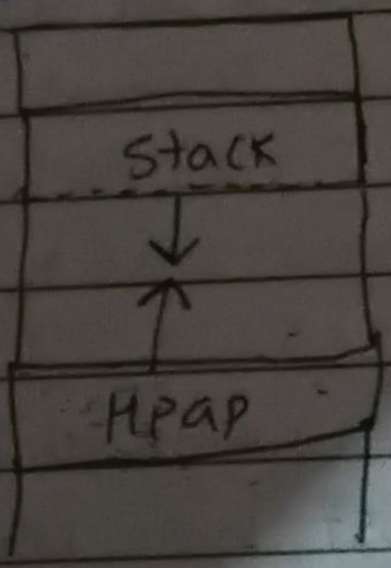
Memory allocated by calloc is initialized to zero.

NOTE - It is not the case with malloc. Memory allocated by malloc is some initialized with garbage value.
calloc both return null when sufficient is not available in the heap

Date / /
Page No. 11

Heap Memory

- Heap is the segment of memory where dynamic memory allocation take place.
- Unlike stack where memory is allocated or deallocated in a defined ordered, heap is an area of memory where memory is allocated or deallocated without any order. or randomly. There are certain builtin function that can help in allocating / deallocating some memory space at runtime.



Date: / /
Page No.

Linked List

It is a dynamic data structure that consists of a sequence of elements, each of which is called a node.
Unlike arrays,
linked list do not require a contiguous block of memory. instead,
they are spread out across memory and connected using pointers. Each node in a linked list contains data and a pointer to the next node in the sequence. This allows for efficient insertions and deletions.

Type of linked lists.

1. Singly linked list:- Each node points to the next node in the sequence, and the last node points to 'NULL'. This allows traversal in only one direction.

2. Doubly Linked list:- Each node has two pointers: one pointing to the next node and another pointing to the previous node. This allows traversal in both directions.

3) Circular Linked list:- Similar to singly or doubly linked lists, but the last node points back to the first node, forming a circle.

- Create a node (using 1 Pointer, 2 Pointer)
- Insertion (beg, end, Pos)
- Deletion (beg, end, Pos)
- Traversal with count

Date : / /
Page No. 13

- Self referential structure :- It is a structure which contains a pointer to a structure of same type

```
struct abc{
        int a;
        char b;
        struct abc *self;
          };          (Pointer).

struct node {
        int data;
        struct node *link;
        };          (address of another variable)
```

fopp(temp):- This function is used to free or deallocate the dynamically allocated memory. and helps in reducing memory wastage.

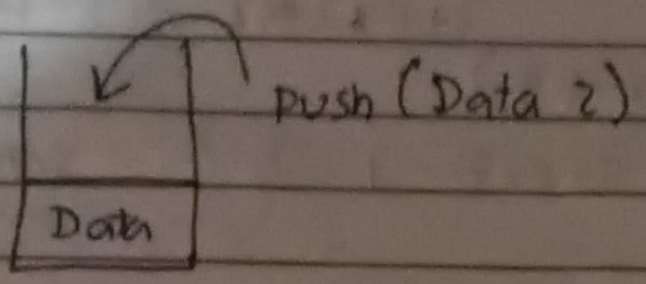
Date : / /

Page No.

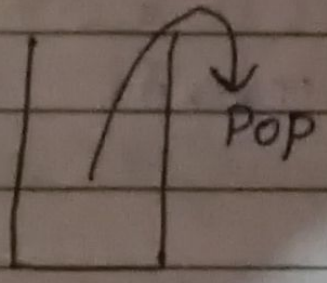
STACK

Stack :- It is a linear list where all insertion and deletion are allow only at the end, call top of the stack

1) Primary stack operation :- a) Push (Data); insert data onto stack

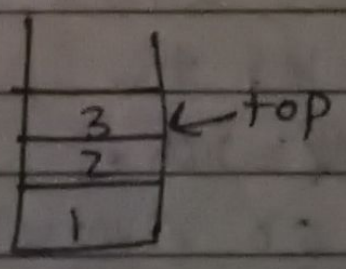


b) ~~Pop(Data)~~ pop(); delete the last inserted element from the stack



2) Secondary stack operation :-

a) top(); return the last inserted element without removing it.



b) size(); return the size or the number of element in the stack.

c) isEmpty(); return true if stack is empty, else

POW(a,b) = a^b
(In code) (In text)

Date : / /
Page No.

isfull( ); returns true if the stack is full, else return false;

3+5^6-7 3+15625-7 = 15621
254+10-9 255
365*52*7+10/10 132860+1 = 132861
3+5*(7-4)^2 3+5*3^2 = 3+75^2 = 228
8*(5^4+2)-6^2//9*3)
8*(625+2)-36/27 = (8*627)-1 = 5016-1 = 5015

Paranthesis has highest priority

Infix Notation: {a(+)b}
operands
operators

operators are written blw their operands.

Prefix Notation :- +ab

In which operators is written before the operands. It is also called as polish Notation

Postfix Notation: ab+

In which operators is written after the operands, so it is called postfix Notation.
It is also known as suffix Notation or reverse polish Notation

Postfix Notation is type of notation which is most suitable for a computer to calculate any expression. It is universally excepted Notation for designing arithmetic and logical unit of the CPU.

( ) > * / > + -

Date: / /
Page No.

- Any expression entered into the computer is first converted into postfix Notation, stored in Stack and calculated

Associativity :-

( )
^
* / left to right
+ - left to right

STACK IMPLEMENTATION USING LINKED LIST

- Use linked list when the size of the stack is not known in advanced
- Top of the stack, we will select beginning of the linked list as the top of the stack.
- Time complexity of adding a node at the beginning is ~~1~~. O(1).
- Time complexity removing the first top is O(1)
- Time complexity of adding a node at the end is O(1)
- Time complexity of removing the last node O(n)
- delete the key node of Singly linked list requires traversal

Time Complexity

	Insert a node	Delete a node
1)	At the beg → O(1)	At beg O(1)
2)	At the end → O(1)	At end O(n)

- Stack overflow occurs when there is no space left to dynamically allocate the memory. In that case malloc function will return NULL.
- Stack underflow occurs when top is ~~=~~ NULL.

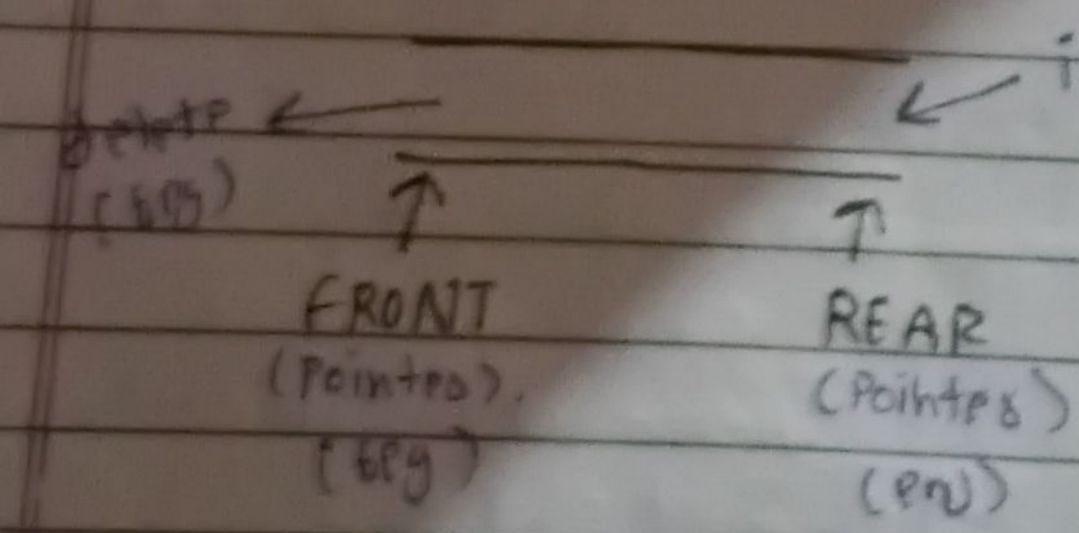
- It is a linear ds which operates in first in first out or last in last out.
- It is named queue as it behaves like a real world queue.

Ex- Queue of cars in a single lane,
Queue of people waiting at foodcounters etc

- It is simple datastructure that allows adding and removing element in a particular order.
- The order is ~~[illegible]~~ 'FIFO' (first in first out). 'LILO' (Last in Last out).

Basic operation :-

1) Enqueue :- addition of element litem.
2) Dequeue :- deletion of element litem.



The insertion is performed at the end called 'REAR' of the Queue.
The deletion is performed at the beginning (called) FRONT' of the Queue.

0	1	2
~~2~~	3	4

F = -1 ~~0~~ 1 (while dequeue front is +1)
R = -1 ~~0~~ + 2 (while enqueue Rear is +1)

Difference b/w stack & Queue

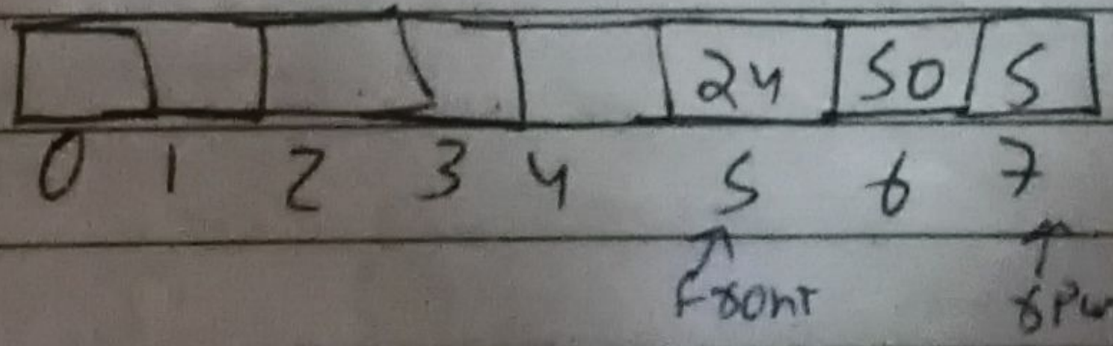
STACK	QUEUE
• Incase of stacks, Insertion / delRtion ~~of~~ is always performed at one end i.e top of the stack	• Incase of Queue, Insertion is performed at the end and deletion is always performed at the beginning.
• Stack follow 'LIFO'	• Queue follow 'LILO' / 'FIFO'.

Enqueue :-
This operation adds an element or item to the rear of the queue.
If Queue is Empty, The added element become the first element of the Queue.

Dequeue :-
This operation removes an element from the front of the queue.
If Queue is Empty, This operation can't be performed

F:- Mostly Each of our queue will be maintain by linear array queue and two pointer variables Front containing the location of the front element of the Queue and the rear containg the location of the rear element of the Queue.

Drawback of Queue using Array



- Consider this case when front != 0 and rear max-1 this means we can't insert new elements in the queue, even though space is available to insert 5 elements in the queue.

(iii) The Calloc () Contains 2- arguments i.e no. of elements and size of each memory.

E.g →

```
int *p;
p=(int*)calloc(10,
    sizeof(int));
```

(iii) The malloc () Contain One arguments i.e. Size of data-type.

E.g

```
int *p.
p=(int *)malloc(size of
    (int));
```


ns→ Calloc ()

(i) Calloc () Stands for Contiguous allocation.

(ii) It's allocates a multiple block of memory of Same datatype/size.

Malloc ()

(i) malloc () Stands for memory allocation.

(ii) It's allocates a sigle block of memory of Specified Size

V.V.I

Calloc ()	Malloc ()
(iv) It is slower than malloc.	(v) It is faster than Calloc.
Syntax :-	Syntax :-
p=(data-type*)Calloc(n, size));	p=(data-type*)malloc (si

Postfix → Prefix = operator + operand 2 + operand 1

Prefix → Postfix = operand 1 + operand 2 + operator

Postfix → Infix = operand 2 + operator + operand 1

Prefix → Infix = operand 1 + operator + operand 2

Postfix → Prefix = operator + operand2 + operand1

Prefix → Postfix = operand1 + operand2 + operator

Postfix → Infix = operand2 + operator + operand1

Prefix → Infix = operand1 + operator + operand2

reverse

Prefix to postfix

operand1 + operand2 + operator

*-A/BC-/AKL

reverse :- LKA/-CB/A-*

① ② ③

*	⇒AK/	-	⇒AK/L-	/	⇒		BC/
K		AK/		C			
L		K		AK/L-			

A		*		
BC/	⇒ABC/-	ABC/-	⇒	ABC/-AK/L-*
AK/L-		AK/L-		

④ ⑤

reverse

SP

nd top + operand top-1 + operator

x to postfix

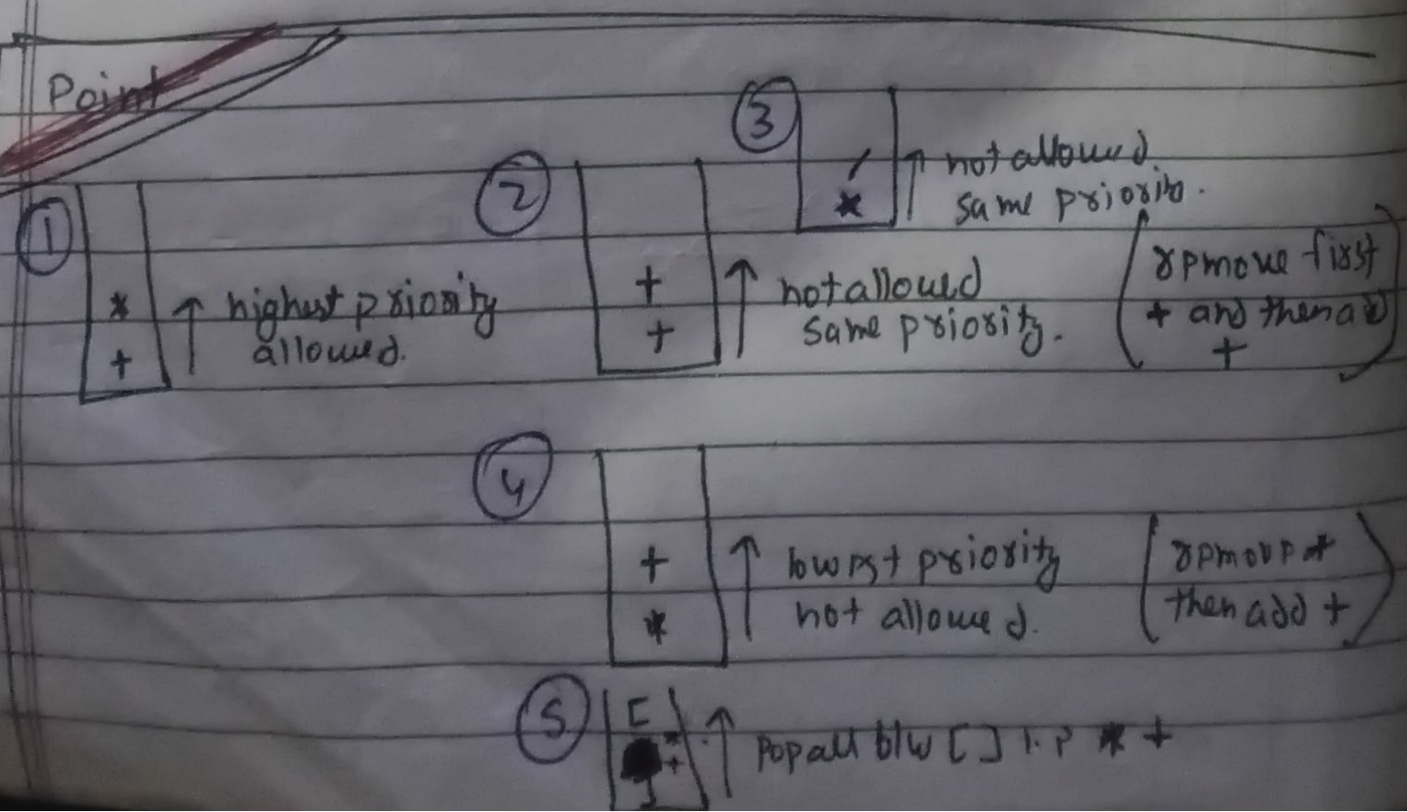
(A+B/C) (A-*A+B/C

Date : / /
Page No.

(A+B x (C+D)/F + D x E)

Symbol	Stack	String (Post fix)
[	[	
A	[	A
+	[+	A
B	[+	AB
x	[+*	AB
[	[+*[	AB
C	[+*[	ABC
+	[+*[+	ABC
D	[+*[+	ABCD
]	[+*[+] → [+*	ABCD+
/	[+* → [+/	ABCD+*
F	[+/	ABCD+*F
+	[+	ABCD+*F/+
D	[+	ABCD+*F/+D
x	[+*	ABCD+*F/+D
E	[+*	ABCD+*F/+DE
]		ABCD+*F/+DE*+

Point



Q. Infix to prefix
(A-B/C)*(A/K-L)

Reverse:- (L-K/A)*(C/B-A)

Symbol	Stack	Prefix
(	)	
L	)	L
-	)-	L
K	)-	LK
/	)-/	LK
A	)-/	LKA
)	)-/)	LKA/-
*	*	LKA/-
(	*(	LKA/-
C	*(	LKA/-C
/	*(/	LKA/-C
B	*(/	LKA/-CB/
-	*(-	LKA/-CB/-
A	*(-	LKA/-CB/A
)	*	~~LKA/-CB/A-*~~
		LKA/-CB/A-*

Rule:-
Reverse
infix to postfix rule
Reverse - ans

Reverse.
*-A/BC-/AKL

ans

Date: / /
Page No

Hashing

It is the process of converting a key into a specific index in a hash table using a hash function for efficient data storage and retrieval.

Components of ~~the~~ Hashing

- Key : Input (string or no.) to the hash function
- Hash function : converts the key into a hash index
- Hash Table : store key-value pairs in an array-like structure at the computed index.

Key → Hash Function → Hash Table

Key	value
0	87
1	93
2	
3	64

(HashTable)

Types of Hash function

1) Division Method

N = 1000

Key = 123456789

H(Key) = Key % N

= 123456789 % 1000

= 789

2) Mid Square Method

Q.) N = 1000 (4bit)

Key = 121

H(Key) = h(Key × Key)

= $(121)^2$

= 14641

= 464 (3bit)

Q.) N = 100 (3bit)

Key = 121

H(Key) = $(121)^2$ = 14641

= 46086 (2bit)

Depend what you wanna choose

Digit folding

3) Digit Extraction Method

Q.) N = 1000 (4bit)

Key = 789456123

h(Key) = 789+456+123

= 1368

again 1368

= 136+8 = 144 (3bit)

Q.) N = 100 (3bit)

Key = 789456123

h(Key) = 78+94+56+123

= 243

again 243

= 24+3 = 27 (2bit)

Date: / /
Page No.

Hashing Need

	Hashing		Array	
Insertion	O(1)		O(1)	
Search	O(1)	O(n) (due to collision)	O(n) (unsorted array)	O(logn) (sorted array)

Collision :- It occurs in hashing when two different keys map to the same hash value.

Collision Resolution Technique (used to maintain performance)

Separate Chaining / Open hashing	Open addressing / Close hashing
(Each hash table index points to a linked list storing keys with the same hash value).	(resolves collisions by probing for the next available slot).
load factor = no. of element / Table size.	- linear Probing - Quadratic Probing - Double Hashing.
threshold load factor is decided by self like 0.5, 0.9 If threshold load factor is reached rehashing occurs.	
Rehashing :- process of resizing the hash table and re-computing hash	

Date : / /
Page No.

linear Probing :-

56, 79, 41, 44, 21, 43, 11, 29

$H'(K) = K \% h$

$H(K,i) = (H'(K) + i) \% h$

$(i = 0, 1, 2 \dots)$

- $56 \% 10 = 6$ $(h = 10)$
  $(6+0) \% 10 = 6$

- $79 \% 10 = 9$
  $(9+0) \% 10 = 9$

- $41 \% 10 = 1$
  $(1+0) \% 10 = 1$

- $44 \% 10 = 4$
  $(4+0) \% 10 = 4$

- $21 \% 10 = 1$
  $(1+0) \% 10 = 1$
  $(1+1) \% 10 = 2$

- $43 \% 10 = 3$
  $(3+0) \% 10 = 3$

- $11 \% 10 = 1$
  $(1+0) \% 10 = 1$
  $(1+1) \% 10 = 2$
  $(1+2) \% 10 = 3$
  $(1+3) \% 10 = 4$
  $(1+4) \% 10 = 5$

0	29
1	41
2	21
3	43
4	44
5	11
6	56
7	
8	
9	79

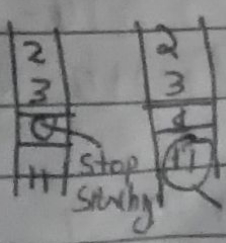
Primary clustering.

- $29 \% 10 = 9$
  $(9+0) \% 10 = 9$
  $(9+1) \% 10 = 0$

Advantage : No extra space

disadvantage.

- ~~Insertion~~ ~~O(1)~~
- deletion hard (Because rehashing)



(Here does can put $ symbol deleted element so that we can search below it too.)

- Primary clustering
- Secondary clustering
- Searching $O(n)$

index repeat after some time and we have to find index to store

Date : / /
Page No.

Quadratic Probing

$h'(k) = k \% n$ $\quad n = 10$

$h(k,i) = (h'(k) + i^2) \% n$

$(i = 0, 1, 2, 3 \dots)$

89, 18, 49, 58, 69, 29

$89 \% 10 = 9$

$(9 + 0^2) \% 10 = 9$

$18 \% 10 = 8$

$(8 + 0^2) \% 10 = 8$

$49 \% 10 = 9$

$(9 + 0^2) \% 10 = 9$

$(9 + 1^2) \% 10 = 0$

$58 \% 10 = 8$

$(8 + 0^2) \% 10 = 8$

$(8 + 1^2) \% 10 = 9$

$(8 + 2^2) \% 10 = (8+4) \% 10 = 12 \% 10 = 2$

$69 \% 10 = 9$

$(9 + 0^2) \% 10 = 9$

$(9 + 1^2) \% 10 = 0$

$(9 + 2^2) \% 10 = (9+4) \% 10 = 3$

$29 \% 10 = 9$ $\quad (9 + 4^2) \% 10 = (9+16) \% 10 = 5$

$(9 + 0^2) \% 10 = 9$

$(9 + 1^2) \% 10 = 0$

$(9 + 2^2) \% 10 = 3$

$(9 + 3^2) \% 10 = 18 \% 10 = 8$

0	49
1	
2	58
3	69
4	
5	29
6	
7	
8	18
9	89

Disadvantage
- Secondary clustering
- deletion hard.
- ~~Deletion OK~~
- space available but we can't insert
- Searching O(n)

Advantage
No primary clustering
No extra space

Date : / /
Page No.

Double Hashing

$h_1(K) = K \bmod n$ — prime no.

$h_2(K) = 8 - (K \bmod 8)$ — prime no

$n = 11$

disadvantage
Searching O(n)

Advantage
No Primary clustering
No Secondary clustering
No Extra Space

$h(K,i) = (h_1(K) + i\,h_2(K)) \% n$
$(i = 0, 1, 2, 3 \ldots)$

20, 34, 45, 70, 56

Soln

$20 \% 11 = 9$
$8 - (20 \% 8) = 8 - 4 = 4$

$(9 + 0(4)) \% 11 = 9 \% 11 = 9$

$34 \% 11 = 1$
$8 - (34 \% 8) = 8 - 2 = 6$

$(1 + 0(6)) \% 11 = 7$

$45 \% 11 = 1$
$8 - (45 \% 8) = 8 - 5 = 3$

$(1 + 0(3)) \% 11 = 1$
$(1 + 1(3)) \% 11 = 4$

$70 \% 11 = 4$
$8 - (70 \% 8) = 8 - 6 = 2$
$(4 + 0(2)) \% 11 = 4$
$(4 + 1(2)) \% 11 = 6$

0	
1	34
2	
3	56
4	45
5	
6	70
7	
8	
9	20
10	

$56 \% 11 = 1$
$8 - (56 \% 8) = 8$
$(1 + 0(8)) \% 11 = 1$
$(1 + 1(8)) \% 11 = 9$
$(1 + 2(8)) \% 11 = 17 \% 11 = 6$
$(1 + 3(8)) \% 11 = 3$

Q. Key 1, 3, 12, 4, 25, 6, 18, 20, 8
length of Hash table = 10
hash function = i^2 %10 and linear probing

i) Resultant hash table?

ii) max probe value?

0	20
1	1
2	8
3	
4	12
5	25
6	4
7	6
8	18
9	3

~~$h(i) = i^2$ %10~~

~~$= i^2 \cdot 1 \cdot 10$~~

$= 1$

$h(1) = 1^2 \cdot 1 \cdot 10$

$= 1$

$h(3) = 3^2 \cdot 1 \cdot 10$

$= 9$

$h(4) = 4^2 \cdot 1 \cdot 10$

$= 6$

$\Rightarrow 1^2 \cdot 1 \cdot 10 = 1$

$\Rightarrow 3^2 \cdot 1 \cdot 10 = 9$

$\Rightarrow 12^2 \cdot 1 \cdot 10 = 144 \cdot 1 \cdot 10 = 4$

~~$(4+1) \cdot 1 \cdot 10 = 5$~~

$\Rightarrow 4^2 \cdot 1 \cdot 10 = 16 \cdot 1 \cdot 10 = 6$

$\Rightarrow 25^2 \cdot 1 \cdot 10 = 625 \cdot 1 \cdot 10 = 5$

$\Rightarrow 6^2 \cdot 1 \cdot 10 = 6$

$(6+1) \cdot 1 \cdot 10 = 7$

$\Rightarrow 18^2 \cdot 1 \cdot 10 = 324 \cdot 1 \cdot 10 = 4$

$(4+1) \cdot 1 \cdot 10 = 5$

$(4+2) \cdot 1 \cdot 10 = 6$

$(4+3) \cdot 1 \cdot 10 = 7$

$(4+4) \cdot 1 \cdot 10 = 8$

$\Rightarrow 20 \cdot 4 \cdot 10 = 0$

$\Rightarrow 8^2 \cdot 1 \cdot 10 = 64 \cdot 1 \cdot 10 = 4$

$(4+1) \cdot 1 \cdot 10 = 5$

$(4+2) \cdot 1 \cdot 10 = 6$

$(4+3) \cdot 1 \cdot 10 = 7$

$(4+4) \cdot 1 \cdot 10 = 8$

$(4+5) \cdot 1 \cdot 10 = 9$

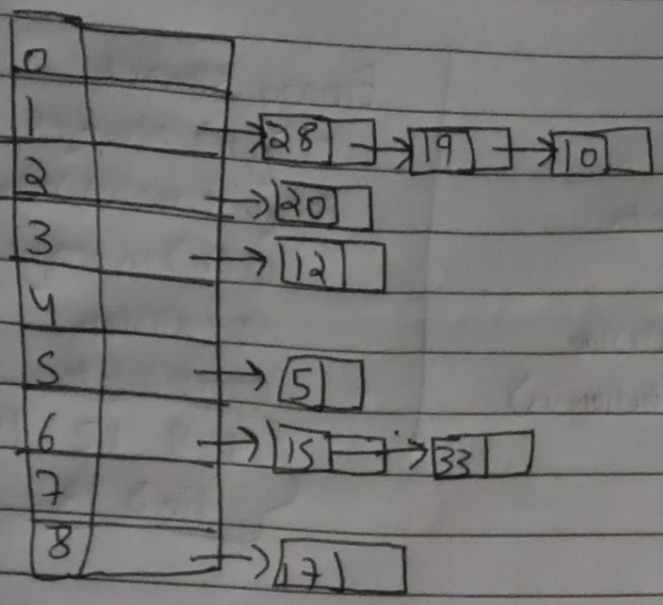
$(4+6) \cdot 1 \cdot 10 = 0$

$(4+7) \cdot 1 \cdot 10 = 11 \cdot 1 \cdot 10 = 1$

$(4+8) \cdot 1 \cdot 10 = 12 \cdot 1 \cdot 10 = 2$

Date : / /
Page No.

Q-



Max chain length = 3
Min chain length = 0 (as 0, 4, 7 all index has 0 element) min no chain
avg chain length = $\frac{9}{9} = 1$

28, 19, 10, 20, 12, 5, 15, 33, 17 => Total element 9
Total size = 0+8+1 = 9
$\therefore \frac{\text{tot elem}}{\text{tot size}} = \frac{9}{9} = 1$

Hashset :- It is a collection that store unique element. It is unordered. It internally use hashmap to avoid duplicacy.

1 4 2 3 4 1 3 - this won't be added

insert - O(1)
search - O(1)
delete - O(1)

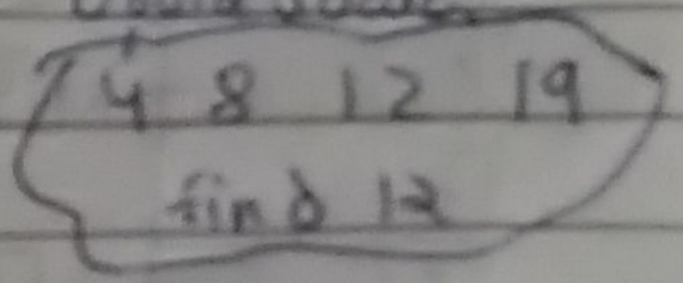
Hashmap :- It is a data structure that stores key-value pairs where each key is unique.

(If key-value again inserted then value will be updated)
china 180
china 150 (180 change to 150)

key	value
fuel	Price
cNG	70
petrol	80

Date : / /
Page No.

	Hashing	Binary Search
Speed	O(1) to O(n)	O(logn) always
data order	Works on unsorted data	Require sorted data.
insert / delete	O(1)	O(n) in array
Collission	need to handle collision	NO collision
USP Cases	Quick lookups (ex- dictionaries)	Ordered data



Date: / /
Page No.

Stack (n-1) (-1)
- Push, Pop, isFull, isEmpty → Array / linked list

Queue
- Enqueue, Dequeue → Array / linked list

- Infix to postfix
- Prefix to postfix
- Infix to prefix

Evaluate Postfix

Postfix → Prefix, Infix
Prefix → Postfix, Infix

- Create a node (using 1 pointer, 2 pointer)
- Insertion (beg, end, pos)
- Deletion (beg, end, pos)
- Traversal with count

Sorting	time		span	
insertion	$O(n^2)$		$O(1)$	(1 time) store key in temp and find right index and place
selection	$O(n^2)$		$O(1)$	(1 time) select key and find min then swap key with min
Bubble	$O(n^2)$		$O(1)$	(n-1 times) largest elements come to end by swaping with adjacent
merge	$O(n \log n)$	$O(n \log n)$	$O(n)$	
Quick	$O(n^2)$ (when given array is already sorted)	$O(n \log n)$	$O(\log n)$	

Traversal in stack
- Preorder
- Inorder
- Postorder

Date : / /
Page No.

1. Insertion sort :- insert each element into its correct place in the sorted part of the list. [O(n²) (time) O(1) (space)]

2. Selection sort :- find the smallest element in the unsorted part and swap it to its correct position [O(n²) O(1)]

3. Bubble sort :- Repeatedly swap adjacent elements if they are in the wrong order. [O(n²) O(1)]

4. Merge sort - split the list in half, sort each half, and merge the sorted halves [O(nlogn) O(n)]
(follow divide and conquer approach)

5. Quick sort - pick a pivot, partition the list around the pivot, and sort each partition. [O(nlogn) O(logn)
O(n²) - If array is already sorted]
Pivot ← random element
median element
first element
last element

Sorting Algorithm - It is a method used to arrange elements in a particular order, typically ascending or descending. Sorting makes data easier to search, analyze and display in structured way.

Tree :- It is type of data structure that represent a hierarchical relationship b/w data elements called nodes.
Tree is a connected graph without any cycle.
Binary - Tree :- tree data structure where each node has atmost 2 children.

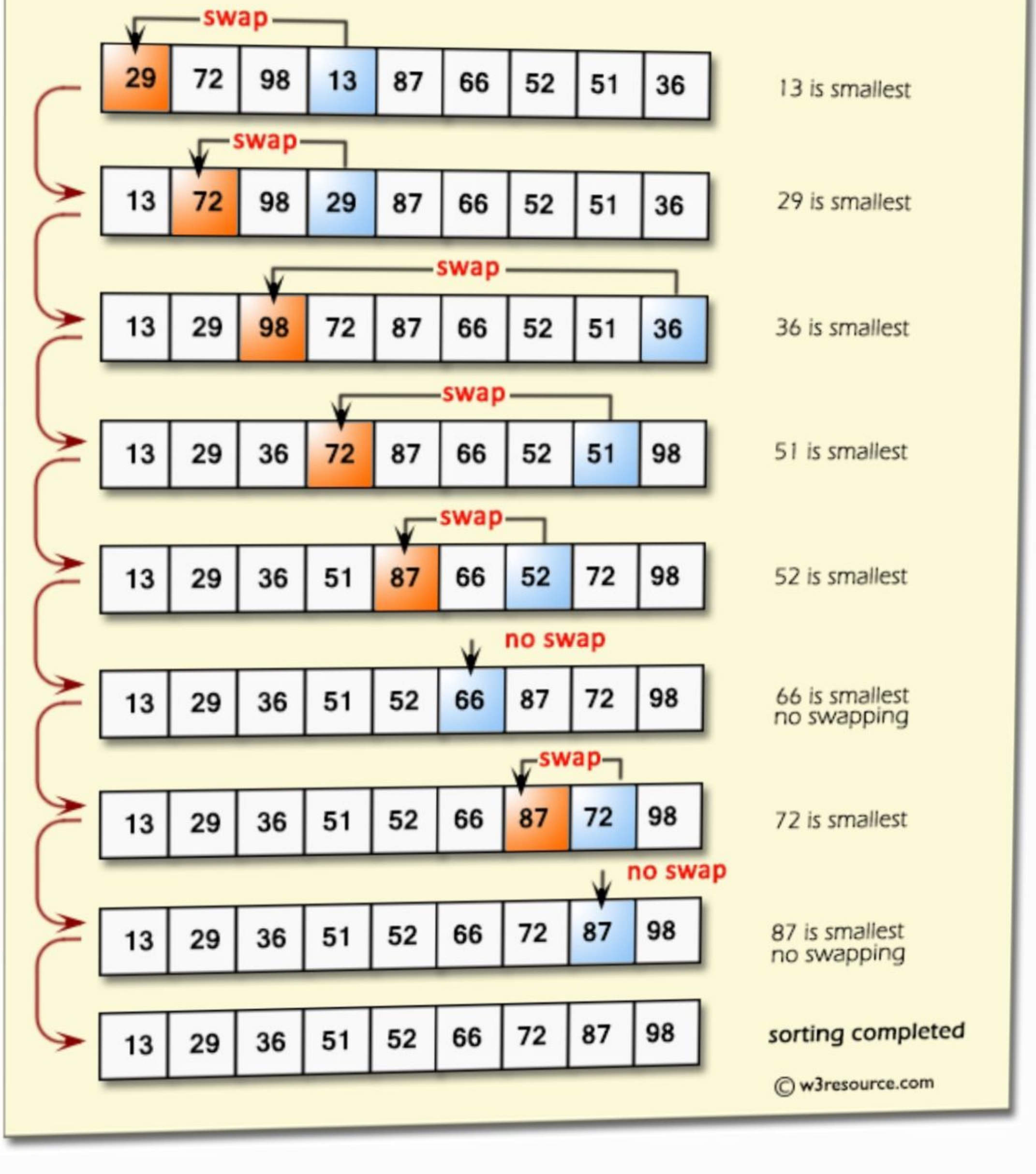
Traversal in Tree

Preorder Traversal :- visit the root first, then left subtree and finally right subtree.

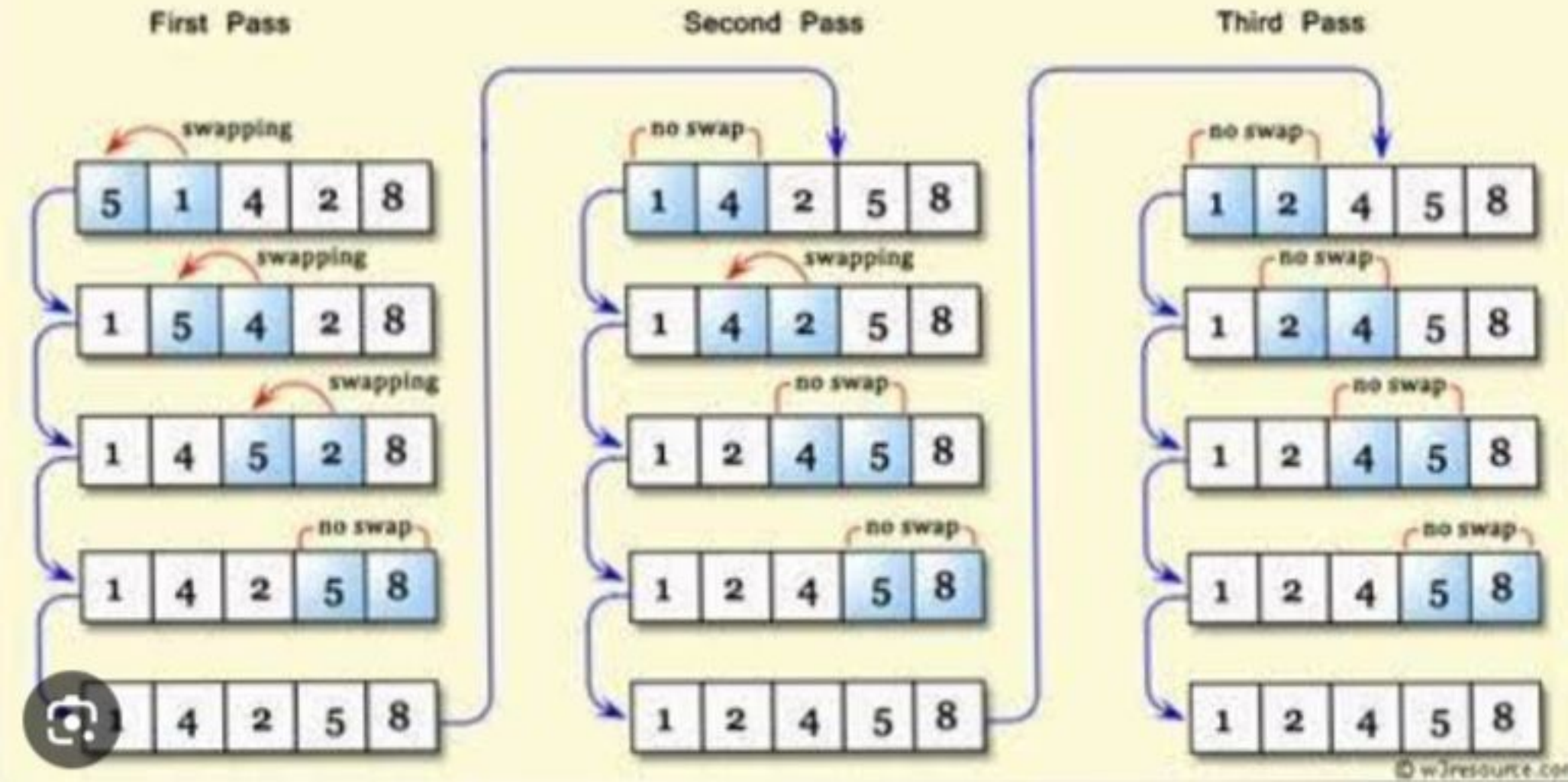
Inorder Traversal :- visit the left subtree first, then the root, and then right subtree.

Postorder Traversal :- visit the left subtree first, then the right subtree

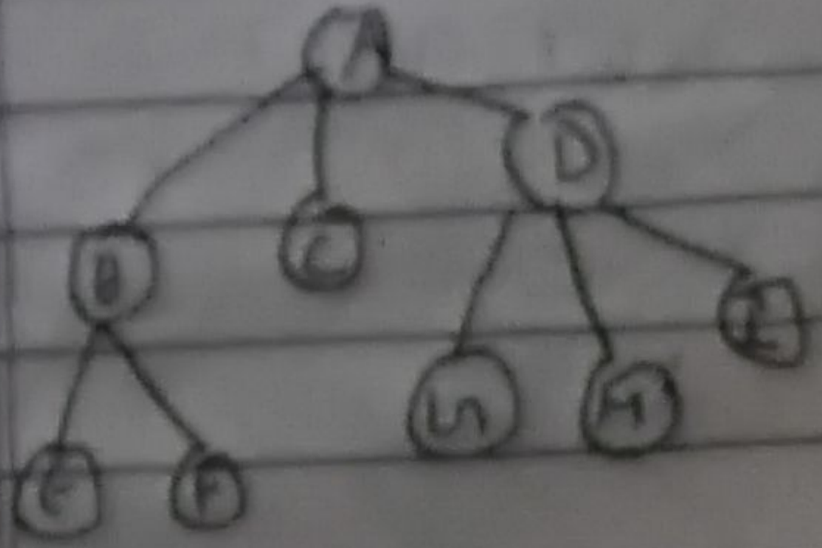
Selection Sort



Bubble Sorting

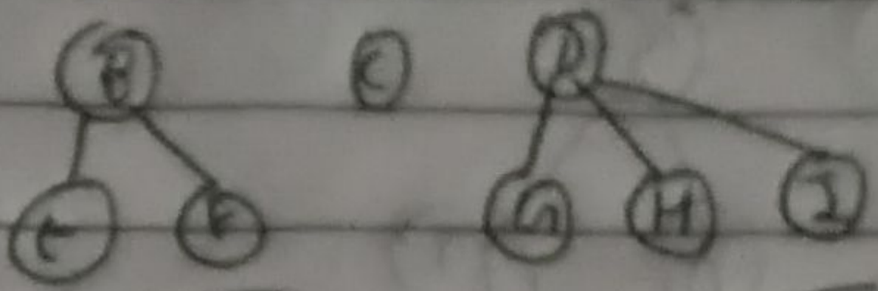


Node - element in a tree
Root - Topmost node having no parent
Parent - Node with children
Child - Node with a parent
Leaf - Node with no children.
Ancestor - Any node above a given node
Descendant - Any node below a given node.
Level - Distance from the root where root is level 0
Sibling - Nodes with same parent
Depth - Distance from node to root, same as node's level
height - longest path from node down to a leaf
degree - Number of children a node has
edge - link b/w nodes (i.e parent to child). n node = n-1 edge
Internode:- Node with at least one child.



- Path A to I = (A,D)(D, I)
(Way to node to node)

- forest (collection of disjoint tr



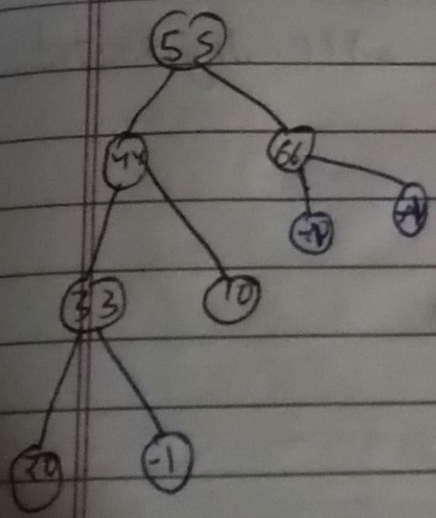
(HPBP A is removed)

So call 3 BPrime
disjoint (Independent)

Date : / /
Page No.

Representation of binary tree.

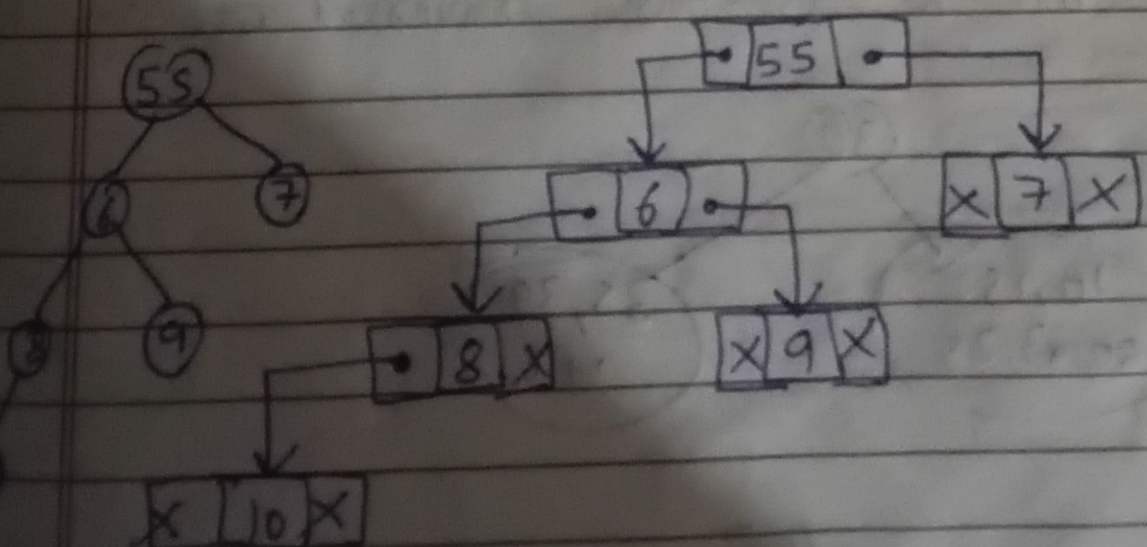
1) Array Representation of Binary tree
   i) Root is stored in a[0]
   ii) Node occupies a[i]
   left child -> [2*i+1]
   right child -> [2*i+2]
   Parent node -> [(i-1)/2]



a[i]	0	1	2	3	4	5	6	7	8
	55	44	66	33	10	~~41~~	~~41~~	20	-1

2x0+1=1, 2x0+2=2 → 0
2x1+1=3, 2x1+2=4 → 1
2x2+1=5, 2x2+2=6 → 2
~~2x3+1=7~~, 2x3+2=8 → 3

2) LinkedList Representation of Binary tree

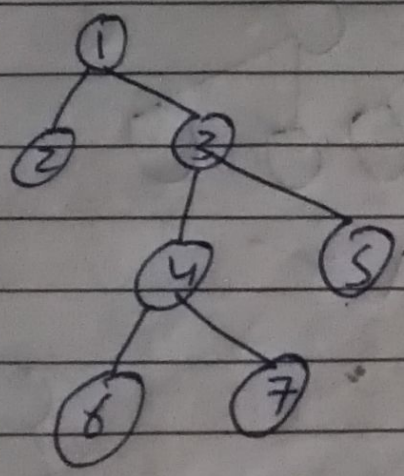


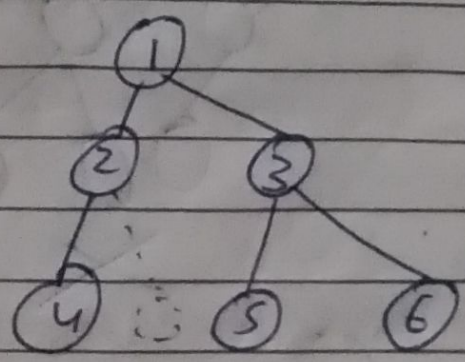
Date : / /
Page No.

Types of Binary Tree. (Important)

1) Full / Proper / Strict Binary tree

all nodes have either 0 or 2 children.

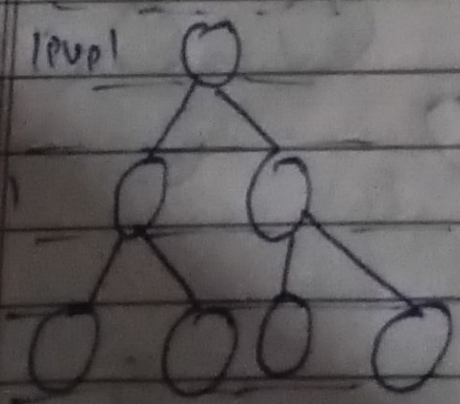


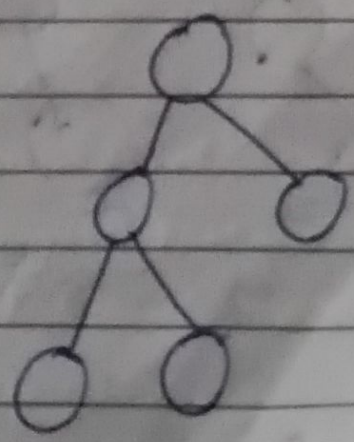


2 has 1 child

2) Perfect Binary tree
(to ignore leaf level)

Internal node have two children + all leaves nodes are on same level.

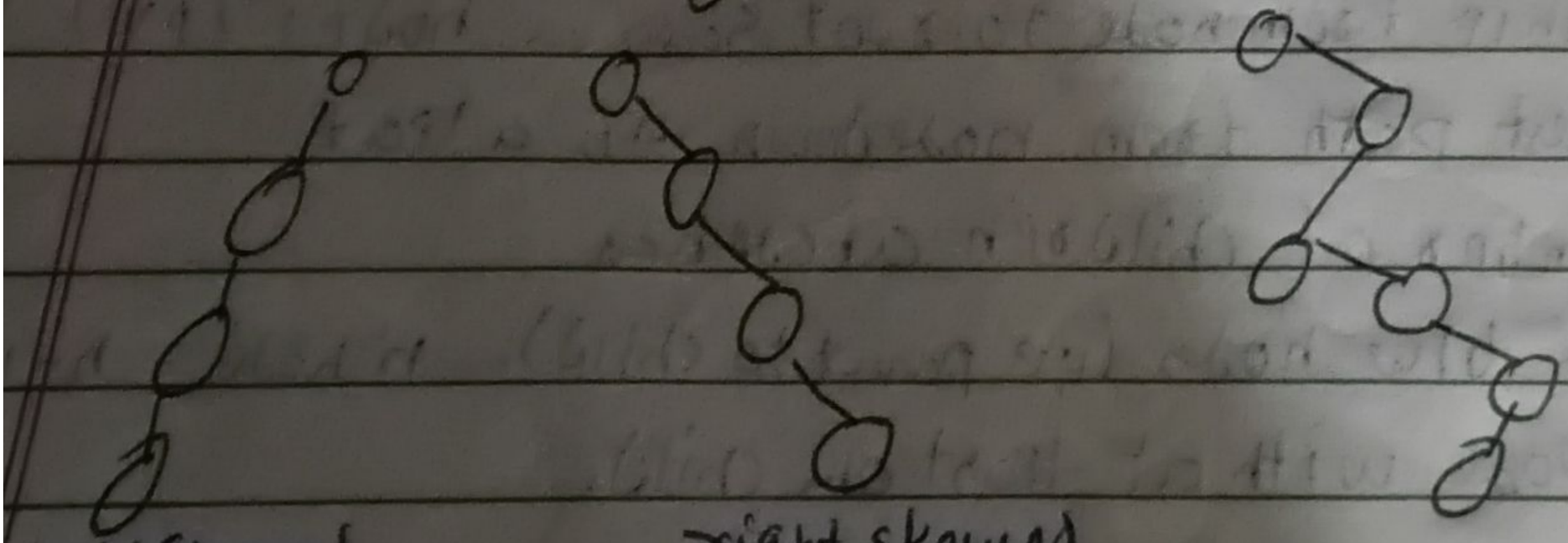




Date : / /
Page No.

3) Complete binary tree All levels are completely filled except last level + last level must have its key as left as possible.

4) Degenerate Binary tree (each internal node has 1 child)



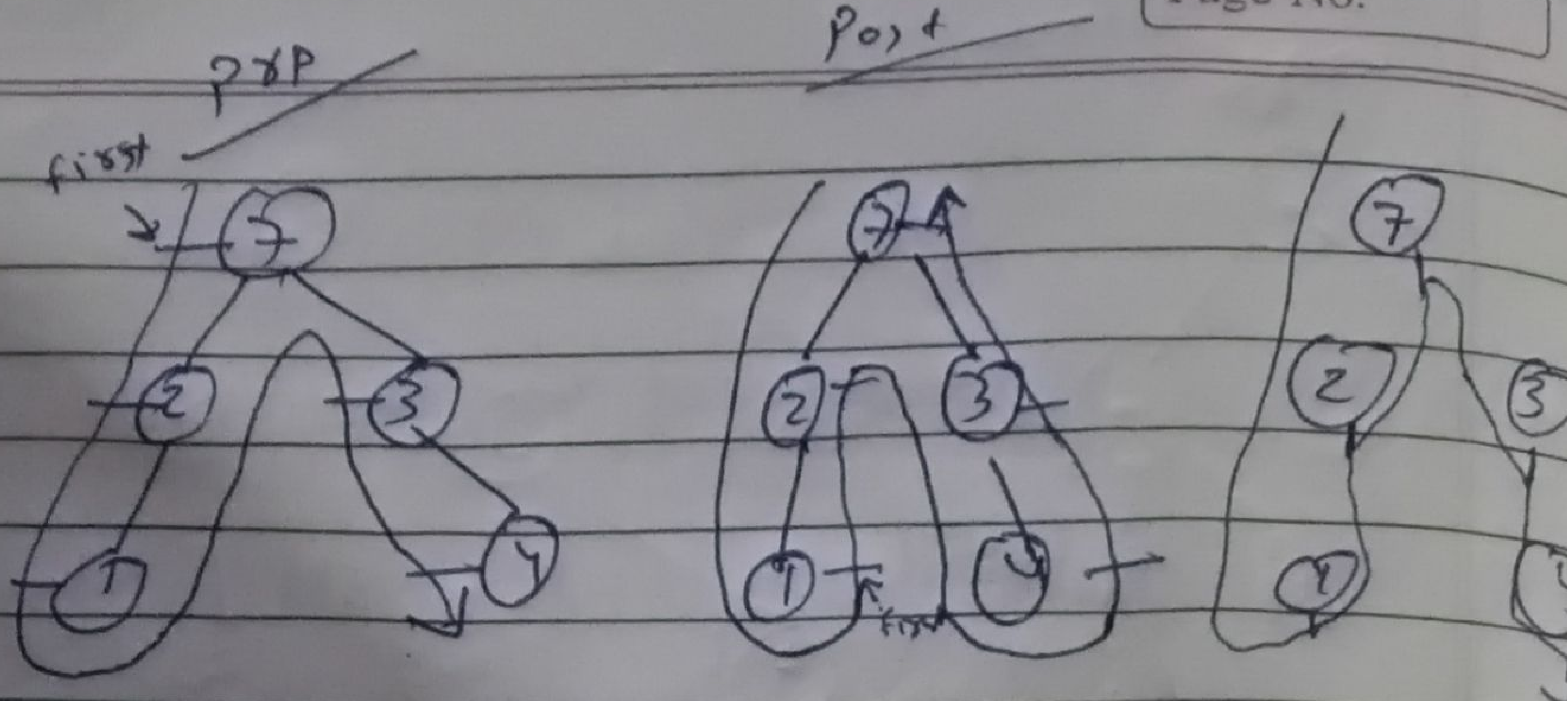
left skewed Binary tree

right skewed Binary tree

Page No.

Inorder

Trick ~~Not for Exam~~



Inorder :- 7 2 1 A B 4

Pre (root left right) => 7 2 1 3 4

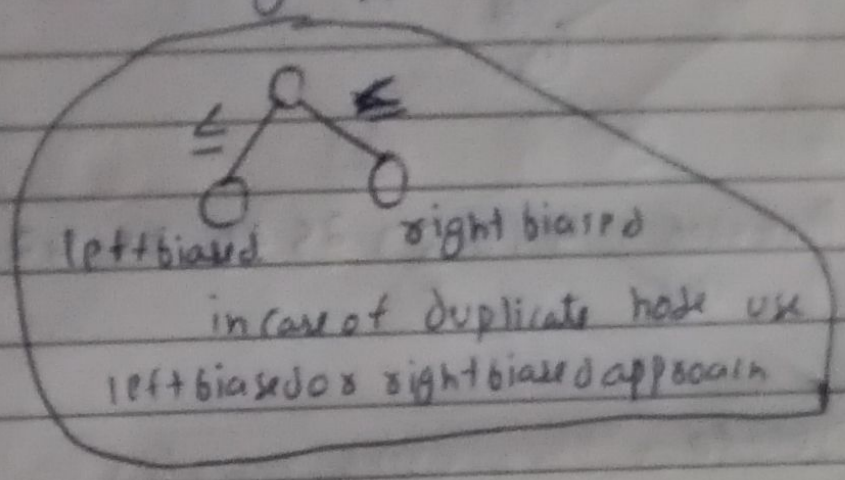
Post (left right root) :- 1 2 4 3 7

In (left root right) = 1 2 7 3 4

```
Preorder Traversal
id preorder(struct Node* root) {
  if (root == NULL)
      return;

  printf("%d ", root->data); // Visit the root
  preorder(root->left);      // Traverse the left subtree
  preorder(root->right);     // Traverse the right subtree


/ Inorder Traversal
oid inorder(struct Node* root) {
   if (root == NULL)
       return;

   inorder(root->left);       // Traverse the left subtree
   printf("%d ", root->data); // Visit the root
   inorder(root->right);      // Traverse the right subtree
}

// Postorder Traversal
void postorder(struct Node* root) {
    if (root == NULL)
        return;

    postorder(root->left);     // Traverse the left subtree
    postorder(root->right);    // Traverse the right subtree
    printf("%d ", root->data); // Visit the root
```

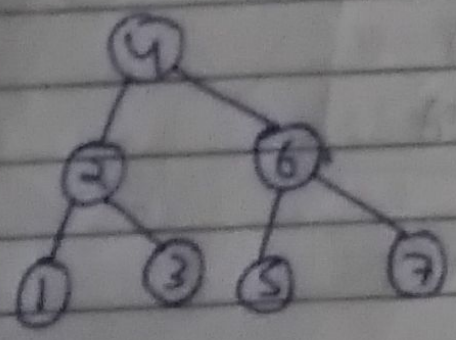
Binary Search Tree (BST)

Property :-
- left subtree of a node contains only nodes with keys lesser than the node's key
- right subtree of a node contains only nodes with keys greater than the node's key.
- left and right subtree each must also be a binary search tree
- There must be no duplicate nodes (BST may have duplicate values with different handling approaches)



Create Binary Search tree

4, 2, 3, 6, 5, 7, 1



(1,2,3) < 4 < (6,5,7)

1 < 2 < 3

5 < 6 < 7

Smaller element existin left

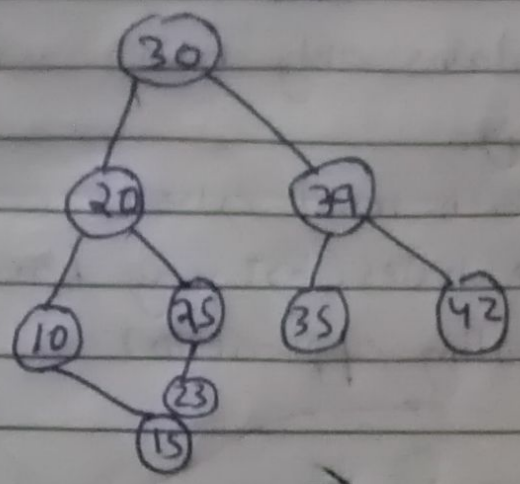
larger element exist in right

Inorder :- 1 2 3 4 5 6 7

- An inorder traversal of a BST produces the key in sorted/ascending order.

Q. Preorder traversal sequence of BST is
30, 20, 10, 15, 25, 23, 39, 35, 42. find Postorder :- ?

Soln Inorder :- 10, 15, 20, 23, 25, 30, 35, 39, 42
left root right

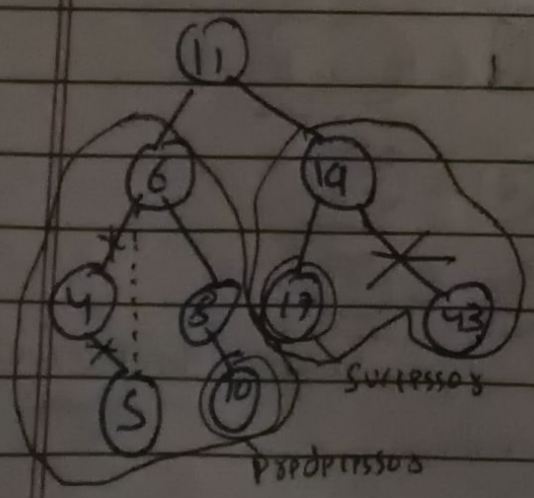


Postorder: 15 10 23 25 20 35 42 39 30

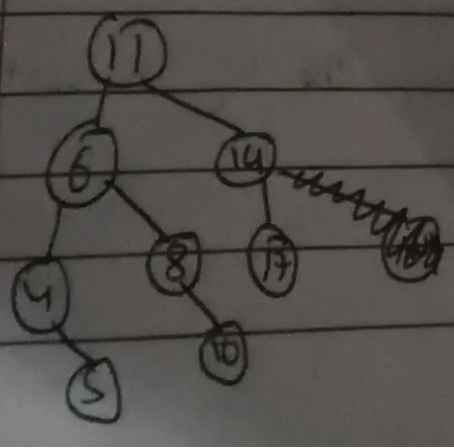
Delete.

11, 6, 8, 19, 4, 10, 5, 17, 43, 49, 3,

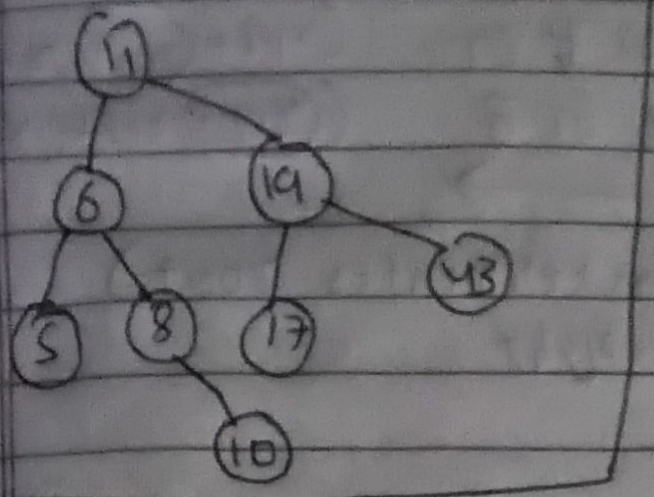
Conditions
i) 0 child
ii) 1 child
iii) 2 child

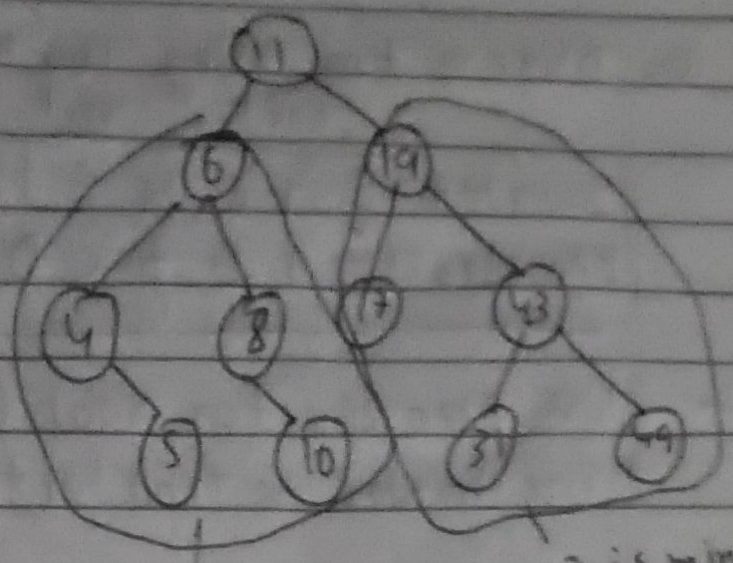


i) remove 43



ii) remove 4





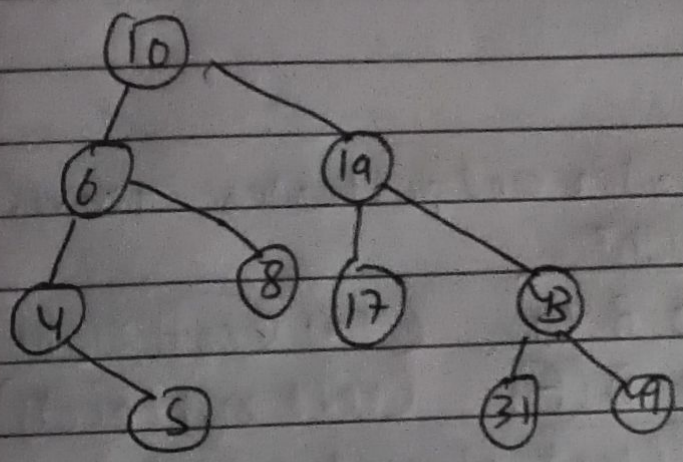
10 is max
∴ predecessor

17 is min
∴ successor

iii) remove 11

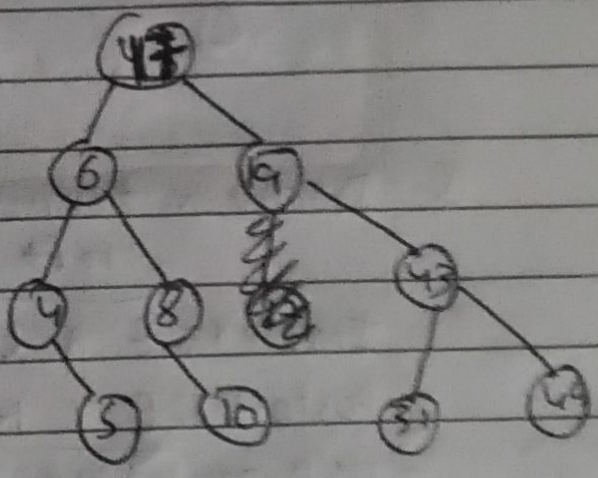
a) inorder predecessor

in left subtree max element



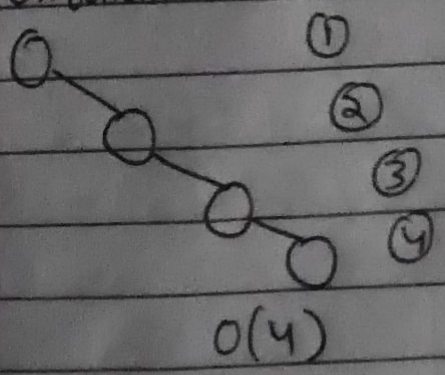
b) inorder successor

in right subtree min element



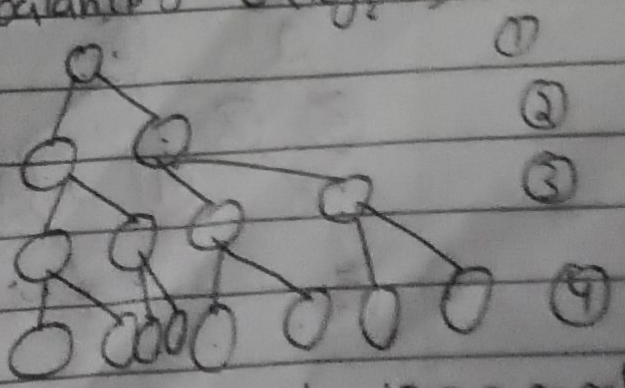
Searching in BST (Time complexity)

Case-I unbalanced O(n)



O(4)

Case-II balanced $O(\log_2 n)$



$\log_2 15 = 3.90 \simeq 4$

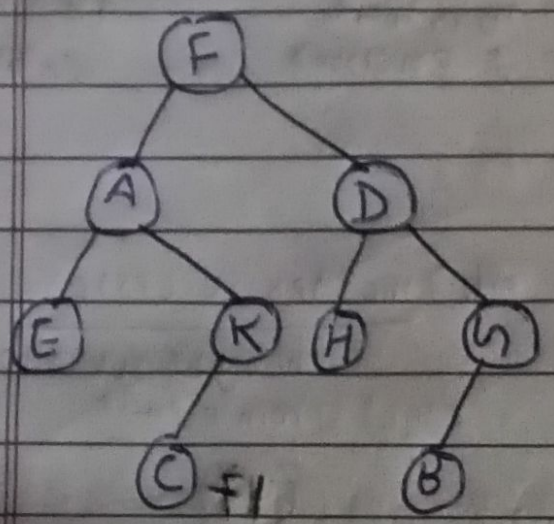
Q. Draw a tree using inorder and preorder traversal.

Left root right

Inorder : E A C K F H D B G (left root right)

Preorder : F A E K C D H G B (root left right)

Soln
- Use Preorder for root (first element is root)
- Use Inorder to find left and right



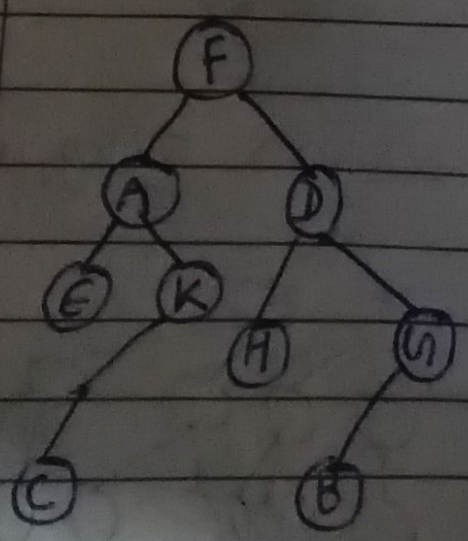
Q. Draw a tree using inorder and postorder traversal.

left right

Inorder : E A C K F H D B G (left root right)

Postorder : E C K A H B G D F (left right root)

Soln
- use postorder for root (last element is root)
- use inorder to find left and right



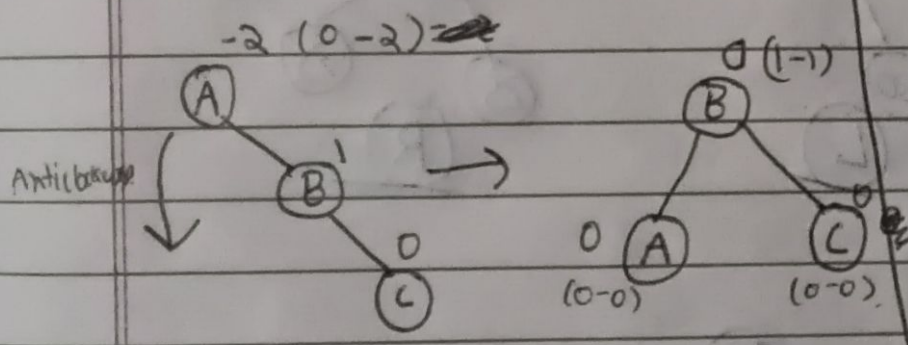
Date : / /
Page No.

AVL Tree (named after Adelson-Velsky and Landis)
It is self-balancing Binary search tree where difference in the heights of the left and right subtrees for any node is at most 1.
This balance is achieved through rotation during insertion and deletion

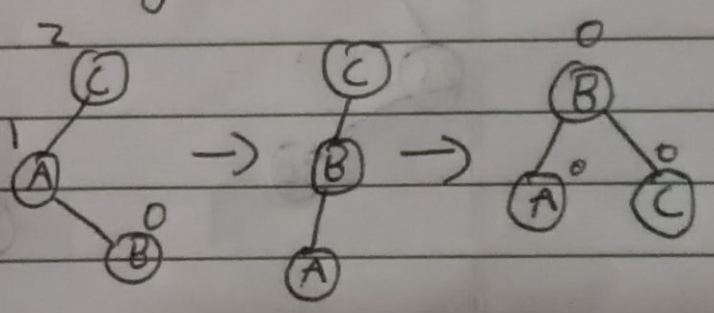
Balanced factor = height of left subtree - height of right subtree
If balance factor is not [-1, 0, 1] then tree is unbalanced

Rotating the subtrees in an AVL Tree
An AVL tree may rotate in one of the following 4 ways to keep itself balanced.

i) Left rotation. (RR rotation

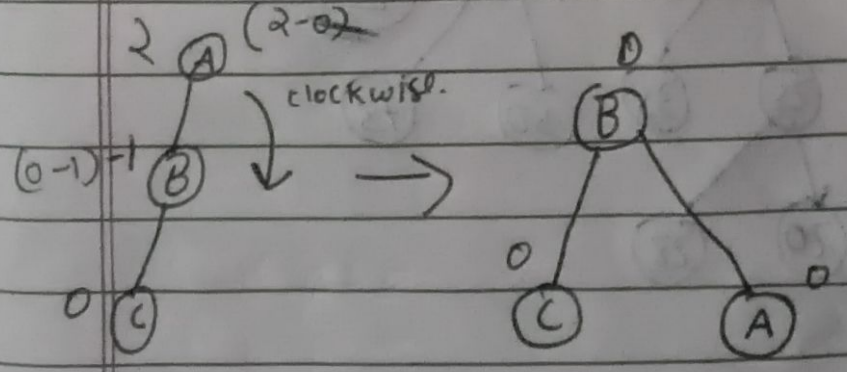


iii) Left-right rotation

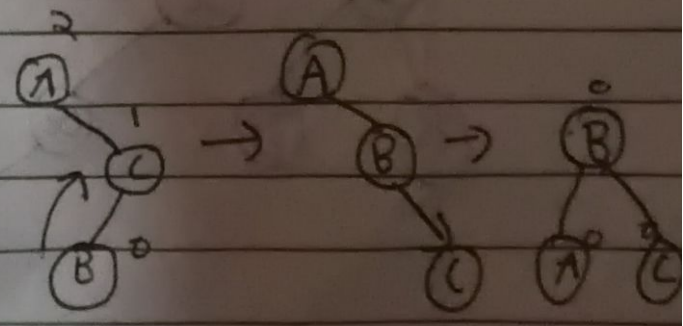


(LR → LL)

ii) right rotation / LL Rotation.



iv) right-left rotation.

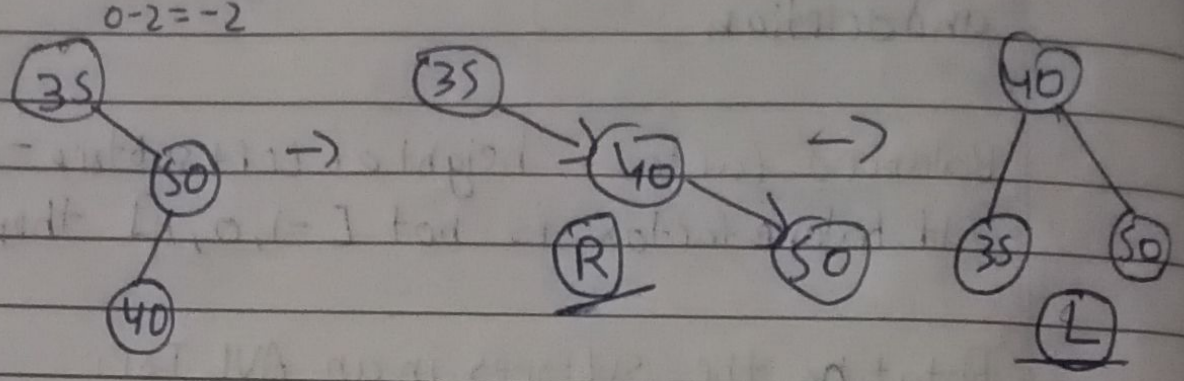


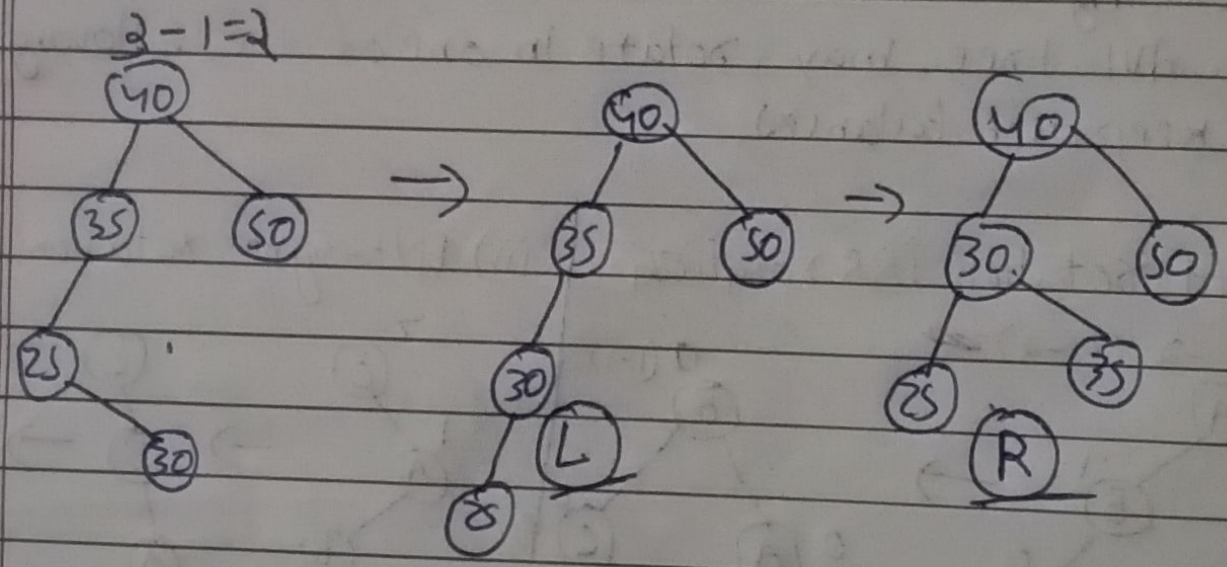
(RL → RR)

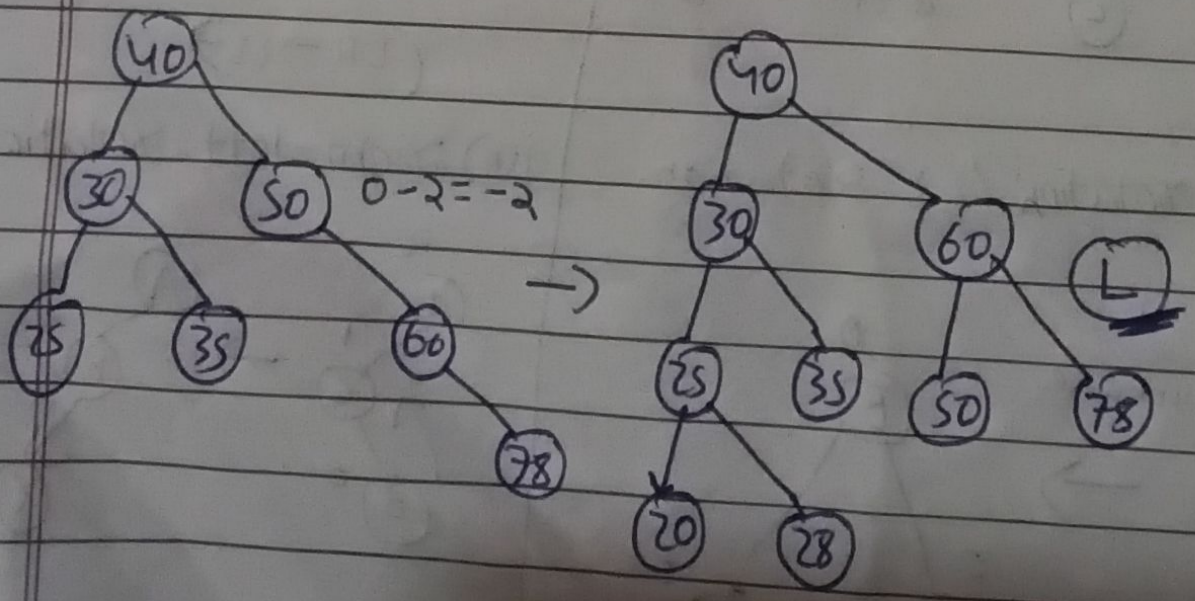
Date : / /
Page No.

Q- How many rotations are required during construction of an AVL tree if the following elements are added in order given

35, 50, 40, 25, 30, 60, 78, 20, 28







3 left, 2 right

Date : / /
Page No.

Graph (Tree is graph but graph (cyclic) is not Tree)

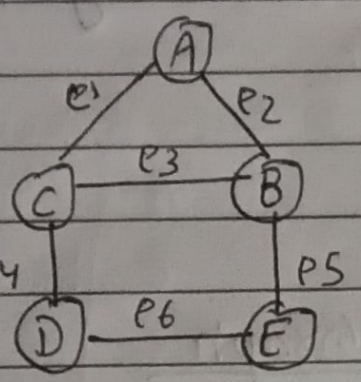
It is non-linear data structure consisting of vertices (nodes) and edges that connect pair of vertices.

G(V, E)
graph vertices edge

V(A, B, C, D, E)

E(e1, e2, e3, e4, e5, e6)

E(A-B, B-C, C-A, D-C, D-E, B-E) (both way to represent edge.)

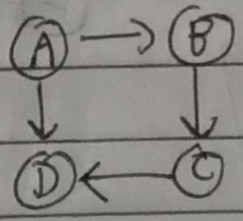


G(5, 6)

Appl^n of Graph.
1) Shortest path in maps.
2) Social Network (facebook, insta)
3) Machine learning.
4) computer vision to detect object and image

Types of Graph

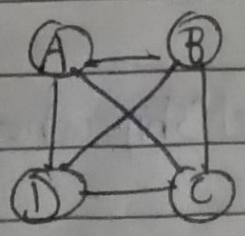
1) Directed graph



(Edges have direction)

It indicate a specific path from one vertex to another vertex.

2) Undirected graph.



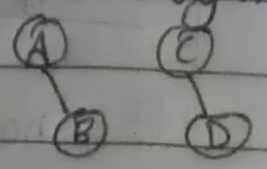
(Edges have no direction)

In this graph

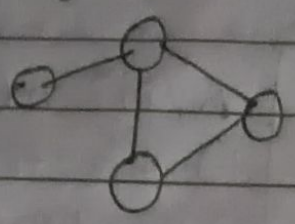
3) Connected graph :- directed and undirected graph are connected graph.

Date : / /
Page No.

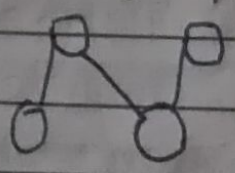
4) Unconnected graph: Where not all vertices are joined by ~~lines~~ {edges}. Some parts are completely separate.



5) Cyclic graph: contains atleast one cycle.

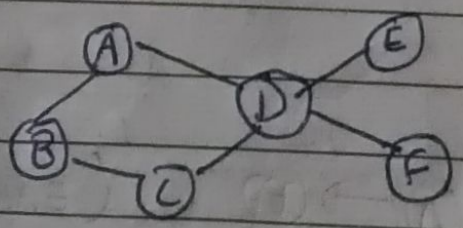


6) Acyclic graph: contains No cycle.



Terminology of Graph

1) PAth:- way to reach initial node to terminal node.



A → F

Path 1) A → D → F

Path 2) A → B → C → D → F

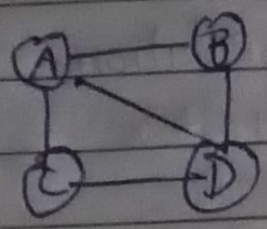
2) Adjacent node :- rel"ship where two vertices are connected by edge.

Adjacent of D : A, C, E, F

~~By Edge~~

Date : / /
Page No.

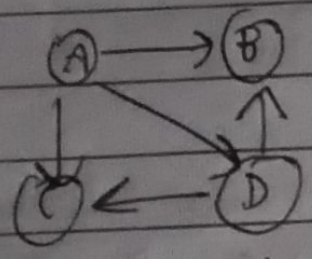
3) Degree : total no of edges connected to nodes



undirected.

Degree (A) = 3 (B, D, C)
Degree (B) = 2 (A, D)

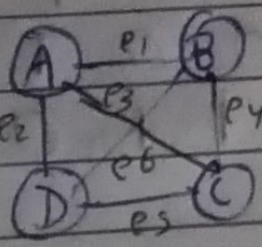
There are two type of degree in directed graph



Directed

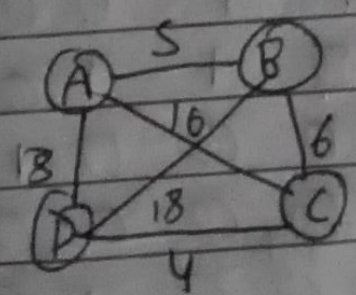
Indegree (A) = 0
Outdegree (A) = 3 (B, D, C)

4) Complete graph :- All nodes of graph are connected to each other.

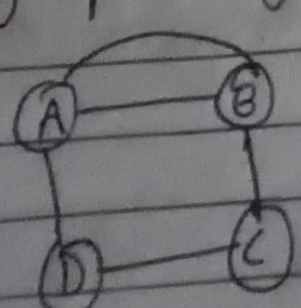


complete graph Edge = $\frac{node(node-1)}{2} = \frac{4^2 \times 3}{2} = 6$

5) Weighted graph :- Value assigned to the edge of graph



6) Multi graph :- graph contain multiple edge b/w two nodes



Date : / /
Page No.

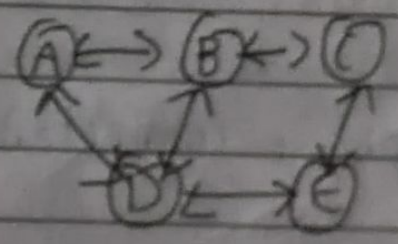
Type of Graph representation

1. Adjacency matrix / sequential representation

$V_{ij} = 1$: edge b/w $V_i \rightarrow V_j$
$V_{ij} = 0$: no edge b/w $V_i \rightarrow V_j$

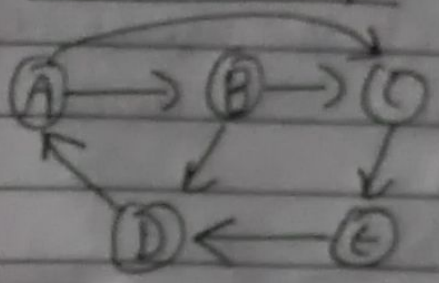
Ex -

Unidirected Graph



(i,j)	A	B	C	D	E
A	0	1	0	1	0
B	1	0	1	1	0
C	0	1	0	0	1
D	1	1	0	0	1
E	0	0	1	1	0

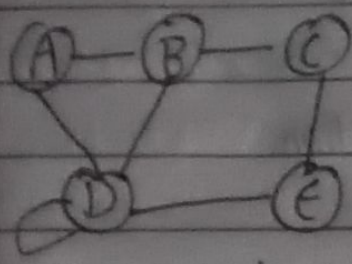
Directed Graph



(i,j)	A	B	C	D	E
A	0	1	1	0	0
B	0	0	1	1	0
C	0	0	0	0	1
D	1	0	0	0	0
E	0	0	0	1	0

Date : / /
Page No.

2. Adjacency Linked representation



Directed graph

vertex	adjacency list
A	B D
B	A C D
C	B E
D	A B E D
E	C D

A → B → D X

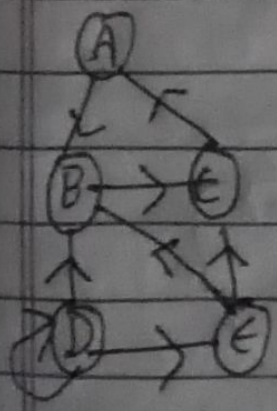
B → A → C → D X

C → B → E X

D → A → B → E → D X (You can see D connected to itself)

E → D → C X

Undirected graph



vertex	adjacency list
A	B
B	C
C	A
D	B D E
E	B C

A → B X

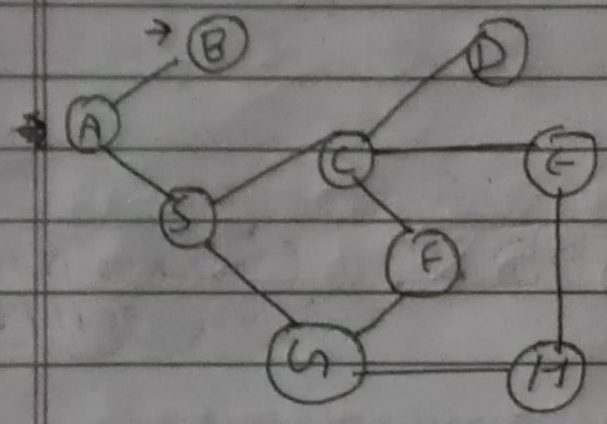
B → C X

C → A X

D → B → D → E X (You can D connected to D)

	bfs	dfs
Traversal	level by level	Path by path
Data structure	Queue	Stack
use case	shortest path	cycle detection
Time comple:	O(V+E)	O(V+E)
	V: vertices, E: edges	

BFS (Breadth first Traversal):



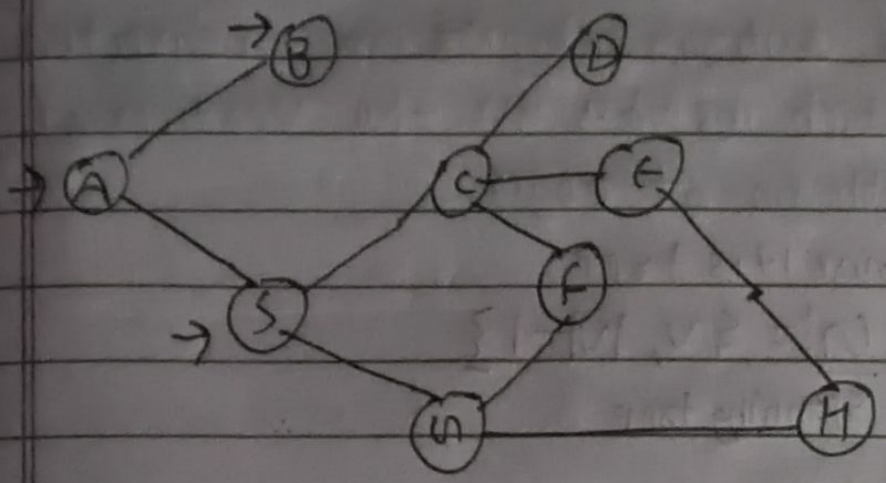
Queue	output
~~B~~	A B S C G D E F H
~~S~~	
~~C~~	
~~G~~	
~~D~~	
~~E~~	
~~F~~	
~~H~~	

queue is Empty Now

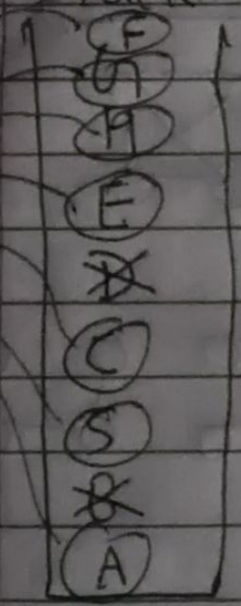
until queue empty
put all adjacent node and remove it from queue and add to string

Date : / /
Page No.

DFS (Depth first Search)



stack



Popout all in end

output

ABSCDEHGF

until

Put in stack and put in stsig too

(D don't have adjacent node more

(B don't have adjacent node more)

Date : / /
Page No.

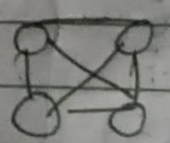
Spanning tree

- Spanning tree is a sub-graph of an undirected connected graph, which includes all the vertices of graph with a minimum possible no. of edges.
- It can't be cyclic because it is tree.

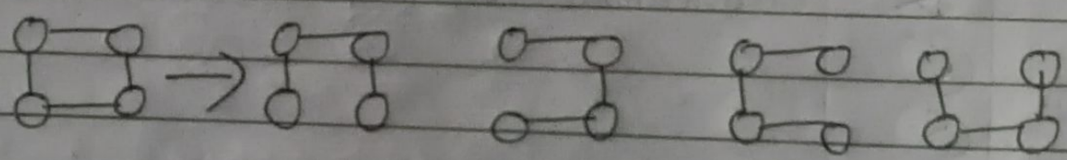
$G = \{V, E\}$ graph $G' = \{V, |V|-1\}$ Spanning tree

- $n^{(n-2)}$ · total no. of spanning tree from complete graph

$4^{(4-2)} = 4^2 = 16$

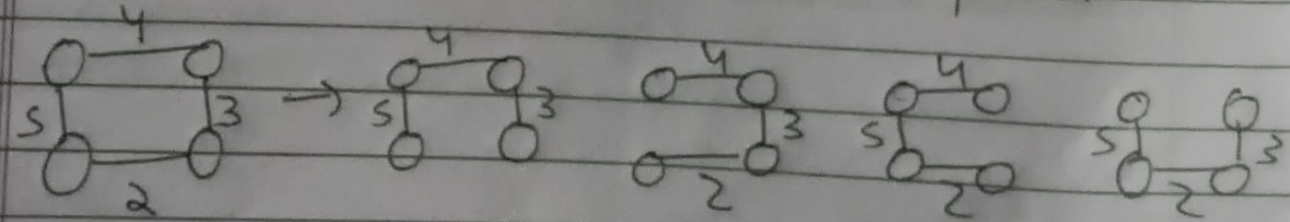


Ex-



Minimum sum spanning tree

Minimum Spanning tree :- Sum of weight of edges is as minimum as possible



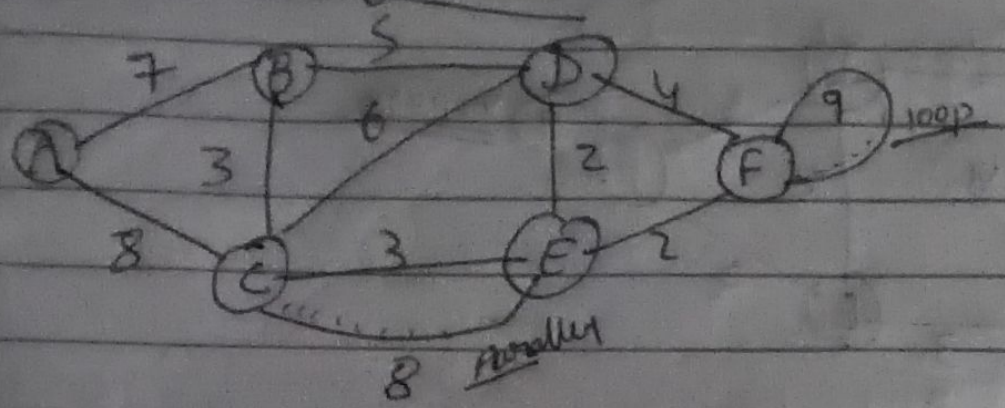
minimum cost spanning tree

- Kruskal's and prim's algo is used to calculate minimum Spanning Tree

Date : / /
Page No.

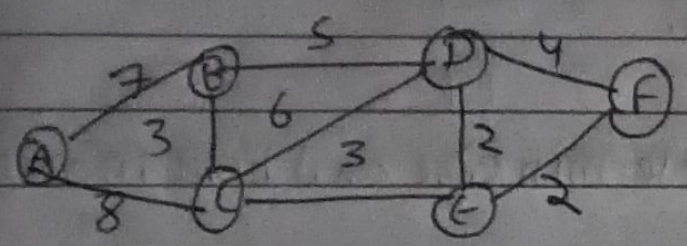
Kruskal's Algorithm

Q.



Sol^n

remove all loop and || edges.



Write in Ascending order weight.

EF - 2

DE - 2

CE - 3

BC - 3

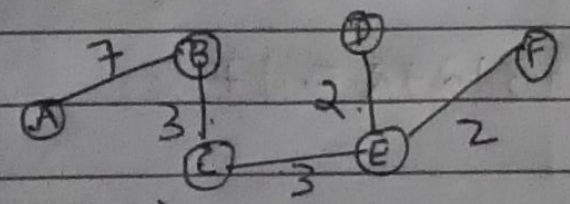
DF - 4 X (else cycle will brcome)

DB - 5 X (else cycle will brcome).

CD - 6 X (else cycle will brcome).

AB - 7

AC - 8 X (else cycle will brcome).



6 vertices, 6-1 edge

6 vertices, 5 edge.

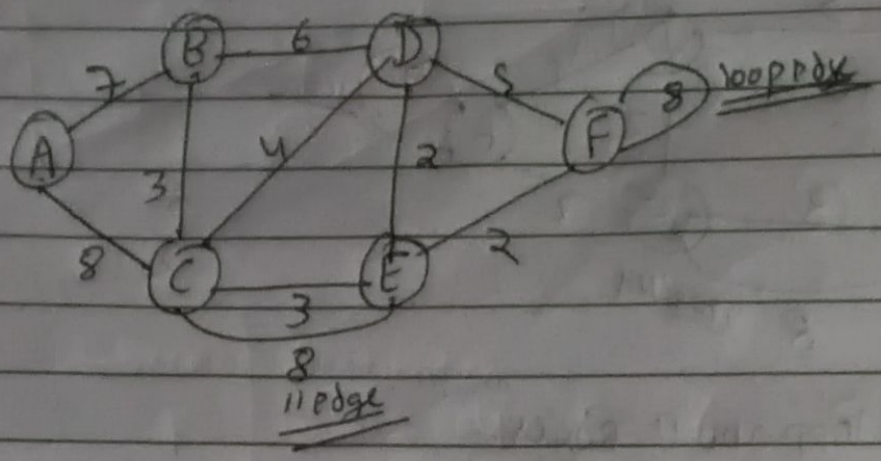
∴ answer is correct

sum = 7+3+3+2+2 = (17) — minimum cost spanning tree

Date : / /
Page No.

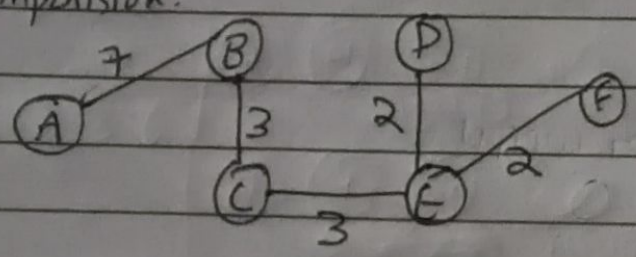
Prim's Algorithm

Q.



Soln Remove loop and 11 edge.

take any node and choose min edge and hold max edge for comparision.



~~8~~ ~~6~~ 4 ~~3~~ ~~2~~ ~~2~~ 5

$7+3+3+2+2 = 17$

Date : / /
Page No.

Dijkstra's algorithm (:- used to find shortest path b/w nodes in a graph)

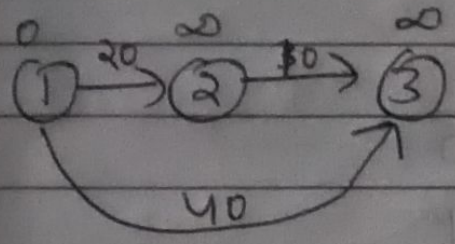
O(ElogV)

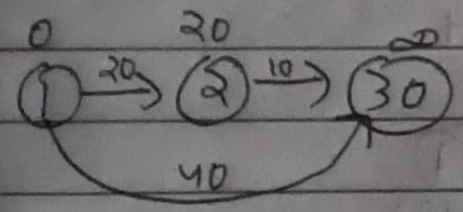
distance of u + cost of u→v < distance of v

If $d(u) + c(u,v) < d(v)$

$d(v) = d(u) + c(u,v)$

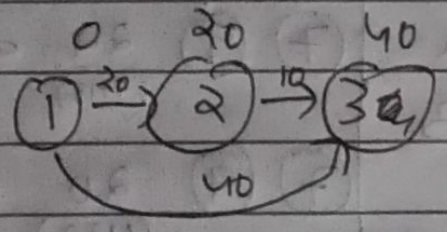
Q.





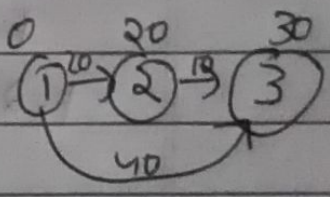
$0 + 20 < \infty$

$d(v) = 20$



$0 + 40 < \infty$

$d(v) = 40$



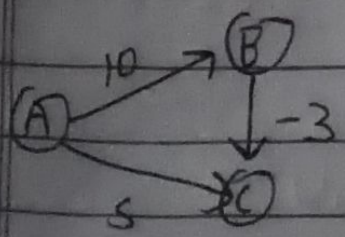
$20 + 10 < 40$

$30 < 40$

$d(v) = 30$

-ve weight Edge

Working.

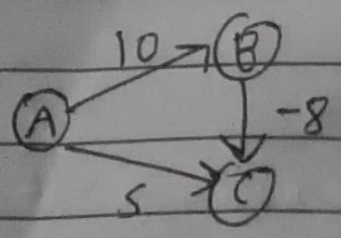


A	B	C
0	∞	∞
	10	5

A, C

A, C, B

Not working.

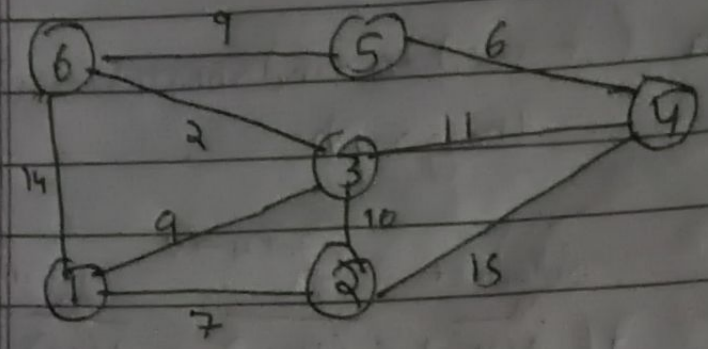


A	B	C
0	∞	∞
	10	5
A, C		
A, C, B	0	5

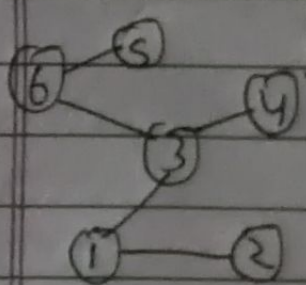
but A→C is 10−8 = 2 which is min so algo not work

Date : / /
Page No.

Q.



Source	Destination				
	2	3	4	5	6
1	∞	∞	∞	∞	∞
~~1~~	(7)	9	∞	∞	14
1,2	(7)	(9)	22 (7+15)	∞	14
1,2,3	(7)	(9)	20 (11+9)	∞	(11) (9+2)
1236	(7)	(9)	(20)	20 (11+6)	(11)
123654	(7)	(9)	(20)	(20)	(11)



Date : / /
Page No.

1. Insertion sort :- insert each element into its correct place in the sorted part of the list.

	sorted array	reverse
best	$O(n)$	$O(n^2)$
(best)	(worst time)	
		$O(1)$ (space)

temp => 0 add
-> data

Heap is a complete binary tree data structure that satisfies the heap property. It is used to implement priority queues.

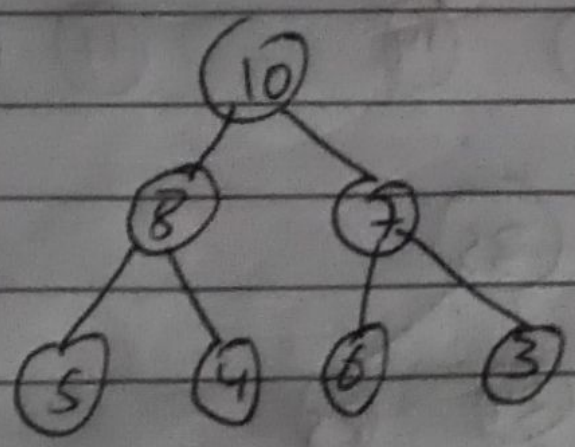
Date: / /
Page No.

Heap tree :

Heap tree condition :-

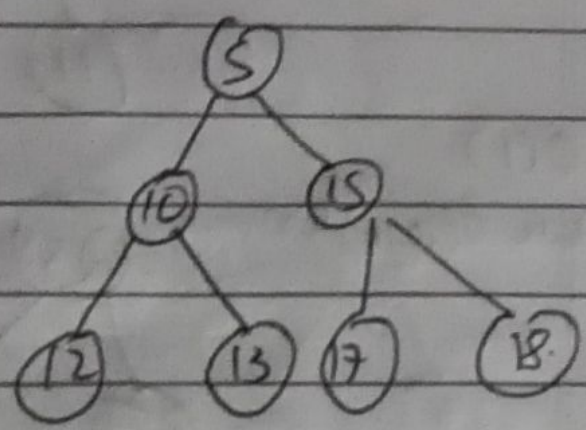
Condition :- 1) structural Property. (∧CBT - almost complete binary tree)
2) ordering Property
- max heap (parent > child)
- min heap (parent < child)

Max Heap



10 > 8, 7 > 5, 4, 6, 3

Min heap



5 < 10, 15 < 12, 13, 17, 18

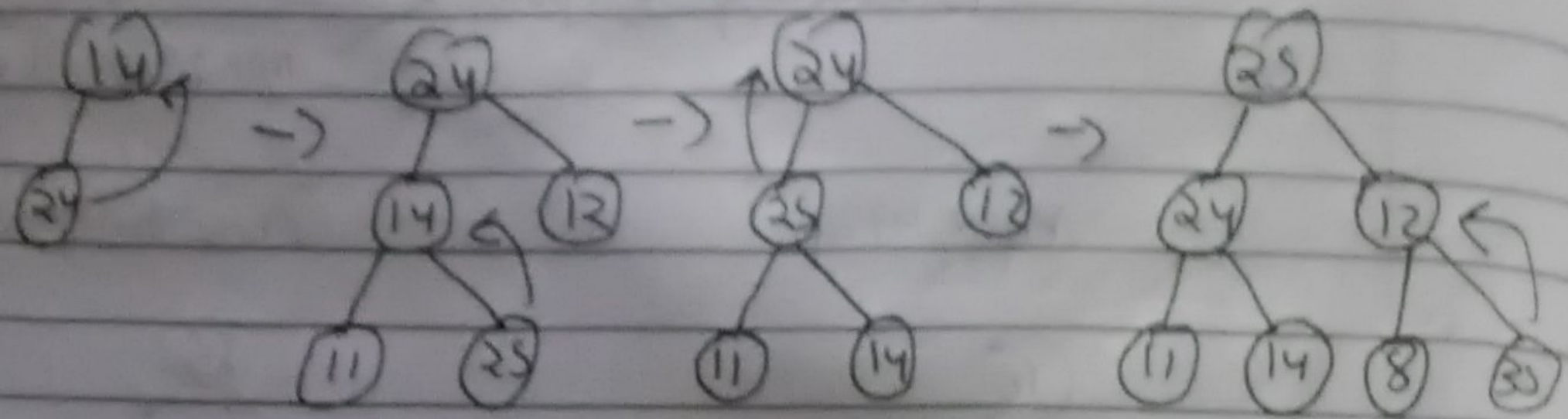
Heap tree Construction / Insertion

- Insert key one by one in given order — O(n log n)
- Heapify method O(n)

MaxHeap :-

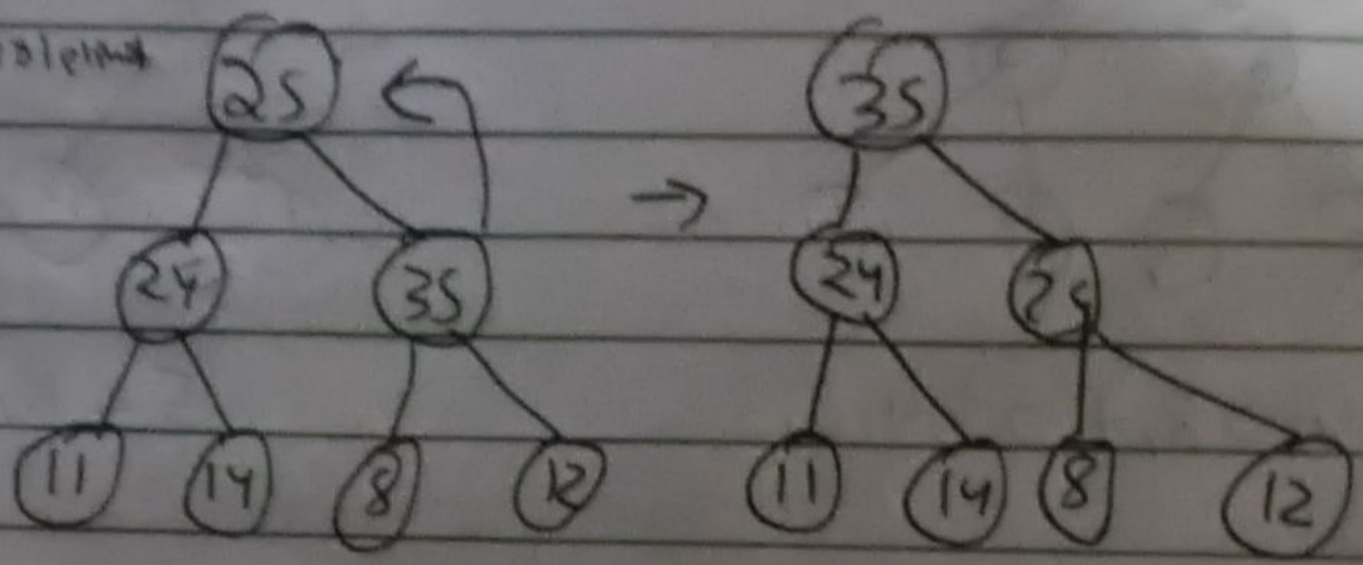
1) one by one Insert

14, 24, 12, 11, 25, 8, 35



a) take O(1)

b) Parent take O(log n) to replace

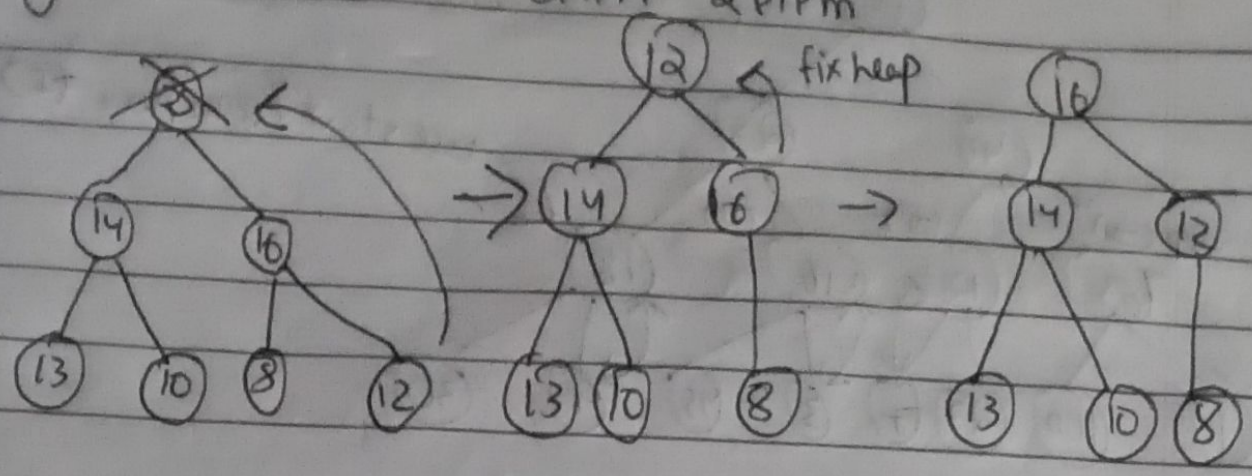


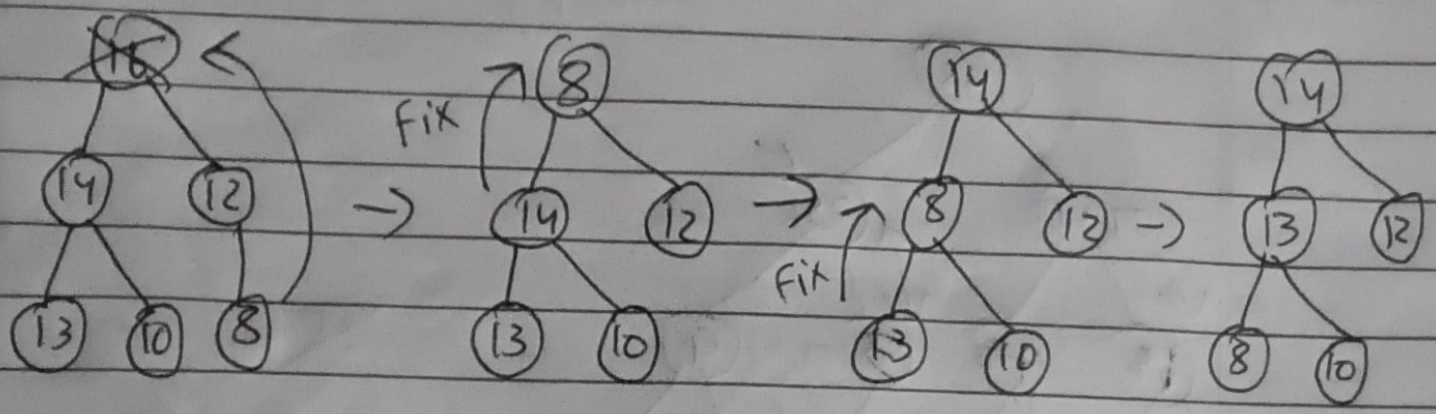
- insert key and compare with parent to follow heap property
- construct it in complete binary tree form

Date : / /
Page No.

from root

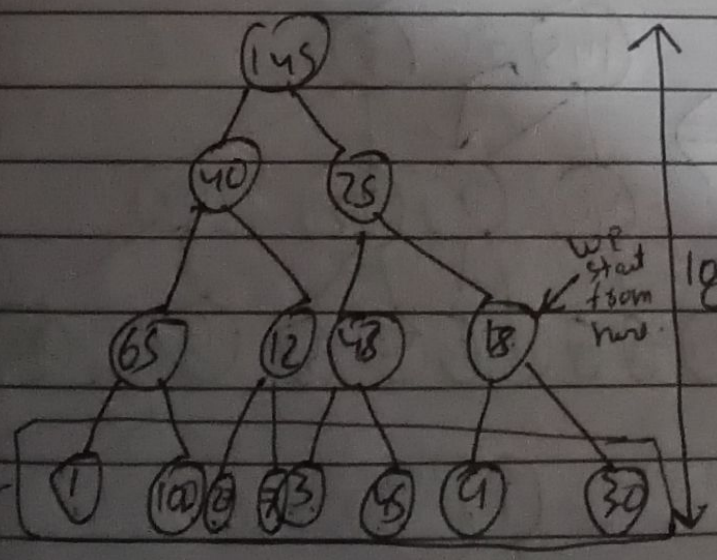
one by one delete :- delete zero





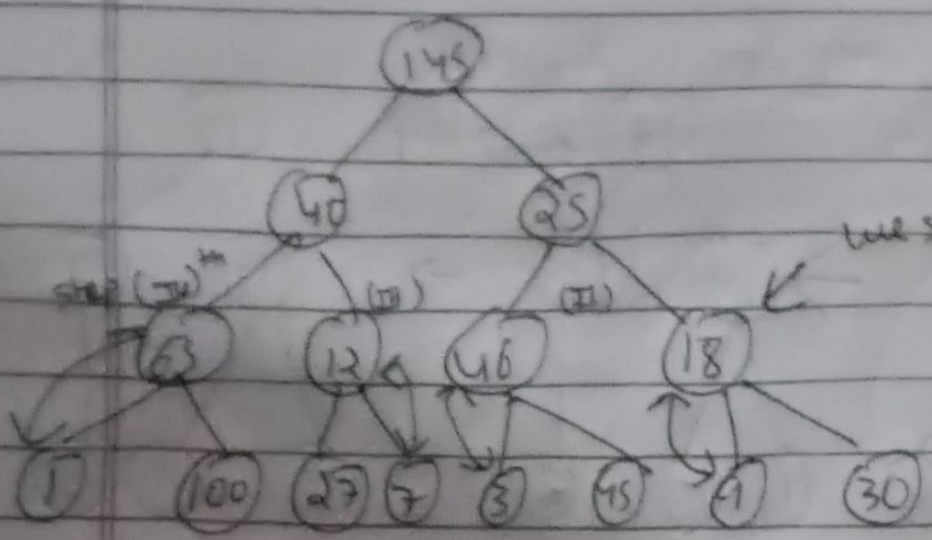
Heapify Method

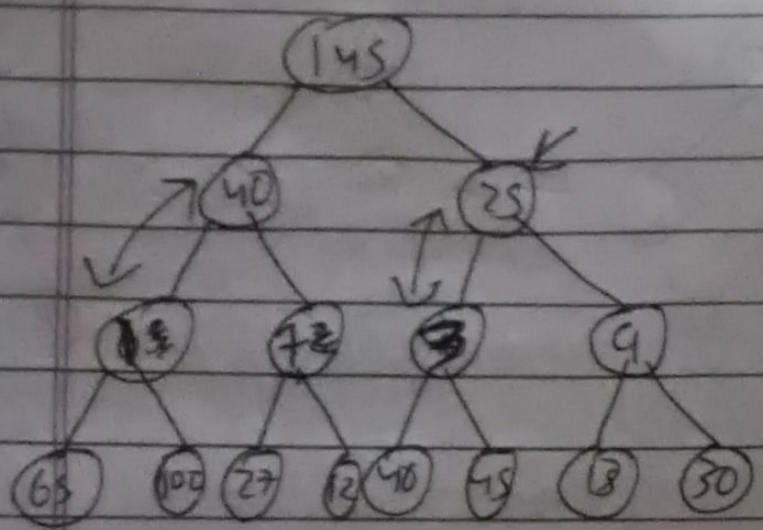
145, 40, 25, 65, 12, 48, 18, 1, 100, 27, 7, 3, 45, 9, 30.

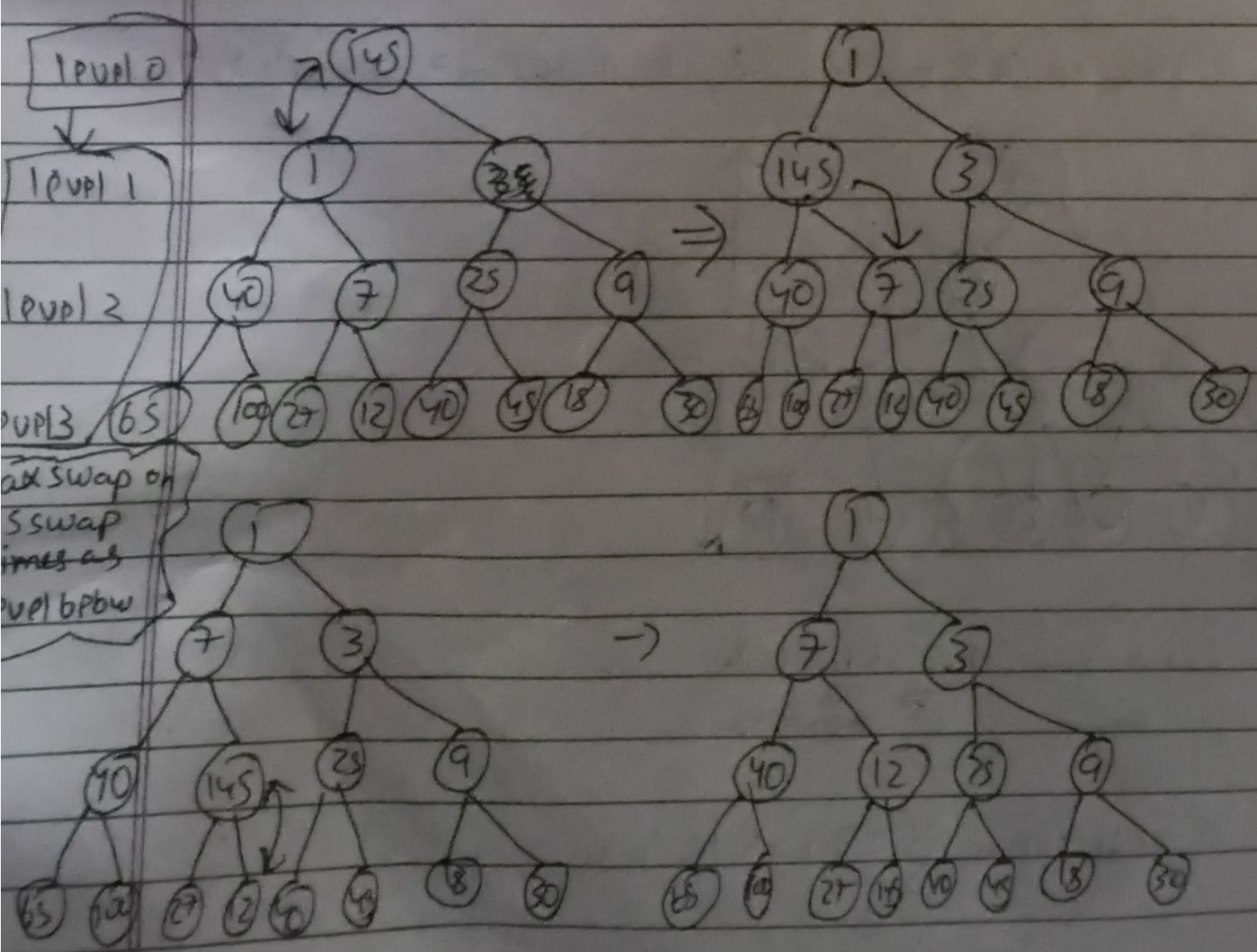


for n node there is n leaf node.

IS Node mean $\frac{15}{2} = 7.5 \simeq 8$ leaf node.

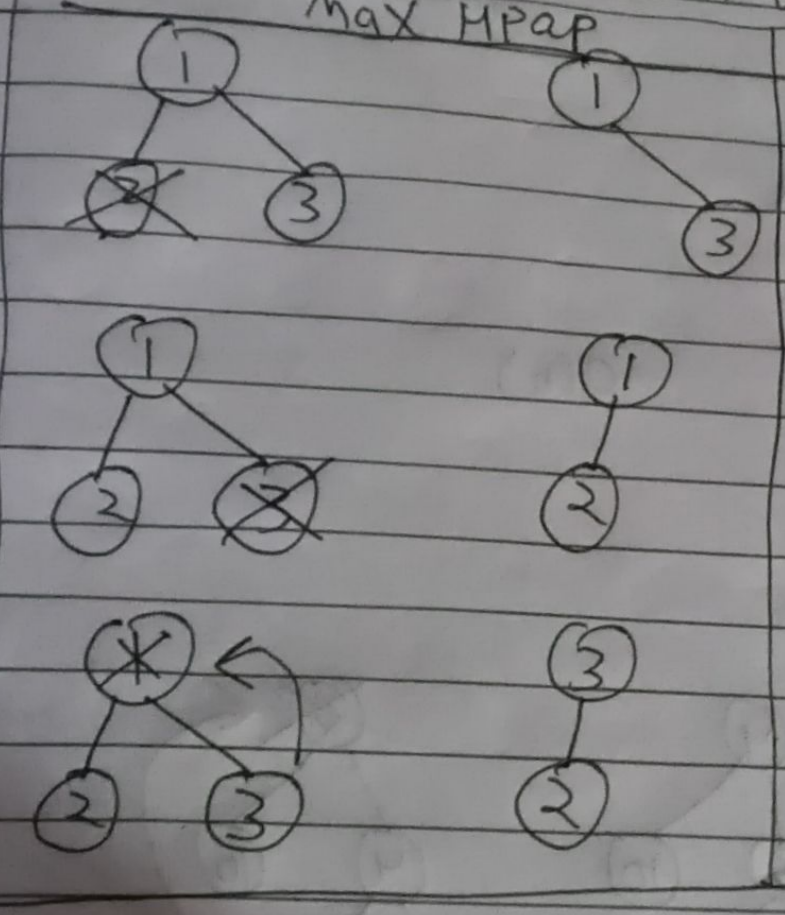

145
40
25
63
12
46
18
1
100
27
7
3
45
9
30


145
40
25
63
72
3
9
65
100
27
12
40
45
18
30


level 0
level 1
level 2
level 3
145
1
35
40
7
25
9
65
100
27
12
40
45
18
30
1
145
3
40
7
25
9
1
7
3
10
145
25
9
65
100
27
12
40
45
18
30
1
7
3
40
12
25
9

Date : / /
Page No.

Deletion in Heap (only root or rightmost element possible otherwise it won't be Almost complet binary tree



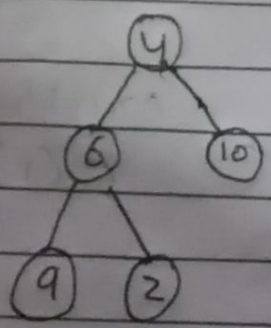
Date : / /
Page No.

Heap sort (inplace, Unstable, O(nlogn))

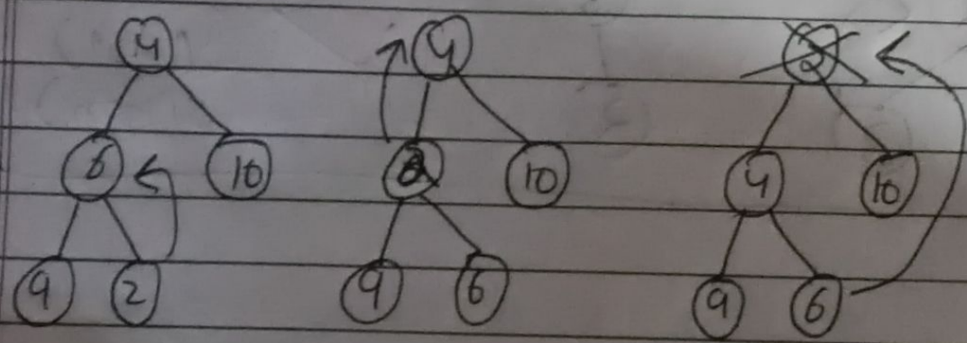
no extra space is used

~~3b 4 3a~~ instead of ~~3a 4 3b~~

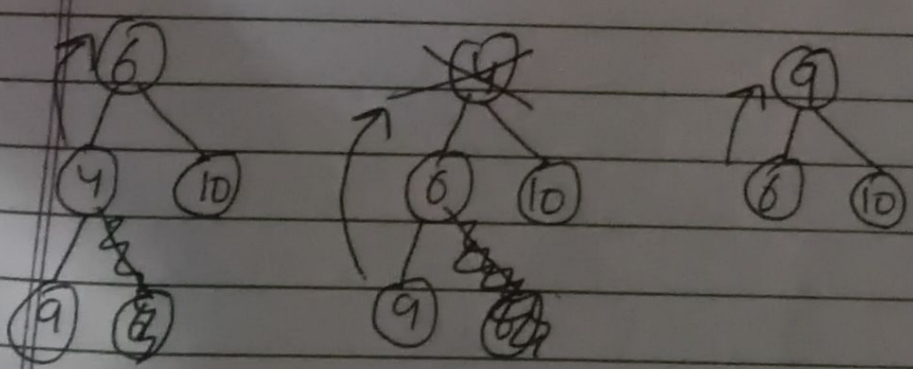
4 6 10 9 2



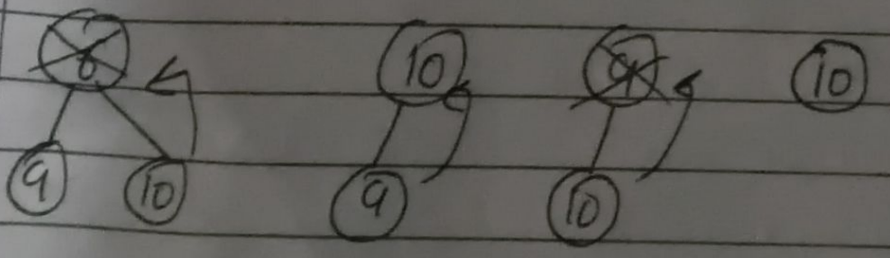
O(n)



2



4



6 9 10

∴ 2 4 6 9 10

Date: / /
Page No.

Priority Queue

~~Max Priority Queue~~
~~Min Priority Queue~~.

- It is an abstract data type similar to regular queue, but each element is assigned a priority.
- Elements are dequeued based on their priority rather than the order in which they were enqued.
- If two element have the same priority, they are served according to their insertion order.

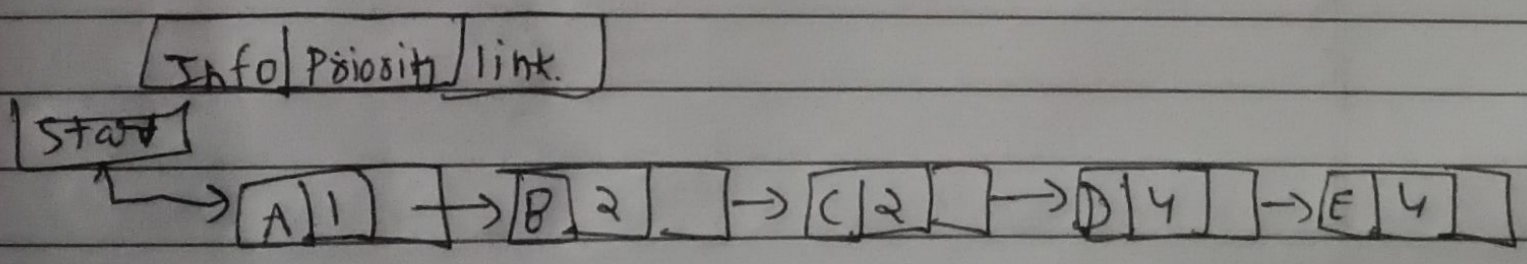
Type of Priority Queue.

1- Max-Priority Queue:- element with highest priority is dequed first.
2- Min-Priority Queue:- element with lowest priority is dequed first.

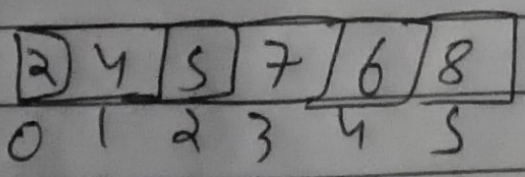
Abstract data type:- datastructure ~~are~~ merged with operations.

Data declaration — Stack
ADT ← operation declaration — Push(9)

Implementation of Priority Queue using linked list

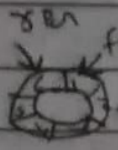


Implementation of Priority Queue using Array

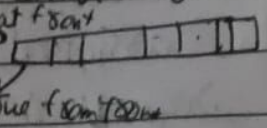


while inserting insert at correct position so that while deletion it take O(1)

Type of Queue.



- Simple Queue.
- Circular Queue - last position is connected back to first position
- Priority Queue
- Double Ended Queue (Deque) :- can be added & removed from both the front and rear.



Disadvantage of Queue / Stack

- sequential access ✗
- memory wastage ✓, overflow and underflow issue ✓ (In Array.)
- limited operation

	HashMap	Hashtable
Synchronized	No	Yes
Thread-Safe	No	Yes
Null Keys and Null values	One null key ,Any null values	Not permit null keys and values
[redacted]		
Performance	Fast	Slow in comparison
[redacted]		

Max no of edge min no of edg
Dense Graph Sparse Graph

	Array		stack / linked list		Binary search tree	Hashing	
Insert	O(1)	O(n)	O(1)	O(1)	O(log n) O(n)	O(1) O(n)	O(1)
delete	O(1)	O(n)	O(1)	O(n)	O(log n) O(n)	O(n)	O(1)
search	O(n)				O(log n) O(n).	O(n)	O(1)

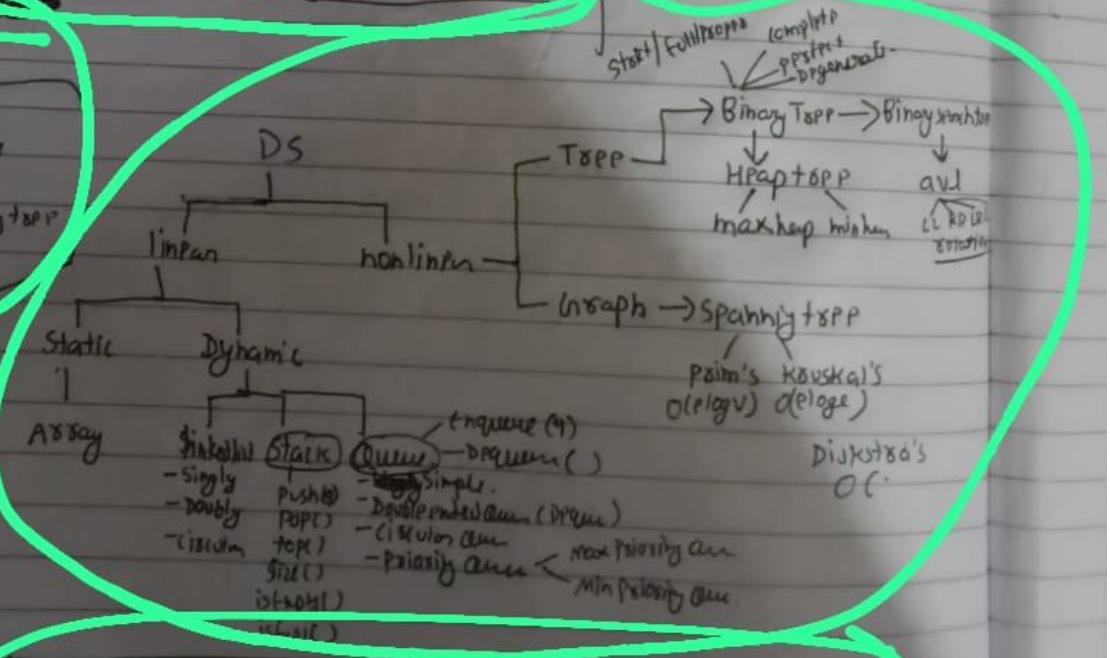
Sorting	time	space
Insertion sort	O(n) O(n²)	O(1)
Selection Sort	O(n²)	O(1)
bubble sort	O(n²)	O(1)
Merge sort	O(nlogn)	O(n)
Quick sort	O(nlogn)	O(logn)
heap sort	O(nlogn)	O(1)

Searching	
BFS	O(V+E)
DFS	O(V+E)
LoS	O(n)
BoS	O(logn)

$\frac{n(n-1)}{2}$ total edge

$n^{(n-2)}$ total spanning tree

$\frac{n}{2}$ leaf node



Hashing : collision

Hashtable | Hashset | Hashmap

Separate chaining open hashing
- Sphashing

open addressing / close hashing
- linear Probing
- quadratic Probing
- double hashing

Breadth-First Search (BFS) - Pseudocode

plaintext Copy code

```
BFS(graph, start):
    Initialize an empty queue Q
    Initialize an empty set VISITED to keep track of visited nodes
    Enqueue the START node into Q
    Add START to VISITED

    WHILE Q is not empty:
        node = Q.dequeue()  // Remove and return the front of the queue
        PROCESS node        // Perform the desired operation (e.g., print)

        FOR each neighbor of node in graph:
            IF neighbor is not in VISITED:
                Mark neighbor as visited
                Enqueue neighbor into Q
```

Depth-First Search (DFS) - Pseudocode

Recursive Approach:

plaintext Copy code

```
DFS(graph, node, visited):
    IF node is not in VISITED:
        Mark node as visited
        PROCESS node        // Perform the desired operation (e.g., print)

        FOR each neighbor of node in graph:
            DFS(graph, neighbor, visited)
```

1. Prim's Algorithm (Minimum Spanning Tree)

c Copy code

```
// Simplified Prim's Algorithm Pseudocode

function prim(graph, start_vertex):
    initialize visited[] = {false} for all vertices
    initialize key[] = {infinity} for all vertices
    set key[start_vertex] = 0
    for i = 0 to number of vertices:
        u = vertex with minimum key[u] and not visited
        mark u as visited

        for each adjacent vertex v of u:
            if v is not visited and weight(u, v) < key[v]:
                key[v] = weight(u, v)
    return key[]
```

2. Kruskal's Algorithm (Minimum Spanning Tree)

c Copy code

```
// Simplified Kruskal's Algorithm Pseudocode

function kruskal(graph):
    sort edges by weight
    initialize disjoint-set (union-find structure)
    initialize mst as empty list

    for each edge (u, v):
        if u and v are in different sets:
            add edge (u, v) to mst
            union(u, v)

    return mst
```

3. Dijkstra's Algorithm (Shortest Path)

c Copy code

```
// Simplified Dijkstra's Algorithm Pseudocode

function dijkstra(graph, start_vertex):
    initialize dist[] = {infinity} for all vertices
    dist[start_vertex] = 0
    initialize visited[] = {false} for all vertices

    for i = 0 to number of vertices:
        u = vertex with minimum dist[u] and not visited
        mark u as visited

        for each adjacent vertex v of u:
            if dist[u] + weight(u, v) < dist[v]:
                dist[v] = dist[u] + weight(u, v)

    return dist
```