Expt :- 01

1] Find Avg of first n Num

2] Find first Repeating element of Array

3] find the greatest & Smallest in Array write

4] Squaring odd Position element.

5] Display for loop Pattern

```
*
# #
* * *
# # # #
```

5] →
```
#include <stdio.h>
int main() {
    int n, i;
    float sum = 0.0, avg, num;
    printf("Enter the Number of element");
    if (scanf("%d", &n) != 1 || n <= 0) {
printf("Invalid input! Number of element
Must be a positive integer.\n");
    return 1;
}
for (i=0; i<n; i++) {
    printf("Enter Number %d:", i+1);
    if (scanf("%f", &num) != 1) {
printf("Invalid input! Please enter
valid number.\n");
    return 1;
```

```
    sum += num;
}
Avg = sum/n;
printf("Avg = %.2f\n", Avg);
return 0;
}
```

Output :-

```
Enter the Number of elements: 4
Enter Number 1: 7
Enter Number 2: 3
Enter Number 3: 5
Enter number 4: 8
Avg = 5.75
```

11 → Code :-

```
#include <stdio.h>
int main() {
int n, i;
float sum = 0.0, avg, num;
printf("Enter the Number of elements: ");
if (scanf("%d", &n) != 1 || n <= 0) {
printf("Invalid input! number of element
Must be a positive integer.\n");
return 1;
}
sum = num;
printf("Avrag = %.2f\n", avg);
return 0;
}
```

Output :- Enter the value of n: 13
the Avg of first 13 numbers
is 7.00

12 → Code :-

```
#include <stdio.h>
int main() {
int arr[6] = {2,4,5,7,4};
int i, j;
for (i = 0; i < 6; i++) {
for (j = i + 1; j < 6; j++) {
if (arr[i] == arr[j]) {
printf("first repeating Number is
%d\n", arr[i]);
return 0;
}
}
}
printf("Repeating Number found.\n");
return 0;
}
```

Output :-

```
First repeating Number is: 4
```

3)

```
#include <stdio.h>
int main () {
int arr [5] = {10, 4, 56, 3, 89};
int i;
int big = arr [0];
int small = arr [0];
for(i=0; i<5; i++) {
if (arr [i] > big)
big = arr[i];
if (arr [i] < small)
small = arr [i];
}
Printf (" greatest no = %d \n" big);
Printf (" Smallest no = %d \n, small);
return 0;
}
```

Output :- greatest Number = 89
Smallest Number = 3

4) code :-

```
#include <stdio.h>
int main ()
int arr [ ] = {1, 2, 3, 4, 5, 6};
int n = Sizeof (arr) / Sizeof (arr[0]);
int n = Sizeof (arr) / Sizeof (arr (0));
Printf (" original array : \n");
for (int i = 0; i < n; i++) {
Print ("%d", arr [i]);
}
for (int i = 1; i < n; i += 2) {
arr [i] = arr [i] * arr [i];
}
Printf (" \n\n Array after Squaring
elements at odd Positions \n")
for (int i = 0; i < n; i++) {
Printf ("%d", arr [i]);
}
return 0;
}
```

output :-

Original array :-

1 2 3 4 5 6 7

Array after Squaring elements at odd Positions :-

1 4 3 16 5 36 7

~~To Sep~~

Expt:- 02

1]

```
#include <stdio.h>
int main (){
int a[] = {56,36,89,57,01,00,67
          59};
 int k, K, i, found = 0;
Printf ("Enter Key to be Searched
Scanf ("%d", &K);
 for(i=0; i<8; i++)
{
 if (a[i] == k)
  { Printf ("\nK FOUND AT INDEX:
          %d", i);
    found = 1;
    break;
  }
}
if(found == 0)
Printf("\nKEY NOT FOND IN
        ARRAY");
   return 0;
}
```

2]

```
#include <stdio.h>
   int main()
   {
 int A[100], n, m, l, h, i
Printf ("Enter number of Array elements
  : ");
Scanf("%d", n);
 Printf ("Enter Key to be Searched:")
  Scanf("%d", K);
  Printf("\n Enter Sorted Array
          elements: \n");
 for (i=0; i<n; i++)
   {
 Scanf("%d", a[i]);
}
l = 0; h = (n-1);
  while (n >= 1)
   {
   m = (int) ((l+n)/2);
    if (h == a[m])
{ Printf ("\n element found at index
      : %d", m);
  break;
}
 else
{
if (h > a[m])
 {
 l = m+1;
}
```

```
if(n < a[m]);
{
h = m + ;
}
}
}
if(n < l)
Printf("\n Element not found);
return 0;
}
```

Output :-

```
Enter Number of Array elements: 6
Enter key to be Searched: 23
 23
4 5
6 7
5 6
7 8
9 8
```

Q3] Difference bet^n linear search & binary Search.

linear Search	Binary Search
i] Time complexity is low	Time complexity is O
ii] works on both Sorted & Converted	only works on sorted data
iii] It is less efficient, especially for large data sets	It is more efficient especially for large data sets
iv] Code is Simpler	code is complex
v] It use Sequenti Searching Approach	It uses divide & Conquer Approach

(4) State limitation of linear Search interms of time complexity.

⇒ A linear Search runs in an worst linear time & makes at most Comparision where n is the length of the list. linear Search is ravely practical because other Search Algorithm & Schemes Such as binary search Algorith etc.

24/7/25

Expt:- 03

1Q Sort element
Ascending order Using bubble.

```
#include <stdio.h>
int main () {
int a [100], n, i, j, x;
Printf ("Enter num. of element in Array");
Scanf ("%d", &n");
Printf ("Enter Array elemen:\n");
for(i=0; i<n; i++) {
   Scanf("%d", &a[i]);
}
for(i=0; i<n-1; i++);
{
for(j=0; j<n+i; i++)
{
if (a[i] > a[j+1]) {
   x = a[j];
      a[j] = a[j+i];
      a[j+1] = x
   }
}
}
return 0;
}
```

Output:-

```
Enter number of elements
 is Array: 4
Enter Array elements
 88
 66
 44
 22

Sorted Arrayin Ascending
 22
 44
 66
 88
```

2] Sort element Descending order Using Selection

```
#include <stdio.h>
int main() {
  int a[100], n, i, j, t;
Printf("Enter no of elements in Array");
Scanf("%d", &n);
Printf("Enter array elemen:\n");
 for(i=0; i<n; i++) {
  {
  if(a[i] < a[i])
   {
    t = a[i];
    a[i] = a[i];
   a[j] = t;
  }
 }
}
Printf("\n Sorted array element in
   descending order:\n");
   for(i=0; i<n; i++) {
 Printf("%d\n", a[i]);
   }
   return 0;
  }
```

Output

```
Enter Number of elements in Array: 4
Enter Array elements:
22
44
66
88
Sorted Array elements in descending
88
66
44
22
```

Q3] Find Number of Compare Require in bubble Sort Method & show the following list having 5 No. 100, 200, 300, 400, 500 ?

100 200 300 400 500

100 x 200 300 400 500

100 200 x 300 x 400 500

100 200 300 400 x 500

Algorithm.

1] Start
2] Compare each element & swap if the first element is greater than second element
3] After each full Pass largest Unsorted element moves to the end of Array
5] Stop.

Q 4]

Pass I,

500	-20	30	14	50
-20	500	30	14	50
-20	500	30	14	50
-20	500	30	14	50

Pass II,

-20	500	30	14	50
-20	30	500	14	50
-20	14	500	20	50

Pass III,

-20	14	500	30	50
-20	14	30	500	50

Pass IV)

-20	14	30	500	50
-20	14	30	50	500

Expt:- 04

1] Sort elements in a Ascending order using insertion sort?

→
```
#include <stdio.h>
int main()
{
int a[100], n, temp, i, j, k;
Printf ("Enter no of elements in Array:\n");
Scanf ("%d" &n);
Printf ("\n Enter elements of Array:\n");
for (i = 0; i<n; i++)
{
Scanf ("%d", & a[i]);
}
for (i=1; i<n; i++)
{
For (j=0; j<i; j++)
{
if (a[i] <a[j])
{
temp = a[i]
For (K= i-1; k>=j; K-1)
{
a[K+1] =a[k];
}
a[j] = temp;
}
}
```

```
Printf ("\n Array after Pass-1.d:", P);
{
Printf ("-1.d", a[i]);
}
++P;
}
Printf ("\n\n the Sorted elementsare:");
for (i= 0; i<n; i++)
{
Printf ("\n -1.d", a[i]);
}
return 0;
```

Output:-

```
Enter Number of elements in Array:
5
Enter elements of Away:
9000
569
6
8589
3456

The sorted elements are:-
6
569
3456
8589
9000
```

2] What is the output of insertion sort after the 2nd iteration given Sequence of NO :- 7, 3, 5, 1, 9, 8, 4, 6

→ 1st interaction :-

7 3 1 9 4 8 6

3 7 1 9 4 8 6

2nd Interaction :-

3 7 1 9 4 8 6

1 3 7 9 4 8 6

∴ After 2nd interaction the result is

→ 1, 3, 7, 9, 4, 8, 6

③ Sort the following Number Using Radix Sort :-

① 100 225 390 4130 956 9[illegible]

Pass 1	0	1	2	3	4	5	6	7	8	9
0100	0100									
0225						0225				
0390	0390									
4130	4130									
0956							0956			[illegible]
0099										099
5431		5431								

Pass 2	0	1	2	3	4	5	6	7	8	9
0100	0100									
0390										039
4130				4130						
5431		5431		[illegible]						
0225						0225				
0956						0956				[illegible]
0099										0

Pass 3	0	1	2	3	4	5	6	7	8
0100		0100							
0225			0225						
4130		4130							
5431					5431				
0456									
0390				0390					
0099	0099								

Pass 4	0	1	2	3	4	5	6	7	8	9
0099	0099									
0100	0100									
4130					4130					
0225	0225									
0390	0390									
5431						5431				
0956	0956									

2] 25, 6, 99, 145, 239, 20, 18

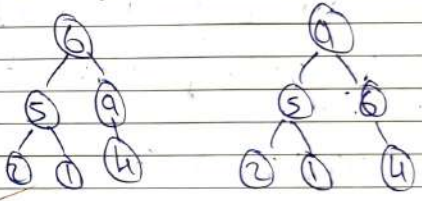
	0	1	2	3	4	5	6	7	8	9
25						25				
6							6			
99										99
145						145				
239										239
26	26									
18									18	
20			20							
25			25							
145										145
6	06	06								
18		18								
99										
239			239							99

	0	1	2	3	4	5	6	7	8	9
06	06									
18	18									
20	20									
25	25									
239			239							
145		145								
99	99									

Q4] Sort the following elements Using heap sort :-

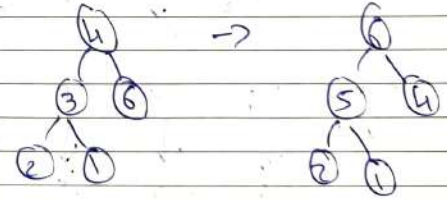
(1) arr [6, 3, 9, 2, 1, 4]

Pass 1



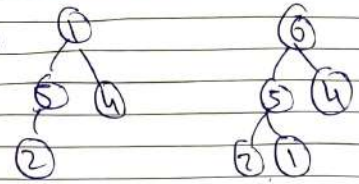
arr: [9, 5, 6, 2, 1, 4]
[9, 5, 6, 2, 1, 9]

Pass (2)



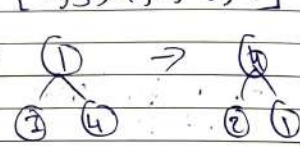
arr [6, 5, 4, 2, 1, 9]
arr [1, 5, 4, 2, 6, 9]

Pass 3



arr = [6,5,4,2,1,9]
arr = [1,5,4,2,6,9]

Pass 4 ⇒



arr = [4,2,1,5,6,9]
arr = [1,2,4,5,6,9]

Q5] Sort following elements using Shell Sort

① arr = [22,18,29,7,9,55,21,8]

Pass 1

22	18	29	7	9	55	21	8
↓	↓	↓	↓	↓	↓		
22	18	29	7	9	55		

Pass 1

22	18	29	7	9	55	21	
↓	↓	↓	↓	↓	↓	↓	
22	18	29	7	9	55	21	

Pass 2

9	18	21	7	22	55	29	8
↓	↓	↓	↓	↓	↓	↓	↓
9	18	21	7	22	55	29	8

Pass 3

9	7	21	8	22	18	29	55
9	7	21	8	22	18	29	55
7	9	21	8	22	18	29	55
7	9	21	8	22	18	29	55
7	8	9	21	22	18	29	55
7	8	9	21	22	18	29	55
7	8	9	21	22	18	29	55
7	8	9	18	21	22	29	55
7	8	9	18	21	22	29	55

1314

Experiment :- 08

1] Write a C program for expression conversion using (convert infix to postfix or prefix expression

→
```
#include <stdio.h>
#include <type.h>
char stack [100]
int top = -1
void push (char x) {
stack [++top] = x;
}
char pop() {
if (top == -1)
return -1;
else
return stack (top--)
}
int priority (char x) {
if (x == '(')
return 0;
if (x == '+' || x == '-')
return 1;
if (x == '*' || x == '/')
return 2;
if (x == '^')
return 3;
return -1;
}
```

```
int main() {
char exp[100];
char *e, x;
printf("Enter infix expression:");
scanf("%s", exp);
e = exp;
printf("Postfix expression:");

while (*e != '\0') {
if (isalnum (*e)) {
printf("%c", *e);
}
else if (*e == '(') {
push (*e);
}
else if (*e == ')') {
while ((x = pop()) != '(')
printf("%c", x);
}
else {
while (top != -1 && priority (s
[top]) >= priority (*e))
printf("%c", pop());
push (*e);
}
e++;
while (top != -1) {
printf("%c", pop());
}
return 0;
}
```

Output :- A+(B*C-(D/E^

Postfix :- ABC*DEF^/G*-

Q(ii) Convert infinix expression into prefix Using Stack

A + (B * C - (D / E ^ F) * G) * H

Input	Stack	output
H	*	H
*	*)	H
)	*)	H
G	*)	HG
*	*)*	HG
)	*)*)	HG
F	*)*)	HGF
^	*)*) ^	HGF
E	*)*) ^	HGFE
/	*)*) /	HGFE^
D	*)*) /	HGFE^D
C	**)	HGFE^D/
+	*) *	HGFE^D/
-	*) -	HGFE^D/*
*	*) - *	HGFE^D/*
B	*) - *	HGFE^D/*B
C	+	HGFE^D/*B-C
+	+	HGFE^D/*B-C*
A	+	HGFE^D/*B-C*A+
		HGFE^D/*B-C*A+

Q(iii) Write a C Program to evaluate infinix or postfix expression

→
```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
int pop();
void push(int);
char postfix[size];
int stack[size], top=-1;

int main()
{
int i, a, b, result, pval;
char ch;
for(i=0; i<size; i++)
{
stack(i) = -1
}
printf("\n Enter a postfix
expression");
scanf("%s", postfix);
for (i=0; postfix[i]!='\0'; i
{
ch = postfix[i];
if (isdigit(ch)
{
push (ch-'0');
{
case '+':
result = a+b;
```

```
        break;
    case '-':
        result = a-b;
        break;
    case '*':
        result = a*b;
        break;
    case '/':
        result = a/b;
        break;
    }
    push(result);
  }
 }
 pval = pop();
 printf("\n the post fix evaluation is
    %d\n", pval);
 return 0;
}
void push(int n)
{
  if (top < SIZE-1)
    stack[++top] = n;
  else
  {
    printf("\n stack is empty\n");
    exit(-1);
  }
}
```

Output:

```
Enter a post fix expression: 12+42*
-42$+
the post fix evaluation is : 6
```

Q(iv) Evaluate Prefix expression using stack

+ - * + 1 2 / 4 2 1 $ 4 2

I/P	OP1	Stg	OP2	result	stack
2					2
4					2, 4
$	4	$	2	16	2, 16
1					16, 1
2					16, 1, 2
4					16, 1, 2, 4
/	4	/	2		16, 1, 2
2					16, 1, 2, 2
1					16, 1, 2, 1
+	1	+			16, 1, 2
*	3	*			16, 1, 6
-	6	-			16, 5
+	16	+			21

Algorithm

1. (a) Scan the infixix expressions from left to right.
(b) If the scanned character is an operand output it.
(c) Else,
(1) If the Precedence of the scanned operator is greater than the Precedence of the operator in the Stack.
(2) Else, Pop all the operators from the Stack which are greater than or equal to in Precendence than that of Scanned operation. After doing that Push the Scanned operator to the Stack
(d) If the scanned character is an '(' Push it to the Stack.
(e) If the Scanned character is an ')' pop the Stack & output it Until a '(' is encountered & discard both Parenthesis.
(f) Repeat step 2-6 Until infixix exp is Scanned
(g) Print the output
(h) Pop & output from the Stack Until It is not empty.

2. [1] Scan the expression from right to left
[2] Scan the character one by one
[3] If the character is an operand, copy it to the prefix notation output
[4] If the character is a closing parenthesis then push it to the Stack.
[5] If the char is an opening parenthesis pop the elements in the Stack Until we find the corresponding closing Parenthesis.
[6] If the char Scanned is an operator
- If the operator has precendence, greater than or equal to the top of the stack, Push the operator to the Stack
- If the operator has precendence lesser than the top of the Stack pop it to the Prefix notation o/p & the check the Above Condition Again with the New top of the Stack
[7] After all the char are Scanned, reverse the Prefix notation output.

3. · Initialize an empty Stalks
· Scan Postfix expression from left to
· If operand → push into S
· If operator → pop two operands fro
Apply operation. Push result back
· After Scanning → top of the Stac
result.

Initialize an empty stack.
- Scan prefix expression from right to
- if operands ⇒ Push into S.
- if operator ⇒ POP two operands from S
- Apply operation, push result back
- After scanning → top of the stack is result.

PAGE NO.
DATE / /

Experiment :- 05

Q] WAP to Perform ~~operations on Array~~ Singly linked list for the following operation

```
#include <stdio.h>
#include <stdio.h>

void create();
void display();
void insert_begin();
void insert_begin();
void insert_end();
void insert_pos();

struct node {
    int info;
    struct node *Next;
};
struct node *start = NULL;

int main() {
    int choice;

while (1) {
  Printf("\n ---- Menu ----\n");
  Printf("1: create\n");
  Printf("2 Display\n");
  Printf("3 Insert at beginning\n");
  Printf("4 Insert at end\n");
```

```
Printf("6 exist\n");
Printf("\nPease enter your choice:");
scanf("%d", &choice);

Switch(choice) {
case I:- create(); break;
case II:- display(); break;
case III:- Insert_begin(); break;
case IV:- Insert_end(); break;
case V:- Insert_Pos(); break;
case VI:- exist(0);

defaut:- Printf("Please enter Action
          from the options!.\n");
  }
}
return 0;

for (int i=0; i<n; i++) {
  Printf("enter no of Nodes:");
scanf("%d", &val);
New node = (Struct node) mok(size
     of Struct node));

Newnode→ info = val;
New node → Next = Null;

if (Start = Null) {
    Start = New node;
    temp = Start;
```

```
{
 else {
  temp → Next = New node;
    temp = New node;
  }
 }
 Printf("created:\n");

 id display () {
 Struct node * temp. Start;
  if (Start = NuLL) {
  Printf("list is empty:\n");
     return 0;

 }
  Printf("NULL\n");

 id Insert_begin () {
     int val;
  Printf(" Enter value:");
     scanf("%d", val);

Struct node * New node = (Struct
          (Struct node);
 (Size of Node));
New node → info = val;
New node → Next = NULL;

 if (Start = NULL) {
    Start = New node;
  } Else {
```

```
struct node* temp = start;
  while (temp→ next! = Null)
    temp= temp→ Next;
     temp→ Next = New Node;

{
    PrintF ("-/.d insert at end a"val

IF (temp= NULL) {

  PrintF ("-/.d Inserted after Position
    -/.d\n", val, pos);
```

Expt:- 06

Q] Doubly linked list write a Name driven C-Program that implements doubly linked list for the following operations:- (create, display, insert a node delete a Node, search a Node

⇒ Code:-

```
# include <Stdio.h>
  # include <Stdio.h>
     type def Struct node{
        int data;
        Struct node * Prev;
         Struct node * Next;
     } node;
     Node * Head = NULL;
Node * create ngde {
  Node * New Node = (Node*)
Malloc (size of (Node);
  New node → data = data;
New node → Prev : NewNode →
Next = NULL;
return * NewNode;
  }
void create list () {
    int data, n, i;
Print f ("Enter no of Nodes:");
Scan f ("-/. d", n);
```

```
head = NULL;
Node * temp = NULL;
Node * temp = NULL;
for (i=0; i<n; i++) {
Printf("Enter no for node :%d:");
Scanf("%d", &data);
Node * Newnode = create Node
(data);
if (head == NULL) {
    head = New node;
        temp = head;
temp -> Next = New node;
  New node -> Prev = temp;
   temp = New Node;
  }
 }
 Printf(" list Created Successfully
}
 void display list () {
    if (head = = NULL) {
   Printf (" list is empty!\n);
     return;
   }
Node * temp = head;
 Printf (" list : ");
 while (temp) {
 Printf("%d", temp -> data);
  temp = temp -> Next;
 }
 Printf("\n");
}
```

```
New node -> Next = temp -> Next;
 if (temp -> Next)
     temp -> Next -> Prev = New node;
temp -> Next = New node;
 New Node -> Prev = temp;
}
Printf (" Node inserted \n");
}
 void delete node() {
 int pos;
 Printf(" Enter Position to
 delete:");
 Scanf("%d", to pos);
  if (head = = NULL) {
  Printf ("list is empty\n");
  return 0;
  }
Node * temp = head;
  Put :;
  if (pos = = 1) {
   head = head -> Next;
   if (head)
   head -> Prev = NULL;
    free (temp);

} else
  for (i=1; i < Pos+1 temp; i++)
     temp = temp -> Next;
  if (!temp) {
  Printf (" Invalid Position \n");
    return;
```

```
}
if (temp -> Prev)
    temp -> Prev -> Next = temp -> Next;
if (temp -> Next)
    temp -> Next -> Prev = temp -> Prev;
free (temp);
}
Printf (" Node deleted : \n");
}
temp = temp -> Next;
    Pos ++;
}
Printf (" value not found in the
list \n");
}
void Count Nodes(){
    int count = 0;
    Node * temp = head;
    while (temp) {
        count++;
        temp = temp -> Next
    }
    Printf (" Total Nodes : %d \n", count
    }
    void reverse list() {
    if (!head){
    Printf (" list is empty : \n");
        return;
    }
    Node * temp = NULL, * current =
    head;
```

```
while (current) {
    temp = current -> Prev;
    current -> Prev = current -> Next;
    current -> Next = temp;
    current = current -> Prev;
}

int main()
    int choice;
    while (1) {
Printf (" \n .... Doubly linked list
    .. .. \n");

Printf (" 1. Create list \n");
Printf (" 2 Display list \n");
Printf (" 3 Insert node \n");
Printf (" 4 Delete node \n");
Printf (" 5 Search node \n");
Printf (" 6 Count node \n");
Printf (" 7 Reverse list \n");
Printf (" 8 exit \n");
Printf (" Enter goes choice (1-8):
    Scanf ("%d", & choice);

Switch (choice) {

case 1 : Create list (); break;
case 2 : display list (); break;
case 3 : insert node (); break;
case 4 : delete node (); break;
case 5 : Search node (); break;
```

```
case 6: Search Node(); break;
case 7: reverse list(); break;
case 8: Exit(0);

default: Printf("Invalid choice
    try again\n");
```

PAGE No:
DATE / /

Expt:- 07

1] Primitive operations on Stack: Pop Push is empty, is full;

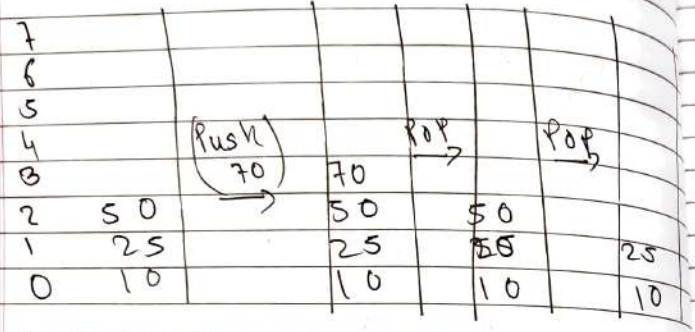
→
```
#include <stdio.h>
#define Max 100
int Stack [MAX]
int top = -1,
void push (int n);
  void pop ();
  void is empty ();
  void is full ();
  int main()
  {
  int value, choice;
Printf("Enter '1' to Push:\n");
Printf("Enter 2 to POP:\n");
Printf("Enter "3" to check if Stack
is empty:\n").
Printf("Enter '4' to check if
Stack is full:\n);
Printf("Enter '5' to exit:\n")
  While (1)
{
  Printf("Enter your choice:
Scanf("%d", &choice);
  Switch (choice)
{
```

```
case 1: Printf ("Enter value:")
        scanf("%d", &value);
        push (value);
        break;

case 2:  POP()
         break;

case 3: is empty();
        break

case 4: is full();
        break;

case 5: Printf("Exiting...\n");
        return 0;

default: Printf (" Invalid input!!
}
return 0;
}
void push (int n)
   {
   if (top == Max-1)
   {
   Printf ("stack overflow
   }
   else
   {
   Printf ("%d element popped ou
\n", Stack [top--]);
   }
```

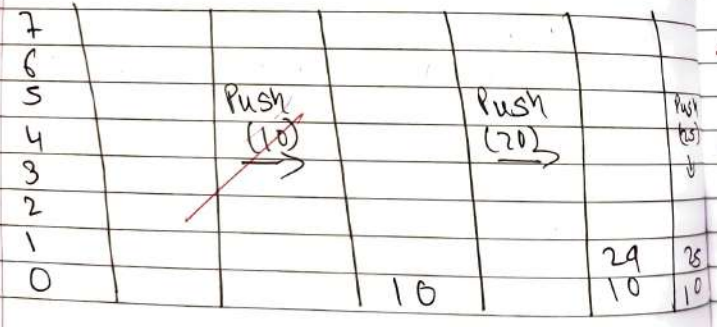
```
void is empty ()
  {
  if (top == -1)
  {
   Printf ("Stack is empty!\n");
    }
     else
     {
Printf("Stack is not empty!\n");
 }
 }
  void is full()
    {
    if (top == Max-1)
    {
     Printf("Stack is full!\n");
    }
    else
    {
    Printf("Stack is not full!\n")
    }
    }
```

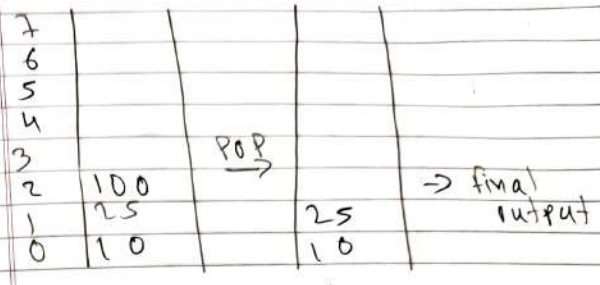
11] Stack Size for Push (10), Push (25), Push (50), Push (70), Pop, Pop, Push (100), Pop & draw final output.

Push (50)



Push (100)





Stack [8] → (Size, 8)
Push (10) → Stack [10]
Push (20) → Stack [10, 20]
Pop → Stack [10]
Push (25) → Stack [10, 25]
Push (50) → Stack [10, 25, 50]
Push (70) → Stack [10, 25, 50, 70]
Pop → Stack [10, 25, 50]
Pop → Stack [10, 25]
Push (100) → Stack [10, 25, 10
Pop → Stack [10, 25]

final output: Stack [10, 25]

Expt:- 09

Queue..

```
#include <stdio.h>
#include <stdlib.h>
#define Size 5
    int Queue [Size]
    int front=-1, rear=-1;
      void insert (){
        int value;
        int (rear==size-1){
  Printf(" Queue is full overflow
\n");
} else;
Printf(" Enter value to insert:");
Scanf ("%d", &value);
  if (front==1){
  front = 0
  rear++;
  Queue [rear] = value;
Printf(" \n inserted %d \n", value
}
}
void delete (){
if (front==-1 || front> rear){
Printf(" Queue is empty (underflow)
\n
} else{
Printf (" Deleted %d \n", Queue
```

```
void display() {
 if (front==-1 || front>rear){
Printf(" Queue is empty \n");
} else {
 Printf (" Queue elements");
 For (int i= front; i<=rear; i++){
Printf("%d", Queue [i]);
}
 Printf (" \n");
}
}
Switch (choice) {

case 1: insert ();
        break;

case 2: delete ();
        break;

case 3:
        display()
        break;

case 4: exit (0)
        default:

Printf (" Invalid choice\n");
}
return 0;
```

Output:

1. Insert
2. Delete
3. Display
4. exit

Enter your choice:
Enter value to insert : 10
inserted 10

Mean ...
Enter your choice = 1
Enter value to insert: 20
inserted: 20

Mean ....
Enter your choice: 1
Enter value to insert: 20
inserted 20

02 Perform following operation on linear Queue in Array Size 10 with this operation insert (10), insert (50), delete, insert(100)

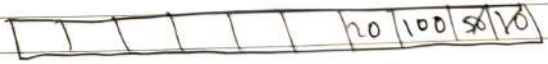
→ Insert (10)

rear: 0 , front = 0
insert (50)

rear = 1 front = 0

insert (100)

insert (20)



rear: 2 front = 0

Delete Insert (25)

insert (200)

Q1] Explain the concept of Priority Queue

→ A Priority Queue is a special type of Queue where each element is assigned a priority & element are derived based on their priority, not just their Arrange order

2] Write Algorithm of insertion & deletion of linear Queue.

→ Insertion:-

1] check if the Queue is in overflow condition

2] If no overflow condition check if front is equal to -1 & change it to 0.

3] Increase rear by 1, equate the rear to the value entered

4] Print the value inquired.

Expt:-10

- Circular Queue.

```
#include <stdio.h>
#include <stdlib.h>
#define Max 6
int front=-1;
int rear=-1;
int Queue[Max];

void enqueue (int val);
void dequeue ();
void display ();
int main (){
  int choice value;
  while (1) {
Printf ("..... Menu ....");
Printf ("1 enqueue\n");
Printf ("2 dequeue\n");
Printf ("3 display\n");
Printf ("4 exit\n");
Printf ("choose option;");
Scanf("%d", &choice);
Switch
```

case1:-

```
Printf (" Enter value to enqued")
Scanf ("%d" &value);
break;
```

```
case 2:
    dequeue();
    break;

case 3:
    display();
    break;
case 4:
    exit(0);
    break;
default:
    Printf("Invalid input");
    }
}
return 0;
}
Printf("%d inserted Successfully",x
  }
}
void dequeue(){
if (front == -1){
  Printf("Queue empty");
}
else
{
  int item = Queue[front];
  if (front == rear){
  front = rear = -1;
  }
    else
{
front = (front+1) % Max;
}
Printf("%d" dequed", item);
}
}
void display(){
 if (furont == -1){
   Printf("Queue empty");
}
  else{
int i = front; of ("Queue elements
  while (1){
Printf("%d", Queue[i]);
  if (i == rear){
     break
  }
  i = (i+1) % Max;
  }
}
}
```

Output

1 Menu
2 Dequeue
3 display
4 exit

Choose option : 1
Enter value to be enqueue : 12
12 enque successfully

.... Menu .....
Choose option : 3
Queue elements : 12 13

..... Menu .....
Choose option : 2
12 dequed successfully

Expt :- 11

Operations on Binary tree

```
#include <stdio.h>
#include <stdlib.h>
    Struct node {
    int data;
    Struct node * left;
    Struct node * right;
};

Struct node * create (int val)
Struct node * insert left (Struct no
* root , int val);
Struct node * insert right (Struct n
* root , int val);
void delete Subtree (Struct nod
root);
void delete left (Struct, node *
root);
void delete right (Struct _node *
root);
void display (Struct, node * roo

int main () {
Struct node * root = create (
insert left (root; 2);
insert right (root; 2);
insert right (root → left, 4);
insert (root → left, 5);
```

```
Printf("original tree : ");
    display (root);
    Printf("\n");
delete left (root)
display (root);
return 0;
}

Struct node* create (int val){

    if (New = NULL) {
    Printf("Memory Alloc failed");
      exit (1);
    }

New -> data = val.
New -> left = NULL
New -> right = NULL;
    return New;
}

root -> right = (reate (val);
return root -> right;
}

void display (Struct node* root){
    if (root == NULL) {
        return;
    }
Printf ("%d ", root -> data);
    display (root -> left);
    display (root -> right);
```

3

output :-

original tree : 1 2 4 8 5 3 6 7

left Subtree deleted

Tree After deletion: 1 3 6 7

Q2 Pre order, Post order, inorder traversal.

```
→ #include <stdio.h>
  #include <stdlib>

  Struct node {
    int data;
    Struct node* left;
    Struct node* right;
  };

Void Pre order (Struct node* r
  void post order (Struct node* ro
  void inorder (Struct node* roo

  int main() {

  int choice;
  while (1) {
  Printf (" --- Menu --- \
```

```
Printf("1. Pre_order traversal");
Printf("2. Post_order traversal");
Printf("3. In order traversal");
Printf("4. exit");
scanf("%d", &choice);
switch (choice) {

case 1:

  Pre order (root);
   break;

case 2:

  Post order (root);
    break;

Case 3:

  in order (root);
   break;

(case 4:
  void inorder (struct node *root)
if (root == NULL) {
     return;
}
inorder (root -> left)
Printf("%d", root -> data);
}
```

Output:-

```
---- Menu ----

1 Preorder traversal
2 Postorder traversal
3 inorder traversal

Choose option: 1

1 2 4 8 5 3 6 7

---- Menu ----

Choose option:- 2
8 4 5 2 6 5 7

---- Menu ----

Choose option: 3

8 4 2 5 1 6 3 7
```

Expt:- 12

Q->
```
#include <stdio.h>
#define V 5

void init (int arr [V] = [V]){
  int i, j;

  for (i=0; i<V; i++)
  for (j=0; j<V; j++)
  arr [i] [j] = 0;
  }
  void insert Edge (int arr [V] [V]
  int j) {
  arr [i] [j] = 1;
  arr [j] [i] = 1;
  }
  void print adj Matrix (int arr [V]
[V]) { int i, j;
Printf (" Adj Matrix: \n");
for (i=0; i<V; i++) {
for (j=0; j<V; j++) {
Printf ("%d ", arr [i] [j]);
  }
  Print ("\n")
  }
  }

int main () {
int Adj Matrix [V] [V];
```