PAGE No.
DATE / /

* Experiment 1 *

WAP to implement array operations -

(a) Find average of 10 numbers using array.

```
#include <stdio.h>
   int main()
{
    int n[10];
    int s=0;
    float a;
    printf("Enter 10 number :\n");
    for(int i=0; i<10; i++)
{
    printf("Number %d :", i+1);
    scanf("%d", &n[i]);
    s+ = n[i];
}
    a = s/10.0;
    printf("The average of 10 numbers is :
    %.2f\n", a);
    return 0;
}
```

Enter 10 numbers :

2
5
7
4
8
3
2
7
7

The average of 10 numbers is : 5.30.

⑥ Display the following pattern —

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

```
#include <stdio.h>
int main() {
  for (int i=1 ; i<=4 ; i++)
{
  char c = (i%2==1) ? '*' : '#' ;
  for (int j = 1 ; j<=i ; j++)
  printf("%c", c);
  printf ("\n");
}
  return 0;
}
```

Output :-

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

PAGE No.
DATE / /

(c) Find first repeating number in an array.

```
#include<stdio.h>
int main()
{
  int n ;
  printf(" Enter Number:");
  scanf("%d", &n);
  int a[n];
  printf("Enter %d elements :\n", n);
   for ( int i=0 ; i<n ; i++)
   scanf ("%d" , &a[i]);
   int f= -1 ;
    for (int i=0; i<n ; i++)
  {
   for (int j= i ; j<n ; j++)
  {
     if ( a[i] ==a[j])
    {
      f= a [i];
       break ;
    }
  }
    if (f != -1 )
        break ;
  }
    if ( f != -1 )
      printf(" first repeating no is :
          %d\n", f);
```

```
else
    printf(" NO repeating no found ");
    return 0;
}
```

output :

Enter the Number : 4

Enter 4 elements

1

2

3

4

NO repeating Number found.

4) Find greatest and smallest element in array.

```
#include <stdio.h>
int main()
{
    int n;
    printf(" Enter number ");
    scanf(" %d ", &n);
    int a [n];
    printf("Enter %d elements : \n", n);
    for ( int i=0 ; i<n ; i++ )
    {
    scanf(" %d ", &a[i]);
    }
```

```
int g = a[0];
int s = a[0];
for (int i=1; i<n; i++)
{
    if (a[i] > g)
        g = a[i];

    if (a[i] < s)
        s = a[i];
}
    printf("Greatest : %d\n", g);
    printf("Smallest : %d\n", s);
    return 0;
}
```

Output :

Enter number : 5

Enter 5 elements

1

2

3

4

5

Greatest element : 5

Smallest element : 1

PAGE NO.
DATE / /

(c) Square odd element numbers of array.

```
#include <stdio.h>
int main ()
{
    int n;
    printf("Enter number");
    scanf("%d", &n);
    int a [n];
    printf("Enter %d elements :\n", n);
    for (int i=0 ; i<n; i++)
    scanf("%d", &a[i]);

    for (int i=0 ; i<n ; i++)
{
    if( a[i]%2 ==0)
    {
        a[i] = a[i] * a[i];
    }
}
    printf("After squaring :");
    for (int i=0 ; i<n ; i++)
    printf("%d", a[i]);
    printf("\n");
    return 0;
}
```

Output:

Enter Number of elements : 2
Enter 2 elements :
2
4
Array after squaring:
2 4 .

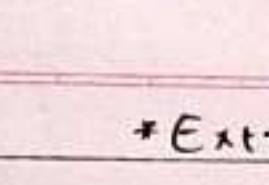
PAGE No.
DATE / /

Extra

(a)

```
#include <stdio.h>
int main()
{
    int i, j;
    for (i=0; i<5; i++)
    {
        for (j=0; j<i; j++)
        {
            printf("*");
        }
        printf("\n");
    }
```

Output :-



(b)

```
1
1 2
1 2 3
1 2 3 4
```

```c
#include <stdio.h>
int main ()
{
  int i, j;
  for (i=1 ; i<=4 ; i++)
{
  for (j=1 ; j<=i ; j++)
  printf("%d ", j);
  printf("\n");
}
  return 0;
}
```

~~Do~~

(c)

```
1
2 2
3 3 3
4 4 4 4
```

```c
#include <stdio.h>
int main()
{
  for(int i= 0; i<5 ; i++)
{
  for( int j= 0 ; j<i ; j++ )
    printf("%d", i);
    printf("\n");
}
  return 0;
}
```

(d)

```
1
2 2
3 3 3
1 2 3 4
5 5 5 5 5
```

```
#include<stdio.h>
int main()
{
    for(int i=1; i<=5 ; i++)
    {
        for(int j=1 ; j<=i ; j++)
        {
            if (i%2==0)
                printf("%d", j);
            else
                printf("%d", i);
        }
        printf("\n");
    }
    return 0;
}
```

⑥

```
*
# #
$ $ $
? ? ? ?
```

```
#include<stdio.h>
int main()
{
    for(int i=1 ; i<=4 ; i++)
    {
    for(int j=1 ; j<=i ; j++)
}
    if( i==1)
     printf("*");
else   if( i==2 )
     printf("#");
else   if ~~("#")~~( i==3)
     printf("$");
else  if (i==4 )
     printf("?");
    }
     printf("\n");
}
  return 0;
}
```

(f)

```
#include <stdio.h>
int main()
{
    int i, j, s;
    int r=4;
    for (i=0; i<r; i++)
{
    for (s=0; s<i; s++)
    printf(" ");

    for (j=0; j<r-i; j++)
{
    printf(" *");
}
    printf("\n");

}
    return 0;
}
```

PAGE No.
DATE / /

* Experiment - 2 *

a Write a c program to perform linear using foll data [56, 36, 89, 57, 01, 06, 0, 67, 59].

Search item 1 & 55 from above list & check weather it is found or not.

```
#include <stdio.h>
int main() {
  int arr[] = {56, 36, 89, 01, 0, 67, 59};
  int keys[] = {1, 55}, size=7, found, i, j;

  for(j=0; j<2; j++) {
    found=0;
    for(i=0; i<size; i++) {
      if(arr[i] == keys[j]) {
printf("Item %d found at index %d\n", key[j][i]
        found = 1;
        break;
      }
    if
    printf(! found)
printf("\n Item %d not found in list\n", keys[j][i]
}
    return 0;
```

Output

Item1 found at index 3

Item55 not found at list,

① Search the item 01

Output:

Enter key to be searched : 01

key found at Index : 4

② Search the item 55 .

output :

Enter key to be searched : 55 .

key not found in array .

③ Search data using Binary search.

```
#include<stdio.h>
int main()
{
    int a[100], n, m, i, h, found = 0K;
    printf("Enter Numbers of elements to searched:");
    scanf("%d", &n);
    printf("\n Enter sorted Array elements : \n");
    for ( i=0; i < n; i++ )
    {
    scanf("%d", &a[i]);
    }
    i=0 ; h = (n-1);
    while (h==1)
    {
    m = (int) ((i+h)/2);
    if (c==a[m])
    }
```

```
        printf("\n Element found AT INDEX: %d", m);
    }
    else
    {
        if (k > a[m])
        {
            l = m + 1;
        }
        if (k < a[m])
        {
            h = m - 1;
        }
    }
    if (h < l)
    printf("\n Element Not found");
    return 0;
}
```

Output :-

Enter Numbers of elements to Searched

Enter key to be searched 2

Enter sorted array elements

5 6 6

Element not found at index: 0

(3) compare linear search & binary search.

Linear Search	Binary Search
Time complexity is 1010	Time complexity is O.
works on both sorted and converted data.	only works on sorted data.
It is less efficient, especially for large dataset.	It is more efficient, especially for large dataset.
Code is simpler.	Code is more complex.
It uses sequential searching approach.	It uses divide and conquer approach.

(4) State limitations of linear search in terms of time complexity.

A linear search runs in a worst linear time and makes at most n comparisions, where n is the length of the list. Linear search is rarely practical because other search algorithms and schemes, such as the binary search algorithm and hash tables, allow significantly faster searching for all but short lists.

① WAP in C to copy the elements of one array into another array in reverse order.

```
#include<stdio.h>
int main()
{
    int a[100], r[100], n;
    printf("Enter the number of elements");
    scanf("%d", &n);
    printf("Enter %d elements", n);
    for(int i=0; i<n; i++)
    {
        r[i] = a[n-1-i];
        scanf("%d", &a[i]);
    }
    for (int i=0; i<n; i++)
    {
        r[i] = a[n-1-i].
    }
    printf("Reversed array:\n");
    for (int i=0; i<n; i++)
    {
        printf("%d", r[i]);
    }
    return 0;
}
```

Output

Enter the number of elements: 5
Enter 5 elements

Reversed array

1
2
3
4
5

Algorithm :

① Start
② Input variable (i, a for array, r to store reversed array, n to input number of elements and i for loop.
③ Input no of elements.
④ use for loop to input elements.
⑤ use for loop to reverse the element.
⑥ print the reversed array.
⑦ Stop

(2) WAP to count total number of duplicate elements in array.

```
#include<stdio.h>
int main()
{
  int a[50], n, c=0, i, j;
  printf("Enter no of elements : ");
  scanf("%d", &a[i]);

  for(int i=0; i<n; i++)
{
  for(int j=i+1; j<n; j++)
  {
    if(a[i]==a[j])
    {
      c++;
      break;
    }
  }
  printf("Total duplicate elements %d", c);
  return 0;
}
```

output :

Enter no of elements : 5

Enter 5 elements

2

7

9

2

Total duplicate elements : 1

(Algorithms)

① Start

② Declare a flow array "n" for storing no of elements in array, "c" for count of duplicate elements.

③ Input the no. of elements.

④ Print the statement in(for) the array.

⑤ Input elements in the array.

⑥ use to for loop and use if condition to check is a[i] == a[j] if true

⑦ Print c total no of duplicate elements.

⑧ Stop.

(8) WAP in C to print all unique elements in an array appear.

```
#include<stdio.h>
int main()
{
    int a[50], n, v;
    printf("Enter no of elements");
    scanf("%d", &n);
    printf("Enter no elements");
    for(int i=0; i<n; i++)
    scanf("%d", &a[i]);
    printf("Unique elements are");
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
        {
            for(i!=j && a[i]==[j])
            {
                a=0;
                break;
            }
        }
        if(a)
            printf("%d", a[i]);
    }
    return 0;
}
```

Output:

Enter number of elements : 3

Enter 3 elements

1

2

3

Unique elements are

Algorithm

(1) Start

(2) Declare a (for array), n (for no of elements), a (for unique), i, j (for loop)

(3) Enter no of elements (n)

(4) Input elements using for loop

(5) Check if they are unique.

(6) Print array elements.

(7) Stop.

(6a) WAP to seprate odd and even integers into seprate arrays

```
#include<stdio.h>
int main()
{
    int a[50], e[50], o[50];
    int n, ev=0, od=0;
    printf("Enter no of elements");
    scanf("%d", &n);
    printf("Enter %d elements\n");
    for (int i=0; i<n; i++)
    scanf("%d", &a[i]);
    for (int i=0; i<n; i++)
    {
    if (a[i] %2 == 0)
        e[ev++] = a[i];
    else
        o[od++] = a[i];
}
printf("Even elements\n");
    for (int i=0; i<ev; i++)
printf("%d", e[i]);
    for (int i=0; i<od; i++);
printf("%d", o[i]);
    return 0;
}
```

OUTPUT :

Enter no of elements : 5
Enter 5 elements :
1
2
3
4
5

Even Elements
2
4

Odd Elements
1
3
5

```
    }
    printf("Even elements:\n");
    for (int i=0; i<ev; i++)
    printf("%d", e[i]);
    for (int i=0; i<od; i++);
    printf("%d", o[i]);
    return 0;
}
```

Output:

Enter no of elements: 5

Enter 5 elements:

1

2

3

4

5

Even Elements

2

4

Odd Elements

1

3

5

(6) WAP to count the total no of words in a string.

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s[100];
    int i, words = 1;

    printf("Enter a string:");
    scanf("%[^\n]", str);
    for (i=0; str[i] != '\0'; i++) {
        if (str[i] == ' ' && str[i+1] != ' ' && str[i+1] != '\0') {
            words++;
        }
    }
    printf("Total no of words = %d\n", words);
    return 0;
}
```

Output:

Enter a string - Hello world.
Total no of words : 2

Experiment 3 :

1) Sort elements in ascending order using bubble Sort .

```
#include<stdio.h>
int main()
{
    int a[] = { 5, 1, 4, 2, 8 };
    int n = sizeof(a) /sizeof (a[0]);
    int i, j, t;
    printf(" Original Array");
     for ( i= 0 ; i<n ; i++)
       printf("%d", a[i]);

    For (i=0 ; i<n-1 ; i++)
{
    For (j= 0 ; j<n-1 ; j++)
{
    if (a[j] > a [j+1] )
    {
        t = a [j] ;
        a [j] = a [j+1];
        a [j+1] = b;
    }
}
}
```

```
printf("Sorted Array:");
for(i=0; i<n; i++)
printf("%d", a[i]);

return 0;
}
```

Output :-
Original array:
5 1 4 2 8
Sorted array:
1 2 4 5 8

(2) Sort elements in descending order using Selection Sort.

```
#include <stdio.h>
int main()
{
    int a[] = {5, 1, 4, 2, 8};
    int n = sizeof(a)/sizeof(a[0]);
    int i, j, m, t;

    printf("original array:");
    for(i=0; i<n; i++)
        printf("%d", a[i]);

    for(i=0; i<n-1; i++)
    {
```

```
        m = i;
        for (j = i+1; j < n; j++)
    {
        if (a[j] > a[m])
        {
            m = j;
        }
    }
        t = a[i];
        a[i] = a[m];
        a[m] = t;
}
printf("Sorted array in descending
        order");
for (i = 0; i < n; i++)
    printf("%d", a[i]);

    return 0;
}
```

Output :

Original array :
5 1 4 2 8
Sorted array in descending order :
8 5 4 2 1

(3) Find the no of comparisons required in bubble sort method of following data having 5 numbers : 100, 200, 300, 400, 500

Given Numbers : 100, 200, 300, 400, 500

100 200 300 400 500
100 200 300 400 500
100 200 300 400 500
100 200 300 400 500

Hence only 1 pass and 4 comparisons take place as the given numbers are already sorted.

Algorithm:

1- Start with the list of Numbers
2- Compare each pair of adjacent elements and swap if the first element is greater than second element. Continue comparing and sorting.
3- After each full pass the largest unsorted element moves to the end of array.
4- Repeat the process of 2 step on the unsorted array.
5- Stop when all elements are in the correct order

④ Sort the given array in selection sort for ascending order and show diagramatic representation of every iteration of the loop : 500, -20, 30, 14, 50

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

Hence, only 4 passes and 5 comparisons take place.

Algorithm :-

1- Set min TO 0
2- Search min element of the array
3- Swap value of the location of min
4- Increment min TO point the next element location.
5- Repeat till sorted
6- Stop.

* Experiment - 4

a. Sort element in ascending order using insertion Sort.

```
#include <stdio.h>
int main()
{
    int a [] = { 7, 3, 5, 1, 9, 8, 4, 6 };
    int n = sizeof (a) / sizeof (a [0]);
    int i, j, k ;

    for (i=1 ; i<n ; i++ )
{
    k= a [i] ;
    j= i-1 ;
    while (j>=0 && a[j] >k)
{
    a [j+1] = a [j] ;
        j= j-1 ;
}
    a [j+1] = k ;
    if (i== 2)
{
    printf("Array");
    for (int i=0 ; i<n ; i++)
    printf(" %d ", a [i] );
    printf ("\n");
}
}
```

```
printf(" Sorted Array");
    for (i=0 ; i<n ; i++)
        printf("%d ", a[i]);
        return 0;
}
```

Output :-

Array

3 5 7 1 9 8 4 6

Sorted Array:

1 3 4 5 6 7 8 9

b) Sort element in ascending order using radix sort.

```
#include <stdio.h>
int getmax (int a[], int n)
{
    int m = a[0];
    for (int i=1 ; i<n ; i++)
    {
        if (a[i] > m)
            m = a[i];
        return m;
    }
}
void Sort (int a[], int n, int exp)
{
    int a[50], c[10] = {0};
    for (int i=0 ; i<n ; i++)
        c[(a[i]/exp) % 10] + 1;
    for (int i=1 ; i<10 ; i++)
        c[i] += c[i-1];
```

```
for (int i=n-1 ; i>=0 ; i--)
{
    a[c[(a[i]/exp)%10]-1] = a[i];
    a[i];
    c[(a[i]/exp)%10]--;
}
    for (int i=0 ; i<n ; i++)
        a[i] = a[i];
}
    void radixSort (int a[], int n)
{
    int m = getMax (a, n);
    for (int exp = 1; m/exp>0 ; exp*=10)
        Sort (a, n, exp)
}
    int main ()
{   int a[] = {7, 3, 5, 1, 9, 8, 4, 6};
    int n = sizeof (a) / sizeof (a[0]);

    radixSort (a, n);
    printf ("Sorted array:");
    for (int i=0 ; i<n ; i++)
    printf ("%d", a[i]);
    return 0;
}
```

Output :-

Sorted Array:

1, 3 4 5 6 7 8 9

(c) What is the output of insertion sort after the 2nd iteration given the following sequence of numbers:

7, 3, 5, 1, 9, 8, 4, 6

7 3 5 1 9 8 4 6

3 7 5 1 9 8 4 6

3 5 7 1 9 8 4 6

1 3 5 7 9 8 4 6

1 3 5 7 8 9 4 6

1 3 4 5 7 8 9 6

1 3 4 5 6 7 8 9

Algorithm:

1. If it is the 1st element, it is already sorted element.
2. Pick next element.
3. Compare with all the elements in the sorted sub list.
4. Shift all the elements in the sorted sub list that is greater than the value to be sorted.
5. Insert the value.
6. Repeat until the list is sorted.
7. Stop.

(d) Sort the following number using radix sort

(i) 100, 225, 390, 4130, 956, 99, 5413

	0	1	2	3	4	5	6	7	8	9
0100										
0225						0225				
0390										
4130										
0956							0956			
0099										0099
5413			5413							8

	0	1	2	3	4	5	6	7	8	9
0100		0100								
5413										
0225										
4130										
0956										
0390										
0099										

	0	1	2	3	4	5	6	7	8	9
0100		0100								
5413										
0225			0225							
4130		4130								
0956										0956
0390				0390						
0099										

	0	1	2	3	4	5	6	7	8	9
0100										0390
0390				4130						
4130			5413							
5413				0225						
0225						0956				
0956										0099
0099										

	0	1	2	3	4	5	6	7	8	9
0099	0099									
0100	0100									
4130					4130					
0225	0225									
0390	0390									
5413						5413				
0956	0956									

∴ Sorted array:

[0099, 0100, 0225, 0396, 0956, 4130, 5413]

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

	0	1	2	3	4	5	6	7	8	9
025						025				
006							006			
099										099
145						145				
239										239
020	020									
018										

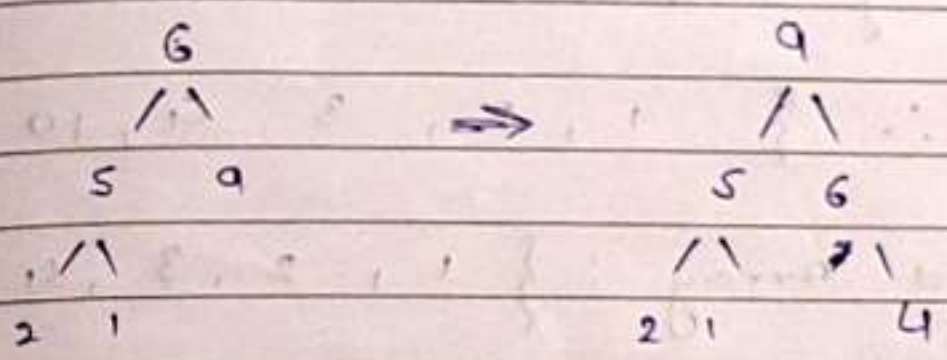
	0	1	2	3	4	5	6	7	8	9
020			020							
25			025							
145						145				
006	006									
018		018								
099										099
239				239						

	0	1	2	3	4	5	6	7	8	9
006	006									
018	018									
020	020									
025										
239				239						
145		145								
099	099									

Sorted Array :- 006, 018, 020, 025, 099, 145, 239.

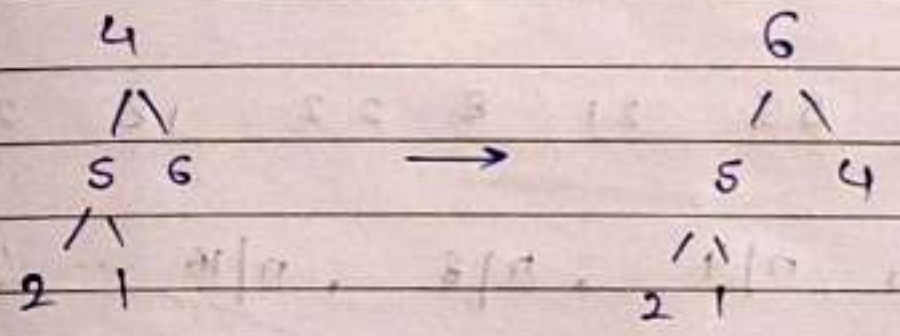
(c) Sort the following elements using Bop Sort.

(i) 6, 5, 9, 2, 1, 4.



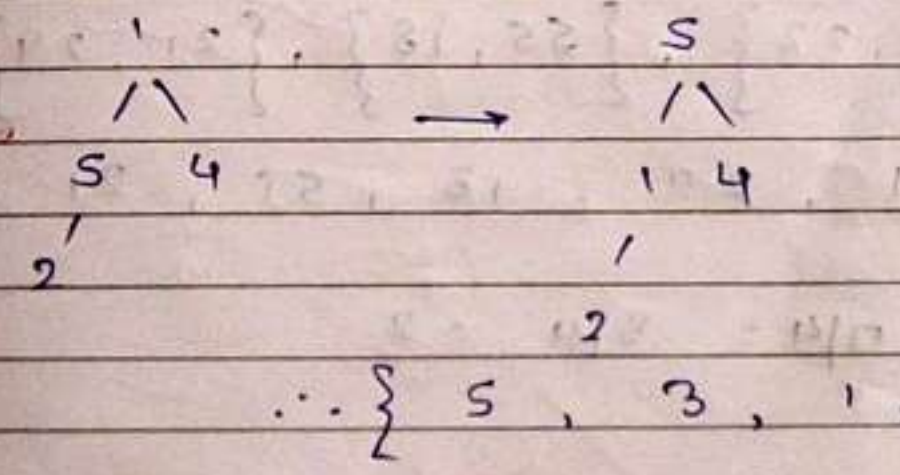
Array: { 9, 5, 6, 2, 1, 4 }

∴ { 4, 5, 6, 2, 1, 9 }



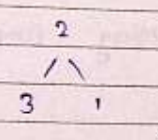
∴ { 6, 5, 4, 2, 1, 9 }

{ 1, 5, 4, 2, 6, 9 }



∴ { 5, 3, 1, 2, 10 }

{ 2, 3, 1, 4, 10 }



{ 3 , 2 , 1 , 4 , 10 }

∴ { 1 , 2 , 3 , 4 , 10 }

Sorted Array : { 1 , 2 , 3 , 4 , 10 }

(f) Sort the following using shell short.

0	1	2	3	4	5	6	7
22	18	29	7	9	55	21	8

∴

9 55 21 8 22 18 29 7

n/2 , n/4 , n/8 , n/16 1

n=8

n/2 = 8/2 = 4

{ 9 , 22 } , { 55 , 18 } , { 21 , 29 } , { 8 , 9 }

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

n/4 = 8/4 = 2

{9, 18, 21, 7} {22, 55, 29, 8}

{7, 9, 18, 21} {8, 22, 29, 55}

$n/8 < 8/8 = 1$

(c) 22, 18, 22, 7, 9, 55, 21, 8

$n/2$, $n/4$, $n/8$, $4/16 \ldots 1$

$n = 8$.

$n/2 = 8/2 = 4$

{22, 9} {18, 55} {29, 21} {7, 8}

{9, 22} {18, 55} {21, 29} {7, 8}

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

$n/4 = 8/4 = 2$

{9, 18, 21, 7} {22, 55, 29, 8}

{7, 9, 18, 21} {8, 22, 29, 55}

→ 7, 9, 18, 21, 8, 22, 29, 55

$4/8 = 8/8 = 1$.

7 9 18 21 8 22 29 55

7 9 18 21 8 29 55

7 9 18 21 8 22 29 55

7 9 18 21 8 22 29 55

∴ 7, 8, 9, 18, 21, 22, 29, 55

[Sorted Array]

(ii) 3, 10, 15, 12, 1, 5, 2, 6

n/2, 4/4, n/8, n/16 ... 1

n/2

n/2 → 8/2 = 4

{3, 1} {10, 5} {15, 2} {12, 6}

{1, 3} {5, 10} {2, 15} {6, 12}

⇒ 1, 3, 5, 10, 2, 15, 6, 12

n/4 = 8/4 = 2

{1, 5, 2, 6} {3, 10, 15, 12}

{1, 2, 5, 6} {3, 10, 12, 15}

1 , 2 , 5 , 6 , 3 , 10 , 12 , 15

n/8 : 8/8 = 1 .

1 , 2 , 5 , 6 , 3 , 10 , 12 , 15

1 2 5 6 3 10 12 15

1 2 5 6 3 10 12 15

1 2 5 6 3 10 12 15

∴ 1 , 2 , 3 , 5 , 6 , 10 , 12 , 15 .

[Sorted Array] .

PAGE NO.
DATE / /

* Experiment - 5 *

WAP that implements singly linked list for the following operations create, insert by beginning. Insert node at middle, insert node at end.

```
#include <stdio.h>
#include <stdlib.h>
void create();
void display();
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("5. Insert position\n");
printf("6. Exit \n");
printf("Please enter your choice:");
scanf("%d", &choice);
```

```
switch (choice) {
    case 1: create(); break;
    case 2: display(); break;
    case 3: insert_begin(); break;
    case 4: insert_end(); break;
    case 5: insert_pos(); break;
    case 6: exit(0);
    default: printf("Please enter valid option from
                the options\n");
    }
}
return 0;
}
void create() {
    int n, val;
    struct node *newnode, *temp;
    printf("Enter no. of nodes:");
    scanf("%d", &n);
    for (int i=0; i<n; i++) {
        printf("Enter value for node %d:", i+1);
        scanf("%d", &val);
        newNode = (struct node*) malloc(size of
                struct node));
        newNode->info = val;
        newNode->next = NULL;
        if (start == NULL) {
            start = newNode;
            temp = start;
```

```
} else {
    temp -> next = newNode;
    temp = newNode;
  }
}
printf("List created:\n");

id display () {
struct node * temp = start;
    if (start == NULL) {
        printf("List is empty:\n");
        return;
    }
    printf("List :");
    while (temp != NULL) {
        printf("List is empty :\n");
        return 0;
    }

    printf("List :");
    while (temp != NULL) {
    printf("List is empty :\n");
        return
}
    printf("NULL\n");

id insert_begin () {
    int val;
    printf("Enter value :");
    scanf("%d", &val);
```

```
struct node * newNode = (struct node *) malloc
                        (struct node) );

newNode  → info = val;
newNode  → next = start;
    start = newNode

printf(" %d inserted at beginning in", val);
}

void insert_end ()
{
    int val;
printf("Enter value");
scanf("%d", &val);

struct node * newNode = (struct node *) malloc
                        (size of struct node));
newNode → info = val;
newNode → next = NULL;

if (start = NULL) {
    start = newNode;
} else {
    struct node * temp = start;
    while (temp → next != NULL)
    temp = temp → next;
    temp → next = newNode;
}
printf("%d inserted at end in", val);
```

```
}
if (temp == NULL) {
    printf("Position not found.\n");
}

struct node * newNode = (struct node *) malloc
                        (sizeof (struct node));

newNode → info = val;
newNode → next = temp → next;
temp → next = newNode;

printf("%d inserted after position
        %d\n", val, pos);
```

Nov 3/11

* Experiment 6 *

WAP to perform operations on doubly linked list

```
#include <stdio.h>
#include <stdlib.h>
void create ();
void display ();
void insert_begin ();
void insert_end ();
void search_node ();
int count_nodes ();
struct node
{
    int info;
    struct node * next;
    struct node * prev;
};
struct node * start = NULL;
int main() {
    int ch;
    while (1)
{   printf("\n create \n 2 :- Display \n 3 :- Insert
    at beginning \n 4 :- Insert at end \n 5 :- Insert
    at any position \n 6 :- Search node \n 7 :-
    count nodes \n 8 :- Enter your choice ");
    scanf("%d", &ch);
    switch (ch)
{   case 1 : create (); break;
    case 2 : display (); break;
    case 3 : insert_begin (); break;
    case 4 : insert_end (); break;
```

Experiment 6

perform operations on doubly list

```
<stdio.h>
<stdlib.h>
ue ();
play ();
sert_begin ();
er_end ();
h_node ();
t_nodes ();
e

%d
  node * next;
  node * prev;

node * start = NULL;
in()) {
  ch.
ile (1)
f("1. Create 1n2:- Display 1n 3 :- Insert
beginning 1n4:- Insert atend 1n5 :- Insert
any position 1n6 :- Search node 1n7 :-
nt nodes 1n8 :- Enter your choice ");
nf("%d", &ch);
tch (ch)
se 1 : create (); break;
e 2 : display (); break;
e 3 : insert_begin (); break;
e 4 : insert_end (); break;
case 5 : insert_any (); break;
case 6 : search_node (); break;
}
    return 0;
}
void create()
{ struct node * temp, *ptr;
  temp = (struct node *) malloc (sizeof (struct node));
  if (temp == NULL)
  { printf("memory allocation failed \n");
    return 0;
  }
  printf("Enter details : ");
  scanf("%d", &temp -> info);
  temp -> next = NULL;
  temp -> prev = NULL;

  if (start == NULL)
  { start = temp;
  }
  else
  { ptr = start
    while (ptr -> next != NULL)
    { ptr = ptr -> next;
    }
    ptr -> next = temp;
    temp -> prev = ptr;
  }
}
```

```
void display()
{   struct node * ptr;
    if (start == NULL)
    {   printf("\n empty list :");
        return;
    }
    ptr = start;
    printf(" list elements are :");
    while (ptr != NULL)
    {   printf("\n %d", ptr -> info);
        ptr = ptr -> next;
    }
    printf("\n");
}
void insert_beign ()
{   struct node * temp;
    temp = (struct node *) malloc (sizeof
                (struct node));
    if (temp == NULL)
    {   printf("\n memory Allocation failed: \n");
        return;
    }
    printf(" Enter details :");
    scanf("%d", & temp -> info);
    temp -> next = NULL;
    temp <- prev = NULL;

    if (start == NULL)
    {   start = temp;
    }
```

```
else
{   ptr = start;
    while (ptr -> next != NULL

if (start != NULL)
{   start -> prev = temp;
    start = temp;
    }
}
    void insert end ()
{     create ();
    }
    void insert - any ()
{     int pos, i = 1;
    struct node * temp, * ptr;

if (start != = NULL)
{   start -> prev = temp;
    start = temp;
    return;
    }
}
}
    void search_node ()
{   int key, pos = 1;
    struct node * ptr = start;
    if (start == NULL)
    {
        printf("In List is empty");
        return 0;
    }
```

```
printf(" Enter elements to search : ");
scanf("%d", &key);
while (ptr -> info != key)
{   if (ptr -> info == key)
        {
        printf(" Element %d found at position
                %d \n", key, pos);
        return 0;
    }
        ptr = ptr -> next;
        pos++;
    }
    printf(" Element %d not found \n", key);
}
int count = node()
    int count = 0;
    struct node * ptr = start;
    while (ptr != NULL)
{       count++;
        ptr = ptr -> NULL;
    }
        return count;
}
```

PAGE No.
DATE / /

* Experiment - 7 *

```
#include <stdio.h>
# define max 5
void push (int);
int pop ()
void display ();
int stk [max];
int top = -1;
int main () {
        int ch, data;
        do
{ printf("\n1: Push \n2: Pop \n3: display \n4: exit );
  printf("\n Enter your choice:");
  scanf("%d", &ch);
  switch (ch)
  {     case 1:
        printf("Enter data to push:");
        scanf("%d", &data);
        push (data);
        break;

        case 2:
        data = pop ();
printf("data popped is %d", data);
        break;

        Case 3:
          display ();
          break;
```

```
case 4:
    printf("\n Exiting ");
    break;
default:
    printf("\n Invalid choice entered");
}   while (ch!=4);
    return 0;
}
void push (int data)
{   if (top == max-1)
    {
        printf("\n Stack is full :");
    }
    else
    {
        top++;
        stk[top] = data;
        printf("\n data inserted \n");
    }
}
int pop()
{   int data = 0;
    if (top == -1)
    {
        printf("\n Stack is empty :");
    }
    else
    {
        data = stk[top];
        top--;
    }
```

```
    return data;
}
    void display ()
{
    int i;
    if (top == -1)
{
    printf("Stack is empty:");
}
    else
    {   printf("Stack is :");
        for (i=0; i<=top; i++)
{
    printf("%d", &stk[i]);
}
}
}
```

Experiment - 8

(i) Convert infix to postfix expression

```
#include <stdio.h>
#include <ctype.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;
  return 0;
}
int main (){
  char exp [100];
  char * e, x;
  printf("Enter the expression");
```

```
scanf("%s", &exp);
printf("\n");
e = exp;

    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 (priority(stack[top]) >= priority(*e))
          printf("%c", pop());
      }
        e++;
}     while (top != -1)
      {
        printf("%c", pop());
      }
        return 0;
}
```

[ii] Evaluate postfix expression:

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#define SIZE 40

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("\nEnter a postfix expression: ");
scanf("%s", postfix);

for (i=0; postfix[i] != '\0'; i++)
{   ch = postfix[i];

    if (isdigit(ch))
{
        push(ch-'0');
}
```

```
else if (ch == '+' || ch == '-' || ch == '*' || ch == '/')
        b = pop();
        a = pop();

    switch (ch)
    case '+' :
        result = a + b;
        break;
    case '-' :
        result = a - b;
        break;
    Case '/' :
        result = a/b;
        break;
    case '*' :
        result = a * b;
        break
    Case '%' :
        result = a % b;
        break;
    }
        push (result);
    }
}
    pEval = pop();
printf("In The postfix evaluation is : %d \n" pEval);
    return 0;
}
```

```
void push (int n)
{
   if (top < SIZE - 1)
   {
      stack [++top] = n;
   }
      else {
            printf("stack is full!\n");
            exit (-1);
         }
}
   int pop()
   {     int n;
         if (top > -1)
         {
            n = stack [top]
            stack [top--] = n-1;
            return n;
         }
      else
   {     printf("stack is empty!\n");
         exit (-1);
      }
   }
```

PAGE No.
DATE / /

(5) Convert infix expression into prefix using stack. Infix expression : A + [B * C - (D / E * F)] * G

Input	Stack	Output
A		A
*	*	A
(	* (	A
G	* (	A G
*	* (*	A G
(	* (* (	A G
F	* (* (	A G F
^	* (* (	A G F
E	* (* (^	A G F
/	* (* (^ /	A G F E
D	* (* (^ /	A G F E ^
)	* (	A G F E ^ D
-	* (-	A G F E ^ D /
C	* (-	A G F E ^ D / * G
*	* (- *	A G F E ^ D / * G C
B	* (- *	A G F E ^ D / * G C B
)	*	A G F E ^ D / * G C B * -
*	+	A G F E ^ D / * G C B * - *
A	+	A G F G ^ D / * G C B * - * A +

Evaluate prefix expression using stack

prefix expression : + - * 12/42 | $42

Input	Stack	Output
2		2
4		2, 4
$	2$4=16	16
1		16, 1
2		16, 1, 2
4		16, 1, 2, 4
/	2/4=0.5	16, 1, 0.5
2		16, 1, 0.5, 2
1		16, 0.5, 2, 1
+	2+1=3	16, 0.5, 3
*	0.5*3=1.5	16, 1.5
-	1-1.5=-0.5	16, -0.5
+	16+(-0.5)=15.5	15.5

* Algorithms

1. Infix → Postfix

Step 1 : Scan infix expression from left to right
Step 2 : Operand → add to postfix directly
Step 3 : '(' → push to stack
Step 4 : ')' pop from stack until '(' is found
Step 5 : Operator → pop all operators from stack of greater or equal precedence then push current operator

Step 6 : After scanning → pop remaining operator to postfix

2. Infix → Prefix

Step 1 : Reverse the infix expression
Step 2 : Swap '(' with ')' in reversed exp.
Step 3 : Convert the new expression to postfix (using same algorithm as above)
Step 4 : Reverse result → that the prefix expression

Evaluation of Postfix

Step 1 : Scan left → right
Step 2 : operand → push into stack
Step 3 : operator → pop top 2 value, apply value apply operator push result back.

Step 4: At end → only one value remains in stack is result.

4. Evaluation of Prefix
Step 1: Scan expression right → left
Step 2: operand → push onto stack
Step 3: Operator → pop top 2 values, apply operation, push result back
Step 4: At end → only 1 value remains in stack → result.

PAGE No.
DATE / /

* Experiment - 12 *

```
#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 insertEdge(int arr[V][V], int i, int j){
    arr[i][j] = 1;
    arr[j][i] = 1;
}

void printAdjMatrix(int arr[V][V]) {
    int i, j;
printf("Adjacency matrix:\n");
 for (i=0; i<V; i++) {
 for (j=0; j<V; j++) {
     printf("%d", arr[i][j]);
}
     printf("\n");
  }
}

   int main() {
   int adjmatrix[V][V];
   int (adjmatrix);
```

PAGE No.
DATE / /

```
insertEdge(adjmatrix, 0, 1);
insertEdge(adjmatrix, 0, 4);
insertEdge(adjmatrix, 1, 2);
insertEdge(adjmatrix, 1, 3);
insertEdge(adjmatrix, 1, 4);
insertEdge(adjmatrix, 2, 3);
insertEdge(adjmatrix, 3, 4);

printAdjMatrix(adjmatrix);
return 0;
}
```

Output:

Adjacency matrix:

```
0 1 0 0 1
1 0 1 1 1
0 1 0 1 0
0 1 1 0 1
1 1 0 1 0
```

Nahi

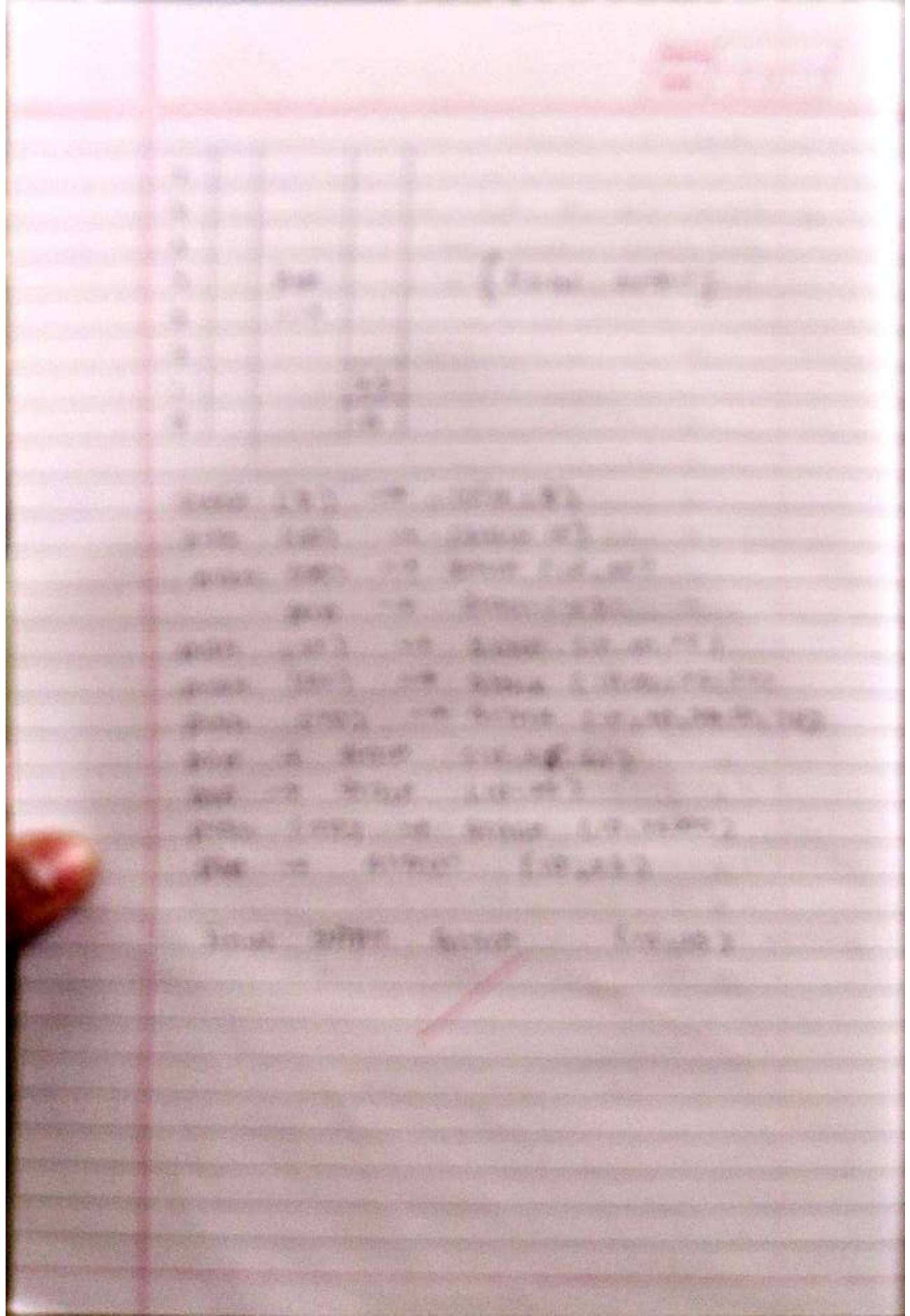
PAGE No.
DATE / /

Exp - 7 Conti.

① Stack Size - 8 for push (10), push (20), push (25), push (50), push (70) pop, pop, push (100) pop & draw final output:

7							
6							
5							
4							
3		push(70) →	70	pop push →		pop →	
2	50		50		50		
1	25		25		25		25
0	10		10		10		10

7							
6							
5							
4		push(10)		push(20)		Push(20	
3		→		→		→	
2							
1					20		25
0			10		10		10



7				
6				
5				
4		POP		[Final output]
3		→		
2				
1			25	
0			10	

Stack [8] → (size, 8)

push (10) → (Stack 10)

push (20) → Stack (10, 20)

pop → Stack (10)

push (25) → Stack (10, 20, 25)

push (50) → Stack (10, 20, 25, 50)

push (70) → Stack (10, 20, 25, 50, 70)

pop → Stack (10, 25, 50)

pop → Stack (10, 25)

push (100) → Stack (10, 25, 100)

pop → Stack (10, 25)

Final output Stack : (10, 25)

PAGE No.
DATE / /

Experiment 9

Perform primitive operation on linear queue.

```
#include <stdio.h>
#define MAX 100
int queue [MAX];
int front = -1   rear = -1;
void enqueue (int n);
void dequeue ();
void display ();
int main () {
printf("Enter '1' to enqueue :\n");
printf("Enter '2' to dequeue :\n");
printf("Enter '3' to display :\n");
printf("Enter '4' to exit :\n");
while (1) {
printf("Enter choice");
scanf("%d", &choice);
switch (choice) {
case 1: printf("Enter value").
        scanf("%d", &value);
        enqueue (value);
        break;
case 2: dequeue ();
        break;
case 3: display ();
        break;
case 4: printf ("exiting :-\n");
        return ()
}
```

default : printf("Invalid input \n");
}
}

void enqueue (int n){
if (rear = max-1)
printf("Queue overflow \n");
} else {
if (front == -1) {
front = 0; }

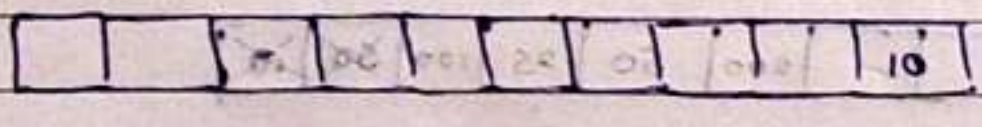
rear ++;
queue [rear] = n;
printf("%d", enqueue \n", n);
}
}

void display (){
if (front == || front > rear) {
printf("Queue is empty \n");
} else {
for int (i = front ; i < rear; i++) {
printf ("%d", queue [i]);
}
printf("\n");
}
}

PAGE No.
DATE / /

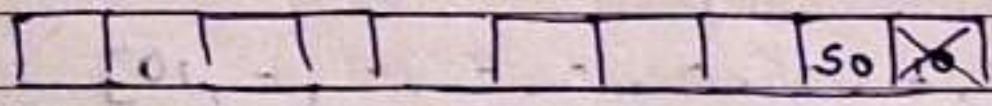
② Perform following operation on linear queue in array size 10 with this operation insert (10) insert (50) delete insert (100) insert (20) delete insert (25) insert (200)

insert 10



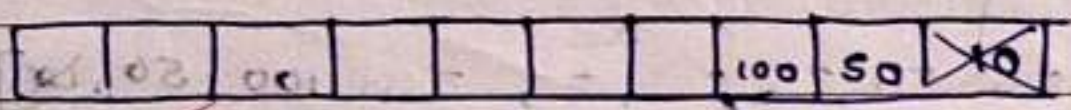
Rear = 0 Front = 0

insert 50

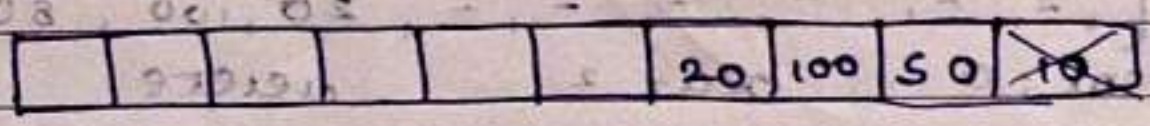


Rear = 1 Front = 0

Delete



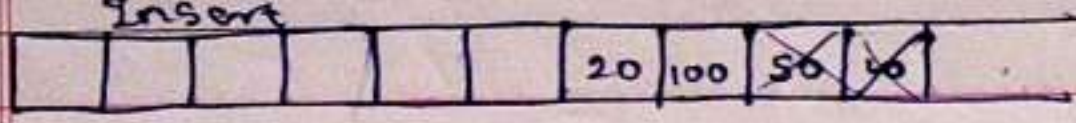
insert 100



Rear = 2 Front = 0

Delete

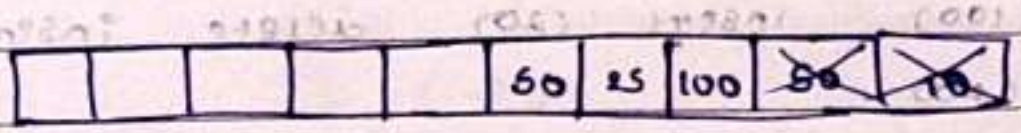
Insert



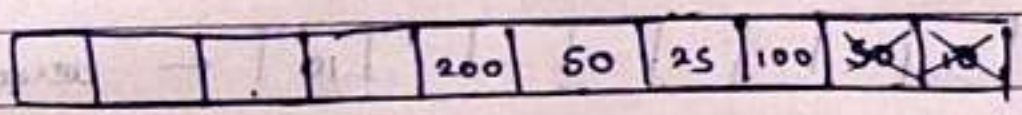
PAGE No.
DATE / /

Rear = 1 Front = 0

insert 25



Insert 200



rear = 2 front = 0

insert (10) empty front = 10 rear = -1

[10, -, -, -, -, -, -, -, -, -]

insert (50) front = 0 , rear 0

[-, -, -, -, -, -, -, -, 50,]

front = 0 , rear = 1 insert 100

[-, -, -, -, -, -, -, 100, 50,]

front = 0 rear = 1 insert 20

[-, -, -, -, -, -, 20, 100, 50,]

front = 0 rear 2 delete

[-, -, -, -, -, -, 20, 100, 50,]

front = 0 , rear = 1

PAGE No.
DATE / /

insert (25)

[-, -, -, -, -, 25, 20, 100, 80, 10]

Front : 0 rear = 2

insert (200)

[-, -, -, -, 200, 25, 20, 100, 80, 10]

Front = 0

rear = 3.

10/11

PAGE No.
DATE / /

* Experiment - 10 *

```
#include <stdio.h>
#define SIZE 5

int queue[SIZE];
int front = -1, rear = -1;

void insert(int value) {
    if ((rear+1) % SIZE == front) {
    printf("Queue is fully\n");
} else {
        if (front == -1)
            front = 0;
        rear = (rear+1) % SIZE;
        queue[rear] = value;
    printf("Inserted %d\n", value);
    }
}

    void delete() {
if (front == -1) {
    printf("Queue is empty\n");
} else {
    printf("Deleted: %d\n", queue[front]);
    if (front == rear)
        front = rear = -1;
    else
    front = (front+1) % SIZE;
}
}
```

PAGE No.
DATE / /

```
void display() {
  if (front == -1) {
    printf("Queue is empty\n");
  } else {
    printf("Queue elements:");
    int i = front;
    while (1) {
      printf("%d", queue[i]);
      if (i == rear)
        break;
      i = (i+1) % SIZE;
    }
    printf("\n");
  }
}

int main() {
  int choice, value;
  while (1) {
    printf("\n--- Circular Queue Menu ---\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch choice {
    case 1:
      printf("Enter value to insert");
      scanf("%d", &value);
      insert(value);
      break;
```

```
case 2 :
    delete ();
    break;
case 3 :
    display();
    break;
case 4 :
    return 0;
default:
    printf("Invalid choice!\n");
}
}
```

PAGE No.
DATE / /

* Theory:

What are advantages of circular queue
The size of the queue is fixed, so if it gets full no more elements can be added.

Its very more complicated to implement than a single linear queue because of circular wrap around.

If we don't handle to full condition properly it can cause overflow errors.

Write a algorithm for insertion & deletion on circular queue

S1 if front = 0 & rear = max - 1
"queue overflow"
if front = rear + 1
"queue overflow"

S2 If front = -1 & rear = -1
front = rear = 0
else if rear = max - 1 & front != 0
rear = 0
rear = rear + 1
end

S3 queue [rear] = val

S4 exit.

* Difference between Stack & Queue.

Stack	Queue
It is linear data structure in which elements are pushed & popped from end called 'top' of stack	It is linear data structure in which elements inserted from rear end & deleted from front end
In stack only one pointer variable called top	The queue two position variables are used called front & rear.
If top = max-1 then it is Stack Overflow	If rear = max-1 then it is queue overflow

PAGE No.
DATE / /

* Assigment *

* Data structure : List & explain operation on DS
* Find order of the foll function

```
for (i=0; i<n ; i++)
printf("Hello");
for (j=1; j<n ; j=j*2)
    printf("Hi");
for (i=n; i>0; i=i/2)
for (j=1; j<n ; j=j*2)
    printf("Hi");
```

n is searching technique search key=9 in the given ways using efficient seach technique 9, 15, 4, 9, 25, 6

Sort following element using radix sort
25, 651, 43, 222, 91

Write a note on Singly link list

Convert infix to prefix
(A+B*C-(D/E)$F/G*H)

Evaluate given postfix expression
3, 6, 4, 7, +, * 6 / 3

Draw expression Tree using given expression.
([A+B)*C)/(D$(E-F))

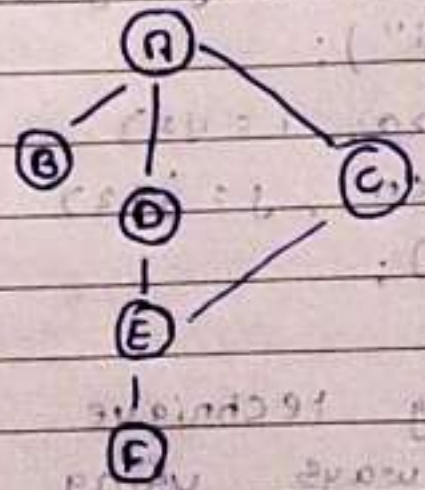
Construct BST & Traverse in pre, Pin, post orders
52, 40, 60, 45, 35, 65, 55, 100, 93, 36, 49

PAGE No.
DATE / /

Refer given graph & ans the question.

- Adjacency matrix representation.
- Adjacency list representation.

Path from A → M. in degree & node of



PAGE No.
DATE / /

* Experiment - 11 *

```
#include <stdio.h>
#include <stdlib.h>
struct node {
int data;
struct node * left, * right;
};
struct node * create (int value) {
struct node * newnode = (struct node *)
malloc (sizeof (size of (struct node));
newNode → data = value;
newNode → left = newnode → right = NULL;
return newNode;
}
struct node * insert (struct node * root
                      int value) {
if (root == NULL) {
return create Node(value);
if (value < root → data)
root → left = insert (root → left value);
else if (value → root → data)
}
void inorder (struct node * root)
if (root ! NULL) {
inorder (root → left);
print ("%d", root → data);
```

```
}
void postorder (struct node * root) {
if (root != NULL) {
postorder (root → left);
postorder (root → right);
printf("%d", root → data);
}
}
struct node * Search (struct node * root, int
if (key < root → data)
return search (root → left, key);
return search (root → right, key);
}
}
struct node * Search (struct node * inorder) {
if (key < root → data)
return Search (root → left, key);
return search (root → right, key);
}
struct node * findmin (struct node
* root) {
while (root → left != NULL)
root = root → left;
return root;
}
```

```
struct node * findmin (struct no
{
while (root → left != NULL)
root = root → left;

printf("In Binary Tree menu\n");
printf(" 1. Insert in 2. Display (Inorder
printf(" 3 - Search 4. Delete in 5. E
Scanf("%d", &choice);
switch(choice) {

case 1: printf("Enter value to inse
scanf("%d", &value);
root = insert (root, value
break;
case 2: printf("Inorder Traversa
inorder (root);
printf ("\n");
break;
case 3: printf("Preorder Tra
inorder (root);
print ("\n");
break;
case 4: printf("postorder
postorder (root);
printf("\n");
break;
```

```
struct node * findmin (struct node * root)
{
   while (root → left != NULL)
   root = root → left
```

```
printf("\n Binary Tree menu\n");
printf(" 1. Insert \n 2. Display (Inorder) \n 3. Display");
printf(" 5 - Search \n 6. Delete \n 7 Exit");
scanf("%d", &choice);
   switch(choice) {
```

```
case 1: printf("Enter value to insert :");
        scanf("%d", &value);
   root = insert (root, value);
        break;
case 2: printf(" Inorder Traversal");
        inorder (root);
        print ("\n");
        break;
case 3: printf(" Preorder Traversal:");
        inorder (root);
        print ("\n");
        break;
case 4: printf(" Postorder Traversal:");
        postorder (root);
        printf("\n");
        break;
```

```
case 5 : printf("Enter value to search: ");
         scanf("%d", &value);
         if (search(root, value))
         printf("%d found \n", value);
         else {
         printf("%d not found \n", value);
         break;
case 6 : printf("Enter value to delete: ");
         scanf("%d", &value);
         root = delete node(root, value);
break
case 7 :
         exit(0);
default : printf("Invalid choice \n");
}
return 0;
}
```

```
2. C Program Binary Tree Trav

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

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

struct node *createNode
struct node *newNode = (s
          (size(struct no
newNode->data = value
newNode->left = NULL
newNode->right = NUL
return newNode;
}

struct node *insert(stru
if (root == NULL)
   return createNode(v
else if (value < root->
   root->left = insert
else if (value > root->
   root->right = inser
return root;
}
```

PAGE No.
DATE / /

2. C Program : Binary Tree Traversal

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

struct node{
    int data;
    struct node *left;
    struct node *right;
};
struct node *createNode(int value){
  struct node *newNode = (struct node*)malloc
                (sizeof(struct node));
  newNode->data = value;
  newNode->left = NULL;
  newNode->right = NULL;
  return newNode;
}
struct node *insert(struct node *root, int value){
  if (root == NULL)
    return createNode(value);
else if (value < root->data)
  root->left = insert(root->left, value);
else if (value > root->data)
  root->right = insert(root->right, value);
  return root;
}
```

PAGE No.
DATE / /

```
void inorder (struct node * root) {
  if (root != NULL) {
    inorder (root → left);
    printf ("%d", root → data);
    inorder (root → right);
  }
}

void preorder (struct node * root) {
  if (root != NULL) {
    printf("%d", root → data);
    preorder (root → left);
    preorder (root → right);
  }
}

void postorder (struct node * root) {
  if (root != NULL) {
    postorder (root → left);
    postorder (root → right);
    printf("%d", root → data);
  }
}

int main() {
  struct node * root = NULL;
  int choice, value;
  while (1) {
```

PAGE No.
DATE / /

```
printf("\n\n -- Binary Tree menu --\n");
printf(" 1. Insert Node \n");
printf(" 2. Inorder Traversal\n");
printf(" 3. Preorder Traversal\n");
printf(" 4. Postorder Traversal\n");
printf(" 5. Exit\n");
printf("Enter your choice:");
scanf("%d", &choice);

switch (choice){
  case 1:
    printf("Enter value to insert:");
    scanf("%d", &value);
    break;
  case 2:
    printf("Inorder Traversal:");
    inorder(root);
    break;
  case 3:
    printf("Preorder Traversal:");
    preorder(root);
    break;
  case 4:
    printf("Postorder Traversal:");
    postorder(root);
    break;
  case 5:
    exit(0);
  default:
    printf("Invalid choice!\n");
}
return 0;
}
```

PAGE No.
DATE / /

Q.2) What are the advantages of circular queue? Show diagramatic representation

1. Efficient memory utilization:
unlike linear queue, space is reused after deletion.

2. Prevents overflow:
In linear queue, once rear reaches the end no more insertion are possible even if there are empty slots at the front.

3. Fixed size for optimized work: Well for fixed size memory

4. Faster operations:
Insertion & deletion are O(1)

MAY 10th

PAGE No.
DATE / /

* Experiment -12 *

Write a program to implement following operations on Graph: creation & display.

```
#include <stdio.h>
#define MAX 10

void createGraph (int graph[MAX][MAX], int vertices){
int i, j, edges, src, dest;

for (i=0; i<vertices ; i+1){
for (j=0; j<vertices ; j++){
  graph [i] [j] =0;
  }
}
printf("Enter Number of edges: ");
scanf("%d", & edges);
graph [src] [dest ]:1 ;
graph [dest ] [src ] = 1;
}
}

void displayGraph (int graph[MAX][MAX], int
        vertices) {
  int i ,j ;
printf("In Adjacency Matrix :\n");
for (i = 0; i < vertices ; i++) {
for (j= 0; j < vertices ; j++) {
  printf(" %d", graph [i][j] );
}
  printf("\n");
}
}
```

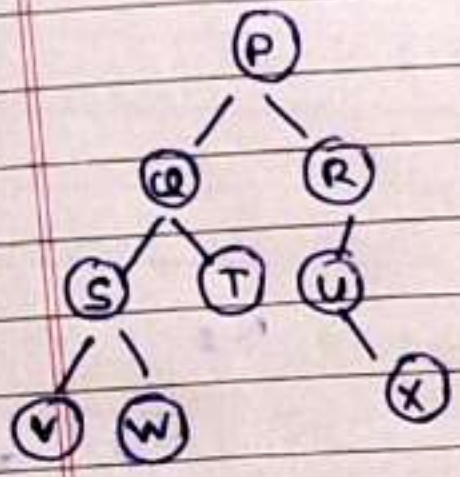
PAGE No.
DATE / /

```
int main () {
    int graph [MAX] [MAX];
    int vertices;

    printf("Enter number of vertices:");
    scanf("%d", &vertices);

    createGraph (graph, vertices);
    displayGraph (graph, vertices);

    return 0;
}
```

2nd
* 12Th Exp on Another Book *

PAGE No.
DATE / /

1

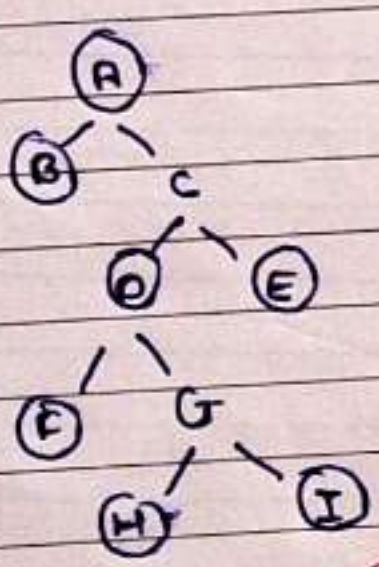


Preorder: P → Q → S - V - W - T - R - U - X

Inorder - V - S - W - Q - T - P - X - U - R

Postorder - V - W - S - T - Q - X - U - R - P

2



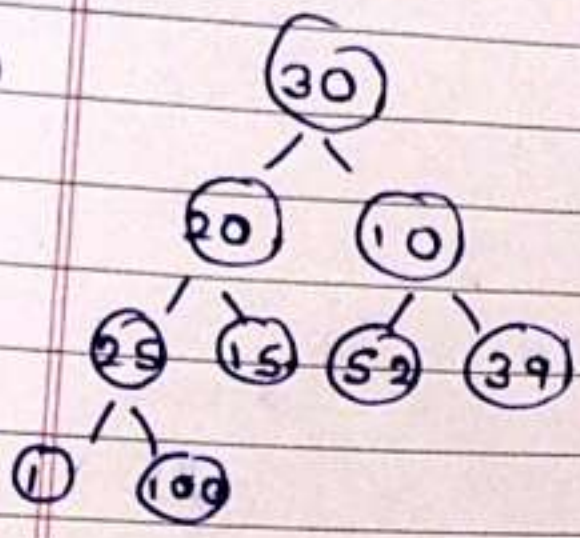
pre order - A → B - C - D - F - G → H → I → E

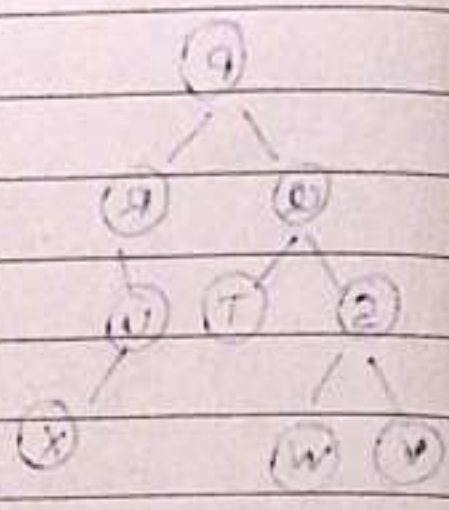
Postorder - B → F → H → I → G → D → E → C → A

Inorder - B → A → F → D → H → G → I → C → E

PAGE No.
DATE / /

(ब)





Preorder : 30-20-25-1-15-10-52-39-100

Inorder : 1-25-100-20-15-52-10-39-1

Postorder : 1-100-25-15-20-52-39-10-30

(स)