Experiment 1

WAP to implement array operations -

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 the 10 numbers is : %.2f\n", a);
return 0;
}

Output :

Enter 10 numbers :
2
5
7
4
8

3
2
7
7

The average of 10 numbers is : 5.30

b) Display the following pattern -

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

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

Output -

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

Find first repeating number in an array

```
#include <stdio.h>
int main ()
{
    int n, j;
    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++)
    {
        for (int j = 1; 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.
```

d) - 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
```

e) 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 elements : 5
Enter 5 elements

2
4
7
9
5

Array After squaring :

2 4 49 81 6

21/7/25

Experiment 2

a) Search data using linear search consider the quantities to perform using 56, 36, 89, 57, 1, 0, 61, 59. Search no 1 then 55

```
#include <stdio.h>
int main()
{
int a[] = { 56, 36, 89, 57, 1,
            61, 59};
int i, n, f;
printf(" Array :");
for ( i=0 ; i<7 ; i++)
    printf("%d", a[i]);
printf(" Enter number to search");
scanf("%d", &n);
f=0;
for (i=0 ; i<7 ; i++)
{
if (n == a[i])
{
printf(" Number found at
        index %d ", i);
f=1;
break;
}
}
```

```
if ( f s = 0)
        printf (" Number not found ");
    return 0;
}
```

Output

```
Array : 56 36 89 57 1 0 61 59
        Enter number to search 1
        Number found at index 4
        Enter number to search 55
        Number not found
```

Using Binary search

```
# include <stdio.h>
int main() {
int arr [ ] = {0, 1, 36, 56, 57, 5
              , 67, 89}
int n = 8;
int ; Rm, low = 0, h = n-1,
mid ;

printf(" Enter the ; tem to sear
    " );
scanf (" %d, & ; tem);

while (low ; <= high) {
    mid = (low + high) / 2;
    found = 1;
    printf (" Item %d found
      at position %d ", item, mi
      );
break ;

} else if ( item < arr [mid])
{
    high = mid - 1 ;
} else {
    low = mid + 1;
}
if (! found) {
    printf (" Item %d
      not found in the
```

M T W T F S S

Page No.:

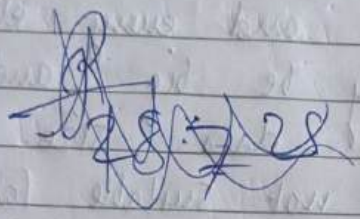
Date:

FIONA

~~While True:~~

~~# Show menu and handle choices~~

~~• Loop continues until user chooses to exit~~



Difference between linear and Binary search.

Linear search	Binary search
Checks all elements of array.	Divides array & searches in halves
Works on both sorted and unsorted array	Works only on sorted array
Sequential scan	logarithmic time
Simple and Straight forward	Slightly more complex
Slower on larger database	much faster on larger database

Limitation of linear search in terms of Time complexity

Inefficient for larger Data set :
Linear search consumes ~~more~~ more time as it checks each and every element as the array may be big and it also depends on array data size of data types. Hence it is not suitable for larger arrays.

No early stopping
If the element is at the end of the array or not

25/7/25

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", 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 :

1
2
3
4
5

Reversed array

5
4
3
2
1

Algorithm

1) Start
2) Input variable 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 elements using
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;
    printf (" Enter no of elements :");
    scanf (" %d ", & a [i]);

    for (int i = 0, i < n ; i++)
    {
        for (in 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 :

1
2
7
9
2

Total duplicate elements : 1

Algorithm

1) Start
2) Declare a for array n for no of elements in array, c for count of duplicate elements
3) Input the no of elements
4) Print the statement to ask user to input elements
5) Input elements for the array
6) use to for loops and use if condition to check if a[i] == a[j] if true (++c)
7) Print c (total no. of duplicate elements)
8) Stop

3) WAP in C to print all unique elements in an array appear once ↑ array

```
#include <stdio.h>
int main()
{
    int a[50], n, v;
    printf("Enter no of elements :");
```

```
scanf("%d", &n);
printf("Enter no of 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++)
{
    if(i!=j && a[i]==a[j])
    {
        u=0;
        break;
    }
}
    if(u)
        printf("%d", a[i]);
    }
    return 0;
}
```

Output

```
Enter number of elements : 3
Enter 3 elements :
1
2
1
Unique elements are :
2
```

Algorithm

1) Start
2) Declare a for array, n for no of elements a for unique, i and j for loop
3) Enter no of elements
4) Input elements using for loop
5) Check if they are unique
6) Print array elements
7) Stop

4) WAP to seperate odd and even integers into seperate 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", 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])
    printf (" odd elements : \n ");
    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
```

5) WAP to find second smallest element in given array.

```
#include <stdio.h>
int main()
{
    int a[50], n;
    int f = INT_MAX, s = INT_MAX;
    printf("Enter no. of elements");
    scanf("%d", &n);
    printf("%d elements", n);

    for(int i = 0; i<n, i++)
        scanf("%d", &a[i]);
        if(a[i]<f)
        {
            s=f;
            f=a[i];
        }
        if(s != INT_MAX)
            printf("No smallest element");
        else
            printf("Second smallest: %d",
                n, s);
        return 0;
}
```

Output -

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

Second Smallest : 2

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, 7 };
    int n = size of (a) / size of (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 - i - 1 ; j++)
    {
      if (a [j] > a [j+1])
      {
        t = a [j];
        a [j] = a [j+1];
        a [j+1] = t;
      }
    }
  }
}
```

```
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, 2 };
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;
            }
        }
        b = a[i];
        a[i] = a[m];
        a[m] = b;
    }
    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 comparisions required in bubble sort method of the following data having 5 numbers : 100, 200, 300, 400, 500

given number - 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 comparision take place as the given numbers are already sorted

Algorithm

1) Start with the list of numbers
2) Compare each ~~other~~ pair of adjecent element greater than second 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 element are in the ~~correct~~ correct order (after n-1 passes)

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

```
#include <stdio.h>
void selectionSort(int arr[], int n) {
    for(int i = 0 ; i < n-1 ; i++) {
        int minIdx = i;
        for(int j = i+1 ; j < n ; j++)
            if(arr[j] < arr[minIdx])
                minIdx = j;
        }
    }
    int temp = arr[i];
    arr[i] = arr[minIdx];
    arr[minidx] = temp;
    printf("Iteration %d : ", i+1);
    for(int k = 0 ; k < n ; k++) {
        printf("%d ", arr[k]);
    }
    printf("\n");
}
}
int main() {
    int arr[] = { 500, -20, 30, 14
                , 50 };

    int n = sizeof(arr) / sizeof(arr
                [0]);
```

```
selectionSort(arr, n);
printf("Final sorted array: ");
for(int i = 0; i < n; i++) {
    printf("%d ", arr[i]);
}
return 0;
}
```

12/8/25

Experiment 4:

Write 'c' program to sort, from given array using Insertion Sort and Radix Sort

1) Sort element in ascending order using Insertion Sort.

```
#include <stdio.h>

void insertionSort (int arr [], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr [i];
        j = i - 1;

    while (j >= 0 && arr [j] > key) {
        arr [j + 1] = ~~key~~ arr [j];
        j--;
    }
        arr [j + 1] = key;
    }
}
    int main () {
        int arr [] = { 7, 3, 5, 1, 9,
                       8, 4, 6};
        int n = size of (arr) / size of (arr
                [0]);
        insertion Sort (arr, n);
```

```
printf("Sorted array in ascending
        order:");
    for (int i = 0; i < n; i++)
        printf("%d", arr[i]);

    return 0;
}
```

ii) Sort elements in ascending order using Radix sort

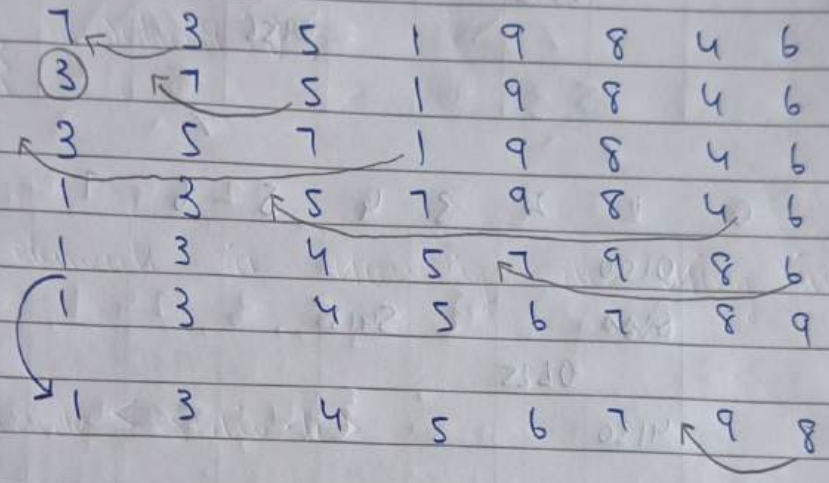
```
#include <stdio.h>

int getMax(int arr[], int n) {
    int max = arr[0];
    for (int i = 1; i < n; i++)
        max = arr[i];
    return max[i];
}

void countSort(int arr[], int n, int exp)
{
    int output[n];
    int count[10] = {0};

    for (int i = 0; i < n; i++)
        count[(arr[i] / exp) % 10]++;

    for (int i = 1; i < 10; i++)
        count[i] += count[i-1];
```

```
for (int i = n - 1; i >= 0; i--)
    output[count[(arr[i] / exp) % 10] - 1] = arr[i];
    count[(arr[i] / exp) % 10]--;
}

void radixSort(int arr[], int n) {
    int max = getMax(arr, n);

    for (int exp = 1; max / exp > 0;
    exp *= 10)
        countSort(arr, n, exp);
}

int main() {
    int arr[] = {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 interaction given the following sequence of numbers: 7, 3, 5, 9, 8, 7, 6



d) Sort the following no. using radix sort

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

	0	1	2	3	4	5	6	7	8	9
0100										
0225										
0390						0225				
4130										
0956										
0099							0956			
5413			5413							009

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

	0	1	2	3	4	5	6	7	8	9
0100		0100								
5413		~~0100~~			5413					
0225			0225							
4130		4130								
0956										0956
0390				0390						
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, 0390, 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							
025			025							
145					145					
006	006									
018		018								
099										099
239				239						

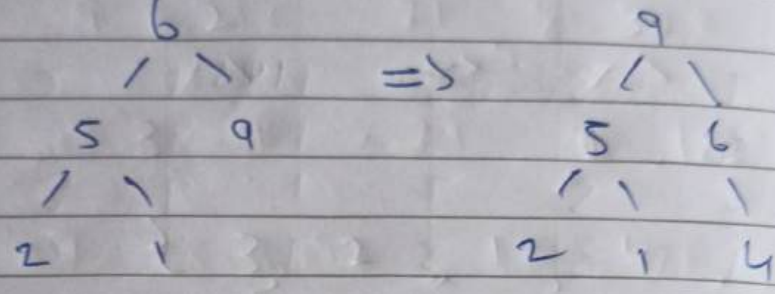
	0	1	2	3	4	5	6	7	8	9
006	006									
018	018									
020	020									
025										
239			239							
145		145								
099	099									

Sorted array : 006, 018, 020, 025, 099, 145, 239

e) Sort the following elements using heap sort

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

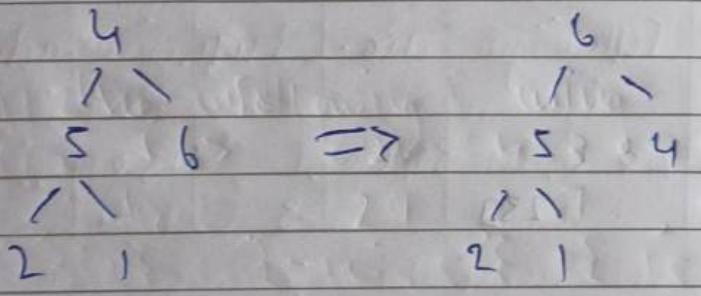
pass 1



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

∴ { 4, 5, 6, 2, 1, 9 }

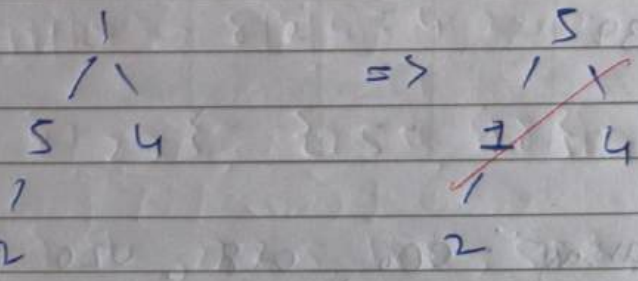
pass 2



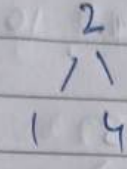
∴ { 6, 5, 4, 2, 1, 9 }

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

pass 3

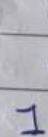


{2, 1, 4, 5, 6, 9}



pass 4 ∴ {4, 1, 2, 5, 6, 9}

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

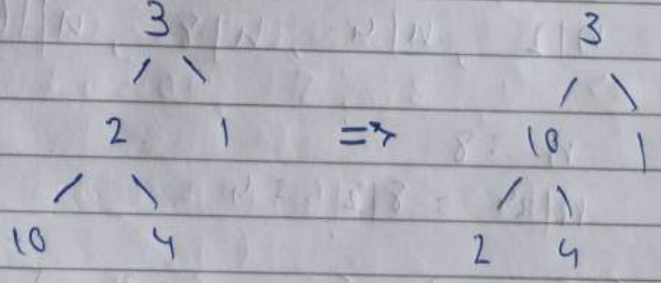


{2, 1, 4, 5, 6, 9}

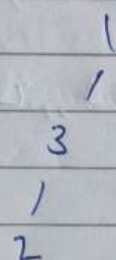
∴ Sorted array : {1, 2, 4, 5, 6, 9}

(ii) 3, 2, 1, 10, 4

pass 1



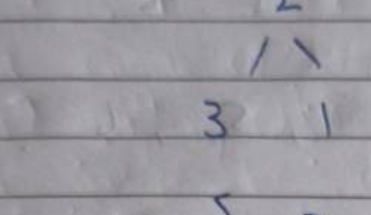
pass 2



$\therefore$ { 5, 3, 1, 2, 10 }

{ 2, 3, 1, 4, 10 }

pass 3



{ 3, 2, 1, 4, 10 }

$\therefore$ { 1, 2, 3, 4, 10 }

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

(f) Sort the following using Shell sort

(i) 22, 18, 29, 7, 9, 55, 21, 8

0 1 2 3 4 5 6 7

9 55 21 8 22 18 29 7

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

n = 8

n/2 = 8/2 = 4

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

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

(i) 22 , 18 , 29 , 7 , 9 , 55 , 21 , 8

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

n/8 = 8/8 = 1

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

7, 9, 18, 21, 8, 22, 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, n/4, n/8, n/16 ... 1

n/2 , n/4

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/2 = 8/2 = 4

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]

Experiment 5: Singly Linked List

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

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

struct node {
    int info;
    struct node *next;
};

struct node *start = NULL;

int main(){
    int choice;
    while (1) {
        printf("\n Menu :");
        printf("\n 1: Create");
        printf("\n 2: Display");
        printf("\n 3: Insert at beginning");
        printf("\n 4: Insert at end");
        printf("\n 5: Insert at any position
        ;
        printf("\n Enter choice :");
        scanf("%d", &choice);
```

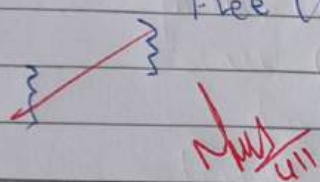
```
Switch (choice) {
    case 1 : Create (); break;
    case 2 : display(); break;
    case 3 : insert_begin (); break;
    case 4 : insert_end (); break;
    case 5 : insert_any (); break;
    default : printf("\n Invalid choice\n");
    }
}
    return 0;
}
void create () {
    struct node *temp, *ptr;

    temp = (struct node *) malloc (sizeof (
            struct node));
    if (temp == NULL) {
        printf ("Memory allocation failed
            \n");
        return;
    }

    printf ("Enter details :");
    scanf ("%d", &temp -> info);
    temp -> next = NULL;

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

```
ptr -> next = temp;
    }
}
void display ()
    struct node *ptr;
    if ( start == NULL) {
    printf (" \n Empty list \n");
    return;
}
    ptr = start;
    print (" list elements : ");
    while (ptr != NULL) {
    printf (" %d", ptr -> info);
    ptr = ptr -> next;
    }
    printf ("\n");
}
void insert_begin () {
    struct node* temp;
    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 = start;
    start = temp;
}
```

```
void insert_end() {
    create();
}
void insert_any() {
    int pos, i = 1;
    struct node *temp, *ptr, *prev;

    if (start == NULL) {
    printf("List is empty, cannot insert
    at position.\n");
    return;
}

    printf("Enter position to insert:");
    scanf("%d", &pos);

    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;

if (pos == 1) {
    temp -> next = start;
    start = temp;
    return;
}
```

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

struct node {
    int data;
    struct Node * prev;
    struct Node * next;
};

struct Node * head = NULL;
struct Node * tail = NULL;

void insertAtBeginning();
void insertAtEnd();
void deleteNodeByValue();
void printList();

int main() {
    int choice
    do {
        print
```

```
ptr = start;
while ( ptr != NULL && i < pos ){
        prev = ptr;
        ptr = ptr -> next;
        i++;
}

if (i == pos){
    prev -> next = temp;
    temp -> next = ptr;
}else{
    printf (" Position out of range \n");
    free (temp);
}
```



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

struct node {
   int info;
   struct node* prev;
   struct node* next;
};

struct node* head = NULL;
struct node* tail = NULL;

void create();
void display_forward();
void display_backward();
void insert_begin();
void insert_end();
void insert_pos();
void delete_node();
void count_nodes();
void search();
```

```
int main() {
    int choice;
    while (1) {
        printf("\nMenu\n");
        printf("1. Create first node\n");
        printf("2. Display forward\n");
        printf("3. Display backward\n");
        printf("4. Insert at the beginning\n");
        printf("5. Insert at the end\n");
        printf("6. Insert at position\n");
        printf("7. Delete node by value\n");
        printf("8. Count nodes\n");
        printf("9. Search for a value\n");
        printf("10. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: create(); break;
            case 2: display_forward(); break;
```

```
            case 3: display_backward(); break;
            case 4: insert_begin(); break;
            case 5: insert_end(); break;
            case 6: insert_pos(); break;
            case 7: delete_node(); break;
            case 8: count_nodes(); break;
            case 9: search(); break;
            case 10: exit(0);
            default: printf("Invalid choice\n");
        }
    }
    return 0;
}

// Create (insert at end)
void create() {
    struct node* temp = (struct
node*)malloc(sizeof(struct node));
    if (!temp) {
        printf("Memory allocation failed\n");
        return;
```

```
    }
    printf("\nEnter data: ");
    scanf("%d", &temp->info);
    temp->next = NULL;
    temp->prev = tail;

    if (head == NULL) {
        head = tail = temp;
    } else {
        tail->next = temp;
        tail = temp;
    }
}

void display_forward() {
    struct node* ptr = head;
    if (head == NULL) {
        printf("\nList is empty\n");
        return;
    }
    printf("\nList (forward):\n");
```

```
    while (ptr != NULL) {
        printf("%d <-> ", ptr->info);
        ptr = ptr->next;
    }
    printf("NULL\n");
}

void display_backward() {
    struct node* ptr = tail;
    if (tail == NULL) {
        printf("\nList is empty\n");
        return;
    }
    printf("\nList (backward):\n");
    while (ptr != NULL) {
        printf("%d <-> ", ptr->info);
        ptr = ptr->prev;
    }
    printf("NULL\n");
}
```

```
void insert_begin() {
    struct node* temp = (struct
node*)malloc(sizeof(struct node));
    if (!temp) {
        printf("Memory allocation failed\n");
        return;
    }
    printf("\nEnter data: ");
    scanf("%d", &temp->info);
    temp->prev = NULL;
    temp->next = head;

    if (head == NULL) {
        head = tail = temp;
    } else {
        head->prev = temp;
        head = temp;
    }
}

void insert_end() {
```

```
    printf("Enter position to insert at
(starting from 1): ");
    scanf("%d", &pos);

    struct node* temp = (struct
node*)malloc(sizeof(struct node));
    if (!temp) {
        printf("Memory allocation failed\n");
        return;
    }
    printf("Enter data: ");
    scanf("%d", &temp->info);

    if (pos == 1) {
        temp->prev = NULL;
        temp->next = head;
        if (head != NULL) head->prev = temp;
        head = temp;
        if (tail == NULL) tail = temp;
        return;
    }
```

```
struct node* ptr = head;
for (i = 1; i < pos - 1 && ptr != NULL; i++)
{
    ptr = ptr->next;
}

if (ptr == NULL) {
    printf("Invalid position\n");
    free(temp);
    return;
}

temp->next = ptr->next;
temp->prev = ptr;

if (ptr->next != NULL)
    ptr->next->prev = temp;
else
    tail = temp; // inserted at end
```

```
    ptr->next = temp;
}

void delete_node() {
    int key;
    printf("Enter value to delete: ");
    scanf("%d", &key);

    struct node* temp = head;
    while (temp != NULL && temp->info !=
key) {
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Value %d not found\n", key);
        return;
    }

  if (temp->prev != NULL)
     temp->prev->next = temp->next;
```

```
    else
        head = temp->next;

    if (temp->next != NULL)
        temp->next->prev = temp->prev;
    else
        tail = temp->prev;

    free(temp);
    printf("%d deleted\n", key);
}

void count_nodes() {
    struct node* temp = head;
    int count = 0;
    while (temp != NULL) {
        count++;
        temp = temp->next;
    }
    printf("Total nodes: %d\n", count);
}
```

```
void search() {
    int key, pos = 1, found = 0;
    printf("Enter value to search: ");
    scanf("%d", &key);

    struct node* temp = head;
    while (temp != NULL) {
        if (temp->info == key) {
            printf("Value %d found at position
%d\n", key, pos);
            found = 1;
            break;
        }
        temp = temp->next;
        pos++;
    }
    if (!found)
        printf("Value %d not found\n", key);
}
```

M T W T F S S

Page No.: FIONA

Date :

Experiment 7:

Stack menu driven program for push pop display and exit

```
#include <stdio.h>
#include <stdlib.h>
#define size 5
    void push();
    void pop();
    void display();
    int stack [size];
      int top = -1;
      int main()
    {
        int ch;
        printf(" 1 Push in 2. POP in 3.
                display in 4. exit");
    do
      {
        printf("Enter choice");
          Scanf("%d, &ch);
          switch (ch)
          {
            case 1:
            Push();
            break;
            case 2:
            pop()
            break;
    case 3:
    display();
```

```
        break;
        case 4;
        exit (0)
        break;
    }
}

void pop()
{
    if( top == -1)
        printf(" Stack is empty ");
    else
    {
        printf (" Deleted element = %d"
            stack [top];
            top --;
    }
}

    void push()
    {
        int v;
        if (top == max-1)
            printf (" over flow ");
        else
        {
        print(" Enter element : ");
            scanf (" %d ", &v);
            stack [++top]
        }
    }
```

```
void display()
{
    if (top == -1)
        printf(" Empty ");
    else
    {
        print(" Element are :");
            for (int i = 0 ; i <= top , i++)
        {
            printf(" %d", stack[i])
        }
        print ("\n");
    }
}
```

Output

1. PUSH 2. POP 3. DISPLAY 4. EXIT
Enter choice : 1
Enter element : 1

Enter choice : 3
Element are : 1

Enter choice : 4

Experiment - 8

1) 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 == ')')
}

else
{
    while (priority[stack[top]] >=
    priority(*e)) printf("%c", pop());
    push(*e);
}

e++;
}

while (top != 1)
{
    printf("%c", pop());
}
    return 0;
}
```

o/p

3) 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, pEval;
    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');
    }
```

```
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 ("\n 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-- ] = -1;
        return n;
    }
else
    { printf (" stack is empty \n");
        exit (-1);
    }
}
```

O/P

M T W T F S S

Page No.: FIONA

Date :

2) Convert infix expression into prefix using stack : Infix Expression : A+(B*C-(D/E^F)*G)*H

Input	Stack	Output (Postfix)
H		H
*	*	H
)	*( )	H
G	*( )	HG
*	*(*	HG
)	*(*(	HG
F	*(*(	HGF
^	*(*(^	HGF
E	*(*(^	HGFE
/	*(*(/	HGFE^
D	*(*(/	HGFE^D
)	*(*	HGFE^D/
-	*(-	HGFE^D/*G
C	*(-	HGFE^D/*GC
*	*(-*	HGFE^D/*GC
B	*(-*	HGFE^D/*GCB
)	*	HGFE^D/*GCB*-
+	+	HGFE^D/*GCB*-*
A	+	HGFE^D/*GCB*-*A
		HGFE^D/*GCB*-*A+

solve again

4) Evaluate prefix expression using stack:
prefix expression: + - * +12142$42

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

Solve again

M T W T F S S

Page No.: FIONA

Date:

Algorithms :

1) Infix → Postfix

Step 1 : Scan infix expression from left → 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 with greater or equal precedance, 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 the reversed exp

Step 3 : Convert the new expression to postfix (using same algorithm as above)

Step 4 : Reverse result → that's the prefix expression

3) Evaluation of postfix

Step 1 : Scan left → right

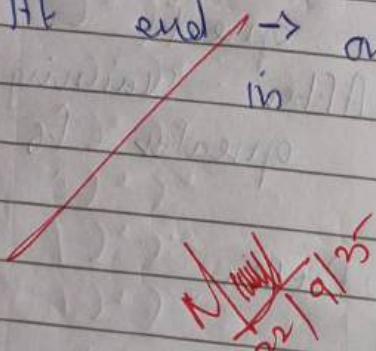
Step 2 : Operand → push into stack

Step 3 : Operator → pop top 2 value apply operator push result back

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



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

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

=> +A*-*BC*/D^EFGH

~~infix~~	~~into~~ prefix evaluation	
2		2
4		2, 4
$		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, 2, 1
+		16, 1, 2, 3
*		16, 1, 6
-		16, 5
+		21

Experiment 9 : Queue

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

#define SIZE 5

void enqueue();
void dequeue();
void display();

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

int main()
{
   int ch;
   printf("1. Enqueue\n2. Dequeue\n3. Display\n4
          . Exit\n");
   do
   {
      printf("\nEnter your choice:");
      scanf("%d",&ch);

      switch(ch)
      { case 1: enqueue(); break;
        case 2: dequeue(); break;
        case 3: display(); break;
        case 4: exit(0);
        default: printf("\n Invalid choice!\n");
      }
```

```
    }
  } while (ch != 4);
    return 0;
}

void enqueue()
{
    int element;
    if(rear == SIZE-1)
    {
      printf("\n Queue is full...");
    }
    else
    {
      printf("\n Enter element to insert:");
          scanf("%d", &Element);

        if( front == -1)
            front = 0;
        rear ++;
        queue [rear] = element;
        printf("\n Inserted -> %d, element);
    }
}

void dequeue()
{
    if (front == -1 || front > rear)
    {
        printf("\n Queue is empty...");
    }
```

```
else
{
    printf("\n Deleted element -> %d", queue[front
        ]);
    front ++;
  }
}

void display()
{
  If(front == -1 || front > rear)
  {
    printf("\n Queue is empty...");
  }
  else
  {
    printf("\n Queue elements are :\n");
      for(int i = front ; i <= rear ; i++)
      {
        printf("%d\t", queue[i]);
      }
        printf("\n");
  }
}
```

Output:

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your choice :

Q2)

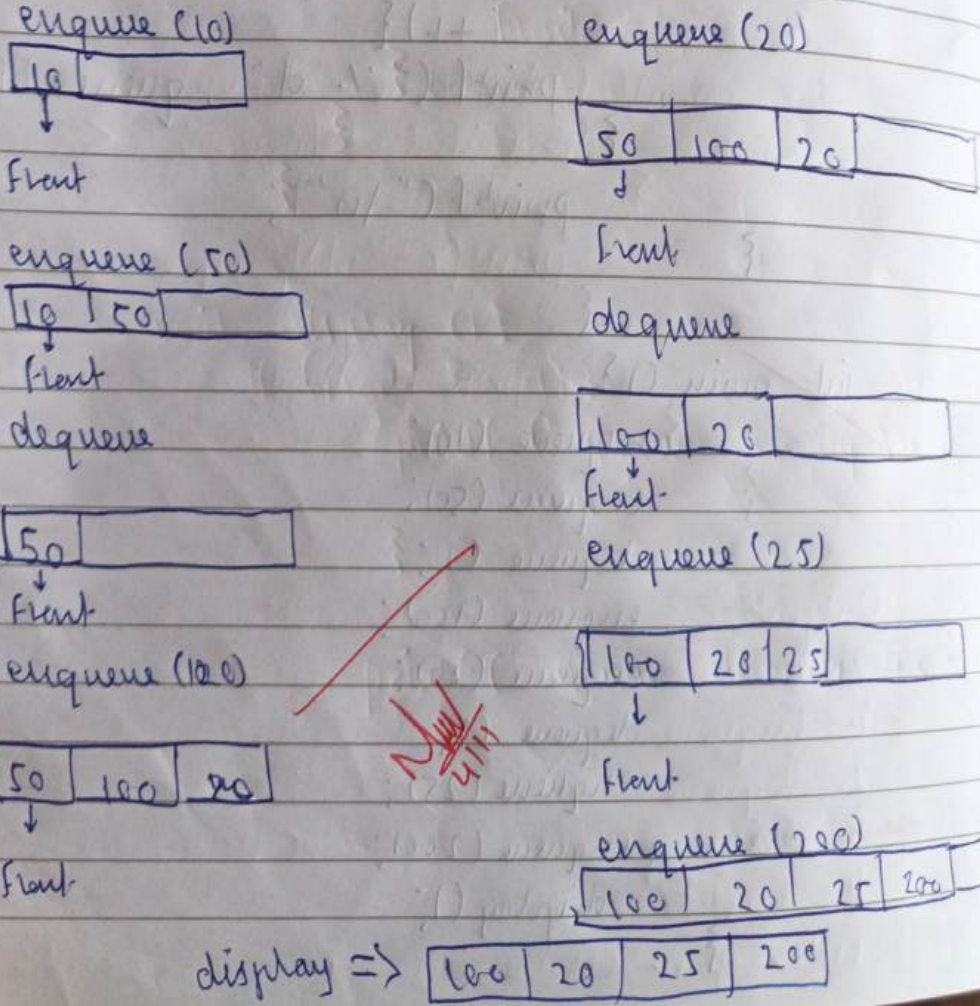
```
#include <stdio.h>
#define MAX 100

int queue [MAX];
int front = -1;
int rear = -1;

void enqueue (int num) {
        If (rear = MAX -1) {
            printf ("Over flow !");
            return;
        }

If (front == -1 && rear == -1) {
        front = rear = 0;
    } else {
        rear ++;
    }
    queue [rear] = num;

    printf ("%d inserted into queue \n" num)
    }

void dequeue () {
        If (front == -1) {
            If (front == -1) {
                printf ("Underflow\n");
            }

            int last = queue [front];
            front ++;
```

```
    if (front > rear) {
        front = rear = -1;
    }
    printf("%d deleted from queue\n", last);
}

void display() {
    if (front == -1 && rear == -1) {
        printf("Queue Empty");
        return;
    }
    printf("Current Queue :-\n");
    for (int i = front; i <= rear; i++) {
        printf("%d", queue[i]);
    }
    printf("\n");
}

int main() {
    enqueue(10);
    enqueue(50);
    dequeue();
    enqueue(100);
    enqueue(20);
    dequeue();
    enqueue(25);
    enqueue(200);
    display();
}
```

Output :

10 inserted
50 inserted
10 deleted
100 inserted
20 inserted
50 deleted
25 inserted
200 inserted
current queue :-
100 20 25 200

Diagramatic representation



Experiment - 10

Q1)

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

int queue [MAX];
int front = -1;
int rear = -1;

void enqueue (int num) {
    if ((front == 0 && rear == MAX-1)
        (front == rear +1)) {
        printf ("overflow \n");
        return;
    }

if (front == -1 && rear == -1) {
    front = rear = 0;
} else if (rear == MAX-1 && front!
    =0) {
    rear = 0;
} else {
    rear ++;
}
    queue [rear] = num;

    printf ("%d inserted into queue
        .\n", num);
}
```

```
void delete (){
    If (front == -1 && rear == -1){
        printf (" UNDERFLOW \n")
        return;
    }

int val = queue [front];
printf (" Deleted element : %d \n", val);

If (front == rear){
    front = rear = -1;
} else {
    If (front == MAX -1){
        front = 0;
    } else {
        front ++;
    }
}
}

void display (){
    If (front == -1){
        printf (" Queue empty \n");
        return;
    }

    printf (" Queue elements are : ");
```

```
If (rear >= front){
    for (int i = front ; i <= rear ; i++
    ){
    printf("%d", queue[i]);
    }
} else {
    for (int i = front ; i < MAX ; i++
    ){
    printf("%d", queue[i]);
    }
    for (int i = 0 ; i <= rear ; i++)
    {
    printf("%d", queue[i]);
    }
}
    printf("\n");
}

int main (){
    int choice, num;
    while (1){
        printf("\n --- Circular Queue Menu
                --- \n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice:");
        scanf("%d", &choice);
```

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

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

Experiment - 11

Q11

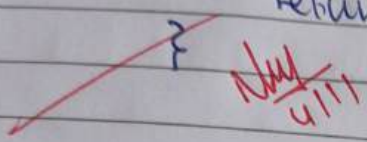
```
# Include < stdio.h>
# include < stdlib.h>

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

Struct node * Create Node (int value) {
    Struct node * new Node = ( Struct
        node *) malloc (size of
            Struct node)

    new Node -> data = value;
    new Node -> left = new Node -> 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)
        root -> right = insert (root
        -> right, value);
    }
    return root;
```

```
void inorder ( struct Node * root ) {
    If (root != NULL) {
    inorder (root -> left);
    printf (" %d ", root -> data);
    inorder (root -> right;
    }
}

Struct node * search (Struct node *
        root , int key)
If (root = Null || root -> data
        == key)
    return root;
If (key < root =>data)
    return search (root -> left
        , key);
return search (root -> right, key)
    }

Struct node * min value Node (Struct
        node * node) {
    Struct node * current =
        node ;
While (current && current ->
    left != Null)
        current = current -> left;
    return current;
}

Struct node * delete Node ( Struct node
    * root int key)
```

```
If (root == Null ) return root;

If( key < root -> data)
    root -> left = delete (root -> left,
    key

else if (key > root -> data)
        root -> right = deleteNode(root
        -> right, key );
        else {
            If (root -> left == NULL) {
            Struct node *temp = root -> left;
            free (root);
            return (root);
            return temp;
            }

            Struct node * temp = minValueNode
                                 (root -> right);
            root -> data = temp -> data;
            root -> right = deleteNode(root
                            -> right, temp -> data);
        }

        return root;
}

int main () {
        Struct node * root = Null;
        int choice, val;
        while (1) {
            printf (" 1. Insert ");
```

```
printf (" 2. Display ");
printf ( 3. Search ");
printf ( 4. Delete ");
printf (" Enter your choice :");
    scanf ("%d", &choice;

switch (choice){
    case1 :
        printf (" Enter value to insert
                :");
        scanf ("%d ,&val);
        break ;

    case 2 : printf (" Traversal (:");
        inorder (root);
        break ;

    Case 3 :
        printf (" Enter value to
                search ");
        scanf ("%d", &val);
        If search (root, val)
            printf (" Found ");
        else;
            printf (" Not found ");
        break;
Case 4 :
    printf(" Enter value to
            delete:");
    scanf ("%d", &val);
    root = delete Node(root
        , val);
```

```
        printf(" deleted");
        break;
    case 5:
        exit(0);

    default:
        printf(" Invalid choice !");
    }
}
    return 0;
}
```



Experiment 12

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

void initializeMatrix (int graph [V] [V]) {
        for (int i = 0 ; i < V ; i++) {
            for (int j = 0 ; j < V ; j++) {
                graph [i] [j] = 0;
            }
        }
    }

void insertEdge (int graph [V] [V] , int v1,
    int v2) {
        if (v1 >= 0 && v1 < V && v2 >= 0 && v2
            < V) {
            graph [v1] [v2] = 1;
            graph [v2] [v1] = 1;
        }
    }

void printMatrix (int graph [V] [V]) {
        for (int i = 0 ; i < V ; i++) {
            printf (" %d", graph [i] [j]
            );
        }
        printf ("\n");
    }
int main () {
```

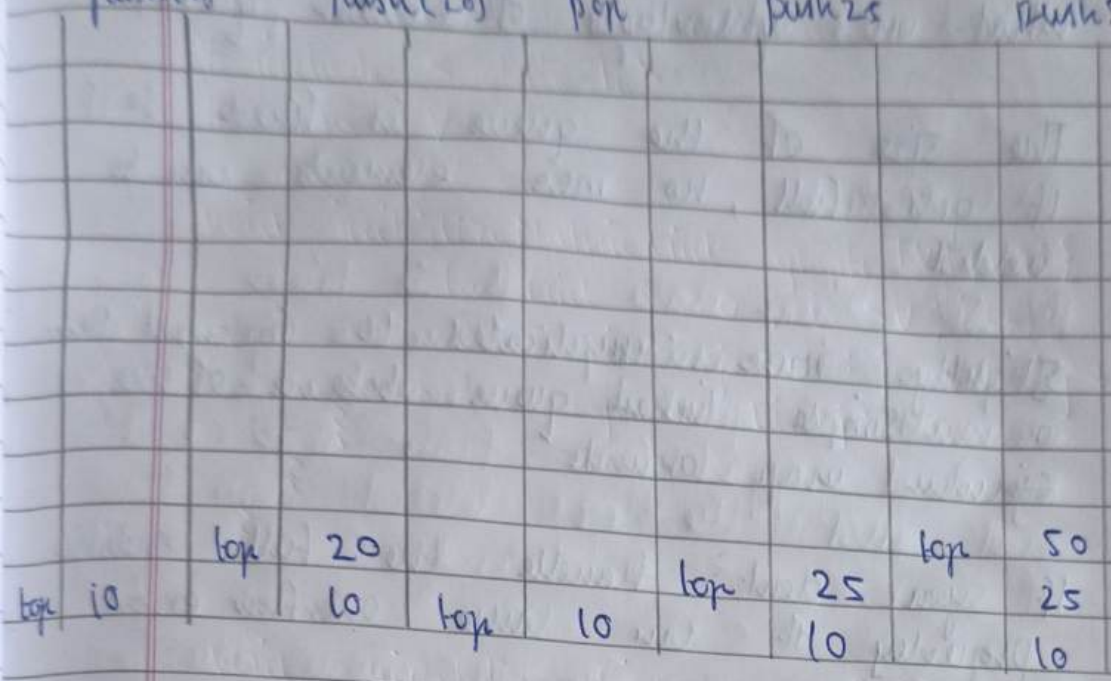
```
int graph[V][V];
initializeMatrix(graph);

insertEdge(graph, 0, 1);
insertEdge(graph, 0, 4);
insertEdge(graph, 1, 2);
insertEdge(graph, 1, 3);
insertEdge(graph, 3, 4);

printMatrix(graph);

return 0;
}
```

Output:

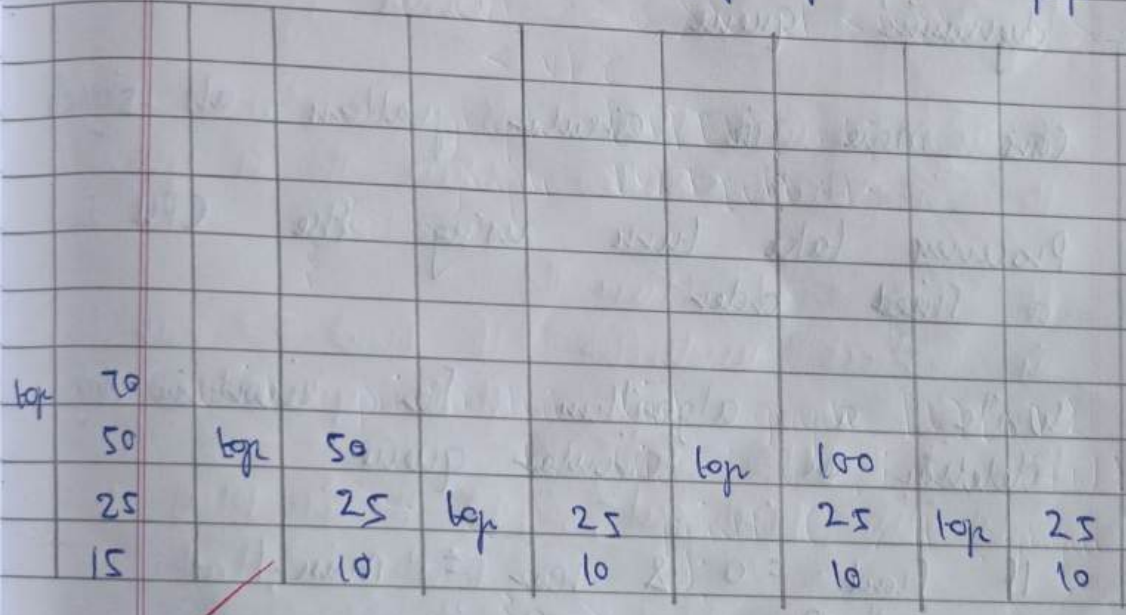
```
0 1 0 0 1
1 0 1 1 0
0 1 0 0 0
0 1 0 0 1
1 0 0 1 0
```

Exp 7 Theory

push(10) push(20) pop push25 Push50



push 30 pop pop push100 pop



Exp 10

What are advantages of circular queue

The size of the queue is fixed, so if it gets full, no more elements can be added

Slightly more complicated to implement than a simple linear queue because of the circular wrap around.

If you don't handle the full condition properly it can cause overflow errors

Can't grow or shrink easily like a dynamic queue

Can move in circular pattern at signals

Processes take turn using the CPU in a fixed order

Write a algorithm for insertion by deletion on circular queue

If front = 0 & rear = max-1
"Queue overflow"
If front = rear + 1
"Queue overflow"

If front = -1 by rear = -1
front = rear = 0

Else if rear = max-1 & front != 0
rear = 0
End
queue [rear] = val
exit

Algorithm for circular queue

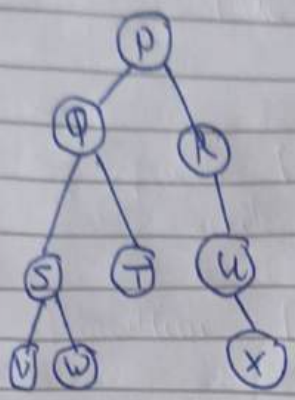
If front = -1
write "underflow"
go to 5.4
[End of if]
Set val = Queue [front]
If front = Rear
set front = Rear = 1
else
If Front = Max -1
set front = 0
else
Set front = front + 1
[End of if]
[End of if]
Exit

Write applications of Queue
Used in call logs
It is used to in traffic light systems
It is used in online ticket booking system
Used in call centre support
Used in memory management

CPU uses queue to schedule process

Stack	Queue
It is a linear data structure in which elements are pushed ~~by~~ and popped from one end called the 'top' of stack.	It is linear data structure in which elements are inserted from rear end deleted from front end
In stack only one pointer variable called top	In queue the pointer variable are used called front and rear
If top = max-1 then it is stack overflow	If rear = max-1 then it is queue overflow
If top == -1 then it is stack underflow	If front = = rear then it is queue underflow
It is LIFO ie last in, first out.	It is FIFO ie First in, last out.

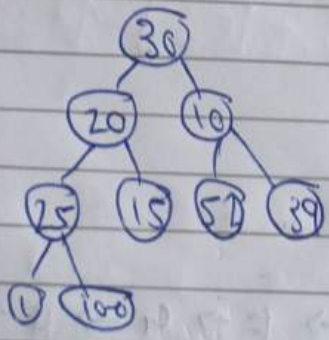
Exp 11 -



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

Inorder : V → S → W → Q → T → P → X → U → R

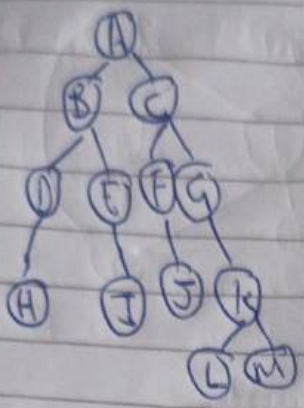
30 , 20 , 10 , 25 , 15 , 52, 39, 1 , 100



Pre : 30 → 20 → 10 → 1 → 15 → 25 → 52 → 39 → 100

In : 1 → 25 → 100 → 20 → 15 → 52 → 10 → 39

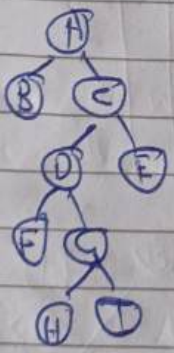
Post : 1 → 100 → 25 → 15 → 20 → 52 → 39 → 10 → 30



Pre : A → B → D → H → E → I → C → F → G → K → L → M

In : H → D → B → E → I → A → J → F → C → L → K → M → G

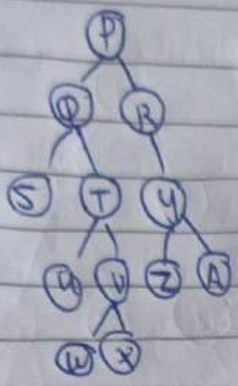
Post : H → D → I → E → B → I → F → L → M → K → G → C → A



Pre : A → B → C → D → F → G → H → I → E

In : B → A → F → D → H → G → I → C → E

Post : B → F → H → I → G → D → E → C → A

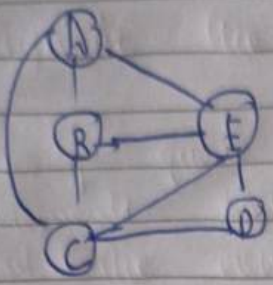


Pre : P → Q → S → T → U → V → W → X →
K → Y → Z → A

In : S → Q → U → T → W → V → X
→ P → R → Z → Y → A

Post-order : S → U → W → X → V → T → Q
R → Y → Z → A → P

Exp 12



	A	B	C	D	E
A	0	1	1	0	1
B	1	0	1	0	1
C	1	1	0	1	1
D	0	0	1	0	1
E	1	1	1	1	0

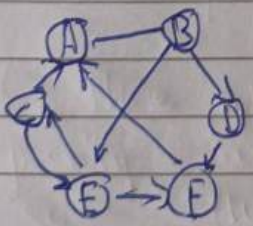
A → [B] → [C] → [E]

B → [A] → [C] → [E]

C → [A] → [B] → [D] → [E]

D → [C] → [E]

E → [A] → [B] → [C] → [D]



	A	B	C	D	E	F
A	0	0	0	0	0	0
B	1	0	0	1	1	0
C	1	0	0	0	1	0
D	0	0	1	0	0	1
E	0	0	1	0	0	1
F	1	0	0	0	0	1

A → Null
B → [A] → [B] → [E]
C → [A] → [E]
D → [C] → [B]
E → [C] → [E]
F → [A]

Application of graph

In CS - represent the flow of computation

google
facebook
World wide web
operating
maping
microsoft excel
social media
Biochemical application

Exp: 9

i) Explain the concept of priority queue

Priority queue is a collection in which elements can be inserted at any time, but the elements which can be removed is one with higher ~~prio~~ priority

In some cases elements are inserted at any position based on intrinsi ordering rather than strict ordering and tell us which element got deleted.

Priority queues are the queues in which we can use intrinsic ordering of elements

The elements with higher priority are processed before any element with lower priority

If there where 2 elements of same priority and both were processed, so they will get processed on First Come First Serve Basis

ii) Algorithm for insertion and deletion of linear queue

Insertion

Step 1: Check if queue is full or not

Step 2: If queue is full ~~or not~~
print "Queue overflow" and
If it is not overflow condition
Increment rear and add element

Step 3: Stop

Deletion:

Step 1: Check if queue is empty
or not

Step 2: If queue is empty print
"queue underflow" and exit
the performance

Step 3: If it is not empty remove
the element from queue and
increment "front"

Step 4: Stop