EXPERIMENT 1:

Write program for array:

1) Find average of 10 numbers using an array

```
#include<stdio.h>
int main()
{
  int i, a[10], s=0;
  float av;
  printf("Enter 10 numbers:-\n");
  for(i=0; i<10; i++)
  {
    Scanf("%d", &a[i]);
  }
  for(i=0, i<10; i++)
  {
    s = s + a[i];
  }
  av = s/10.0;
  printf("\n Average of array elements: %f", av);
  return 0;
}
```

output:

```
Enter 10 numbers:
1
2
3
4
5
6
7
8
9
10
Average of array element: 5.500000
```

2) Display the following pattern

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

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

3. Find the first repeating element in an array

```
#include <stdio.h>
int main()
{
    int a[100], n, i, j, ch=0;
    printf("Enter the number of elements in array:");
    scanf("%d", &n);
    printf("Enter the elements of array:\n");
    for(i=0; i<n; ++i)
    {
        scanf("%d", &a[i]);
    }
    for(i=0; i<n; ++i)
    {
        for(j=i+1; j<n; ++j)
        {
            if(a[i]==a[j])
            {
                printf("First repeating element is : %d", a[j]);
                ch=1;
            }
        }
        if(ch==1)
        break;
    }
    if(ch==0)
    printf("No repeating element");
    return 0;
}
```

Output:-
Enter the number of elements in array : 5
Enter the element of array :-
1
2
3
3
4
First repeating element is 3

4) Find the greatest and smallest element in an array

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

```
printf("\n Maximum value in array: %d", ma
printf("\n Minimum value in array: %d", mi
return 0;
}
```

Output:-

Enter the number of array elements : 3

Enter the array elements:-

1 2

3 4

5 4

Maximum value in array : 54

Minimum value in array: 12

5 Squaring the odd position element of array

```
# include <stdio.h>
int main ()
{
    int a[10], i;
    printf ("Enter 10 elements of an array:\n");
    for (i = 0; i <10; i++)
    {
      scanf ("%d", a[i]);
    }
    printf ("The array after squaring the odd
             position of element are:\n");
    for (i = 0; i <10; i++)
    {
      if (i %2 == 0)
      {
        a[i]= a[i] * a[i]
      }
      printf ("%d\n", a[i]);
    }
    return 0;
}
```

Output:

Enter 10 elements of an array:

1

2

3

4

5

6

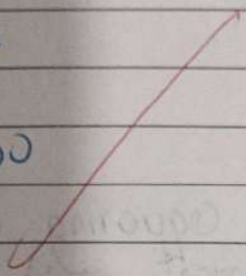
7

8

PAGE No.
DATE / /

The array after squaring odd position elements :

1
4
3
16
5
36
7
64
9
100



15/7/25

Patterns

1)
```
*
* *
* * *
* * * *
* * * * *
```

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

2) 1

2 2

3 3 3

4 4 4 4

5 5 5 5 5

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

3 1 ' ;j--)

1 2

1 2 3

1 2 3 4

1 2 3 4 5

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

4)
```
*
# #
$ $ $
? ? ? ?
```

```
# include <stdio.h>
int main()
{
   int i,j;
   char s[] = {'*', '#', '$', '?'};
    for (i=0; i<4; i++)
    {
        for (j=0; j<=i; j++)
        {
          printf("%c", s[i]);
        }
        return 0;
```

5)
```
*  *  *  *
   *  *  *
      *  *
         *
```

```
#include <stdio.h>
int main()
{
 int i, j;
 for (i=1; i<=4; i++)
 {
 for (j=1; j<i; j++)
 {
   printf(" ");
 }
```

```
for (j=4 ; j >= 1 ; j--)
{
  printf (" * ");
}
printf ("\n");
return 0;
}
```

6)
```
1
1  2
3  3  3
1  2  3  4
5  5  5  5  5
```

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

7)
```
*
* #
* # ?
* # ? $
```

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

8)

```
   *
  * *
 *   *
*   *   *
```

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

EXPERIMENT 2

1) Search data using Linear search consider following list to perform linear search.

56	36	89	57	10	67	59

Search item 1 from the above list and write if the item is found or not.

2) Search data using binary search

3) Compare linear search versus binary search.

4) State limitations of linear search interms of time complexicity

Answers

1) Program

```
#include <stdio.h>
int main ( )
{
    int a[ ] = {56,36,89,57,01,00,67,59};
    int k,i, found=0;
    printf("Enter key to be searched:\n");
    scanf("%d",&k);
    for(i=0; i<8; i++)
    {
        if(a[i]==k)
        {
            printf("In key found at index:%d",i);
```

```
        found = 1;
        break;
    }
    if (found == 0)
    printf("In Key not found in the array:");
    return 0;
}
```

Output:

Enter the key to be searched:
01
Key found at index: 4.

5)
```
#include<stdio.h>
int main()
{
    int a[] = {
```

ii) Enter key to be searched: 55
Key not found in array

Q.2

```
#include <stdio.h>
int main()
{
 int A[100], n, m, l, h, i
 printf("Enter number of array elements:");
 scanf("%d", &n);
 printf("Enter Key to be search:")
 scanf("%d", &k);
 printf("\n Enter sorted array elements:\n");
 for(i=0; i<n; i++)
 {
  scanf("%d", &a[i]);
 }

  l=0; h=(h-1);
  while(h>=l)
  {

   m=(int)((l+h)/2);
   if(h==a[m])
  {

   printf("\nElement found at index: %d", m);
   break;
  }
  else
  {
   if(h>a[m])
   {
     l=m+1;
   }
```

```
        if (h < a[m])
        {
            h = m - 1;
        }
    }
}
    if(h<l)
    printf("\n Element not found);
    return 0;
}
```

Output:

```
Enter number of array element: 6
Enter Key to be search: 23
23
45
67
56
78
98
```

Q.3

	Linear search	Binary search
1.	Time complexity is low	Time complexity is O
2	Works on both sorted and converted	Only works on sorted dat
3.	It is low efficient, especially for large data set	it is more efficient especially for large data set
4.	Code is simpler	Code is more complex
5.	it we sequential searching approach	it uses divide and conquer approoch

Q.4 State limitation of linear search interms of time complexity.

- A linear search runs in an worst linear time and makes at most comparison where n is the length of the list.
- Linear search is rarely practical because other search algorithm and schemes, such as binary search algorithm.
- Not efficient for large dataset
- No improvement with sorted data.

Nawal 9/7/25

Experiment: 3

1) Sort element in ascending order using bubble sort

2) Sort element in descending order using selection sort

3) Find number of comparison required in bubblesort method of the following list

100, 200, 300, 400, 500

4) Sort the given array in ascending order using selection sort method and show diagramatical presentation of iteration of loops
500, -20, 30, 14, 50

1) Program

```
#include <stdio.h>
int main()
{
    int a[100], n, i, j, r;
    printf("Enter no. of elements in array:");
    scanf("%d", &n);
    printf("Enter Array elements :-\n");
    for(i= 0; i<n; i++)
    {
        scanf("%d", &a[i]);
    }
    for(i= 0; i<n; i++)
    {
        for(j= 0; j<n-1; j++)
        {
```

```
        r = a[j];
        if(a[j] > a[j+1])
          {
            a[j] = a[j+1];
            a[j+1] = r;
          }
      }
  }
printf("\n Sorted Array Elements :-\n");
for(i=0; i<n; i++)
  {
    printf("%d\n", a[i]);
  }
  return 0;
}
```

Output :-

Enter no. of elements in array :- 5

Enter Array elements :-

5

3

4

2

1

Sorted Array elements :-

1

2

3

4

5

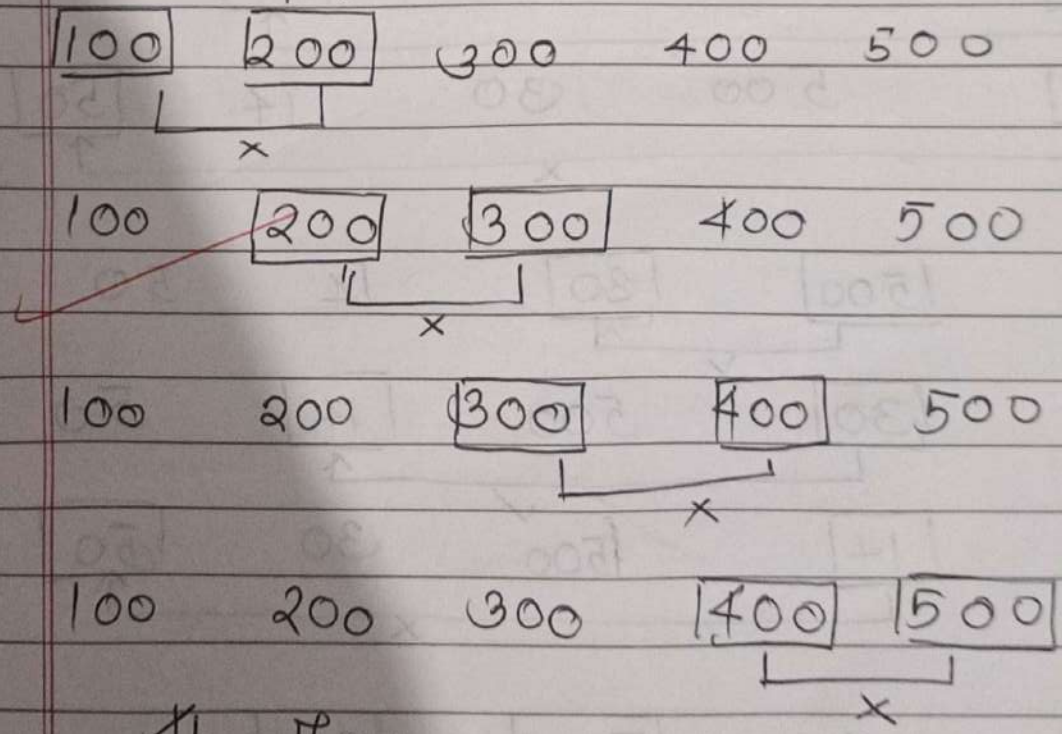
2)
```
#include <stdio.h>
int main()
{
  int a[100], n, i, j, r;
  printf("Enter no of elements in array: ");
  scanf("%d", &n);
  printf("Enter Array Elements :\n");
  for(i=0; i<n; i++)
  {
    scanf("%d", &a[i]);
  }
  for(i=0; i<n; i++)
  {
    for(j=i+1; i<n; j++)
    {
      if(a[i] < a[j])
      {
        r = a[i];
        a[i] = a[j];
        a[j] = r;
      }
    }
  }
  printf("\nSorted Array Elements in descending order :\n");
  for(i=0; i<n; i++)
  {
    printf("%d\n", a[i]);
  }
  return 0;
}
```

Output:
Enter no of elements in array: 3
Enter Array elements:
324
34
0
Sorted Array elements:
324
34
0

3)

100, 200, 300, 400, 500

number of paves = (n-1) (n=5)



Algorithm:

1) Start with the list of no.
2) Compare each element of swap if the first elem is greater than second element.

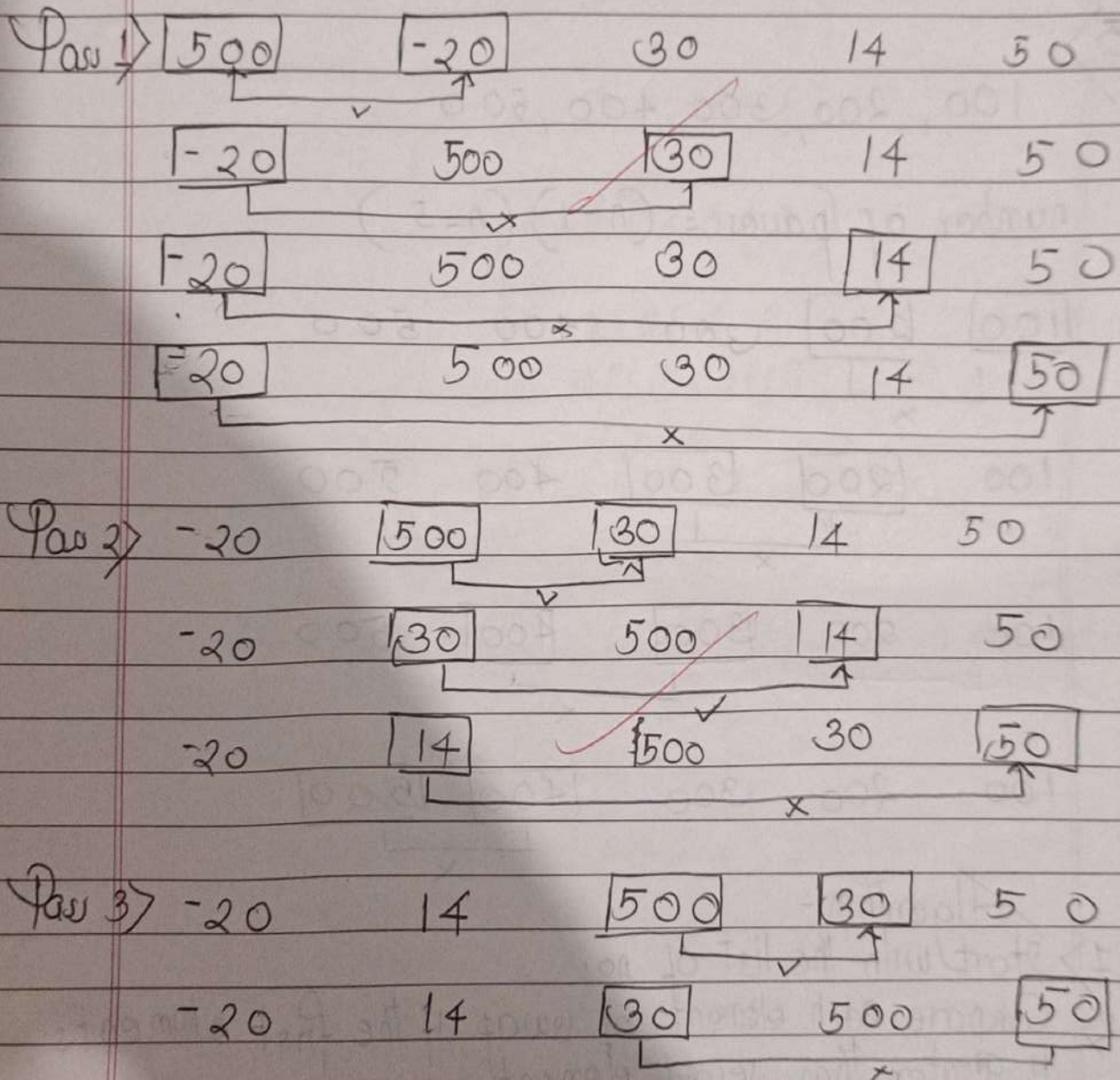
3) After each full pass the largest unsorted element moves to the end of array

4) Repeat the process of 2step on the unsorted array

5) Stop the program

Selection sort:

500, -20, 30, 14, 50



PAGE No.
DATE / /

Pass 1) -20 14 30 500 50

-20 14 30 50 500

* Algorithm :-

1) Start main to 0.
2) Search min element of the array
3) Swap value at selection of main
4) Increment min to point the next element location
5) Repeat till sorted
6) Stop

29/7/25

Experiment: 4

Q. Write a C program to sort data from given array using insertion sort & radix sort

1) Sort element in ascending order using insertion sort

```
#include <stdio.h>
int main()
{
    int a[100], n, i, j, k, r, ii, p=1;
    printf("Enter no. of elements in array: ");
    scanf("%d", &n);
    printf("\n Enter elements of Array :\n");
    for(i=0; i<n; i++)
    scanf("%d,", &a[i]);
    for(i=1; i<n; i++)
    {
        for(j=0; j<i; j++)
        {
            if(a[i] < a[j])
            {
                r = a[i];
                for(k=i-1; k>=j; k--)
                {
                    a[k+1] = a[k];
                }
                a[j] = r;
            }
        }
    printf("\n Array after pass %d: ", n);
    for(ii=0; ii<n; ii++)
    {
        printf("%d", a[ii]);
    }
```

```
    ++p;
  }
  printf("\n\nSorted array in ascending elements :-");
  for(i=0; i<n; i++)
  {
    printf("\n%d", a[i]);
  }
  return 0;
}
```

Output:-

Enter no. of elements in array : 5

Enter elements of Array :-

900
569
8589
6
3456

Array after pass 1 : 569 9000 8589 6 3456
Array after pass 2 : 569 8589 9000 6 3456
Array after pass 3 : 6 569 8589 9000 3456
Array after pass 4 : 6 569 3456 8589 9000

Sorted array in ascending elements :-

6
569
3456
8589
9000

3) What is the output of insertion sort after the 2nd iteration given the following sequence of no: 7,3,5,1,9,8,4,6

7 3 5 1 9 8 4 6

1st Iteration

7 3 1 9 4 8 6

3 7 1 9 4 8 6

2nd Iteration

3 7 1 9 4 8 6

1 3 7 9 4 8 6

After 2nd iteration the result is
1,3,7,9,4,8,6

Sort the following numbers using Radix Sort:-

100 225 390 4130 956 99 5431

Pass 1	0	1	2	3	4	5	6	7	8
0100	0100								
0225						0225			

	0	1	2	3	4	5	6	7	8	9
0390	0390									
4130	4130									
0956							0956			
0099										009
5431		5431								
Pass 2)	0	1	2	3	4	5	6	7	8	9
0100	0100									
0390										039
4130				4130						
5431				5431						
0225			0225							
0956						0956				
0099										009
Pass 3)	0	1	2	3	4	5	6	7	8	9
0100		0100								
0225			0225							
4130		4130								
5431					5431					
0956										09
0390										
0099	0099									
Pass 4)	0	1	2	3	4	5	6	7	8	
0099	0099									
0100	0100									
4130					4130					
0225	0225									
0390	0390									
5431						5431				
0956	0956									

sorted arr = [0099, 0100, 0225, 0390, 0956, 4130, 5

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

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

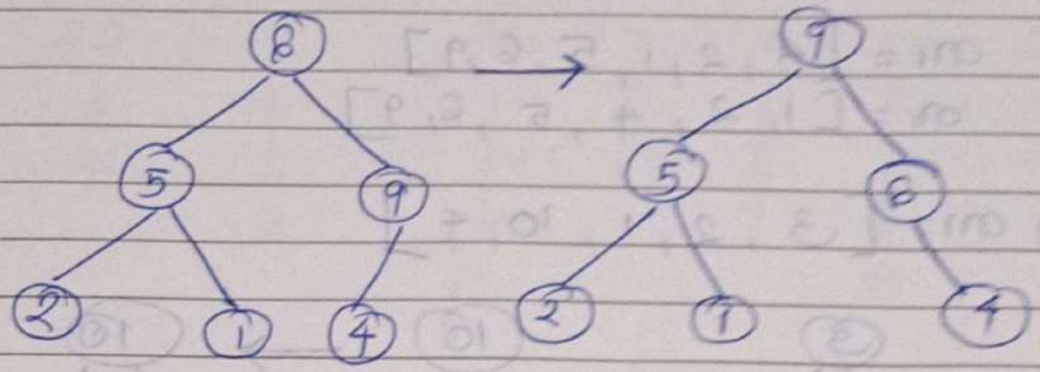
Pass 3)	0	1	2	3	4	5	6	7	8	9
006	006									
018	018									
020	020									
025	025									
239			239							
145		145								
099	099									

Sorted arr = [006, 018, 020, 025, 099, 145, 239]

Q4) Sort the following element using heap sort :-

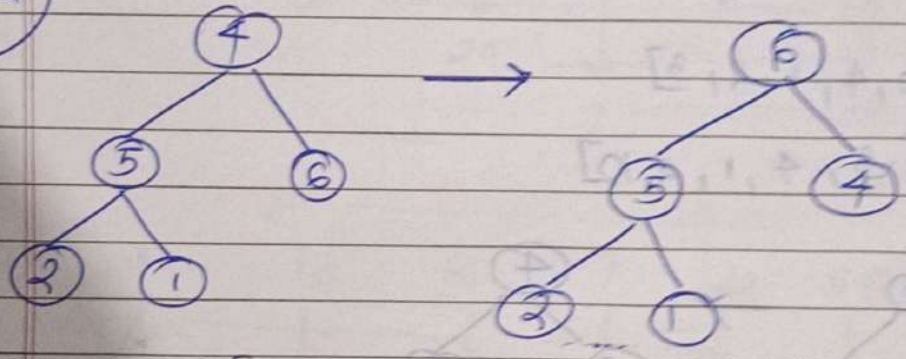
arr [6, 5, 9, 2, 1, 4]

Pass 1)



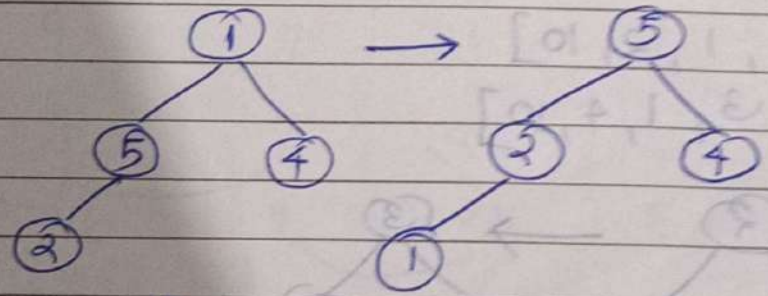
arr = [9, 5, 6, 2, 1, 4]
arr = [4, 5, 6, 2, 1, 9]

Pass 2)



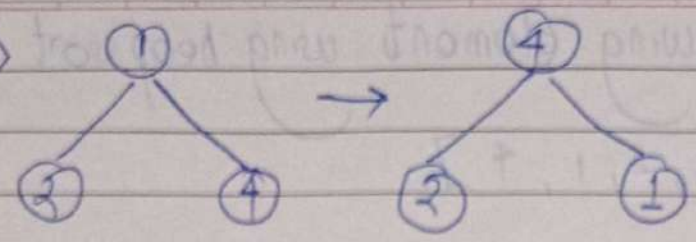
arr = [6, 5, 4, 2, 1, 9]
arr = [1, 5, 4, 2, 6, 9]

Pass 3)



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

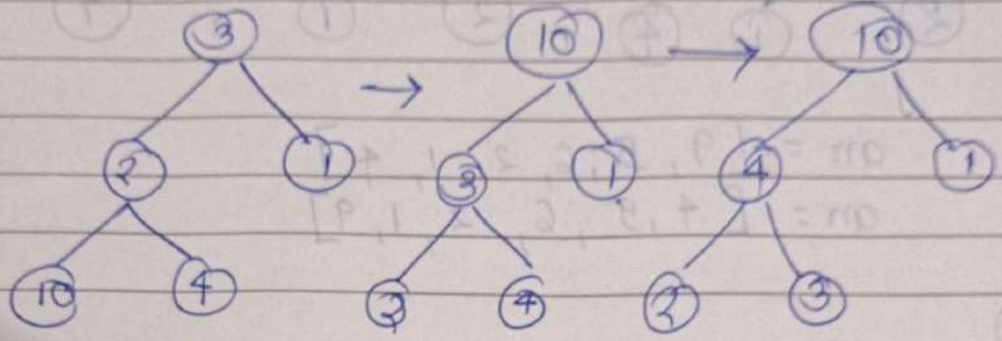
Pass 1)



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

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

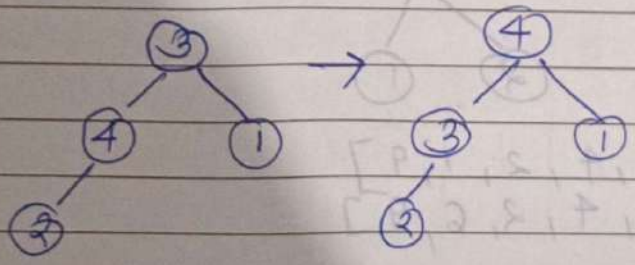
2) arr = [3, 2, 1, 10, 4]



arr = [10, 4, 1, 2, 3]

arr = [3, 4, 1, 2, 10]

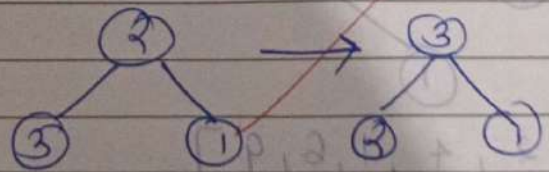
Pass 2)



arr = [4, 3, 1, 2, 10]

arr = [3, 3, 1, 4, 10]

Pass 3)



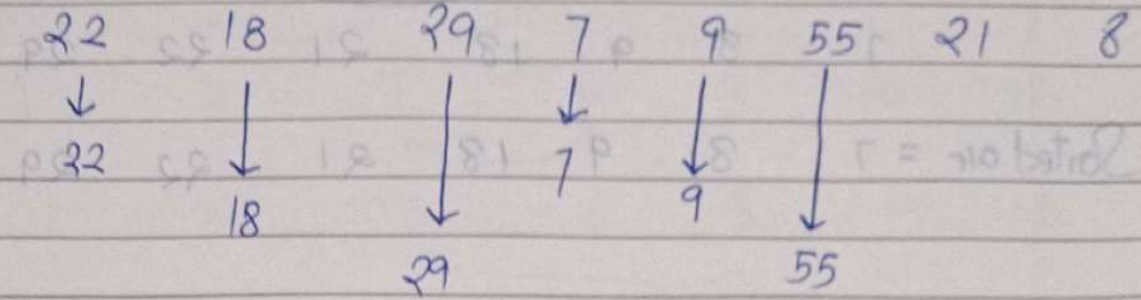
arr = [3, 2, 1, 4, 10]

arr = [1, 2, 3, 4, 10]

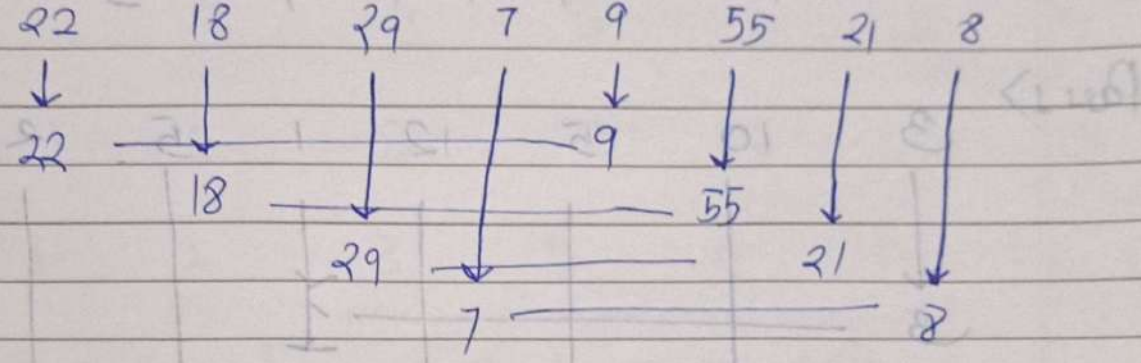
5) Sort the following elements using shell sort

arr = [22, 18, 29, 7, 9, 55, 21, 8]

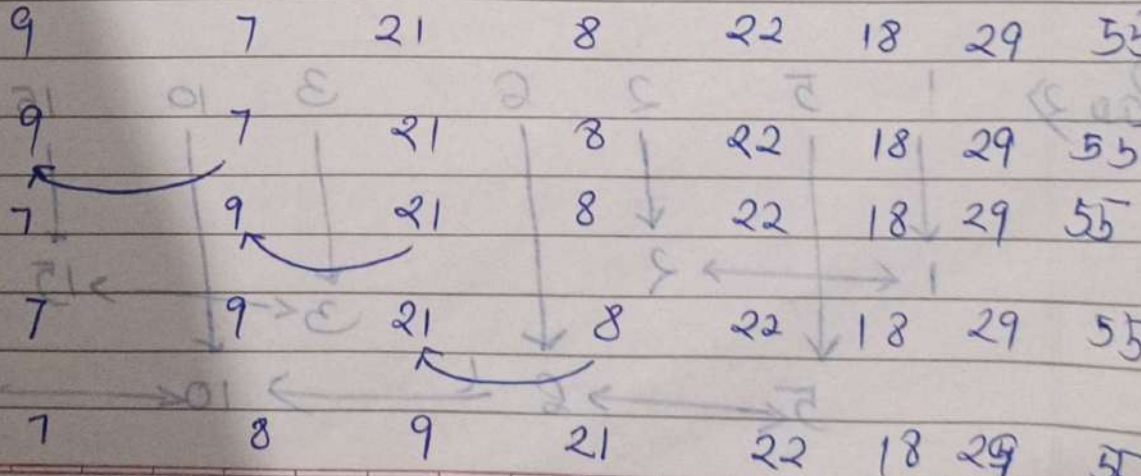
Pass 1)



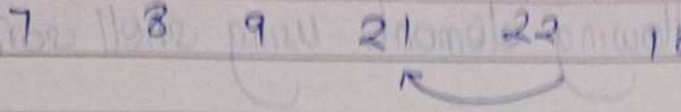
Pass 2)



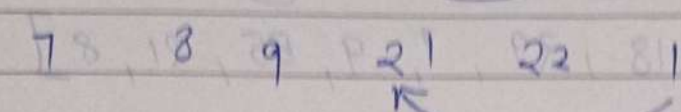
Pass 3)



7 8 9 21 22 18 29 55



7 8 9 21 22 18 29 55

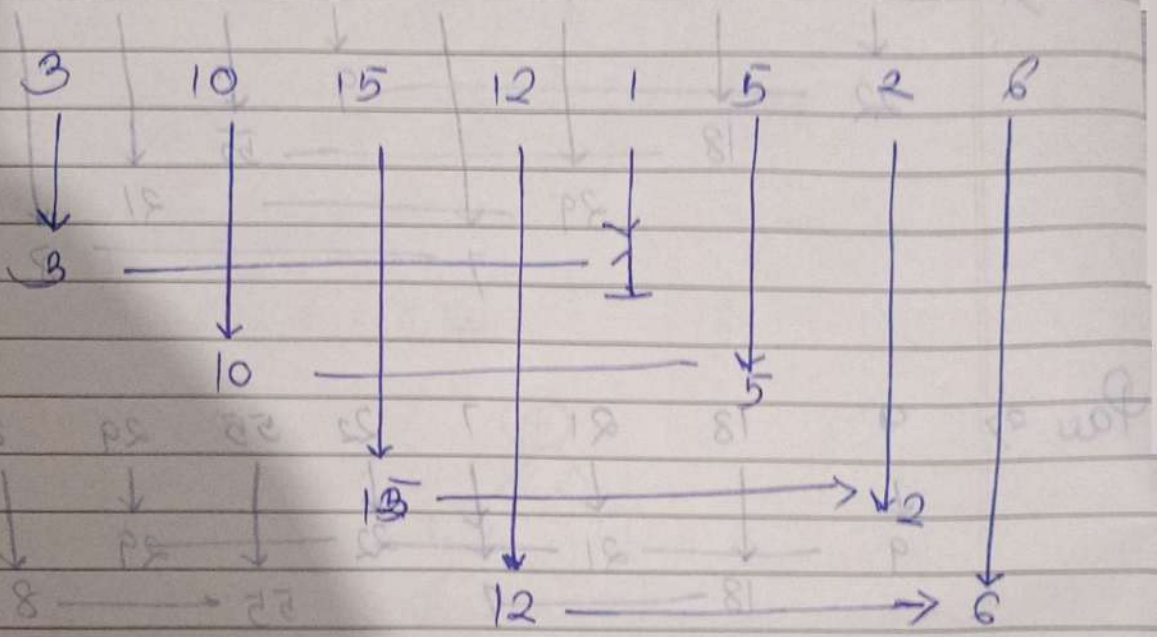


7 8 9 18 21 22 29 55

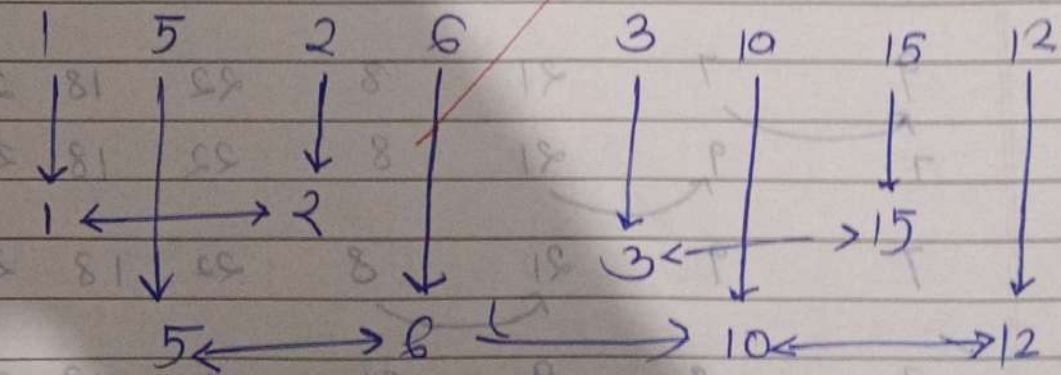
Sorted arr = 7 8 9 18 21 22 29 55

2) arr = [3, 10, 15, 12, 1, 5, 2, 6]

Pass 1>



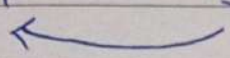
Pass 2>



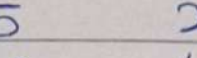
PAGE No.
DATE / /

Pass 3>

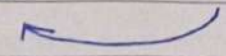
1 5 2 6 3 10 15 12



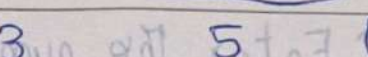
1 5 2 6 3 10 15 12



1 2 5 6 3 10 15 12

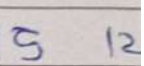


1 2 5 6 3 10 15 12



1 2 3 5 6 10 15 12

1 2 3 5 6 10 15 12



1 2 3 5 6 10 12 15.

Sorted arr = [1, 2, 3, 5, 6, 10, 12, 15].

Q6) Sort elements in ascending order using Radix sort

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main()
{
  int a[20][10], b[20], top[10], len, i, j, k, n, p, pr,
  max;
  printf ("Enter the number of elements:");
  Scanf ("%d", &n);
  if (n > 20)
  {
    printf (" Error : maximum 20 elements allowed\n");
    return 1;
  }
  printf ("\n Enter The elements :\n");
  for (i = 0; i < n; i++)
  {
  scanf("%d", &b[i]);
  }
    max = b[0];
    for (i = 1; i < n; i++)
      {
        if (b[i] > max)
          max = b[i];
      }
      len = (int) log10 (max) + 1;
      for (i = 0; i < len; i++)
        {
          for (j = 0; j < n; i++)
            {
```

```
pos = (int)(b[j] / pow(10, i)) % 10;
a[pos][++top[pos]] = b[j];
}
for (j=0, p=-1; j<10; j++)
{
    for (k=0; k<=top[j]; k++)
    {
        b[++p] = a[j][k];
    }
}
}
printf("\n After Sorting elements:\n");
for (i=0; i<n; i++)
{
    printf("%d", b[i]);
}
printf("\n");
return 0;
}
```

Output:

Enter the number of elements: 5

Enter the elements: 45
35
64
99
78

After sorting the elements:

35 45 64 78 99

12/8

Write a program for expression conversion using stack

i) Convert infix to Postfix or Prefix Expression

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

iii) Write a C program to evaluate infix or postfix Expression

iv) Evaluate prefix expression using stack: prefix Expression
+ - * + 1 2 / 4 2 1 $ 4 2

Exp 8

Program

i)
```
#include <stdio.h>
#include <type.h>
char stack[100];
int top = -1;
void push (char x)
{
    stack[++top] = x;
}
char pop()
{
```

```
if (top == -1)
return -1;
else
return stack[top--];
}
int priority (char x)
{
if (x == '(')
return 0;
if (x == '*' || x == '-')
return 1;
if (x == '*' || x == '/')
return 2;
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(1)!='(')
print("%c",x);
}
else
{
while (priority(stack[top])>=priority(*e))
    printf("%c",pop(1));
    push(*e);
}
e++;
}
while (top1=-1)
{
printf("%c",pop());
}
return 0;
}
```

ALGORITHM

1. Start
2. Initialize empty stack & empty postfix string
3. Scan infix expression left → right
4. If operand → add to postfix
5. If '(' → push to stack
6. If ')' → pop from stack to postfix until '(' is found
7. If operator → pop higher operators from stack to postfix then push current operator
8. After scanning, pop all remaining operators to postfix
9. Result = postfix expression
10. Stop

ii) A ↑ B (+

A + (B * C - (D / E ↑ F) * G) * H

I/P	stack	O/P
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	+A	HGFE$D/*CB*-*

Prefix: + A * - * B C * / D $ E F G H

iii) Program {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("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("\nThe 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 n;
if (top > -1)
{
  n = stack[top--];
  return n;
}
else
{
  printf ("stack is empty!/n");
  exit(-1);
}
}
```

output

Enter a postfix expression : 12+42/*1-42$+

The postfix evaluation is : 6

ALGORITHM

1. Start
2. Initialize an empty stack
3. Scan the postfix expression from left to right
4. For each symbol in expression
   - if symbol is operand <push it onto stack>
   - if the symbol is operator <follow the rules of operator>
5. Repeat until end of expression
6. The final result will be only element left in stack
7. stop

PAGE No.
DATE / /

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

Input	opr 1	sig	opr 2	Result	Output
2					2
4					2, 4
$	4	$	2	16	16
1					16, 1
2					16, 1, 2
4					16, 1, 2, 4
/	4	/	2	2	16, 1, 2
2					16, 1, 2, 2
1					16, 1, 2, 2, 1
+	1	+	2	3	16, 1, 2, 3
*	3	*	2	6	16, 1, 6
-	6	-	1	5	16, 5
+	16	+	5	21	21

Algorithm

1. Start
2. Initialize an empty stack
3. Scan the prefix expression from right to left
4. For each symbol:
   - If operand → push onto stack
   - If operator →
     - Pop two operands from stack
     - Apply operator → get result
     - Push result back on stack
5. After scanning all symbols, the stack's top element is the final result
6. Stop

14/9

Exp-5

Singly linked list code

```
#include<stdio.h>
#include<stdlib.h>
void display();
void create();
void insert_begin();
void insert_end();
void insert_pos();
void delete_all();
void delete_pos();
int count_elem();
  void reverse_u();
  void search();
struct node{
  int info;
  struct node *next;
};
struct node *start=NULL;

int main()
{
  int choice;
  while(1){
  printf("\n1. create\n");
  printf("2. display\n");
  printf("3. insert at begin\n");
  printf("4. insert at end\n");
  printf("5. insert at any position\n");
  printf("6. delete at given position\n");
  printf("7. delete all\n");
  printf("8. count elements\n");
  printf("9 reverse linked list\n");
```

```
printf (" 10. Search for an element \n");
printf (" 11. exit \n");

printf ("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: delete_pos (); break;
  case 7: delete_all (); break;
  case 8: count_elem (); break;
  case 9: reverse_dll (); break;
  case 10: search (); break;
  case 11: exit (0);
  default: printf (Incorrect choice.\n");
  }
}
  return 0;
}
void create ()
{
  struct node * temp * ptr;
  temp = (struct node *) malloc (size of (str
  printf ("Enter data:")
  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 ("\nEmpty list\n");
 return;
}
 ptr = start;
 printf ("\nList elements:");
  while (ptr != NULL)
  {
      printf ("%d", ptr -> info);
      ptr = ptr -> next;
  }
  printf ("\n");
}
void insert_begin ()
{
```

```
struct node * temp, * ptr;
temp= (struct node *) malloc (size of (struct node
    ));
printf ("Enter data:");
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 insert_end(){
    struct node * temp, *ptr;
    temp = (struct node*) malloc (size of (struct node)
    printf ("Enter data:");
    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 insert_pos(){
  struct node *temp,*ptr;
  int i,pos;
  temp=(struct node*)malloc(sizeof(struct node));
  printf("Enter data:");
  scanf("%d",&temp->info);
  temp->next=NULL;
  printf("\n Enter position :");
  scanf("%d",&pos);
  if(pos==1){
    temp->next=start;
    start=temp;
    return;
  }
  ptr=start;
  for(i=1;i<pos-1&&ptr!=NULL;i++){
    ptr=ptr->next;
  }
  if(ptr==NULL){
    printf("position not found\n");
    free(temp);
  }else{
  temp->next=ptr->next;
  ptr->next=temp;
  }
}
```

```
void delete_all (){
    struct node * temp;
    while( start != NULL) {
        temp = start;
        start = start->next;
        free (temp);
    }
    printf (" All values deleted ! \n");
}
void delete_pos(){
    int i, pos;
    struct node * temp, * ptr;
    if (start == NULL){
        printf (" Empty List \n");
        return;
    }
    printf (" Enter position :");
    scanf(" %d", & pos);
    if (pos == 1){
        temp = start;
        start = start -> next;
        free (temp);
        return;
    }
    ptr = start;
    for ( i = 1; i < pos - 1 && ptr != NULL; i++ ){
        ptr = ptr -> next;
    }
    if ( ptr == NULL || ptr -> next == NULL){
        printf (" Position not found \n");
    } else {
        temp = ptr -> next;
```

```
        ptr->next = temp->next;
        free(temp);
        printf("Node deleted\n");
    }
}
int count_elems(){
    int count=0;
    struct node * ptr = start;
    while(ptr != NULL){
        count++;
        ptr = ptr->next;
    }
    printf("Number of elements: %d\n", count);
    return count;
}
void reverse_ll(){
    int len = count_elems();
    if(len == 0) return;
    int arr[len];
    struct node *ptr = start;
    for(int i=0; i<len; i++){
        arr[i] = ptr->info;
        ptr = ptr->next;
    }
    printf("Reversed linked list: ");
    for(int i=len-1; i>=0; i--){
        printf("%d", arr[i]);
    }
    printf("\n");
}
void search(){
```

```
struct node * ptr;
int key, pos = 1, found = 0;
printf("Enter element to search:");
scanf("%d", &key);
ptr = start;
while (ptr != NULL){
    if (ptr -> info == key){
        printf("Element found at position %d",
                pos);
        found = 1;
        break;
    }
    ptr = ptr -> next;
    pos++;
}
if (!found){
    printf("Element not found\n");
}
}
```

Exp-5

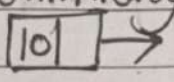
PAGE No.
DATE / /

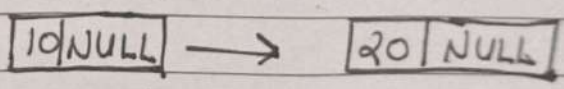
*Theory Question

Array Vs linked list

Feature	Array	linked list
Memory allocation	Contagious, static	Non-contagious dynamic
size	fixed	flexible
Access	direct	sequential
Memory Usage	No-extra pointer needed	Extra memory for pointers

Basic terminologies in linked list

1. Node 

2. Head
3. Tail
4. Next - Head → 

5. Next pointer
6. Empty linked list

1) Node
Basic building blocks of linked list. It contains data & next (address of next node)

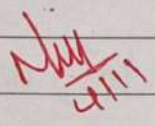
2) Head
Is the reference to the following node

3) Tail
last node where NEXT = NULL

4) Next
Each node has a next reference to the following node

5) NULL pointer
Special value that indicates that ptr does not have object or memory

6) Empty linked list
Singly linked list with no nodes in it.



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

struct node
{
    int info;
    struct node *next;
    struct node *prev;
};
struct node *start=NULL;
void create();
void display();
void insert_begin();
void insert_end();
void insert_pos();
void delete_all();
void delete_pos();
int count_elems();
int main()
{
    int choice;
    printf("Enter '1' to create:\n");
    printf("Enter '2' to Display:\n");
    printf("Enter '3' to Insert at the beginning:\n");
    printf("Enter '4' to Insert at the end:\n");
    printf("Enter '5' to Insert at a position:\n");
    printf("Enter '6' to Delete all elements:\n");
    printf("Enter '7' to Delete at a position:\n");
    printf("Enter '8' to Count the number of elements:\n");
```

```
printf("Enter '9' to search for an element:\n");
printf("Enter '10' to reverse the linked-list:\n");
printf("Enter '11' to exit:\n");
while(1){
printf("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: delete_all();
           break;
   case 7: delete_pos();
           break;
   case 8: count_elems();
           break;
   case 9: search();
           break;
   case 10: reverse_ll();
           break;
   case 11: exit(0);
```

```
        default: printf("\nInvalid input!");
    }
  }
    return 0;
}
void create()
{
    struct node *ptr,*temp;
    int len;
    printf("Enter number of elements:");
    scanf("%d",&len);
    for(int i=0;i<len;i++)
    {
        temp=(struct node*) malloc(sizeof(struct node));
        printf("Enter data:");
        scanf("%d", &temp->info);
        temp->prev = temp->next = 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;
    ptr=start;
    if(start==NULL)
    {
        printf("Empty list!\n");
        return;
    }
    printf("List elements:\n");
    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));
    printf("Enter data:");
    scanf("%d",&temp->info);
    temp->next=start;
    temp->prev=NULL;
```

```
    if(start!=NULL)
    {
    start->prev=temp;
    }
    start=temp;
}
void insert_end()
{
    struct node *temp,*ptr;
    temp=(struct node*) malloc(sizeof(struct node));
    printf("Enter data:");
    scanf("%d",&temp->info);
    temp->next = 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 insert_pos()
```

```
{
  struct node *ptr,*temp;
  int pos;
  temp=(struct node*) malloc(sizeof(struct node));
  printf("Enter data:");
  scanf("%d",&temp->info);
  temp->next=temp->prev=NULL;
  printf("Enter position:");
  scanf("%d",&pos);
  if(pos==1)
  {
    insert_begin();
    return;
  }
    ptr=start;
    for(int i=1;i<pos-1 && ptr!=NULL; i++)
    {
      ptr=ptr->next;
    }
    if(ptr == NULL)
    {
      printf("Position not found!\n");
      return;
    }
    temp->next=ptr->next;
    temp->prev=ptr;
    if(ptr->next !=NULL)
    {
      ptr->next->prev=temp;
```

```
    }
    ptr->next=temp;
}
void delete_all()
{
  struct node *temp;
  while(start!=NULL)
  {
    temp=start;
    start=start->next;
    free(temp);
  }
  printf("All values deleted!\n");
}
void delete_pos() {
  int pos;
  if (start == NULL) {
    printf("Empty list!\n");
    return;
  }
  printf("Enter position: ");
  scanf("%d", &pos);
  struct node *ptr = start;
  for (int i = 1; i < pos && ptr != NULL; i++)
    ptr = ptr->next;
  if (ptr == NULL) {
    printf("Position not found!\n");
    return;
  }
```

```
    if (ptr->prev != NULL)
        ptr->prev->next = ptr->next;
    else
        start = ptr->next;
    if (ptr->next != NULL)
        ptr->next->prev = ptr->prev;
    free(ptr);
    printf("Node deleted!\n");
}
int count_elems()
{
    struct node *ptr;
    int count=0;
    ptr=start;
    if(start==NULL)
    {
        printf("Empty list!\n");
        return;
    }
    while(ptr!=NULL)
    {
        ptr=ptr->next;
        count++;
    }
    printf("Total number of elements in the list:%d\n",count);
    return count;
}
void search()
{
```

```
int pos=1,found=0,key;
struct node *ptr;
printf("Enter element to search:");
scanf("%d",&key);
ptr=start;
while(ptr!=NULL)
{
   if(ptr->info==key)
   {
      printf("Element %d found at index %d\n",key,pos);
      found=1;
      break;
    }
     ptr=ptr->next;
     pos++
  }
  if(found==0)
  {
     printf("Element %d not in the list!\n",key);
  }
}
void reverse_ll() {
   struct node *ptr = start;
   if (ptr == NULL) return;
   while (ptr->next != NULL)
      ptr = ptr->next;
   printf("Reversed linked list: ");
   while (ptr != NULL) {
      printf("%d ", ptr->info);
```

Experiment 7

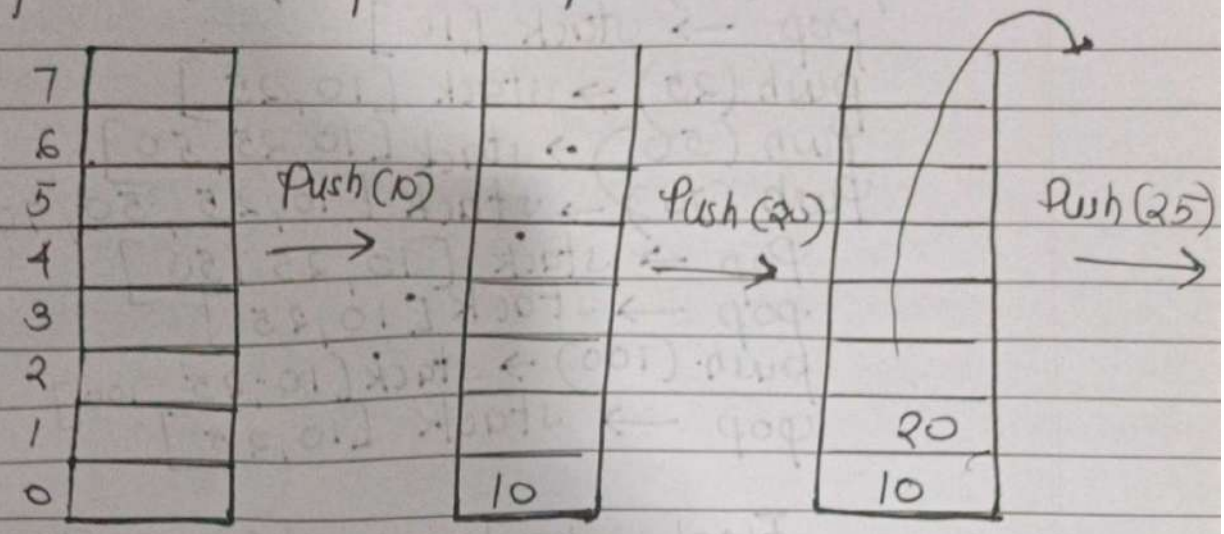
1. Primitive operations on stack : Pop, Push, isEmpty, isFull

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

int stack [MAX];
int top=-1;

void push (int n);
void pop( );
void isEmpty( );
void isFull( );

int main( )
{
  int value choice;
  printf (" Enter '1 to push:\n");
  printf (" Enter '2' to pop:\n");
  printf (" Enter '3' to check if stack is empty:\n");
  printf (" Enter '4' to check if stack is full:\n");
  printf (" Enter '5' to exit:\n");

  while (1)
  {
    Printf ("Enter your choice:");
    scanf(" %d", &choice);
    switch (choice)
    {
      case 1: printf ("Enter value:");
              scanf (" %d", &value);
              push (value);
```

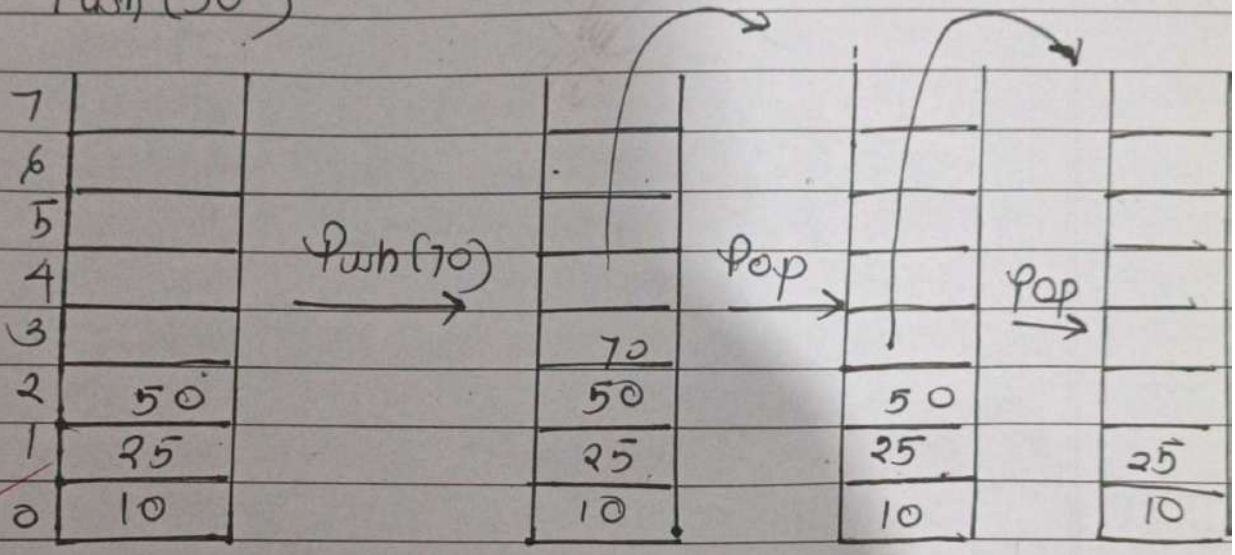
```
        break;
case 2: pop();
        break;
case 3: isEmpty();
        break;
case 4: isfull();
        break;
case 5: printf("Exiting!...\n");
        return 0;
default: Printf("Invalid input.\n");
    return 0;
}
void push(int n)
{
    if (top == MAX-1)
    {
        printf("stack overflow!\n");
    }
    else
    {
        stack[++top]=n;
        printf("%d added to stack\n\n");
    }
}
void pop()
{
    if (top==-1)
    {
        printf("stack underflow\n");
```

```
else
  {
  printf("%d element popped out!\n", stack
                [top--]);
  }
}
void empty()
{
  if (top==-1)
  {
    printf ("stack is empty!\n");
  }
  else
  {
    printf ("stack is not empty!\n");
  }
}
void ifull()
{
  if (top==MAX-1)
  {
    printf ("stack is full!\n"); }
  else
  {
    printf ("stack is not full!\n");
  }
```

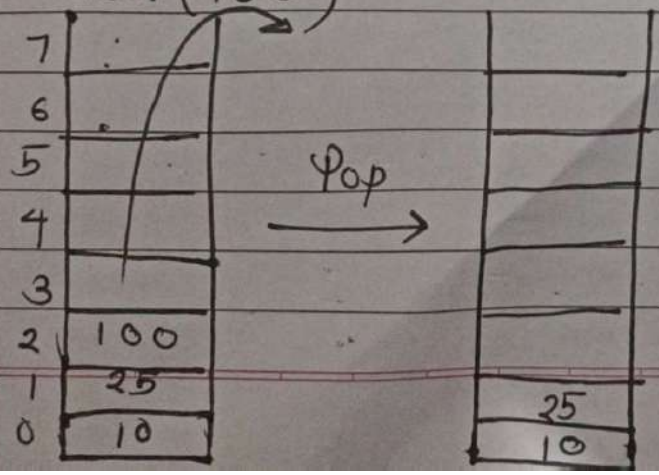
ii) Stack size - 8 for push(10), push(20), push(25), push 50, push(70), pop, pop, push(100), pop and draw final output



Push(50)



Push(100)

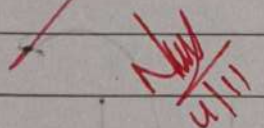


← Final Output

PAGE No.
DATE / /

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

Final output : stack [10,25]



Experiment 9

1. Perform primitive operation on linear queue-insert, display, delete

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

int queue[MAX];
int front=-1, rear=-1;
void enqueue(int n);
void dequeue();
void display();

int main()
{
    int choice, value;
    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;
```

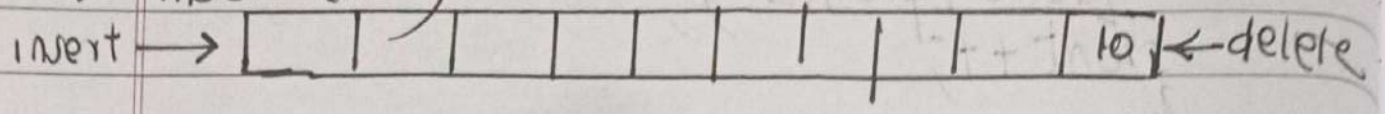
```
case 3: display();
        break;
case 4: printf("Exiting!...\n");
        return 0;
default: Printf("Invalid input\n");
}
}

Void enqueue(int n)
{
if (rear = MAX-1)
{
printf("Queue Overflow\n");
}
else
if (front==-1)
{
front=0;
}
rear++;
queue[rear]=n;
printf("%d enqueued!\n",n);
}
void dequeue()
{
if(front==-1 || front>rear)
{
printf("Queue Underflow!\n");
}
else
printf("%d dequeued from Queue!\n",
```

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

2. Perform following operations on linear queue in array size 10 with this operations insert(10) insert(30) delete, insert(100), insert(20), delete, insert(25), insert(200), insert(15)

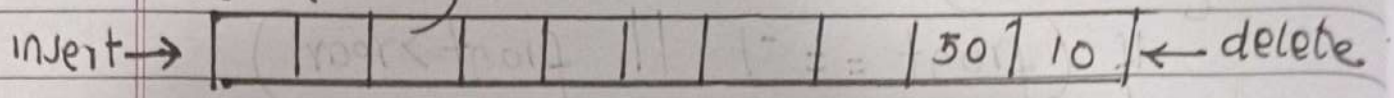
PAGE No.
DATE / /

→ Insert (10)



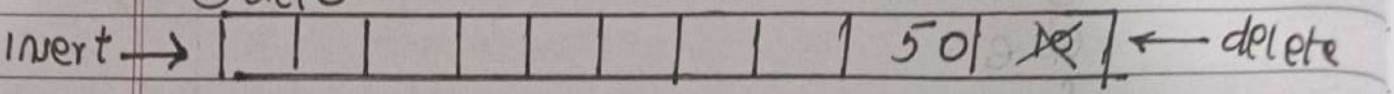
Rear = 0 Front = 0

Insert (50)



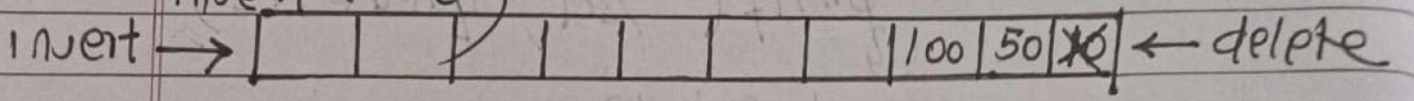
Rear = 1

Delete



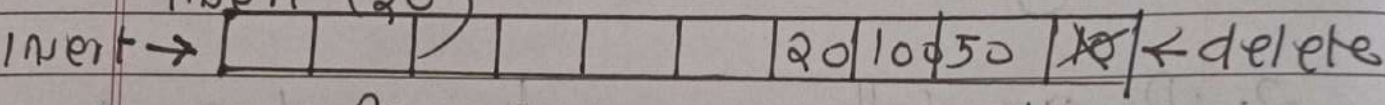
Rear = 0 front = 0

Insert (100)



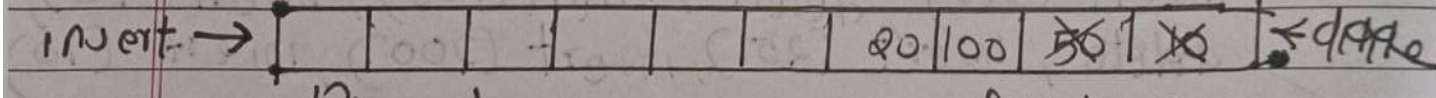
Rear = 1 front = 0

Insert (20)



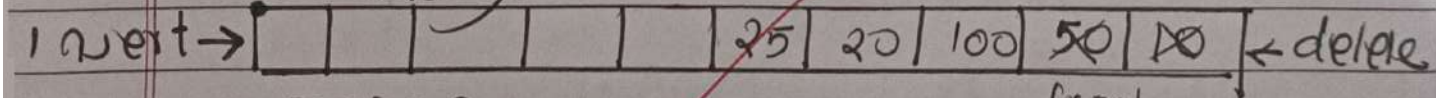
Rear = 2 front = 0

Delete



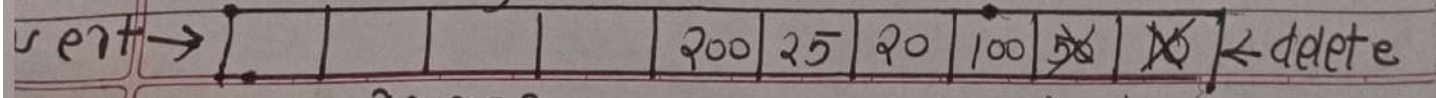
Rear = 1 front = 0

Insert (25)



rear = 2 front = 0

Insert (200)



rear = 3 front = 0

insert(10) if empty front=0, rear=1

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

insert(50) front=0 rear=0

[—, —, —, —, —, —, —, —, 50, 10]

front=0 rear=1

delete

[—, — — — — — — — — 50, ~~10~~]

front=0 rear=0

insert(100)

[—, —, —, —, —, —, —, 100, 50, ~~10~~]

front=0 rear=1

insert(20)

[— — — — — — — 20, 100, 50, ~~10~~]

front=0 rear=2

delete

[— — — — — — — — 20, 100 ~~50~~, ~~10~~]

front=0 rear=1

insert(25)

[—, —, —, —, —, 25, 20, 100, ~~50~~, ~~10~~]

front=0 rear=2

insert(200)

[— — — — 200, 25, 20, 100, ~~50~~, ~~10~~]

front=0 rear=3

PAGE No.
DATE / /

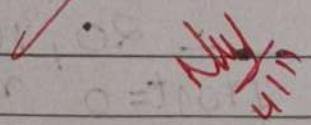
* Theory questions

Q1 Explain the concept of priority queue

A priority queue is a special type of queue where each element is assigned a priority and elements are served based on their priority not just their arrival order.

Q2. Write algorithm of insertion & deletion of linear queue insertion

1. check if the queue is in overflow condition
2. If no overflow condition check if front is equal to -1 and change it to 0
3. Increase rear by 1 equate the rear to the value entered
4. Print the value enqueued



Experiment - 10

```
#include <stdio.h>
#define MAX 5
int queue[MAX];
int rear=-1, front=-1;
void enqueue(value);
void dequeue();
void display();
int main()
{
  int choice, value;
   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("Enterchoice:");
      scanf("%d" &choice);
          switch(choice)
          {
    case1; printf("Enter value:");
           scanf("%d" &value);
           enqueue(value);
            break;
    case 2: dequeue();
            break;
     case 3: display();
            break;
      case4: printf("Exiting!\n");
            return 0;
      default :printf("Invalid input!\n");
    }
}
}
```

```
void enqueue (int n)
{
    if ((rear+1) % MAX == front)
    {
        printf("Queue Overflow!\n");
    }
    else
    {
        if (front == -1)
        {
            front = 0;
        }
        rear = (rear+1) % MAX;
        queue[rear] = n;
        printf("%d enqueued!\n", n);
    }
}
void dequeue()
{
    if (front == -1)
    {
        printf("Queue Underflow!\n");
    }
    else
    {
        printf("%d deleted from queue!\n",
                queue[front]);
        if (front == rear)
        {
            front = rear = -1;
        }
        else
```

```
{
  front = (front + 1) % MAX;
}
}
}
void display()
{
   If (front == -1)
   {
      printf ("Queue Empty!\n");
      ele
   {
   printf ("Queue Elements:\n");
     int i = front;
     while(1)
     {
        printf (" %d", queue[i]);
        If (i == rear) break;
          i = (i+1) % MAX;
        }
        printf ("\n");
     }
```

2. What are advantage of circular queue, show diagrammatic representation

1) Efficient use of memory - all array slot can be use
2) No need for shifting elements - unlike linear que
3. Better performance in repeated enqueue/dequeue operation
4. Practical application - Used in CPU scheduling, buffering

initial (Empty)

[_] [_] [_] [_] [_]

F, R

After enqueue: 10, 20, 30

[10] [20] [30] [_] [_]
R

After dequeue (remove 10)

[_] [20] [30] [_] [_]
F R

Enqueue 40, 50

[40] [20] [30] [50] [_]
R F

Enqueue 60 (wrap around)

[40] [20] [30] [50] [60]
R F

PAGE No.
DATE / /

Theory Question

Q1 Write algorithm for insertion & deletion operation on circular queue insertion algorithm.

1.) Check for overflow condition in queue
2.) Else if front and rear are equal to -1 change it to 0
3.) Else increase rear value by 1
4.) Print value inserted.

deletion algorithm
1.) Check for underflow condition in queue
2.) Else delete the value
3.) Print value deleted

Q2 White and explain application of queue
1. CPU scheduling: processes waiting to be executed by the CPU are stored in a ready queue
2. Breadth first search (BFS) - it uses queue
3. Os - manage access to resources like printers, files, memory.

Experiment - 12

1.

```
#include <stdio.h>
#define V 5
void init (int arr[V][V])
{
   int i, j;
   for (i=0; i<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;
   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];

    init(adjMatrix);

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

    printAdjMatrix(adjMatrix);

    return 0;
}
```

Output

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

Theory questions

Q1 Write all terminologies of graphs with example

Vertex (node): A fundamental unit of graph

Edge: A connection between two vertices

Graph: A collection of vertices & edges.

degree: The number of edges connected to a vertex

Q.2 Write matrix & list representation

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

list

list 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	1	0	0	0
B	1	0	0	1	0	0
C	0	0	1	0	1	0
D	0	0	0	0	0	1
E	1	0	0	0	0	1
F	0	0	1	0	0	0

2nd

list

A: [C]
B: [A, D]
C: [C, E]
D: [F]
E: [A, F]
F: [C]

Experiment -11

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
  int item;
  struct node *left;
  struct node *right;
  };
  void inorder(struct node *root){
   if (root==NULL){
   return; }
  printf("%d" root->item);
  preorder (root->right left);
  preorder(root->right); }
void postOrder(struct node *root){
   if(root==NULL){
  return ; }
post order(root->left)
post order(root->right);
printf("%d", root->item);
struct node *createNode (int item){
struct node *new node = (struct node*)
malloc(size of(struct node));
 new node->item = item;
 new node->left=NULL;
 new node->right=NULL;
 return newnode; }
 struct node *insertleft(structnode *root int item
 root->left=create Node(item)
```

```
return root->left;
struct node *insertRight(struct node *root int item){
struct node * new node = create Node(item);
return root->right; }
int main(){
struct node * root= createNode(1);
insert left (root, 2);
insert Right (root, 3);
insert left (root->left, 4);
printf("Inorder traversal:");
inorder(root);
printf("\nPre order traversal:");
preorder(root);
printf("\nPostorder traversal:");
postorder(root); }
```

1)
```
#include <stdio.h>
#include <stdlib.h>
struct node {
int data;
   struct node * left;
   struct node * right; };
   Struct node * createNode (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 {
   return createnode (value);
```

```
If (value < root → data)
  root → left = insert (root → left, value).
 else if (value → root → data)
  root → right = insert (root → right, value)
  return root; }
struct Node * deleteNode (struct Node * root,
                          int key) {
If (root == NULL) {
return root;
}
If (key < root → data
  root → right = deleteNode (root → right, key);
else { If (root → left == NULL) {
    struct Node * temp = root → right;
    free (root).
    return temp. }
  else if (root → right NULL) {
  struct Node * temp = root → right
  free (root);
  return temp;
}
int main () {
  root = insert (root, 50);
  insert (root, 30).
  insert (root, 10);
  insert (root, 50).
  root = deleteNode (root, 30);
  printf ("After deleting: ");
  return 0;
}
```