Practice Questions.

1. Declare integer array of 5 subjects, accept subject marks from user and find the percentage.

```
#include <stdio.h>
int main()
{

char* ~~arr~~ subjects = { "DSA", "ML", "RDBMS",
  "FAI", "OOP" }

int marks [5], sum=0;
float avg;
printf("Enter subject marks");
for(int i=0; i<5; i++)
{
    printf("Enter marks of %s", subjects[i]);
    scanf("%d", &marks[i]);
    sum+= marks[i];
}

avg = sum/5;

printf("Percentage = %d", avg);

return 0;
}
```

Output:

Enter ~~marks~~ Subject marks:
Enter marks of DSA = 96
Enter marks of ML = 95
Enter marks of RDBMS = 99
Enter marks of FAT = 75
Enter marks of OOP = 82.
Percentage = 89.4%.

2. Write a program to check if the no. is even or odd.

```
#include<stdio.h>
int main()
{
int n;
printf("Enter any number: ");
scanf("%d",&n);
if(n%2==0)
{
printf("The no. is even");
}
else
{printf("The no. is odd");
}
}return 0;
```

Output:
Enter any number: 10
The no. is even.

3. find prime nos. between 1 to 50.

```
#include<stdio.h>
int main()
{
int i, j, prime;
printf("Prime numbers between 1 and 50 are:\n");
for(i=2; i<=50; i++)
{
prime=1;
for(j=2; j*j<=i; j++)
{
if(i%j==0)
{
prime=0;
}break;
}
}
if(prime){
printf("%d",i);
}
}
}return 0;
```

Output:
Prime numbers between 1 and 50 are:
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47.

4. Write a program to accept a no. from the user and check if the no. is pallindrome or not.

```
#include<stdio.h>
int main()
{
    int num, orignal, reversed=0, remainder;
printf("Enter an integer: ");
scanf("%d",&num);

orignal=num;

while (num!=0)
{
    remainder= num %10;
    reversed= reversed* 10+remainder;
    num= num /10;
}

if (orignal==reversed) {
    printf("%d is a palindrome\n",
                    orignal);
}
else {
    printf("%d is not a palindrome\n",
                    orignal);
}
return 0;
}
```

Output:

Enter an integer: 121
121 is a palindrome number.

5. Menu driven monday to sunday weekday, weekend.

```
#include<stdio.h>
int main()
{
    int n;
printf("Enter 1 for monday");
printf("Enter 2 for tuesday");
printf("Enter 3 for Wednesday");
printf("Enter 4 for thursday");
printf("Enter 5 for Friday");
printf("Enter 6 for Saturday");
printf("Enter 7 for Sunday");

scanf("%d",&n);
```

```
switch (n)
  {
    case1:
    printf("It is a weekday");
    break;

    case2:
    printf("It is a weekday");
    break;

    case3:
    printf("It is a weekday");
    break;

    case 4:
    printf("It is a weekday");
    break;

    case 5:
    printf("It is a weekday");
    break;

    case 6:
    printf("It is a weekend");
    break.

    Case 7: Printf("It is a weekend");
    break;

    default:
```

```
    printf("Enter a valid number!");
    break;
  }
  return 0;
}
```

Output:

```
Enter 1 for monday:
Enter 2 for tudesday:
Enter 3 for wednesday:
Enter 4 for thursday:
Enter 5 for friday:
Enter 6 for Saturday:
Enter 7 for Sunday:
6
It is a weekend.
```

Experiment - 1

Q. Write a program to implement array operations:

1. Find avg of 10 nos. using an array

```
#include<stdio.h>
int main ()
{

char *subject = {"DSA", "AI", "ML", "OOP", "RDBMS", "Maths", "Phy", "Chem", "PIC", "PSPD"};

int marks [10], i, sum;
float avg;

printf ("Enter subjects marks: ");

for (i=0; i<10; i++)
{
printf("Enter marks for %s", subject);
scanf ("%d", &marks [i]);
sum += marks [i];
}

avg = Sum /10;
printf("Percentage = %d%%", avg);
return 0;
}
```

Output:

```
Enter subjects marks:
Enter marks for DSA: 90
Enter marks for AI : 90
Enter marks for ML : 92
Enter marks for OOP: 98
Enter marks for RDBMS: 99
Enter marks for Maths: 91
Enter marks for Phy : 97
Enter marks for Chem: 85
Enter marks for PIC: 99
Enter marks for PSPD: 95
Percentage = 93.6%
```

2. Display the following pattern

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

```
#include<stdio.h>
int main()
{
int i;
for (i=1; i<=4; i++){
switch (i){
case 1:
printf ("*\n");
break;
```

```
case 2:
        printf("##\n");
        break;
case 3:
        printf("***\n");
        break;
case 4:
        printf("####\n");
        break;
default:
        break;
    }
}

return 0;
}
```

Output:

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

8) Find 1st repeating element in an array.

```
#include<stdio.h>
int main()
{
int arr[100], n, i, j, found=0;
printf("Enter the number of elements
        in the array: ");
scanf("%d", &n);

printf("Enter %d elements:\n", n);
for(i=0; i<n; i++)
{
    scanf("%d", &arr[i]);
}
for(i=0; i<n-1; i++)
{
    for(j=i+1; j<n; j++)
    {
        if (arr[i]==arr[j])
        {
            printf("First repeating
element is %d\n", arr[i]);
            found=1;
            break;
        }
    }
    if (found)
        break;
}

if (!found)
```

```
{
printf("No repeating elements found.\n");
}
return 0;
}
```

Output:

Enter the number of elements in the array: 6
Enter 6 elements:
5 3 4 5 2 3

First repeating element is 5

4. Find the greatest & smallest element in an array.

```
#include<stdio.h>
int main()
{
int arr[100], n, i;
int max, min;
printf("Enter the number of elements in the array:");
scanf("%d", &n);
printf("Enter %d elements:\n", n);
for(i=0; i<n; i++)
{
scanf("%d", &arr[i]);
}
max = min = arr[0];
for(i=1; i<n; i++)
{
if(arr[i] > max)
{
max = arr[i];
}
if(arr[i] < min)
{
min = arr[i];
}
}
printf("Greatest element = %d\n", max);
printf("Smallest element = %d\n", min);
return 0;
}
```

Output:
Enter the number of elements in the array: 5
Enter 5 element:
10 45 3 22 17

Greatest element = 45
Smallest element = 3

5 program to square the odd positioned elements.

```
#include<stdio.h>
int main()
{
 int arr[100], n, i;
 printf("Enter the number of elements in the array: ");
 scanf("%d", &n);
 printf("Enter %d elements:\n", n);
 for(i=0; i<n; i++)
   {
    scanf("%d", &arr[i]);
   }
 for(i=0; i<n; i++)
   {
    if(i%2 != 0)
    {
     arr[i] = arr[i] * arr[i];
    }
   }
 return 0;
}
```

Output:

Enter the number of elements in the array: 6
Enter 6 elements:
1 2 3 4 5 6

Result:
1 4 9 16 5 36

Practice Questions

Q Print the following patterns

1.
```
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;
}
```

Output:
```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

2.
```
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++)
   {
     if (i%2 == 0)
       {
         for(j=1; j<=i; j++)
           printf("%d", j);
       }
     else {
         for(j=1; j<=i; j++)
           printf("%d", i);
       }
```

```
printf("\n");
}
return 0;
}
```

Output:

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

3.

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

```
#include<stdio.h>
int main()
{
int i;
for(i=0; i<=4; i++)
{
switch (i)
{
case1:
printf("*\n");
break;
```

```
case 2:
printf("# #\n");
break;

case 3:
printf("$$$\n");
break;

case 4:
printf("????\n");
break;

default:
break;
}
}
return 0;
}
```

Output:

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

4.
```
*
* *
* * *
* * * *
```

```
#include<stdio.h>
int main ()
{
int i,j, space;
int rows=4;

for(i=1;i<=rows;i++)
{
  for(space=1; space<=rows-i;
    space++)
  {
  printf("  ");
  }
for(j=1;j<=i;j++){
  printf("*  ");
}
printf("\n");
}

return 0;
}
```

Output:

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

5.
```
* * * *
  * * *
    * *
      *
```

```
#include<stdio.h>
int main ()
{
  int i,j,space;
  int rows=4;

  for(i=0; i<rows; i++)
  {
    for(space=0; space<i; space++)
    {
      printf("  ");
    }
    for(j=0;j<rows-i;j++)
    {
      printf("*  ");
    }
    printf("\n")
  }
  return 0;
```

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

Output:

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

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;
```

Experiment - 2.

Q1. Search Data using linear search:
consider the following list to perform linear search.
{56, 36, 89, 57, 01, 00, 67, 59}

1. Search the item 01 from the above list & write the item is found or not found with the procedure.
2. Search the item 55 from the above list and write the item is found or not found with procedure.

Q2. Search data using binary search.

Q3. Compare linear search v/s binary search.

Q4. State limitations of linear search in terms of complexity.

1.
```
#include<stdio.h>
int main()
{
  int arr[]={56,36,89,57,01,00,67,59};
  int k, i;
printf("Enter search element"); scanf("%d",&k);
 for(i=0; i<8; i++)
  {
```

```
if (K == arr[i])
  {
printf("%d found at index %d",
        arr[i], i);

    return 0;
  }
}

printf("Element not found in array");
  return 0;
```

Output:

if Search element is 01

01 found at index 4.

if search element is 55

Element not found in array.

Q2.
```
#include<stdio.h>
int main()
{

int i, K, l, h, m;
int n;

printf("Enter number of array elements
scanf("%d", &n);

int arr[n];

printf("Enter array elements\n");

for(i=0; i<n; i++)
  {
    printf("Enter element %d: ", i);
    scanf("%d", & arr[i]);
  }

printf("Enter element you want to
        search");
scanf("%d", &K);
l=0, h=n-1;
while (l<=h)
  {
    m=(l+h)/2;
```

```
if (arr[m] == K)
{
  printf("Element found at
        index %d, i);

  break;
}

else
{
  if (K > arr[m])
  {
    ~~print~~ m = l+1;
  }
  else
  {
    m=h-1;
  }
}
}
```

```
if (l>h)
{
  printf("Element not found in
        array!");
}

return 0;
```

3. Linear Search	Binary Search.
1. Scans each element in the list one by one.	1. Repeatedly divides a sorted list in half to find the target.
2. Works on unsorted or sorted data	2. Only works on sorted data.
3. Slower for large datasets	3. Much faster for large sorted datasets
4. Simple to implement	4. More complex due to midpoint calculation and sorting requirement
5. Small or unsorted lists	5. Large, sorted datasets.

4. The limitations of linear search in terms of time complexity are as follows:

a] Inefficient for large data sets: linear search has a time complexity of O(n), where n is the number of elements in the list.

b] No improvement for sorted data: linear search does not take advantages of sorted data.

c] Average and worst case are the same order: The average case & worst case both require scanning about half or all the elements repeatively; so it doesn't perform on average.

d] Not suitable for time critical applications: due to its linear time, it is not ideal for applications where fast searching is necessary, specially with large data sets.

Practice Questions:

1. Write a program to copy the elements of one array into another array in a reverse order.

```
#include<stdio.h>
int main()
{

 int a[100], b[100], i, r, n;
 printf("Enter number of array elements: ");
 scanf("%d", &n);
 printf("Enter array elements:\n");

 for(i=0; i<n; i++)
  {
   scanf("%d", &arr[i]);
  }
 r=n-1;

 for(i=0; i<n; i++)
 {
   b[r]=a[i];
   --r;
 }
 printf("\n New array:\n");
```

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

Output:

Enter number of array elements: 5
Enter array elements:
1 2 3 4 5

New Array:
5 4 3 2 1

2. WAP in C to count total number of duplicate elements in an array.

```
#include<stdio.h>
int main()
{

int arr[100], i, j, n, count=0;
printf(" Enter number of array elements: ");
scanf(" %d ", &n);
printf("Enter array elements:\n");
for(i=0; i<n; i++)
  {
    printf("Enter element %d: ", i);
    scanf(" %d ", &arr[i]);
  }

for(i=0; i<n; i++)
  {
    for(j=i+1; j<n; j++)
      {
        if (arr[i]==arr[j])
        {
          ++count;
        }
      }
  }

printf("Number of duplicate elements in array are %d", count);

return 0;
}
```

Output:

```
Enter number of array elements: 5
Enter array elements:
Enter element 0: 1
Enter element 1: 0
Enter element 2: 1
Enter element 3: 0
Enter element 4: 1

Number of duplicate elements in array:
```

3. WAP in C to print all unique elements in an array.

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

Output:

```
Enter no. of array elements: 5
Enter array elements:
2
5
3
1
4.
All unique elements in array:-
2
1
4.
```

4. WAP to seperate odd & even integer into seperate arrays.

```
#include<stdio.h>
int main()
{

int a[100], even[100], odd[100];
j, n, c1, c2;
printf("Enter no. of array elements:
scanf("%d", &n);
printf("Enter array elements:\n");
for (i=0; i<n; i++)
{
scanf("%d", &a[i]);
}
c1=0, c2=0;
for (i=0; i<n; i++)
{
if (a[i]%2==0)
{
even[c1]=a[i];
++c1;
}
else
{
odd[c2]=a[i];
++c2;
}
}
printf("Even array:\n");
for (j=0; j<c1; j++)
{
printf("%d", even[j]);
}
printf("Odd array:\n");
for (i=0; i<c2; i++)
{
printf("%d", odd[i]);
}
return 0;
}
```

Output:

Enter no. of array elements : 5
Enter array elements:
1
22
33
44
55

Even array : 22 44

Odd array : 1 33 55

5. WAP to find second smallest element in an array.

```
#include<stdio.h>
int main()
{
int a[100], i, min, min2, n;
printf("Enter no. of array elements");
scanf("%d", &n);
printf("\n Enter array elements:\n")
for (i=0; i<n; i++)
{
scanf("%d", a[i]);
}
min = a[0];
for (i=0; i<n; i++)
{
if (a[i] <min)
min = a[i];
}
for (i=0; i<n; i++)
{
if (a[i] != min)
min2 = a[i];
}
for (i=0; i<n; i++)
{
if (a[i] <min2 && a[i] != min)
min2 = a[i];
}
return 0;
}
```

Output:
Enter no. of array elements : 3.
Enter array elements
9
0
3
Smallest element in array : 0

8. WAP to count the total no. of words in a string.

```
#include <stdio.h>
int main()
{
  int i=0, count=0;
  char str[100], c;
  printf("Enter a string: ");
  scanf("%[^]n%", c str, &c);
  while (str[i] != '0');
  {
    if (i==0 || str[i]==' ')
    {
      ++count
    }
    ++i;
  }
  printf("No. of Words in string=%d", count);
  return 0;
}
```

Output:

Enter a string : Ram at an apple

No. of words in string: 4.

Experiment -3.

1. Sort elements in ascending order using bubble sort.

```
#include<stdio.h>
int main()
{

int i, j, n;
int temp=0;
printf("Enter number of array elements:");
scanf("%d",&n);

int arr[n];

printf("Enter array elements:\n");

for (i=0; i<n; i++)
{
  printf("Enter element %d:",i);
  scanf("%d",&arr[i]);
}

for(i=0; i<n; i++)
{
  for(j=0; j<n-1; j++)
  {
    if (arr[j]<arr[j+1])
    {

      arr[j]=temp;

      temp=arr[j];
      arr[j]=arr[j+1];
      arr[j+1]=temp;
    }
  }
}

printf("Array sorted in ascending order:\n")
for(i=0; i<n; i++)
{
  printf("\n%d",arr[i]);
}

return 0;
}
```

Output:
Enter number of array elements: 5
Enter array elements:
Enter element 0: 10
Enter element 1: 5
Enter element 2: 0
Enter element 3: 200
Enter element 4: 42
Array Sorted in ascending order:
0
5
10
42
200

2. Sort elements in ~~ascending~~ descending order using selection sort

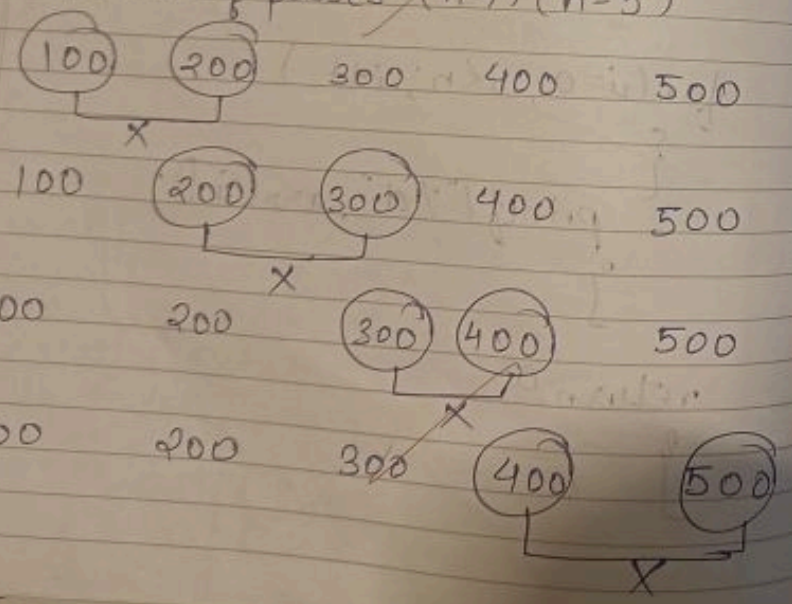
```
#include<stdio.h>
int main()
{
int i,j, n,temp=0;
printf("Enter number of array elements");
scanf("%d", &n);
int arr[n];
printf("Enter array elements:\n");
for(i=0; i<n; i++)
{
printf("Enter element %d:",i);
scanf("%d",&arr[i]);
}
for(i=0; i<n; i++)
{
for(j=i+1; j<n; j++)
{
if(arr[i]<arr[j])
{
temp=arr[i];
arr[i]=arr[j];
arr[j]=temp;
}
}
}
printf("Array Sorted in Descending order\n");
for(i=0; i<n; i++)
{
printf("%d", arr[i]);
}
return 0;
}
```

Output:

```
Enter number of array elements: 5
Enter array elements:
Enter element 0: 10
Enter element 1: 0
Enter element 2: 25
Enter element 3: 2
Enter element 4: 3
Array sorted in descending order:
 0  2  3  10  25
```

3. Find numbers of comparison required in bubblesort method of the following list.
100, 200, 300, 400, 500

Number of passes: (n-1) (n=5)

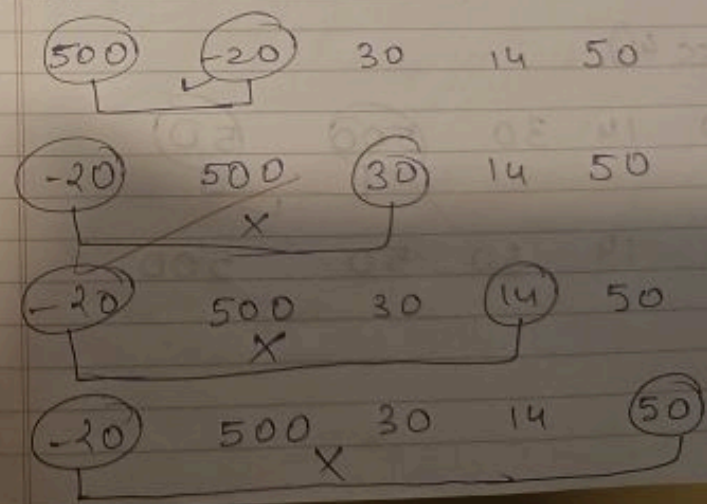


Algorithm:

1. Start with the list of no.
2. compare ~~with~~ each element of swap if the first element is greater than second element.
3. After each ~~follow~~ 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 the program.

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.

Pass 1.)



pass 2.)

-20 500 30 14 50

-20 30 500 14 50

-20 30 14 500 50

-20 30 14 500 500

pass 3.)

-20 14 500 30 50

-20 14 30 500 50

pass 4.)

-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.) Increament min to point the next element location.
5.) Repeat till sorted.
6.) Stop.

Experiment-4.

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

i) Sort element in ascending order using insertion sort.

Ans:
```
#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++)
  {
    printf("Enter element %d", i);
    scanf("%d", &arr[i]);
  }
for(i=1; i<n; i++)
  {
    ~~for(j=0; j<i;~~
  int current= arr[i];
int previous=i-1;
while(previous>=0 && arr[previous]>
                    current)
  {
    arr[previous+1]= arr[previous];
    previous--;
  }
  arr[previous+1]= current;
}

printf("Array sorted in ascending order
    is:\n");
  for(i=0; i<n; i++)
  {
    printf("%d", arr[i]);
  }
  return 0;
}
```

Enter number of array elements: 3
Enter the elements of the array:
Enter element 1: 25
Enter element 2: 1
Enter element 3: 02.
Array sorted in ascending order is:
1 2 25.

Q3. 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 5 1 9 4 8 6

3 7 5 1 9 4 8 6

2nd iteration:

3 7 5 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.

Q4. Sort the following numbers using radix sort.

1. 100 225 390 4130 956 99 5431.

Pass1	0	1	2	3	4	5	6	7	8	9.
0100	100	~~100~~								
0225			~~225~~			225				
0390	390									
4130	4130									
0956							956			
099										99
5431		5431								

Pass2.	0	1	2	3	4	5	6	7	8	9.
0100	0100									
0390			0225							0390
4130				4130						
0956						0956				
0099										0099
5431				5431						
0225										

pass3	1	2	3	4	5	6	7	8	9	0
0100	0100									
0225 ← 4130		0225								
5431	4130									
0956				5431						
0390									0956	
0099			0390							
										0099

pass 4 0099, 0100, 4130, 0225, 0390, 5431, 0956

pass4:	0	1	2	3	4	5	6	7	8	9
0090	0090									
0100	0100									
4130					4130					
0225	0225									
0390	0390									
5431						5431				
0956	0956									

Sorted Array: 90, 100, 225, 390, 956, 4130, 543

2. 256, 99, 145, 239, 20, 18.

pass1	0	1	2	3	4	5	6	7	8	9
256							256			
099										099
145						145				
239										239
020	020									
018									018	

020, 145, 256, 018, 099, 239

pass2	0	1	2	3	4	5	6	7	8	9
020			020							
145					145					
256						256				
018		018								
099										099
239				239						

018, 020, 239, 145, 256, 099

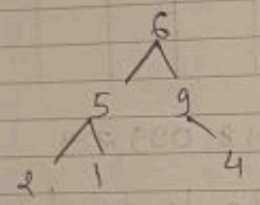
pass3	0	1	2	3	4	5	6	7	8	9
018	018									
020	020									
239			239							
145		145								
256			256							
099	099									

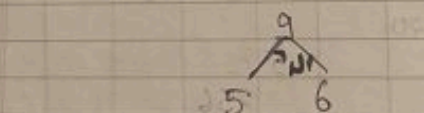
Sorted array: 18, 20, 99, 145, 239, 256

Q5. Sort the following elements using heap sort:

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

pass 1

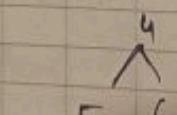


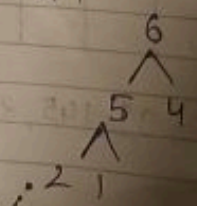


9, 5, 6, 2, 1, 4

4, 5, 6, 2, 1, 9

pass 2.

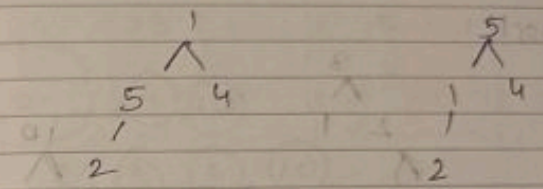




6, 5, 4, 2, 1, 9

pass 3

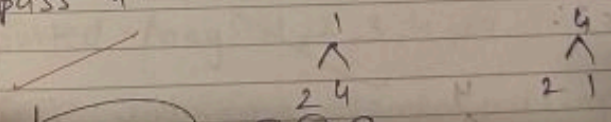
1, 5, 4, 2, 6, 9



5, 2, 4, 1, 6, 9

1, 2, 4, 5, 6, 9

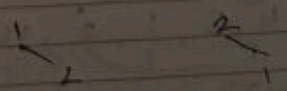
pass 4:



4, 2, 1, 5, 6, 9

1, 2, 4, 5, 6, 9

pass 5:



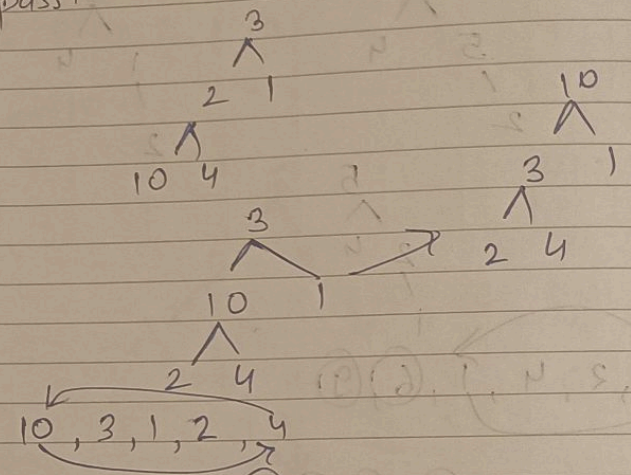
2, 1, 4, 5, 6, 9

1, 2, 4, 5, 6, 9

Sorted Array: 1, 2, 4, 5, 6, 9

ii) 3, 2, 1, 10, 4

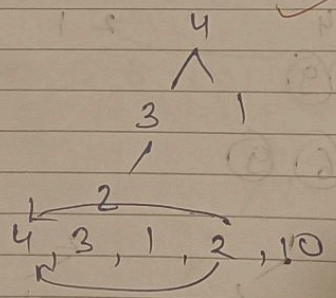
pass 1:



10, 3, 1, 2, 4

4, 3, 1, 2, 10

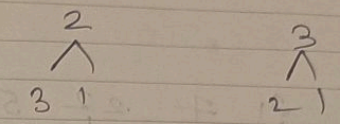
pass 2:



4, 3, 1, 2, 10

pass 3:

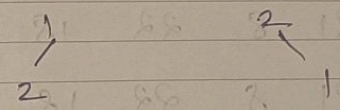
2, 3, 1, 4, 10



3, 2, 1, 4, 10

1, 2, 3, 4, 10

pass 4:



2, 1, 3, 4, 10

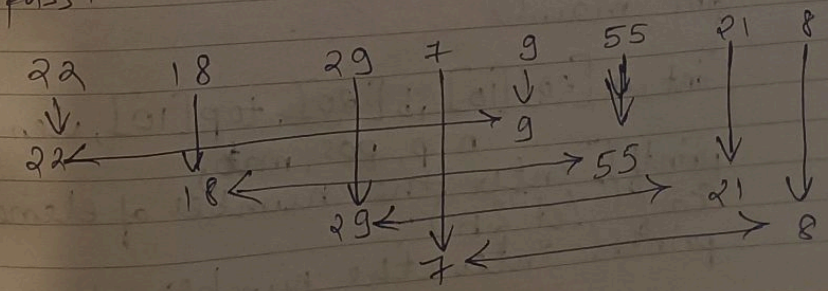
1, 2, 3, 4, 10

Sorted Array: 1, 2, 3, 4, 10.

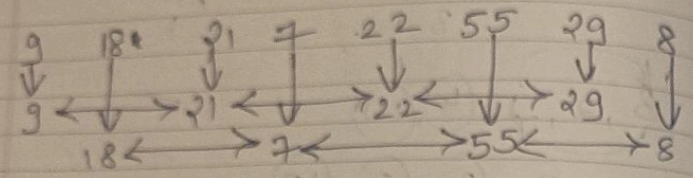
Q5. Sort following elements using shell sort.

1. 22, 18, 29, 7, 9, 55, 21, 8

pass 1:



pass2:



pass 3:

9 7 21 8 22 18 29 55

7 9 21 8 22 18 29 55

7 8 9 21 22 18 29 55

7 8 9 18 21 22 29 55

Sorted Array: 7, 8, 9, 18, 21, 22, 29, 55.

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, pos, mat;
printf("Enter the number of elements:")
Scanf("%d &n);
printf("Enter the number of array
elements: \n");
```

```
printf for(i=0; i<n; i++)
{
    printf("Enter element %d: ", i+1);
    Scanf("%d", &b[i]);
}

max = b[0];
for(i=0; i<n; i++)
{
    if (b[i]>max){
        max == b[i];
    }
len = (int)log10(max)+1;
for(i=0; i<len; i++)
{
    for(j=0; i<10; j++)
    {
        top[j]=-1;
    }
    for(j=0; j<n; j++)
    {
        pos=(int)(b[j]/pos(10, i))%10;
        a[pos][++top[pos]]=b[j];
    }
    for(j=0; n=-1; j<10; j++)
    {
        for(k=0; k<=top[j]; k++)
        {
            b[++p] = a[j][k];
        }
    }
}
```

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

Output:

```
Enter the number of elements: 5
Enter array elements:
Enter element 1: 45
Enter element 2: 35
Enter element 3: 64
Enter element 4: 99
Enter element 5: 78
Sorted Array:
35, 45 64 78 99.
```

Nijoul 27/9/25

Exp - 5

Singly linked list code:

code:-

```
#include<stdio.h>
#include<stdlib.h>
void create();
void display();
void insert_begin();
void insert_end();
void insert_pos();
void delete_all();
void delete_pos();
int count_elems();
void reverse_ll();
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_elems(); break;
  case 9: reverse_ll(); 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(sizeof(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 display()
{
    struct node *ptr;
    if(start==NULL)
    {
        printf("\nEmpty list\n");
        return;
    }
```

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

void insert_begin()
{
struct node *temp, *ptr;
temp=(struct node*) malloc (sizeof(
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(size of
(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 0;
void reverse_dl() {
struct node *ptr;
int Key,pos=1,found=0;
printf("Enter element to search:");
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\n", pos);
      found=1;
    } break;
    ptr=ptr->next;
    pos++;
  }
  if (!found) {
    printf("Element not found\n");
  }
}
```

30/9/25

Experiment-8.

1. Convert infix to postfix

```
#include<stdio.h>
#include<ctype.h>
char stack [100];
int top=-1;
void push (char x)
{
   stack [++top]=x;
}
char pop ()
{
  if (top==-1)
    return 1;
  else
    return stack [top--]
}
int priority (char x)
{
  if (x='(')
    return 0;
  if (x='+'|| x='-')
    return 1;
  if (x='*'|| x='/')
    return 2;
  return 0;
}
int main()
{
  char exp[100];
  char *e ,x;
  printf("Enter the expression:");
  scanf("%s",&exp);
  printf("\n");
  *e=exp;
  while(*e!='\0')
  {
    if (isalnum(*e))
      printf("%c",*e);
    else if (*e=='(')
      push(*e);
    else if (*e==')')
    {
      while ((x=pop())!='(')
      printf("%c",x);
    }
    else
    {
      while (priority(stack[top])=
                priority(*e))
      {printf("%c",pop());
      push(*e);
    }
    e++;
  }
  {while(top!=-1)
  {printf("%c",pop());
}
  return 0;
```

ii Convert infix into prefix using stack.
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	+	HGFE$D/*CB*-*A
		HGFE$D/*CB*-*A+

prefix: +A*-*BC*/D$EFGH.

iii Write a C program to evaluate infix or 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 pop ()
{
 int n;
 if (top > -1)
 {
     n = stack [top]
         stack [top--] = -1;
         return 0;
 }
  else {
  printf ("stack is empty\n");
  exit (-1);
 }
}
```

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 into the stack>
   - if the symbol is operator <follow the rules of operator.
5. Repeat ~~until~~ until end end of expression.
6. The final result will be only element left in stack.
7. Stop.

iv Evaluate prefix expression using stack:
+ - * +12 /42 1 $42

I/P	OP1		OP2	Result	Stack
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 upto stack.
   - If operator →
     - pop two operands from stack.
     - Apply operator → get result.
     - Push result back on stack
5. After scanning all symbols, the stack top element is the final element.

i 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/equal precedence 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 Algorithm.
1. Start
2. Scan from left to right.
3. if the char is operand copy it is to prefix.
4. If the char is closing parenthesis then push to stack.
5. If char scanned is an operator.
6. Stop.

1/9/19

5. Stop.

Experiment 7

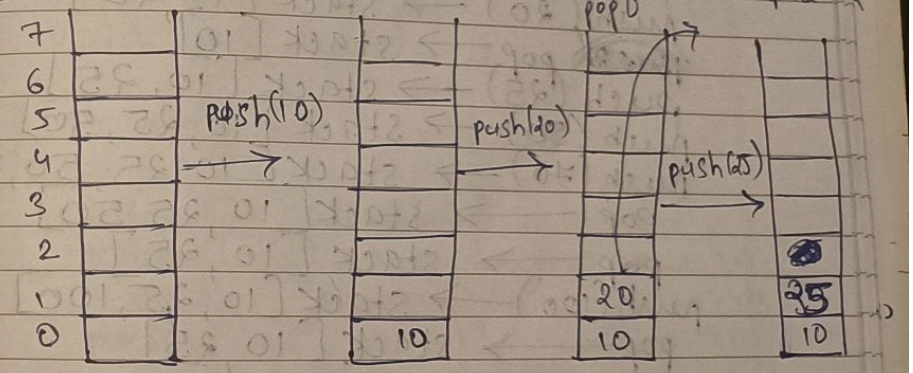
1. primitive operations on stack: Pop, Push, isEmpty, isfull.

```
#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",
           n);
}}
```

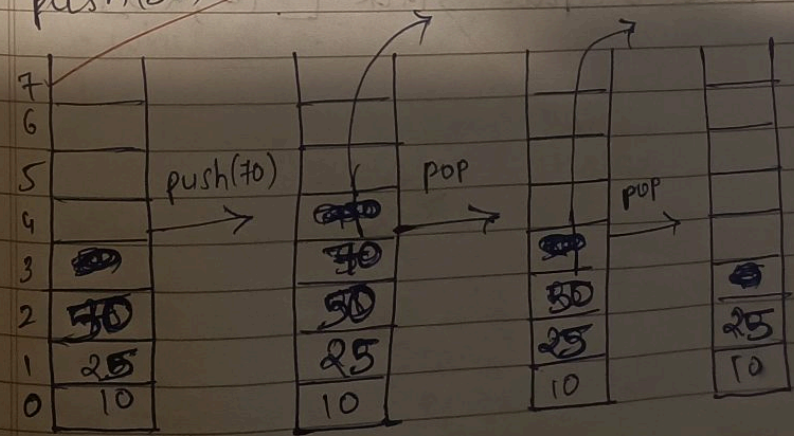
```
void pop()
{
    if (top==-1)
    {
        printf("Stack underflow\n");
    }
    else
    {
        printf("%d element popped out!\n",
                stack[top--]);
    }
}

void isEmpty()
{
    if (top==-1)
    {
        printf("Stack is empty!\n");
    }
    else
    {
        printf("Stack is not empty!\n");
    }
}

void isFull()
{
    if (top==MAX-1)
    {
        printf("Stack is full!\n");
    }
    else
    {
        printf("Stack is not full!\n");
    }
}
```

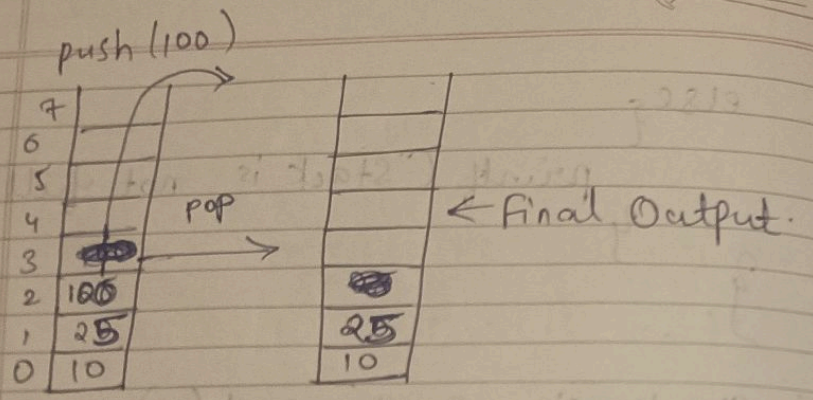
LIFO

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)





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.

" 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(~~front~~ 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 ~~over~~ Underflow!\n");
    }
  else {
    printf("%d dequeued from Queue!\n",
                queue[front]);
    } front++;
  }

void display()
{
  if{(front==-1 && front>rear)
    {
      printf("~~Stack~~ 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 (50), delete, insert (100), insert (20), delete, insert (25), insert (200).

→ insert(10)

insert → [| | | | | | | | | 10] ← delete

Rear = 0 front = 0

insert (50)

insert → [| | | | | | | | 50 | 10] ← delete

Rear = 1 front = 0

Delete

insert → [| | | | | | | | 50 | ~~10~~] ← delete

Rear = 0 front = 0

insert (100)

insert → [| | | | | | | 100 | 50 | ~~10~~] ← delete

Rear = 1 front = 0

insert (20)

insert → [| | | | | | 20 | 100 | 50 | ~~10~~] ← delete

Rear = 2 front = 0

Delete

insert → [| | | | | | | 20 | 100 | ~~50~~ | ~~10~~] ← delete

Rear = 1 front = 0

insert (25)

insert → [| | | | | 25 | 20 | 100 | ~~50~~ | ~~10~~] ← delete

rear = 2. front = 0

insert (200)

insert → [| | | | 200 | 25 | 20 | 100 | ~~50~~ | ~~10~~] ← delete

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~~ , ~~80~~]

front = 0 rear = 2

insert (200)

[_ , _ , _ , _ , 200 , 25 , 20 , 100 , ~~50~~ , ~~80~~]

front = 0 rear = 3.

* 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.

Null 30/9/25

→ 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.

Exp - 5

* 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 [10] →
2. Head
3. Tail
4. ~~Next~~ - Head → [10 | NULL] → [20 | NULL] ← Tail
5. Next pointer
6. Empty linked list. Head → Null

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

(2) Head - The head is a 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.

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("Enter choice: ");
    scanf("%d",&choice);
    ~~switch~~ 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+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");
    }
    else
    {
        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 the advantages of circular queue, show diagrammatic representation

→ ① Efficient use of memory - all array slots can be used.
② No need for shifting elements - unlike linear queues.
③ Better performance in repeated enqueue / dequeue operations.
④ practical applications - used in CPU scheduling, buffering.

Initial.. (Empty)

[_] [_] [_] [_] [_]
F, R.

After enqueue: 10, 20, 30.

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

After dequeue (remove 10)

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

Enqueue 40, 50:

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

Enqueue 60 (wraps around:)

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

* Theory Questions.

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.

Q3. Diff stack & queue.

Stack	Queue
(1) Last in first out	(1) First in, first out
(2) Elements are added at the top	(2) Elements are added at the rear
(3) Elements are removed from the top	(3) Elements are removed from the front.
(4) Function calls, undo operations	(4) Scheduling, buffering task queues.

Q2. Write and explain applications 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 - manages access to resources like printers, files, memory.

Experiment - 6.

1. Implement primitive operations on doubly linked list

DSA_C > C doubly_linkedlist.c > (global) start

```c
#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;
```

main* 11 0 0 0

```c
        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();
```

```c
        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");
}
```

main* 11 0 0 0

```c
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()
```

main* 11 0 0 0

```c
}
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)
```

main* 11:01 0 0

```c
    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)
```

main* 11:01 0 0

```c
    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;
```

main* 11:01 0 0

```c
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);
        ptr = ptr->prev;
    }
    printf("\n");
}
```

Experiment-12.

```
#include<stdio.h>
#define V5
void init(int arr[][V])
{
  int i,j;
  for(i=0; i<V ; i++)
  {
    for (j=0 ; j<V ;j++)
      arr[i][j]=0 ;
  }
}

void insertEdge (int arr[][V] ,int i,int j)
{
  arr [i][j]=1;
  arr [j][i]=1;
}
void printAdjMatrix (int arr[][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];
  int (adjMatrix);
  insertEdge(adjMatrix ,0 ,1);
  insertEdge(adjMatrix ,0 ,2);
  insertEdge(adjMatrix ,1 ,2);
  insertEdge(adjMatrix ,2 ,0);
  insertEdge(adjMatrix ,2 ,3);
  insertEdge(adjMatrix ,0 ,3);
  insertEdge(adjMatrix ,0 ,0);
  insertEdge(adjMatrix ,1 ,3);
  insertEdge(adjMatrix ,4 ,3);
  insertEdge(adjMatrix ,2 ,4);
  printAdjMatrix(adjMatrix);
  return 0;
}
```

Output:

Adjency Matrix:

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

Q1. Write all terminologies of graphs with example.

Ans1. 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.

Q2. Write a matrix & list representation:

Ans2.

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

1st

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

Experiment 11

implement primitive operations on binary tree.

```
#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; }
  inOrder (root->left);
  printf ("%d", root->item);
  inOrder (root->right); }
void preorder (struct node *root) {
  if (root==NULL){
      return; }
  printf ("%d", root->item);
  preorder (root->left);
  preorder (root->right); }
void postOrder (struct node *root){
  if (root==NULL){
      return; }
  postOrder (root->left);
  postOrder (root->right);
  printf ("%d", root->item); }
struct node *createNode (int item){
struct node *newnode=(struct node*)
malloc (sizeof (struct node));
newnode->item = item;
newnode->left = NULL;
```

```
newnode -> right = NULL;
 return newnode; }

struct node *insertleft (struct node *
root int item) {
   root -> left = Create Node (item);
    return root -> left; }
struct node *insertRight (struct node
*root int item) {
 struct node * newnode = create Node
                    (item);
 return root -> right; }
int main() {
 struct node *root = Create Node(1);
 insertLeft (root, 2);
 insertRight (root, 3);
 insert left (root -> left, 4);
 printf (" Inorder traversal: ");
  inOrder (root);
  printf ("\nPreorder traversal: ");
  preorder (root);
  printf ("\nPostorder traversal: ");
  postorder (root); }
```

i)

```
#include<stdio.h>
#include <stdlib.h>
struct node {
int data;
 struct node *left;
 struct node *right; };
struct node* Create Node (int value)
struct Node* newnode = (struct Node*)
malloc (sizeof (struct Node));
newNode -> data = value;
newNode -> left = newNode -> right = NULL;
return newNode; }
struct node * insert (struct Node*
root, int value) {
if (root == NULL) { 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 = delete Node (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, 70);
  insert (root, 50);
  root = deleteNode (root, 30);
printf (" After deleting: ");
 return 0; }
```

Exp-10.

Perform primitive operations on circular queue

```
#include <stdio.h>
#define MAX 5;
int queue[MAX]
int rear=-1, front=-1;
void enqueue(value);
void dequeue();
void display();
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){
case1: printf("Enter value:");
scanf("%d", &value);
        break;
case2: dequeue(); break;
case3: 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 enqueue!\n", n);}}
void dequeue(){
  if (front==-1){
    printf("Queue Underflow!\n");}
  else{ printf("%d deleted from queue", queue[front]);
    if (front==rear){
      front=rear=-1;}
    else{ front=(front+1)%MAX;}}}
void display(){
  if (front==-1){
    printf("Queue Empty!\n");}
  else{ printf("Queue Elements:\n");
    int i=front;
    while(1){
      printf("%d", queue[i]);
      if (i==rear) break;
        i=(i+1)%MAX;}
    printf("\n");}}
```

Q2. What are advantages of circular queue show digrammatic representation.

1.) Efficient use of memory - all array slots can used.

2.) No need for shifting elements - unlike linear queue.

3.) Better performance in repeated enqueue/dequeue operations.

4.) Practical applications - used in CPU scheduling, buffering.

initial (Empty)

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

F, R.

After enqueue: 10 20 30

[10] [20] [30] [ ] [ ]

After dequeue (remove 10)

[_] [20] [30] [_] [_]

Enqueue 40, 50

[40], [20] [30] [50] [ ]

R F

Enqueue 60 (wrap around)

Enqueue 60

[40] [20] [30] [50] [60]

R F

Theory Question

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

→ insertion:

1.) Check for overflow condition.

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:

1.) Check for underflow condition in queue.

2.) Else delete the value.

3.) print value deleted.

Q2. Write & explain applications of queue.

→ 1.) CPU scheduling: processes waiting to be executed by the CPU.

2.) Breadth-first-search.

3.) OS.

Nly 4th