* Experiment 1 -

1.) WAP to implement array operation.
1.) Find avg of 10 no. using array.

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

  int i, num[10], sum=0, avg;

printf ("Enter 10 elements of an array");
 for (i=0; i<10; i++) {
  printf ("Number %d", num+1);
  scanf ("%d", &num[i]);
}
   for (i=0; i<10; i++) {

   sum = sum + num[i];

   }

float average = sum / 10;

  printf ("The average is : %.2f \n",
          avgerage);

  return 0;

}
```

2) Display the following pattern.

O/P -

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

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

int i,j;
char pattern[4] = {'*', '#', '*', '#'};

printf ("Pattern :");

for (i = 0; i<=4; i++) {

for (j=1; j<=i; j++) {

printf if ( i%2 == 0) {

printf ("*");
} else {

printf ("#");
}
}

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

3.) Find first repeating element of array:

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

int n, j, arr[100] i;

printf ("Enter no of elements array:
scanf ("%d", &n);

printf ("Enter elements of array:")
for (i=0; i<n; i++) {

scanf ("%d", &arr[i]);
}

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

printf ("The repeating
element is %d \n", arr[i]);

return 0;

}
```

for (j = i+1; j<n; j++)

4)8 Find greatest & smallest element in an array.

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

int arr[100], n, i;
int min = arr[0], max = arr[0];

printf("Enter no. of elements:");
scanf("%d", &n);

printf("Enter elements of array");
for (i=0; i<n; i++) {
scanf("%d", &arr[i]);

for (i=0; i<n; i++) {

if (min > arr[i]) {
    min = arr[i]
}

if (max < arr[i]) {
    max = arr[i]
}

printf("The greatest & smallest elements of array are %d & %d",
    max, min);
return 0;
}
```

5) WAP squaring of position element.

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

int arr[100], n, i;

printf("Enter no. of elements");
scanf("%d", &n);

printf("Enter elements of array");
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];
}
}

printf("New array is:");
for (i=0; i<n; i++) {

printf("%d", arr[i]);
}
return 0;

}
```

0 1 2 3 4 → i
1 2 3 4 5

* Output:

1) enter an array: 1 2 3 4 5 6 7 8 9 2
the average of ten numbers is: 4.0000

3) enter number of elements in an array: 5
enter elements in an array 3 5 7 8 5
the repeating element is: 5

4) Enter the number of elements: 7
Enter elements of array
1
4
5
8
9
4
3
largest element = 9
smallest element = 1

5) enter number of elements in array 6
enter the elements in array 1
3
4
5
6
7
The new array is:
1 3 16 5 36 7

* Display following pattern:

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

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

int i, j;

for (i=1; i<=4; i++) {

    for (j=1; j<=i; j++) {

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

2)
```
    *
   * *
  * * *
 * * * *
```

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

3)

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

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

4)

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

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

i j
1 → 1,2,3,4 3 → 1,2,3

```
for (j=1; j<=i; j++) {
    printf("*_");
}
printf("\n");
}
return 0;
}
```

5>
```
1
2 2
3 3 3
4 4 4
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);   ("%d_", i)
  }
  printf("\n");
}
return 0;
}
```

6>
```
1
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;

}
```

7>
```
* * * *
 * * *
  * *
   *
```

inverted change i

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

int i,j;
for (i=4; i>=1 ; i++){
   for (j=1; j<=4-i; j++){
      printf("_");
   }
   for (j=1; j<=2*i-1; j++){
      printf("*");
   }
   printf("\n");
}
return 0;
}
```

8)
```
*
* #
* # ?
*# ? $
```

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

OR

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

9.)

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

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

int i,j;

for (i=0;i<=4;i++) {

   for (j=1;j<=i;j++) {

    if (i==1)
       printf("*_");
    elseif (i==2)
       printf("#_");
    elseif (i==3)
       printf("$_");
    else if (i==4)
       printf("?_");
   }

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

}
```

* Experiment 2:

Q1.) Search data using linear search. Consider the following list to perform linear search:

56, 36, 89, 57, 1, 0, 67, 59

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

 int arr[8], i, key;

printf("Enter elements of the array:");
for(i=0; i<8; i++) {
 scanf("%d", &arr[i]);
}

printf("Enter element you want to search:");
 scanf("%d", &key);

for(i=0; i<8; i++) {

 if(arr[i] == key) {
  printf("Element %d is found
          at position: %d", arr[i
  break;
 }
}
```

```
if (i==8)
{
  printf ("\n Element not found.");

}
return 0;

}
```

O/P - enter elements of the array : 56
36
89
57
1
0
67
59
enter element you want to search: 1
element 1 is found at position: 5

O/P - enter elements ~~you want to search~~ of the array: 56
36
87
57
1
0
67
59
enter element you want to search: 55
elenment ~~55~~ not found ~~at position~~.

Q2.) Search data using binary search:

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

  int i, arr [8], h, l, key, m;

  printf ("Enter elements of array in sorted
          order:");
  for (i=0; i<8; i++) {
    scanf ("%d", &arr[i]);

  printf ("Enter element you want to search:
  scanf ("%d", &key);

  l = 0;
  h = 7;

  while (l<h) {
   m = (l+h)/2;
   if( arr[m] == key) {
    printf ("Element %d is found at
            position : %d ", arr[m], m+1)
   }
   else {
    if( key > arr[m]) {
      l = m+1;
    }
    else {
```

```
        h = m-1;
    }
}
}

if (l > h){
    printf ("Element not found.");
}

return 0;

}
```

O/P - enter elements of array in sorted order:

56	0
36	1
89	36
57	56
1	57
0	59
67	67
59	89

element you want to search: 1
element 1 is found at position 2

O/P - enter elements of array is sorted order:

56	0
36	1
89	36
57	56
1	57
0	59
67	67
59	89

enter element you want to search: 55
element not found.

Q3> Compare linear & binary search:

	linear search	Binary search
1.	It works on both sorted & unsorted arrays.	It works only on Sorted arrays.
2.	Time complexity in (n)	Time complexity in (log n)
3.	It is used for small & unsorted data set	It is used for large data sets
4.	Simple to implement	Complex to implement
5.	Compares with each / first element	Compared with middle element
6.	Algorithm type is sequential	Algorithm type divide & conquer.

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

Linear search has limitations in terms of time complexity especially for large datas. It's worst case and average case time complexity is (n) meaning it may need to scan every element in array to determine, if it is present or not, which makes it slower. The binary search is on sorted data. Hence performace is poor & data size grows.

* Extra Questions -

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

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

int arr1[10], arr2[10], i;

printf ("Enter elements of the array:"
for (i=0; i<10; i++) {
    scanf ("%d", &arr1[i]);
}

for (i=0; i<10; i++) {
  arr1[i] = arr2[9-i];
}

printf ("The given array is:");
for(i=0, i<10; i++) {
 printf ("%d", arr1[i]);
}

printf ("The array in reversed order is
for (i=0; i<10; i++) {
 printf ("%d", arr2[i]);
}
return 0;
}
```

Enter elements of array: 1
2
3
4
5
6
7
8
9
10

The given array is: 1 2 3 4 5 6 7 8 9 10
The array in reversed order is:
10 9 8 7 6 5 4 3 2 1

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

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

int arr[10], i, j, count = 0;

printf ("Enter elements of an array:");
for (i=0; i<10; i++) {
    Scanf ("%d", &arr[i]);
}
```

```
for (i=0; i<10; i++) {
    for (j=i+1; j<10; j++) {
        if (arr[i] == arr[j]) {
            count ++;
            break;
        }
    }
}

printf (" The total number of duplic
 elements are : %d", count);

return 0;
}
```

enter elements of array: 1
2
3
4
5
3
4
6
7
1

The total number of repeated elements are :3

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

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

int arr[10], i, j, count;

printf("Enter elements of array:");
for(i=0; i<10; i++){
    scanf("%d", &arr[i]);
}

printf("Unique elements are:");

for(i=0; i<10; i++){
count=0;
for(j=0; j<10; j++){
 if(arr[i] == arr[j]){
   count++;
    }
}

if(count==1){
  printf("%d ", arr[i]);
}
}
return 0;
}
```

```
enter elements of array:1
2
3
4
5
6
7
6
7
8
The unique elements are: 1 2 3 4 5 8
```

4.) WAP to ~~repeat~~ seperate odd and even integers into seperate array.

```
# include <stdio.h>
int main(){
int arr[10] = {1,2,3,4,5,6,7,8,9,
int ~~arr[10]~~, int even[10], odd
int i, evencount=0, oddcount=
```

```
for(i=0; i<10; i++){
   if(arr[i]%2 == 0){
     even[evencount] = arr[i];
     evencount++;
   }
else{
  odd[oddcount] = arr[i];
  oddcount++;
```

```
    }
}

printf ("Even elements :");
for (i=0; i<evencount ; i++) {
    printf ("%d", even[i])
}
printf ("\n");

printf ("Odd elements :");
for (i=0; i<oddcount; i++) {
    printf ("%d", odd[i]);
}
printf ("\n");

return 0;

}
```

Even elements : 2 4 6 8
Odd elements : 1 3 5 7 9 3

5) WAP to find second smallest element of array.

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

int arr[10], i, j, temp;
printf ("Enter elements of array :");
for (i=0; i<10; i++) {
scant ("%d", & arr[i])
}

for ( i=0; i<10 ; i++) {
    for (j=0; j<10 ; j++) {

        if (arr[i] > arr[j])
    {
        temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}
}

for (i=1; i<10; i++) {

    if (arr[i] != arr[0])
{
printf ("Second smallest element
        is %d ", arr[i]);
}
return 0;

}
```

Enter elements of array : 2
4

6
8
10
12
1
5
9
3

The second smallest element is : 2

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

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

 char str [100];
 int i, words =0;

 printf ("Enter a string:");
 fgets (str, sizeof(str), stdin);

 while (str [i] != '\0') {

 if (str[i] == ' ' || str[i] == '\n' ||
    str[i] == '\t') {
 wordscount ++;
 i++;
 }

 if ( i=0 && str [i-1] != ' ' &&
    str [i-1] != '\n' && str[i-1] !=
     '\t') {
  wordsi++;
 }

 printf (" Total number of words : %d\n"
   word count);

 return 0;
 }
```

Enter a string : I am Anushka
Total number of words : 3

25.7.25

Experiment 3.

Q1) Sort elements in ascending order using bubble sort-

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

int arr[10], i, j, temp;

printf ("Enter the elements of the
        array:");
for (i=0; i<10; i++) {
scanf ("%d", &arr[i]);
}

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

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

printf ("Array is ascending order is :");
 for (i=0; i<10; i++) {
 printf ("%d", arr[i])
}

return 0;
}
```

O/P

```
Enter the element of the array: 4
2
5
1
3

Sorted array in ascending order is: 1
2
3
4
5
```

Q2) Sort elements in descending order using selection sort.

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

int arr[10], i, j, tem, maxindex;

printf ("Enter elements of the array:
for (i=0; i<10; i++) {
 scanf ("%d", &arr[i])
}
```

```
for (i=0; i<9; i++) {
    maxindex = i;
for (j=i+1; j<A9; j++) {

    if (a[j] = a[maxindex]){

        maxindex = j;
    }
}

temp = a[i]
a[i] = a[maxindex]
a[maxindex] = temp;

printf (" Sorted array in descending
        order: \n");
for (i=0; i<10; i++) {

printf ("%d", arr[i]);

}
return 0;
}
```

O/P- Enter elements of array: 4
2
5
1

Sorted array in descending order: 1
2
3
4
5

03) Find number of comparisons required in bubble sort method of the following list having 5 members. 100, 200, 300, 400, 500

Pass 1

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

Pass 2

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

Pass 3
100 200 300 400 500
100 200 300 400 500
100 200 300 400 500

Pass 4
100 200 300 400 500
100 200 300 400 500

Q4) Total number of passes: n-1 = 4
Total number of comparisons:
4+3+2+1 = 10

Q4) Sort given array in ascending order using selection sort method & show digramatic representation in every interation of loop:
500, -20, 30, 14, 50

Pass 1
500 -20 30 14 50
-20 500 30 14 50
-20 500 30 14 50
-20 500 30 14 50

Pass 2
-20 500 30 14 50
-20 30 500 14 50
-20 14 500 30 50
-20 14 500 30 50

Pass 3
-20 500 14 ~~30~~ 500 ~~14~~ 30 50
-20 14 30 500 50
-20 14 30 500 50

Pass 4
-20 14 30 500 50
-20 14 30 50 50

Total number of passes: n-1 = 4
Total number of iterations:
4+3+2+1 = 10

Experiment 4

Q1) Sort element in ascending order using Insersion sort.

```
#include <stdio.h>
int main()
{
    int arr[50], n, i, j, temp;
    printf("\n Enter number of
            elements :");
    scanf("%d", &n);
    printf("\n Enter %d elements:",
            n);
    for (i=0; i<n; i++)
    {
        for scanf("%d", &a[i])
    }

for (i=1; i<n; i++) {

    for (j=0; j<i; j++) {

        if (arr[i] < arr[j]){

            temp = arr[j];
        for (int k = i-1; k>=j; k--)
        {
            arr[k+1] = arr[k];
        }
        arr[j] = temp;
        break;
        }

        printf("\n After pass %d", i);
        for (int p=0; p<n; p++)
        {
            printf("%d", arr[p]);
        }
        printf("\n");

    }

    printf("\n Sorted array :");
    for (i=0; i<n; i++)
    {
        printf("%d", arr[i]),
    }
        printf("\n");

    return 0;

}
```

02> Sort element in ascending order using radix sort.

```
#include <stdio.h>
int main()
{
  int a[50], output[50], count[10];
  int n, j, i, max=0, place=1;

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

  printf("Enter %d numbers: \n", n);
  for (i=0; i<n; i++){
    scanf("%d", &a[i]);
    if (a[i] > max)
      max = a[i];
  }

  while (max / place > 0) {

    for (i=0; i<10; i++)
      count[i] = 0;

    for (i=0; i<n; i++)
      count[a[i] / place) % 10]++;

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

    for (i=n-1; i>=0; i--) {
      int digit = (a[i] / place) % 10;
      output[--count[digit]] = a[i];
    }

    for (i=0; i<n; i++)
      a[i] = output[i];

    place *= 10;
  }

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

  return 0;
}
```

03> What is the output of the insersion sort after second iteration :

7, 3, 1, 9, 4, 8, 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

After second iteration the result will be:

Ans: 1 3 27 9 4 8 6

Q4) Sort the following number using radix sort:

Input	Bucket 0	1	2	3	4	5	6	7	8	9
100	100									
225						225				
390	390									
4130	4130									
956							956			
99										99
5431		5431								

100	100									
225			225							
390										390
4130		4130								
956										956
99										99
5431					5431					

Input	Bucket 0	1	2	3	4	5	6	7	8	9
100	100									
390										
4130		4130								390
5431						5431				
225			225							
956							956			
99										99

100		100								
4130		4130								
~~225 5431~~			225							
5431						5431				
956										956
390				390						
099	099									

~~100~~										
0099	099									
0100	0100									
4130						4130				
0225	0225									
0390	0390									
5431								5430		
0956	095									

Ans: 99, 100, 225, 390, 956, 4130, 5431

Q5) 256, 99, 145, 239, 20, 18

Bucket / Input	0	1	2	3	4	5	6	7	8	9
256							256			
99						145				99
145									238	
239										18
20	20									
18										
20			20							
145					145					
256						256				
18		18								
99										99
238				238						
18	018									
20	020									
238			238							
145		145								
256			256							
99	099									

Ans) Sorted array :
18, 20, 99, 145, 238, 256

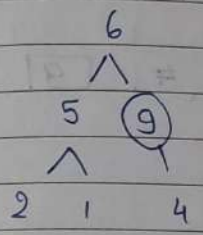
Q6) Sort following using heap sort.

6, 5, 9, 2, 1, 4

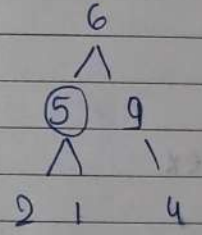
n = 6

(n/2) - 1 = 3 - 1

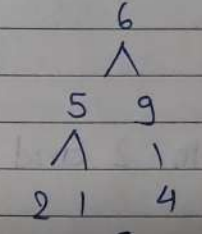
= 2 → index



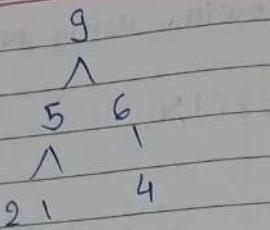
compare 9 with 4



compare 5 with 2 and 1.



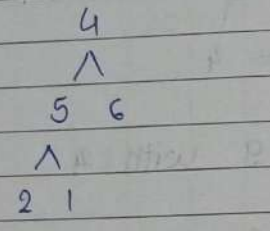
Compare 6 with 5 and 9



9, 5, 6, 2, 1, 4

Swap 9 with 4.

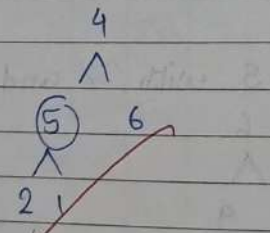
4, 5, 6, 2, 1, #, [9]



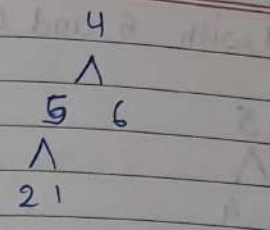
n = 5

(n/2) - 1

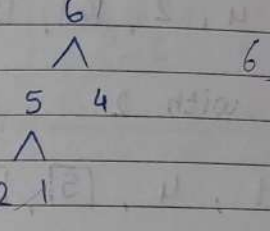
= 1 → index



Compare 5 with 2 and 1



Compare 4 with 5 and 6

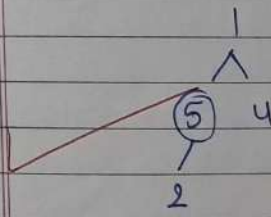


6, 5

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

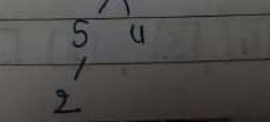
Swap 1 and 6

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

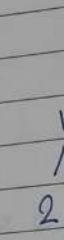


(n/2) - 1 = (4/2) - 1 = 2 - 1 = 1

Compare 5 with 2



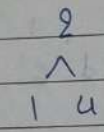
Compare 1 with 5 and 4



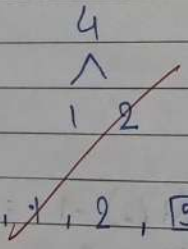
5, 1, 4, 2, [6], [9]

Swap 5 with 2

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



Compare 2 with 1 & 4



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

Swap 2 & 4

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

compare 2 and 1

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

swap 1 and 2

Sorted array:

1, 2, 4, 5, 6, 9

* Extra Questions -
- Algorithms:

a) Bubble sort:
1. Compare 0th element (j) with next / first element (j+1).
2. If (j+1) is ~~gr~~ smaller than j swap the two elements.
3. Move to the next set of elements (1st & 2nd) & repeat step 2
4. Continue this process till you reach the end of the list/array.
5. At end of first iteration, largest element is found at end, exclude that element & repeat from step 1.
6. For n number of elements n-1 iterations will occur.

b) Selection sort:
1. Start from the 0th element of the list (i) and compare with each element.
2. If arr[i] is greater than any element swap the two.
3. At end of first iteration smallest element is found at begining Exclude that & repeat step 1 & 2
4. Repeat step 3 for each position/ element till only one element remains unsorted (largest)
5. Hence, sorted list is obtained.

c) Insersion sort:
1. Start with 1st ~~element~~ index (2nd position)
2. Compare it with element before ie 0th element.
3. If it is smaller, insert in position of 0th element by shifting all others to the right.
4. The move to next element (2nd index) compare with 0th & 1st & insert at ist's appropriate position by making space. with 1st if smaller 0th
5. Repeat with all elements until list is sorted.

d) Radix Sort:
1. Define 10 queus each with buckets from digit 0 to 9.
2. Consider least significant digit of each number in the list to besorted.
3. Insert each number in respective bucket based on least-significant digit.
4. Group all numbers from queue from 0 to 9 in order they have been inserted (top to bottom)

5. Repeat the process with next least significant digit.
6. Continueue till all elements are grouped based on most significant digit.

e.) Shell sort :
1. Start
2. Initialize value of gap size Exh.
3. Divide list into smaller sub parts such that each has equal intervals.
4. Sort sub list using insertion sort.
5. Modify & gap size according to chosen optimal sequence.
6. Repeat from step 2 until list is sorted.
7. Print sorted list.
8. Stop.

f.) Heap Sort :
1. Construct a complete binary tree from array of size n.
2. Find last non-leaf node index using $i = (n/2) - 1$ & heapify nonleaf nodes in reverse to form max heap.
3. Reconstruct array using max heap & swap largest element in first position with last unsorted position
4. Reconstruct complete tree binary after deleting largest element.
5. Repeat till only one unsorted element remains.
6. Print sorted list.

g.) Merge sort :
1. Divide array into two values such that each half is a subarray until each subarray has 1 element.
2. Sort & merge each seperate subarray in reverse order of division.
3. Reapeat till original length array is obtained.
4. Print sorted array.

Algorithm	Best Time	Avg Time	Wast time	Worst space
Linear	$O(1)$	$O(n)$	$O(n)$	$O(1)$
Binary	$O(1)$	$O(\log n)$	$O(\log n)$	$O(1)$
Bubble	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Inserion	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Merge	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Heap	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Radix	$O(nk)$	$O(nk)$	$O(nk)$	$O(n+k)$
Shell	$O(n)$	$O((n \log n^2)$	$O(n \log n^2)$	$O(1)$

17/10

4.9.25

Experiment 5

Q1.) Write a menu driven program that implements singly linked list for following operations:
Create, insert node at beginning insert node & middle & end.
Q2.) Delete first node, Delete node at middle, Delete node at end.
Q3.) Create, display, Count no. of nodes, Reverse, Search.

* Code.:

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

Struct Node{
   int data;
   struct Node *next;
};

Struct Node *head = NULL;

Struct Node* CreateNode (int data);

void createList();
void insertAtBeginning();
void insertAtEnd();
void insertAtPosition();
void deleteAtBeginning();
void deleteAtEnd();
void deleteAtPosition();
void display();
void countNodes();
void reverse();
void search();

int main() {

  int choice;

  while(1) {
  printf("\n Singly Linked List Menu \n");
  printf("1. Create List \n");
  printf("2. Insert At Beginning \n");
  printf("3. Insert At End \n");
  printf("4. Insert At Position \n");
  printf("5. Delete at Beginning \n");
  printf("6. Delete at End \n");
  printf("7. Delete at Position \n");
  printf("8. Display \n");
  printf("9. Count Nodes \n");
  printf("10. Reverse List \n");
  printf("11. Search Node \n");
  printf("12. Exit \n");
  printf("Enter your choice : ");
  scanf("%d", &choice);
```

```
switch (choice) {

case 1: createList(); break;
case 2: insertAtBeginning(); break;
case 3: insertAtEnd(); break;
case 4: insertAtPosition(); break;
case 5: deleteAtBeginning(); break;
case 6: deleteAtEnd(); break;
case 7: deleteAtPosition(); break;
case 8: display(); break;
case 9: countNodes(); break;
case 10 reverse(); break;
case 11: search(); break;
case 12: Exit(0);
default: printf("Invalid choice!\n");
}
}
return 0;
}

struct Node* createNode(int data){
struct Node* newNode = (struct Node*)
malloc(sizeof(struct Node));
newNode -> data = data;
newNode -> next = NULL;
return newNode; }

void createList(){
 int n, data;
 printf("Enter number of node:");
 scanf("%d", &n);
 head = NULL;
 for(int i=1; i<=n; i++){
 printf("Enter data for node %d:", i);
 scanf("%d", &data);
 struct Node *newNode = createNode(data)
 if(head == Null) head = newNode;
 else {
    struct Node *temp = head;
    while(temp -> next != NULL) temp
     = temp -> next;
     temp -> next = newNode;
  }
 }
}

void insertAtBeginning(){
  int data;
  printf("Enter data:");
  scanf("%d", &data);
  struct Node *newNode = createNode(data)
  newNode -> next = head;
  head = newNode;
}

void insertAtEnd(){
 int data;
 printf("Enter data:");
```

```
scanf("%d", &data);
struct Node *newNode = createNode(data);
if (head == NULL) { head = newNode; return; }
struct Node * temp = head;
while (temp -> next != NULL) temp =
temp -> next;
temp -> next = newNode;
}

void insertAtPosition() {
int data, pos;
printf("Enter position & data:");
scanf("%d %d", &pos, &data);
struct Node* newNode = createNode(data);
if (pos == 1) {
newNode -> next = head; head = newNode;
return ; }
struct Node * temp = head;
for (int i=1; temp != NULL && i<pos-
1 ; i++) temp = temp -> next;
if (temp == NULL) { printf("Position
out of range !\n"); free(newNode);
return; }
newNode -> next = temp -> next;
temp -> next = newNode;

void deleteAtBeginning () {
if (head == NULL) { printf("List empty!!");
return ; }
```

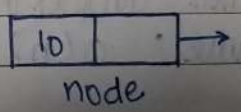
```
struct Node *temp = head;
head = head -> next;
free(temp);

void deleteAtEnd () {
if (head == NULL) { printf("List is
empty !\n"); return ; }
if (head -> next == NULL) { free(head);
head = NULL ; return ; }
struct Node * temp = head;
while (temp -> next -> next != NULL)
temp = temp -> next;
free (temp -> next);
temp -> next = NULL;

void deleteAtPosition() {
int pos, i;
struct node *current = head, *temp;
if (head == NULL) {
printf("List is Empty.\n");
return;
}
printf("Enter Position:");
scanf("%d", &pos);
if (pos <= 0) {
printf("Invalid position!\n");
return 0;
if (pos == 1) { deleteAtBeginning(); return; }
struct *Node * temp = head;
```

```
if (int i=1 ; temp != NULL && i < pos-1
; i++) temp = temp -> next;
if (temp == NULL || temp -> next == NULL)
{ printf ("Position out of range! \n");
return 0; }
struct Node *del = temp -> next;
temp -> next = del -> next;
free(del);
}

void display() {
if (head == NULL) { printf ("List is
empty! \n"); return ; }
struct Node *temp = head;
printf ("List : ");
while (temp != NULL) { printf ("%d ->
temp -> data); temp = temp -> next;
}
printf (" NULL \n");

Void countNodes() {
int count = 0;
struct Node * temp = head;
while (temp != NULL) { count++;
temp = temp -> next; }
printf ("Total Nodes = %d\n", count);

void reverse() {
struct Node * prev = NULL; * curr =
head ; * next = NULL;
while (head curr != NULL) next = curr ->
next ; curr -> next = prev ; prev =
curr ; curr = next; }
head = prev ;
}

void search() {
int key , pos = 1 ;
printf ("Enter element to search: ");
scanf ("%d", & key );
struct Node * temp = head;
while (temp != NULL) {
    if (temp -> data == key) {
printf ("Element %d found at position %d"
key , pos) ; return ; }
temp = temp -> next;
pos ++;
}
printf ("Element not in list );
}
```

04) Basic Terminology of linked list:

a) Node: Basic building block of linked list. It contains data (value) & next (pointer to next node)

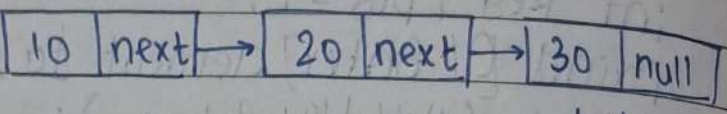
eg



b) Head: The head is a reference or pointer to first node of list. Null or nun.

c) Tail: last node where next = NULL

d) Next:
Each node has a next reference to the following node.

eg. 

e) Null Pointer:
Special value that indicates that pointer does not have object / memory.

f) Empty linked list: Singly linked list with no nodes in it.
eg Head → null.

05)

Array	Linked list
• Resizing not possible.	• Can grow or shrink.
• memory allocation is fixed size/static.	• Size of array thus memory can change/dyna
• Stored in contiguous memory.	• Stored in non-contiguous memory.
• Memory usage is efficient.	• Uses extra memory for pointers.
• Simple to implement	• Complex implementation
• O(1) access time.	• O(n)

* exp 5:

Output:

Singly Linked List Menu

1. Create
2. Insert at Beginning
3. Insert at End
4. Insert at Position
5. Delete at Beginning
6. Delete at End
7. Delete at Position
8. Display
9. Count Nodes
10. Reverse List
11. Search Node
12. Exit

Enter your choice: 1
Enter number of nodes: 5
Enter data for node 1: 1
" 2: 2
" 3: 3
" 4: 4
" 5: 5

Enter your choice: 8
List: 1 → 2 → 3 → 4 → 5 → NULL

Enter your choice: 2
enter data: 0
List: 0 → 1 → 2 → 3 → 4 → 5 → NULL

* Enter your choice: 8 after data.

Enter your choice : 3
Enter data : 6
List : 0 → 1 → 2 → 3 → 4 → 5 → 6 → NULL

Enter your choice : 4
Enter data and position : 3 4
List : 0 → 1 → 2 → 3 → 3 → 4 → 5 → 6 → NULL

Enter yr choice : 5
List : 1 → 2 → 3 → 3 → 4 → 5 → 6 → NULL

Enter yr choice : 6
List : 1 → 2 → 3 → 3 → 4 → 5 → NULL

Enter yr choice : 7
Enter ~~data and~~ position : 4
List : 1 → 2 → 3 → 4 → 5 → NULL

Enter your choice : 9
Total Nodes = 5

Enter your choice : 10
List reversed sucessfully!
List : ~~1 → 2~~ 5 → 4 → 3 → 2 → 1 → NULL

Enter choice : 11
~~Search No~~. Enter element to search: 4
Element 4 found at position 2.

* Experimen 4:
Shell sort -

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

n = 8
n/2 = 4

22 18 29 7 9 55 21 8

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

9 22 18 55 21 29 78

n/4 = 8/4 = 2

{ 9 22 18 55 } { 21 29 7

{ 9 22 18 55 } { 21 29 7

{ 9 18 22 55 } { 7 21 29

{ 9 18 22 55 } { 7 8 21

9 18 22 55 7 8 21 2

n/8 = 8/8 = 1

9 18 22 55 7 8 21 29

9 18 22 55 7 8 21 29

9 18 22 55 7 8 21 29

9 18 22 55 7 8 21 29

7 9 18 22 55 8 21 29

7 8 9 18 22 55 21 29

7 8 9 18 21 22 55 29

sorted array — 7 8 9 18 21 22 29 55

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

n = 8

n/2 = 4

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

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

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

n/4

8/4 = 2

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

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

{1 3 5 10} {2 ~~15~~ 6 15

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

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

sorted array:

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

Experiment 6

Doubly Linked List -

Doubly Linked List.

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

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

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

struct node* start = NULL;

int main() {
    int choice;
    while (1) {
        printf("\n MENU ");
        printf("\n 1. CREATE");
        printf("\n 2. DISPLAY");
        printf("\n 3. INSERT_BEGIN");
        printf("\n 4. INSERT_END");
        printf("\n 5. INSERT_POSITION");
        printf("\n 6. EXIT");
        printf("\n 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_position();
            break;
        case 6:
            printf("Exiting program.\n");
            exit(0);
        default:
            printf("Invalid choice, please try again.\n");
    }
  }
  return 0;
}

void create() {
  struct node* temp;
  struct node* ptr;

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

  printf("\n Enter data: ");
  scanf("%d", &temp->info);
  temp->next = NULL;
  temp->prev = NULL;

  if (start == NULL) {
    start = temp;
  } else {
    ptr = start;
    while (ptr->next != NULL) {
      ptr = ptr->next;
    }
    ptr->next = temp;
    temp->prev = ptr;
  }
  printf("Node created and added at the end.\n");
}

void display() {
```

```
    struct node* ptr;
    if (start == NULL) {
       printf("\n Empty List\n");
       return;
    }
    ptr = start;
    printf("\n List elements are: ");
    while (ptr != NULL) {
       printf("%d ", ptr->info);
       ptr = ptr->next;
    }
    printf("\n");
}

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

    temp->next = start;
    temp->prev = NULL;

    if (start != NULL)
       start->prev = temp;

    start = temp;
    printf("Node inserted at the beginning.\n");
}

void insert_end() {
    struct node* temp;
    struct node* ptr;

    temp = (struct node*)malloc(sizeof(struct node));
    if (temp == NULL) {
       printf("Memory allocation failed\n");
       return;
    }
    printf("\n Enter data to insert at end: ");
```

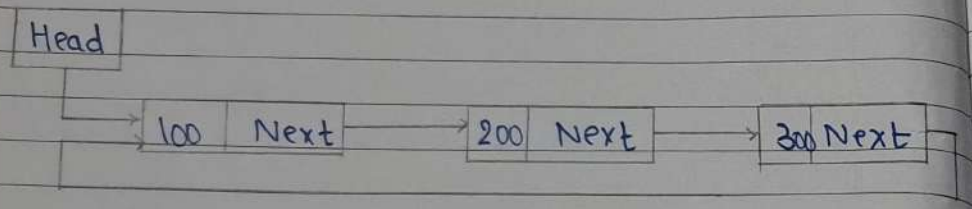
```
    scanf("%d", &temp->info);
    temp->next = NULL;
    temp->prev = NULL;

    if (start == NULL) {
      start = temp;
    } else {
      ptr = start;
      while (ptr->next != NULL) {
        ptr = ptr->next;
      }
      ptr->next = temp;
      temp->prev = ptr;
    }
    printf("Node inserted at the end.\n");
}

void insert_position() {
  struct node* temp;
  struct node* ptr;
  int pos, i;

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

  printf("\n Enter data to insert: ");
  scanf("%d", &temp->info);
  temp->next = NULL;
  temp->prev = NULL;

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

  if (pos == 1) {
    temp->next = start;
    temp->prev = NULL;
    if (start != NULL)
      start->prev = temp;
    start = temp;
    printf("Node inserted at position 1.\n");
    return;
```

```
        }

        ptr = start;
        for (i = 1; i < pos - 1 && ptr != NULL; i++) {
            ptr = ptr->next;
        }

        if (ptr == NULL) {
            printf("Position out of bounds.\n");
            free(temp);
        } else {
            temp->next = ptr->next;
            temp->prev = ptr;

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

            ptr->next = temp;
            printf("Node inserted at position %d.\n", pos);
        }
    }
```

* Explain Circular Linked List:

- In circular singly linked list, last node of the list contains a pointer to the first node of the list.
- We can have circular singly linked list as well as circular doubly linked list.
- We traverse through it until we reach same node where we started.
- It doesn't have a begining or an end.
- Thus there is no null value present in next part of any of the nodes.
- Mainly used in task maintainance in OS.

Diagram:



- Advantages:
Any node can be a starting point, we can traverse the whole list by starting from any point. We just need to stop when first node is visited again.

- Applications:
Personal computers, where multiple applications run. All running application kept in circular LL & the OS gives a fixed time slot. It keeps iterating untill all applications are completed.

11-9-25

Experiment 7

* Primitive operations on stack:-

code:

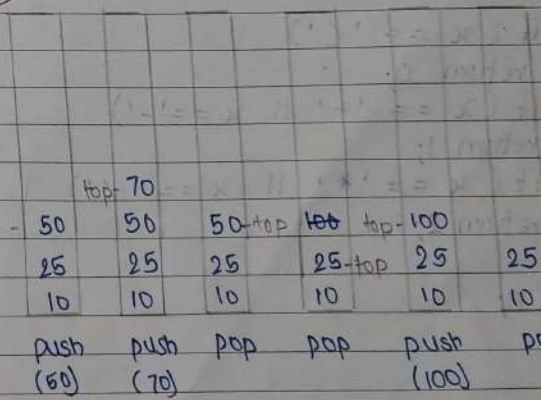
```
# include <stdio.h>
# include <stdlib.h>
# define Max 5

void push();
void pop();
void display();

int S[Max];
int top = -1;

int main() {
   int choice;
   printf ("\n MENU :-");
   printf ("\n 1. Push");
   printf ("\n 2. Pop");
   printf ("\n 3. Display");
   printf ("\n 4. Exit");
   do {
      printf ("\n Enter your choice:");
      scanf ("%d", &choice);

switch (choice) {
      case 1 : push(); break;
      case 2 : pop(); break;
      case 3 : display(); break;
      case 4 : exit (0);
      default : printf ("Invalid cho
}
while (choice != 4);
```

```
return 0;
}

void push(){
 int element;
 if (top == Max-1) {
    printf (" Overflow);
  }
  else {
    printf(" Enter value ");
    scanf (" %d", &element);
    top = top+1;
    s[top] = element;
  }
}
 void pop(){
   int item;
   if (top == -1) {
     printf (" Underflow");
    }
   else {
     item = s[top];
     top--;
     printf (" The deleted element is %d ", item);
   }
  }

void display(){
 if (top == -1){
  printf ("Stack Empty");
 }
else {
printf (" Elements of stack are:");
 for (i=0; i<=top; i++) {
    printf (" \t %d", s[i]);
 }
}
}
```

Q- Stack size - 8 for push(10), push(20), pop push (25), push(50), push(70), pop, pop, push(100), pop & draw final output.

A-

Empty Stack — push (10) — push (20) — pop — push (25)



push (50) — push (70) — pop — pop — push (100) — pop

18.9.25

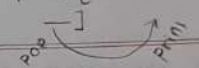
Experiment 8

* Infix to postfix :-

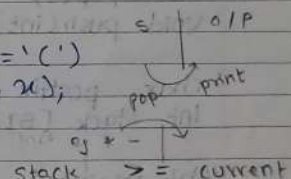
1)

```
# include <stdio.h>
# include <ctype.h>
ABC-
char stack [100];
int top = -1; empty

void push (char x)
{
   stack [++top] = x;
}

char pop(){
   if (top == -1)
    return -1;
   else
    return stack [top--];
}
                top val, then decrement

int priority (char x)
{
   if (x == '(')
    return 0;
   if (x == '+' || x =='-')
    return 1;
   if (x == '*' || x == '/')
    return 2;

 return 0;
}
```

char | stack | O/P
+[| | AB12




```
int main () {

 char exp [100];
 char *e, x;
 printf (" Enter the expression: ");
 scanf ( "%s ",&exp);
 printf ( " \n ");
 e = exp; → begining of string

 while ( *e != '\0') → end
{
        is alphanet /num
   if (isalnum (*e))
     printf ( "%c", *e); → O/P
   else if ( *e == '(')
     push ( *e); → stack
   else if ( * e == ')')
{
          assign & x
     while ((x = pop()) != '(')
          printf ( "%c", x);
    }
   else
   {
     while ( priority ( stack [top]) >= priority (
     printf ( "%c", pop()),
     push ( *e); current o/p to stack
   }
     e ++; next
 }

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



op- Enter the expression: (A+B)
A B +

Enter the expression: A+B*C

A B C * +

3) Evaluate postfix expression :-

```
# include <stdio.h>
# include <ctype.h>      isdigit
# include <stdlib.h>     exit()

# define SIZE 40         = max size of stack

int pop();
void push(int);
                         arrays
char postfix[SIZE];      = array to store expression
int stack[SIZE], top = -1;   number & O/P
                             empty

int main(){

  int i, a, b, result, peval;   final evaluated
  char ch;                      current char

  for (i=0; i < SIZE; i++)
  {
      stack[i] = -1;            all elements to -1
  }
printf("\nEnter a postfix expression :");
scanf("%s", postfix);
```

```
for (i=0; postfix[i] != '\0'; i++)        null
{                                         iterate exp till null
    ch = postfix[i];  → current to ch

  if (isdigit(ch))
  {        ascii ch - ascii of zero
      push(ch - '0');   char to int
  }
  else if (ch == '+' || ch == '-' || ch == '*'
          || ch == '/')                    2
                                           a, b

    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: %
```

```
result turn 0;
}

void push (int n)
{
    if ( top < SIZE -1)
    {
        stack [++top] = n;
    }
    else{

        printf ( "Stack is full ! \n");
        exit(-1);
    }
}

int pop()
{
    int n;

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

op - Enter a postfix expression :
The postfix evaluation is:

2) Convert infix expresion into prefix using stack

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

char	stack	O/P
H	*	H
*	* [	H
[	* [	H
G	* [*	HG
*	* [* [	HG
[	* [* [^	HGF
F	* [* [^	HGF
^	* [* [^	HGF
E	* [* [/ greater	HGFE
/	* [* [/	HGFE^
D	* [*	HGFE^D
]	* [* - greater	HGFE^D/
-	* [-	HGFE^D/*
C	* [- *	HGFE^D/*
*	* [- *	HGFE^D/*
B	*	HGFE^D/*C
]	* + greater	HGFE^D/*C*-
+	+	HGFE^D/*C*-A
A		HGFE^D/*C*-A*

Final expression : + A * - * B C * / D ^ E F G

4.) Evaluate prefix expression using stack:

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

Input	Operand 1	Operand 2	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

* Algorithms :-

① Conversion from infix to postfix expression:

i Read expression from left to right.
ii If an operand (A-Z, 0-9) is encountered add it directly to output.
iii If an operator is encountered,
iv Pop operator from stack to output if they have higher or equal precedence.
v Push the current operator onto the stack.
vi If an opening parenthesis '(' is found, push it into stack
vii If closing paranthesis ')' is found, pop operators from stack to output until opening paranthesis is encountered.
viii After scanning entire expression, pop any remaining operators from stack to output.

② Conversion from infix to prefix expression:

i. Reverse the infix string. Note that while reversing you must interchange left & ')' and right '(' parantheses.
ii Obtain the postfix expression of the given infix expression obtained from step i.
iii Reverse the obtained postfix expression (O to get prefix expression.

③ Evaluating postfix expression:

i Scan the elements of string from left to right
ii If the element is an operand, push it into the stack
iii If the element is an operato two operands from the stack.
iv The first pop is second operand & second pop is first operand.
v Perform arithmetic operation & pus result back into stack
vi When input expression is completely tr pop operand stack & return the

④ Evaluating prefix expression:

i Start from the last element of the expression
ii check the current element.
iii If it is operand, push to stack.
iv If it is an operator, pop two operands from stack (last & second last)
v Perform operation & push elements back to stack
vi Do this till all elements of expression are traversed & return the top of stack which will be the result of the operation.

17/10

25.9.25

Experiment 9

1. Perform primitive operation of linear Queue - Insert, Delete, Display.

* Code -

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

void enqueue();
void dequeue();
void display();
int queue[SIZE];
int front = -1, rear = -1;
    remove      add

int main(){

    int choice;
    printf(" 1. Enqueue \n 2. Dequeue \n 3. Dis
            4. Exit \n");
    do {
        printf (" Enter your choice :");
        scanf ("%d", &choice);
        switch (choice) {
        case 1 : enqueue (); break;
        case 2 : dequeue(); break;
        case 3 : display(); break;
        case 4 : exit (0) ;
        default : printf (" Invalid choice!
    } }
```

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

void enqueue (){

  int element ;
  if (rear == SIZE -1){
  printf ( "\n Queue is full! ");
  }
  else {
  printf ("\n Entre element to insert :");
  scanf ("%d", &element);
  if ( front == -1)
  front = 0;
  rear ++ ;
  queue [rear] = element ;
  }
}

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

 else {
 printf (" \n Deleted element : %d", queue [front]);

 front ++;
 }
}
```

```
void display (){

  if ( front == -1 || front > rear ){
    printf (" \n Que is empty ! ");
  }

   else {
    printf ( " \n Queue elements are : \n");

     for ( int i = front ; i <= rear ; i++){
         printf ( " %d \t", queue [i]);
     }

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

Q1) Write algorithm to enqueue (insert) & deque (deletion) on linear queue.

a.) Insersion:

1. Check the overflow condition.
2. Check if queue is empty. In case it is, then b FRONT and REAR ~~are~~ set to zero. so that n value can be stored at the 0th location.
3. Otherwise, if queue already has some value then REAR is increamented such that i points to the next location in array.
5. The value is stored in the queue at the location pointed by REAR.
6. Exit

b) Deletion:

1. We check for underflow condition. An underflow occurs if FRONT = -1 or FRONT > REAR.
2. If queue has some values, the FRONT is increamented so that it now points to next value in queue.
3. Exit

Q2) Write algorithm of insersion & deletion for circular queues.

a) Insersion:

1. If front = 0 and rear = max-1 the queue will overflow.
2. If front is rear+1 it will also overflow queue.
3. If front = -1 and rear = -1, front = rear = 0
4. Else if rear = max-1 and front != 0
   Rear = 0
5. Else, Rear = rear+1
   End
6. queue [rear] = val
7. exit.

b) Deletion:

1. If FRONT = -1 write underflow.
2. Go to Step four (4) at end of if.
3. Set VAL = queue [front]
5. If front = rear, set front = rear = -1
6. Else, if front = Max-1, set FRONT = 0
7. Else, set front = front+1
8. Exit

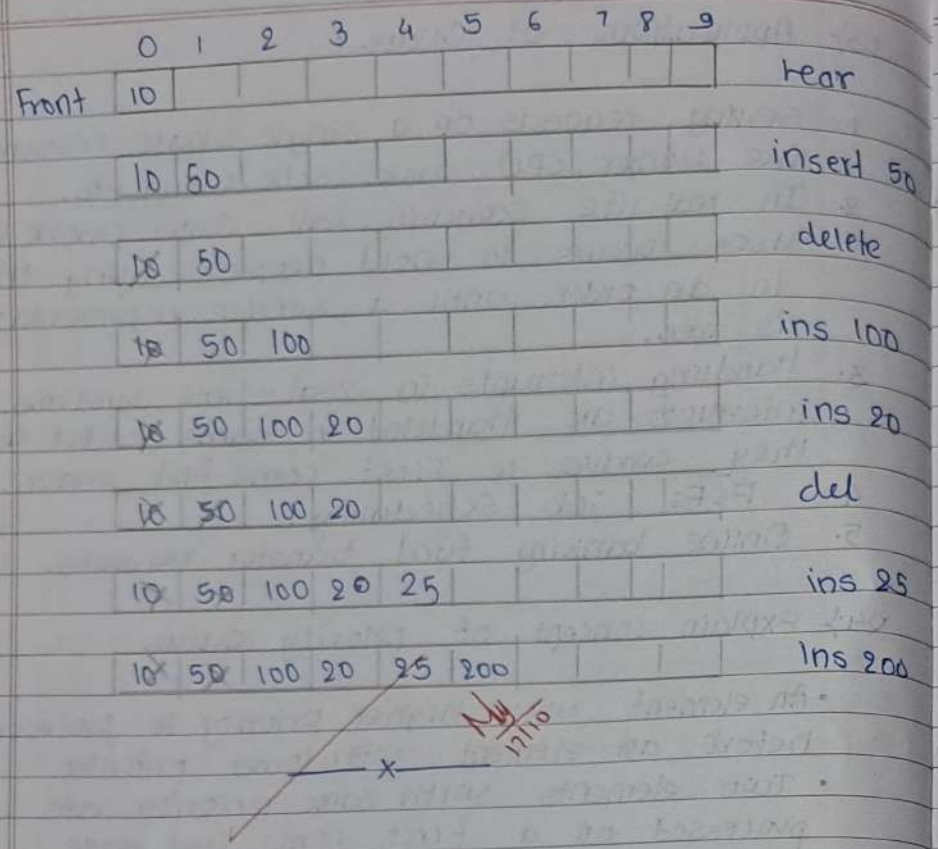
Q3) Applications of Queue:

1. Serving requests on a single share resource like Printer, CPU task scheduling etc.
2. In real life scenario, call center phone s uses queues to hold people calling th in an order, until a service representa is free.
3. Handling interrupts in real-time systems interrupts are handeled in same order they arrive ie First come Fist servi
4. FCFS job Scheduling.
5. Online banking fund transfer requests.

Q4) Explain concept of priority Queue.

- An element with higher priority is pr before an element with lower priorit
- Two elements with same priority processed on a first come first ser (FCFS) basis.
- Types -
  Ascending priority Queue.
  Descending priority Queue.

Q5) Array size 10, insert (10), insert (Insert (100), Insert (20), Delete, Insert (200)



10.10.25

Experiment 10

Q- Write a C program to perform primitive operation on circular Queue

```
#include <stdio.h>
#include <stdlib.h>      exit
#define size 5   max

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

int queue[size];
int front = -1, rear = -1;    empty
int main()
{
    int ch;
    printf("\n1.Enqueue \n2. dequeue \n3.
            display \n4. Exit \n");
do {
        printf("Enter your choice:");
        scanf("%d", &ch);
        switch(ch) {
        case 1: enqueue(); break;
        case 2: dequeue(); break;
        case 3: display(); break;
        case 4: exit(0);
        default: printf("Invalid choice!");
        }
    } while(ch!=4);
    return 0;
}
```


```
void enqueue()
{
  int value;
  if ((front == 0 & rear = size-1) ||
      (rear +1 == front))
  {
     printf ("\n Queue is Full \n");
  }
  else
  {
     printf ("\n Enter value to insert:");
     scanf ("%d", &value);
     if (front == -1)
        front=0;
  rear = (rear +1) % size;
  queue [rear] = value;
  printf ("%d ,insert into queue ", value);
  }
}

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

```
{
   front = rear=-1;
}
else
{
   front = (front +1) % size;
}
}
}
```

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

O/P - 1. Enqueue
2. dequeue
3. Display
4. Exit

Enter your choice : 1
Enter the value to insert : 5
5 inserted into queue

Enter your choice : 1
Enter value to insert : 9
9 inserted into queue

Enter you choice : 3
Queue elements are : 5 9

Q - Applications on priority queue?
A - • Huffman codes:
Priority queue helps build huffman trees for efficient data compression.
• OS Algorithms:
Used for process scheduling to execute high-priority tasks first.
• Traffic light control:
Manages signal priorities based on real-time traffic sensor data.
• Dijikstla's Algorithm:
Select next vertex with smallest dist using priority queue.

17/10

14.10.24

Experiment 11

* Write C program to implement following operations on binary tree: Create, display.

```
#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 0;
    printf ("%d", root -> item);
    preorder (root -> left);
    preorder (root -> right);
}

void postorder ( struct node * root
```

```
void PostOrder (struct node *root)
{
    if (root == NULL)
        return;
    postOrder (root -> left);
    PostOrder (root -> right);
    printf ("%d", root -> item);
}

struct node * (reatenode (item)
{
    Struct node * newNode = malloc(C
sizeof (struct node));

    newNode -> item = item;
    NewNode -> left = NULL;
    newNode -> right = NULL;
    return newNode;
}

struct node * insertAtleft (struct node * root
 int item)
{
    root -> left = Create Node (item);
    return root -> left;
}

Struct node * insertAtRight (struct node *
 root , int item)
{
    root -> right = (reateNode (item);
    return root -> right;
```

```
int main() {

 Struct node * root = (reatenode (7);
  insertAtLeft (root , 5);
  InsertAtRight (root , 6);
  insertAtleft (root -> left , 9);
  insertAtRight (root -> left , 45);
  insertAtRight (root -> right , 40);
  insertatleft (root -> right , 50);
  insertAtLeft (root -> left -> left , 100);
  insertAtRight (root -> right -> right , 26

    printf ("Inorder traversal : ");
    inorder (root);
    print (" \n PreOrder traversal : ");
    preOrder (root);
    printf (" \n Postorder traversal : ");
    postorder (root);
}
```

2.
```
# include <stdio.h>
# iclude <stdlib.h>

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

struct node * createnode (item)
{
```

```
struct node * newnode = malloc (size of (struct
node));
    newnode → item = item;
    newnode → left = NULL;
    newnode → right = NULL;
return newnode;
}

struct node * insertAtleft (struct node * root,
int item)
{
    root → left = createnode (item);
    return root → left;
}

struct node * insertAtright (struct node *
root, int item)
{
    root → right = createnode (item);
    return root & right;

}
```

30.10.24

Experiment 12

Matrix:

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

void init (int arr[V][V]){
  int i,j;
  for (i=0; i<V; i++){
    for (j=0; j<V; j++){
      arr[i][j] = 0;
    }
  }
}

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

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

}
```

O/P -
```
0: 1 1 1 1 0
1: 1 0 1 1 0
2: 1 1 0 1 1
3: 1 1 1 0 1
4: 0 0 1 1 0
```

2. Linked List:

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

struct Node {
    int dest;
    struct Node *next;
};

struct Graph {
    struct Node *adj[V];
};

struct Node *newNode(int dest) {
    struct Node *temp = (struct Node*)
malloc(sizeof(struct Node));
    temp->dest = dest;
    temp->next = NULL;
    return temp;
}

struct Graph *createGraph() {
    struct Graph *graph = (struct
Graph*) malloc(sizeof(struct Gra
for(int i=0; i<V; i++)
    graph->adj[i] = NULL;
  return graph;
}

void addEdge(struct Graph *graph
src, int dest) {
  struct Node *temp = newNode(d
  temp->next = graph->adj[src
  graph->adj[src] = temp;

  temp = newNode(src);
  temp->next = graph->adj[des
  graph->adj[dest] = temp;
}
```

```
void printGraph ( struct Graph * graph ) {
printf ( "Adjacency List : \n" );
for (int i = 0; i < V ; i++) {
    printf ( " %d → ", i );
Struct Node * temp = graph → adj[i];
while (temp) {
printf ( " %d ", temp → dest );
  temp = temp → next;
}
printf ( " \n" );
}
}

int main() {

Struct Graph * graph = createGraph();

addEdge ( graph, 0, 1);
add Edge ( graph, 0, 2);
add Edge (graph, 0, 3);
dddEdge (graph, 0, 4);
add Edge (graph, 1, 3);
add Edge ( graph, 2, 3);
addEdge ( graph, 2, 4);
addEdge (graph, 3, 4);

printGraph (graph);

return 0;
}
```

O/P - Adjacent List :

```
0 → 4  3  2  1
1 → 3  0
2 → 4  3  0
3 → 4  2  1  0
4 → 3  2  0
```

Q2) Explain Terminologies -

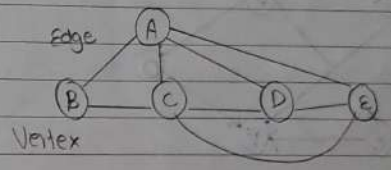
i. Vertex : Each node of graph is represented as a vertex. In the examples, labeled circle is vertex.

ii Edge : Edge represents a path between 2 vertices or a line between two vertices

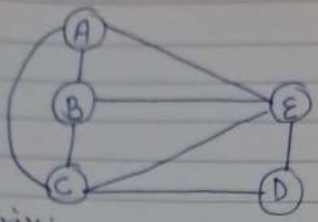
iii Adjacency : Two node or vertices are adjacent if they are connected to each other through an edge.

iv Path : Path represents a sequence of edges between the two vertices.

A34



Q3.)

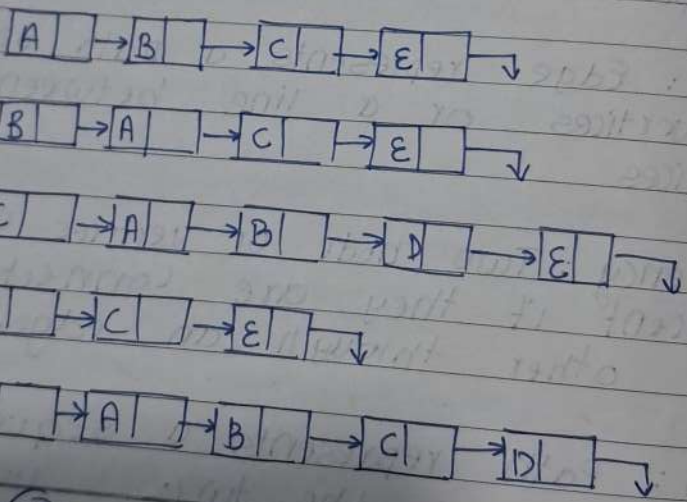


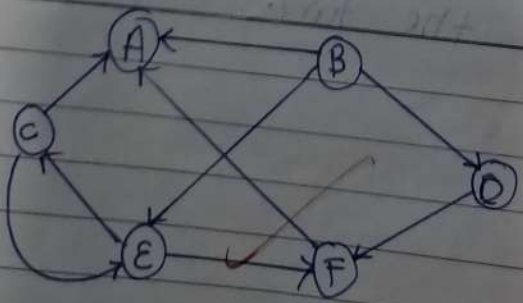
Matrix:

Graph

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

List:





Matrix:

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

List: