ABES Engineering College, Ghaziabad

(Affiliated to Dr.A.P.J AKTU, Lucknow)

Department of Computer Science & Engineering

Estd. 2000

Lab Manual

Session 2019-20 (Odd Semester)

Subject Name : Data Structures Using 'C'

Subject Code : KCS 351

Semester : B.Tech. CSE III (A,B,C)

Faculty Name(s) : Akhilesh Kr. Srivastava
Puneet Kumar Goyal
Priyanka Gupta

Tools & Software

Recommended Systems/Software Requirements:

Turbo C/C++
Code Block
Any Online C/C++ Compiler

Data Structure Lab (KCS 351)/ Data Structure Lab		
List of Experiments		
S.No.	**Program**	**DOMAIN**
1	Program for Array Inserion, Deletion and traversal	Array
2	Program for Insertion in Sorted Array	Array
3	***Program to Find the number which is not repeated in Array of integers, others are present for two times***	***Array***
4	Program For Linear Search	Array
5	Program for Binary Search	Array
6	***Program for Index Sequential Search***	***Array***
7	Program for Bubble, Selection and Insertion Sort	Array
8	***Program for Implementation of Shell Sort***	***Array***
9	Program for Quick Sort	Array
	Program for Randomized Quick Sort	***Array***
	Program for Quick Sort using Median element as Pivot	***Array***
10	Program for Merge Sort	Array
11	Program for Merging of two Sorted Arrays	Array
12	***Program for Finding set elements of A that belongs to set B***	***Array***
13	***Program for Finding set elements of A that does not belong to set B***	***Array***
14	***Program for Set Union***	***Array***
15	***Program for Set Intersection***	***Array***
16	***Program for Set DIFference***	***Array***
17	***Program for Counting Sort***	***Array***
18	***Program for Radix Sort***	***Array***
19	Program for Matrix Addition	Array
20	Program for Matrix Multiplication	Array
21	Program for Matrix transposition	Array
22	Program for Matrix transposition without second matrix	Array
23	Program to Print a given matrix in spiral form	Array
24	Program for Sorting the given Complex Numbers	Array
25	Program for Creation of Max Heap and Min Heap	Heap
26	***Program for Insertion in Max Heap/Min Heap***	Heap
27	***Program for Deletion from Max Heap and Min Heap***	Heap
28	***Program for Realizing Heap as Ascending/Descending Priority Queue***	Heap
Project 1: Program for Heap Sort		
Project 2: Performance Comparision of Sorting ALGORITHMs		
29	***Program for Hash Table Implementation for Basic Hash Function (Without collisions)***	***Hashing***
30	***Program for Hash Table Implementation for Collision Resolution using Linear Probing***	***Hashing***
31	***Program for Hash Table Implementation for Collision Resolution using Quadratic Probing***	***Hashing***

32	***Program for Hash Table Implementation for Collision Resolution using Double Hashing/Rehashing***	***Hashing***
33	***Program for Hash Table Implementation for Collision Resolution using Chaining***	***Hashing***
34	***Finding Anagrams: There are two strings. Find out which characters should be deleted such that both strings contain the same characters (May be in dIFferent Order)***	***Hashing***
35	***There are some numbers in which some are appearing twice but one is not repeated. Find out the number which appears once.***	***Hashing***
36	***There are two arrays containing some elements. Find out what are the elements which are there in both the arrays what are not.***	***Hashing***
37	***find out the values of a,b,c,d (a,b,c,d<=1000) for which a3+b3=c3+d3.***	***Hashing***
Project 3: IdentIFication of tokens and identIFiers and storage in Hash Table		
38	Program for finding the length of a string	String
39	Program for reversing the given string	String
40	Program for finding IF the given string is a palindrome	String
41	Program for finding word count in the Paragraph	String
42	Program for converting all lower case letters to upper case and vice versa in the given sentence	String
43	***Program for finding IF the given word is present in the sentence and at what location***	***String***
44	***Program for sorting the given names in the dictionary order***	***String***
45	***Program for reversing all words in a sentence***	***String***
Project 4: Program for automatic word spelling correction using Minimum Edit Distance		
46	***Program for Decimal to Binary Conversion***	***Stack***
47	***Program for Decimal to Octal Conversion***	***Stack***
48	***Program for Decimal to Hexadecimal Conversion***	***Stack***
49	***Program for Decimal to Any Base Conversion***	***Stack***
50	Program for Stack Primitive Operations	Stack
51	Program for Postfix Evaluation	Stack
52	Program for Infix to Postfix Conversion	Stack
53	Program for Infix to Prefix Conversion	Stack
54	Program for Prefix Evaluation	Stack
55	***Program to check the validity of Parenthesized Arithmetic Expressions using Stack***	***Stack***
56	***Program to check the validity of Bracketed Arithmetic Expressions using Stack***	***Stack***
57	***Program to check IF the given number is a palindrome using stacks***	***Stack***
58	***Program to Reverse the given String using Stack***	***Stack***
Project 5: Program for evaluation of given arithmetic expression. The Expression may have variables and constants		
59	Program for finding factorial of a given number using recursion	Recursion
60	Program for Towers of Hanoi for n disk (user defined)	Recursion
61	Program for Computing A raised to power n using Recursion	Recursion
62	Program for Computing A raised to power n using Divide and Conquer	Recursion
63	Program for finding nth Fibonacci number using Recursion and improving its run time to save stack operations	Recursion
64	Program for finding the GCD of two numbers using Recursion	Recursion

65	Program to reverse the given number using Recursion	Recursion
66	Program of Array Implementation of Linear Queue	Queue
67	Program of Array Implementation of CircularQueue	Queue
68	Program for Array Implementation of Double Ended Queue	Queue
69	Program for Array Implementation of Priority Queue	Queue
70	***Program for 1-D array implementation of Upper Triangular Sparse Matrix***	***Sparse Matrix***
71	***Program for 1-D array implementation of Lower Triangular Sparse Matrix***	***Sparse Matrix***
72	***Program for 1-D array implementation of Tridiagonal Sparse Matrix***	***Sparse Matrix***
73	***Program for Vector Representation of General Sparse Matrix***	***Sparse Matrix***
74	***Program For Linked List Implementation of General Sparse Matrix***	***Sparse Matrix***
75	***Program for Addition of two sparse Matrices***	***Sparse Matrix***
76	Program for Linear Linked List Primitive operations	Linked List
77	Program for Pair wise swap of elements in linked list	Linked List
78	Program to print Linked List contents in reverse order	Linked List
79	Program for Reversing the Linear Linked List	Linked List
80	Program for concatenation of Linear Linked List	Linked List
81	Program for Creation of Ascending Order Linear Linked List	Linked List
82	***Program for Merging two sorted Linked List***	***Linked List***
83	***Program for Union of two sorted Linked List (consider lists as sets)***	***Linked List***
84	***Program for Intersection of two sorted Linked List (consider lists as sets)***	***Linked List***
85	***Program for finding dIFference of two linked list (consider lists as sets)***	***Linked List***
86	***Program for Sorting the Linear Linked List***	***Linked List***
87	***Program for Splitting a Linked List***	***Linked List***
88	***To Detect IF there is any cycle in the linked list. (use two pointers, once moves at a speed of one node, other moves at a speed of two nodes. IF they collide with each other it means there is a cycle.***	***Linked List***
89	Program for Polynomial Addition using Linked List	Linked List
90	Program for Circular Linked List Primitive Operations	Linked List
91	Program for concatenation of Circular Linked List	Linked List
92	Program for Doubly linked list Primitive operations	Linked List
93	Program for Circular Doubly Linked List Primitive Operations	Linked List
94	Program for Linked List Implementaion of Linear Queue	Linked List
95	Program for Linked List Implementaion of Circular Queue	Linked List
96	***Program for Linked List Implementaion of Priority Queue***	***Linked List***
97	Program for Linked List Implementation of Stacks	Linked List
	Project 6: Program for Addition of Two very long Numbers	
	Project 7: Implementatio of Josephus Problem	
98	Program for recursive creation of Binary Tree and Traversals	Binary Tree
99	Program for creation of Binary Tree and finding its height	Binary Tree
100	Program for creation of Binary Tree and finding count of nodes having 2 children	Binary Tree
101	Program for creation of Binary Tree and finding count of nodes having 1 child	Binary Tree
102	Program for creation of Binary Tree and finding count of nodes having 0 child	Binary Tree

103	Program for finding IF the given Binary Tree is complete	Binary Tree
104	Program for Level Order Traversal	Binary Tree
105	Program for finding Balance factor of a given node	Binary Tree
Project 8: Program for Huffman Coding		
Project 9: Creation of Binary Tree from Pre-Order and Inorder Traversal		
Project 10: Program to Create Expression Tree and its Traversal		
106	Program for BST Insertion, traversal, Minimum, maximum and Successor operations	Binary Search Tree
107	Program for BST Deletion	Binary Search Tree
108	Program to convert the given BST to Max Heap	Binary Search Tree
109	Program to check IF a Binary Tree is BST or not	Binary Search Tree
Project 11: Program for Binary Search Tree Deletions		
110	Program for AVL Tree Rotations, Insertion and Traversal operations	AVL Tree
Project 12: Program for Implementation of Interval Tree		
111	Program for BFS on a Graph	Graph
112	Program for DFS on a Graph	Graph
113	Program for Warshall's ALGORITHM for APSP	Graph
114	Program for Dijikstra's ALGORITHM for SSSP	Graph
115	Program for Warhall's ALGORITHM for Transitive Closure	Graph
116	Prorgram for Prim's ALGORITHM for Minimal Spanning Tree	Graph
117	Prorgram for Kruskal's ALGORITHM for Minimal Spanning Tree	Graph
118	Program for topological sorting of given graph	Graph
Project 13: Program for implemenation of Travelling Salesman Problem		

Experiment 1

Object: Write an ALGORITHM for array insertion, deletion and traversal.
DOMAIN: Array

```
ALGORITHM Traverse(A[], N)
BEGIN:
        FOR i=1 TO N DO
              WRITE(A[i])
END;
```

Time Complexity:Ө(N)
Space Complexity:Ө(1)

```
ALGORITHM Insertion(A[], N, i, key)
BEGIN:
        FOR j=N TO i STEP-1 DO
              A[j+1]=A[j]
              A[i]=key
              N=N+1
END;
```

Time Complexity:Ө(N)
Space Complexity:Ө(1)

```
ALGORITHM Deletion(A[], N, i)
BEGIN:
       X=A[i]
        FOR j=i+1 TO N DO
              A[j-1]=A[i]
              N=N-1
       RETURN x
END;
```

Time Complexity:Ө(N)
Space Complexity:Ө(1)

EXPERIMENT No 2

Object-Write an ALGORITHM for insertion in sorted array.
DOMAIN-Array

```
ALGORITHM  Sorted(A[], N, key)
BEGIN:
     i=0
    WHILE A[i]<key DO
         i=i+1
        RETURN i
END;
```

Time Complexity:Ө(N)
Space Complexity:Ө(1)

```
ALGORITHM: INS_sorted(A[], N ,i, key)
BEGIN:
    FOR j=N-1 TO i STEP-1 DO
        A[j+1]=A[j]
    A[i]=key
    N=N+1
END;
```

Time Complexity:Ө(N)
Space Complexity:Ө(1)

Experiment 3

Object:-Write an ALGORITHM to Find the number which is not repeated in Array of integers, others are present for two times.
DOMAIN:-Array

ALGORITHM: Arr_func(A[], N)
BEGIN:

```
    K=0,c,B[20]
    FOR i=0 TO N DO
        c=0
        FOR j=0 TO N DO
            IF A[j]==A[i] THEN
                c=c+1
            IF c==1 THEN
                B[k++]=A[i]
    FOR i=0 TO k DO
        WRITE(B[i])
```

END;

Time Complexity:ϴ(N^2)
Space Complexity:ϴ(1)

Experiment 4

Object- Write an ALGORITHM For Linear Search.
DOMAIN-Searching

```
ALGORITHM Linear_search(A[], N, key)
BEGIN:
        FOR i=1 TO N DO
                IF A[i]==key THEN
                        RETURN i
        RETURN -1
END;
```

Worst Case Time Complexity: O(N)
Best Case Time Complexity: □(1)
Space Complexity: Θ(1)

Experiment.5

Object-Write an ALGORITHM for Binary Search.
DOMAIN-Searching

ALGORITHM Binary_search(A[], N, key)
BEGIN:

```
        HIGH=N-1
        LOW=0
        WHILE LOW<=HIGH DO
                MID=(LOW+HIGH)/2
                IF A[MID]==key THEN
                      RETURN MID
                ELSE
                        IF key<A[MID] THEN
                              HIGH=MID-1
                        ELSE
                              LOW=MID+1
          RETURN -1
```

END;

Worst Case Time Complexity: O(logN)
Best Case Time Complexity: □(1)
Space Complexity: Ɵ(1)

<u>Experiment.6.</u>

Object- Write an ALGORITHM for Index Sequential Search.
DOMAIN-Searching

ALGORITHM: INDsearch(data[N],KEY,index[M][2])
BEGIN:

```
FOR i=0 TO M-1 DO
       IF KEY==index[i][1] THEN
         RETURN index[i][0]
   ELSE
        IF KEY <index[i][1] THEN
             high=index[i][0]-1
             Low =index[i-1][0]+1
              BREAK
   FOR i=low TO high DO
            IF KEY ==data[i] THEN
               RETURN i
   RETURN -1
```

END;

Worst Case Time Complexity: O(N/K+K)
Best Case Time Complexity: [illegible](1)
Space Complexity: ϴ(1)

<u>**Experiment No 7**</u>

Object- Write an ALGORITHM for Bubble, Selection and Insertion Sort.
DOMAIN:Sorting

ALGORITHM: BubbleSort(A[], N)
BEGIN:

```
    FOR i=1 TO N-1 DO
        FOR j=1 TO N-i DO
            IF A[j]>A[j+1]
                k=A[j]
                A[j]=A[j+1]
                A[j+1]=k
```

END;

Worst Case Time Complexity:O(N^2)
Best Case Time Complexity: Omega(N)
Space Complexity:Θ(1)

ALGORITHM: InsertionSort(A[], N)
BEGIN:

```
    FOR i=2 TO N DO
        key=A[i]
        j=i-1
        WHILE j>=1  AND A[j]>key DO
            A[j+1]=A[j]
            j=j-1
        A[j+1]=key
```

END;

Worst Case Time Complexity:O(N^2)
Best Case Time Complexity: Omega(N)
Space Complexity:Θ(1)

ALGORITHM: SelectionSort(A[], N)
BEGIN:

```
    FOR i=1 TO N-1 DO
        min=i
        FOR  j=i+1 TO N DO
            IF A[j]<A[min] THEN
            min=j
        Exchange(A[min], A[i])
```

END;

Time Complexity: Θ(N^2)
Space Complexity:Θ(1)

Experiment No 8

Object- Write an ALGORITHM for Shell Sort.
DOMAIN:Sorting

ALGORITHM shellSort(array, n):

```
    gap = n // 2
    WHILE gap > 0 DO
        FOR i=gap to n DO
            temp = array[i]
            j = i
            WHILE j >= gap AND array[j - gap] > temp DO
                array[j] = array[j - gap]
                j -= gap

            array[j] = temp
        gap //= 2
```

Time Complexity: Θ(N^2)
Space Complexity:Θ(1)

<u>**Experiment 9**</u>

Object-Write an ALGORITHM for Quick Sort.
DOMAIN-Sorting

ALGORITHM: QuickSort(A[],low,high)
BEGIN:

```
    IF low<high THEN
        j=Partition(A[],low,high)
        QuickSort(A[],low,j-1)
        QuickSort(A[],j+1,high)
```

END;

ALGORITHM: Partition(A[],low,high)
BEGIN:

```
    i=low, j=high+1,pivot=A[low]
    DO
        DO
            i=i+1
        WHILE(A[i]<pivot)
        DO
            J=j-1
        WHILE(A[j]>pivot)

        IF i<j THEN
            Exchange(A[i],A[j])
    WHILE(i<j)

     Exchange(A[j],A[low])
    RETURN j
```

END;

Worst Case Time Complexity:O(N^2)
Best Case Time Complexity: □(Nlog$_2$ N)
Space Complexity: □(log$_2$ N)

<u>Experiment.10</u>

Object-Write an ALGORITHM for Merge Sort.
DOMAIN-Sorting

```
ALGORITHM: MergeSort(A[],low,high)
BEGIN:
        IF low<high DO
                Mid=(low+high)/2
                MergeSort(A[],low,mid)
                MergeSort(A[],mid+1, high)
                Merge(A, low,mid,high)
END;
 ALGORITHM: Merge(A[], low,mid,high)
BEGIN:
        i=low,j=mid+1,k=high
        WHILE i<=mid AND j<=high DO
                IF A[i]<A[j] THEN
                        C[k]=A[i]
                        i=i+1
                        k=k+1
                ELSE
                        C[k]=A[j]
                        j=j+1
                        k=k+1
        WHILE i<=mid DO
                C[k]=A[i]
                i=i+1
                k=k+1
        WHILE j<=high DO
                C[k]=A[j]
                J=j+1
               k=k+1
        FOR i=low TO high DO
                A[i]=C[i]
END;
```

Time Complexity: $O(N\log_2 N)$
Space Complexity: □(N)

Experiment.11

Object- Write an ALGORITHM for merging of two sorted arrays.
DOMAIN-Array

ALGORITHM: MergeArr(A[],m,B[],n)
BEGIN:

```
        C[m+n]
        i=1, j=1, k=1
        WHILE i<=m AND j<=n DO
                IF A[i]<B[j] THEN
                        C[k]=A[i]
                        i=i+1
                        k=k+1
                ELSE
                        C[k]=B[j]
                        J=j+1
                        k=k+1
        WHILE i<=m DO
                C[k]=A[i]
               i=i+1
                k=k+1
        WHILE j<=n DO
                C[k]=B[j]
                J=j+1
                k=k+1
        RETURN C
```

END;

Time Complexity: □(N)
Space Complexity: □(N)

Experiment.12

Object-Write an ALGORITHM for finding set elements of A that belong to B.
DOMAIN-Array

ALGORITHM: A_AND_B(A[],m,B[],n)
BEGIN:

```
        C[m+n]
        i=1, j=1, k=1
        WHILE i<=m AND j<=n DO
                IF A[i]<B[j] THEN
                        i=i+1
                ELSE
                        IF A[i]==B[j] THEN
                        C[k]=B[j]
                        i=i+1
                        j=j+1
                        k=k+1
                ELSE
                        j=j+1
                RETURN C
```

END;

Time Complexity: ϴ(N)
Space Complexity:ϴ(N)

Experiment.13

Object-Write an ALGORITHM for finding set elements of A that does not belong to B.
DOMAIN-Array

```
ALGORITHM: A_AND_NOT_B(A[],m,B[],n)
BEGIN:
        C[m+n]
        i=1, j=1, k=1
        WHILE i<=m AND j<=n DO
                IF A[i]<B[j] THEN
                        C[k]=A[i]
                        k=k+1
                        i=i+1
                ELSE
                        IF A[i]==B[j] THEN
                                i=i+1
                                j=j+1
                        ELSE
                                j=j+1
        WHILE i<=m DO
                C[k]=A[i]
                i=i+1
                k=k+1

        RETURN C
END;
```

Time Complexity: Θ(N)
Space Complexity:Θ(N)

Experiment.14.

Object-Write an ALGORITHM for Set Union.
DOMAIN-Array

ALGORITHM: SetUnion(A[],m,B[],n)
BEGIN:

```
        C[m+n]
        i=1, j=1, k=1
        WHILE i<=m AND j<=n DO
                IF A[i]<B[j] THEN
                        C[k]=A[i]
                        i=i+1
                        k=k+1
                 ELSE
                        IF A[i]==B[j] THEN
                                C[k]=B[j]
                                i=i+1
                                j=j+1
                                k=k+1
                         ELSE
                                C[k]=B[j]
                                j=j+1
                                k=k+1
        WHILE i<=m DO
                C[k]=A[i]
                i=i+1
                k=k+1
        WHILE j<=n DO
                C[k]=B[j]
                J=j+1
                k=k+1
RETURN C
```

END;

Time Complexity:Ө(N)
Space Complexity:Ө(N)

Experiment.15.

Object-Write an ALGORITHM for set Intersection.
DOMAIN-Array

ALGORITHM: SetIntersection(A[],m,B[],n)
BEGIN:

```
        C[m+n]
        i=1, j=1, k=1
        WHILE i<=m AND j<=n DO
                IF A[i]<B[j] THEN
                        i=i+1
                ELSE
                        IF A[i]==B[j] THEN
                                C[k]=B[j]
                        i=i+1
                        j=j+1
                        k=k+1
                        ELSE
                                j=j+1
        RETURN C
```

END;

Time Complexity:ϴ(N)
Space Complexity:ϴ(N)

<u>**Experiment.16**</u>

Object-Write an ALGORITHM for set DIFference.
DOMAIN:Array

ALGORITHM: SetDIFference(A[],m,B[],n)
BEGIN:

```
        C[m+n]
        i=1, j=1, k=1
        WHILE i<=m AND j<=n DO
                IF A[i]<B[j] THEN
                        i=i+1
                ELSE
                         IF A[i]==B[j] THEN
                        i=i+1
                        j=j+1
                ELSE
                        C[k]=B[j]
                        j=j+1
                        k=k+1
        WHILE j<=n DO
                C[k]=B[j]
                J=j+1
                k=k+1
        RETURN C
```

END;

Time Complexity:ϴ(N)
Space Complexity:ϴ(N)

Experiment.17.

Object-Write an ALGORITHM for Counting Sort.
DOMAIN-Sorting

ALGORITHM: CountingSort(A[],k,n)
BEGIN:

```
    FOR i = 0 TO k DO
        c[i] = 0
    FOR j = 0 TO n DO
        c[A[j]] = c[A[j]] + 1
    FOR i = 1 TO k DO
        c[i] = c[i] + c[i-1]
    FOR j = n-1 TO 0 STEP-1 DO
        B[ c[A[j]]-1 ] = A[j]
        c[A[j]] = c[A[j]] - 1
    RETURN B
```

END;

Time Complexity: Omega(N)
Space Complexity:Ɵ(N)

Experiment.18

Object-Write an ALGORITHM for Radix Sort.
DOMAIN-Sorting

ALGORITHM: RadixSort(A[],N,d)
BEGIN:

```
    FOR i=1 TO d DO
        Apply counting Sort on A[] at radix i
```

END;

Time Complexity: Ө(N)
Space Complexity:Ө(N)

<u>**Experiment 19.**</u>

Object- Write an ALGORITHM for Matrix Addition.
DOMAIN-Array

ALGORITHM: Matrixadd(A[][], B[][], M,N)

```
BEGIN:C[M][N]
        FOR i=1 TO M DO
                FOR j=1 TO N DO
                        C[i][j]=A[i][j]+B[i][j]
        RETURN C
END;
```

Time Complexity: $\Theta(N^2)$
Space Complexity:$\Theta(N^2)$

Experiment 20

Object-Write an ALGORITHM for matrix Multiplication.
DOMAIN-Arrray

ALGORITHM: Matrixmultiply(A[][], M,N, B[][], P,Q)
BEGIN:

```
        C[M][Q]
        IF N!=P THEN
        FOR i=1 TO M DO
        FOR j=1 T0 Q DO
                        C[i][j]=0
                        FOR k=1 TO N DO
                                C[i][j]=C[i][j]+A[i][k]*B[k][j]
        RETURN C
```

END;

Time Complexity: $\Theta(N^3)$
Space Complexity:$\Theta(N^2)$

Experiment.21

Object- Write an ALGORITHM for Matrix Transposition.
DOMAIN:Array

ALGORITHM: Matrixtranspose(A[][], M,N)
BEGIN:

```
    B[N][M]
    FOR i=1 TO M DO
        FOR j=1 TO N DO
            B[j][i]=A[i][j]
    RETURN B
```

END;

Time Complexity: $\Theta(N^2)$
Space Complexity:$\Theta(N^2)$

Experiment 22

Object-Write an ALGORITHM for matrix transposition without using second matrix.
DOMAIN-Array

ALGORITHM: Matrixtranspose(A[][], M,N)
BEGIN:

```
    FOR i=1 TO M DO
        FOR j=1 TO i DO
            temp=A[i][j]
            A[i][j]=A[j][i]
            A[j][i]=temp
    RETURN A
```

END;

Time Complexity: $\Theta(N^2)$
Space Complexity: $\Theta(1)$

Experiment 23

Object-Write a program to print a given matrix in spiral form.
DOMAIN-Array

ALGORITHM MatrixInSpiralForm(int[][100],int,int)
BEGIN:

```
        int i,k=0,l=0;
         WHILE  k<r and l<c DO
                  for(i=l;i<c;i++)
                      write(a[k][i])
         k++
         FOR(i=k;i<r;i++) DO
                  write(a[i][c-1])
         c--
         IF(k<r) THEN
                  for(i=c-1;i>=l;i--)
                            write(a[r-1][i])
         r--
         IF(l<c) THEN
                  for(i=r-1;i>=k;i--) DO
                           printf("%d ",a[i][l])
         l++;
```

END;

Time Complexity: ϴ(N)
Space Complexity:ϴ(1)

Experiment 25

Object-Write an ALGORITHM for creation of min heap and max heap.
DOMAIN-Heap

```
ALGORITHM MaxHeapIFy(A[],N)
BEGIN:
          FOR i=N/2 TO STEP-1 DO
                    Adjust(A,i,N)
END;

ALGORITHM Adjust(A[],i,N)
BEGIN:
          WHILE 2*i<=N DO
                    j=2*i
                    IF j+1<=N THEN
                              IF  A[j+1]>A[j]
                              j=j+1
                  IF A[j]>A[i] THEN
                              Exchange(A[j],A[i])
                    ELSE
                              BREAK
          i=j
END;

 ALGORITHM MinHeapIFy(A[],N)
BEGIN:
         FOR i=N/2 TO STEP-1 DO
                    Adjust(A,i,N)
END;

ALGORITHM Adjust(A[],i,N)
BEGIN:
          WHILE 2*i<=N DO
                    j=2*i
                    IF j+1<=N THEN
                              IF  A[j+1]<A[j]
                              j=j+1
                    IF A[j]<A[i] THEN
                              Exchange(A[j],A[i])
                    ELSE
                              BREAK
          i=j
END;
```

Time Complexity: Θ(NlogN)
Space Complexity:Θ(1)

<u>**Experiment 26**</u>

Write an ALGORITHM for insertion in min or max heap.

```
ALGORITHM  InsertHeap(A[],N,key)
BEGIN:
        A[N+1]=key
        i=N+1
        WHILE i>1 AND A[i]<A[i/2] DO
              Exchange(A[i],A[i/2])
                I=i/2
        N=N+1
END;
```

Time Complexity: Θ(N)
Space Complexity:Θ(1)

Experiment 27

Write an ALGORITHM for deletion from min or max heap.

```
ALGORITHM DeleteHeap(A[],N)
BEGIN:
        x=A[i]
      A[i]=A[N]
      Adjust(A[],1,N-1)
     RETURN x
END;
```

Time Complexity: Ɵ(logN)

Space Complexity:Ɵ(1)

Experiment 28

Write an ALGORITHM for realizing Heap as Ascending/ Descending Priority Queue

```
ALGORITHM  PQInser(A[],N,key)
BEGIN:
            A[N+1]=key
            i=N+1
            WHILE i>1 AND A[i]<A[i/2] DO
                     Exchange(A[i],A[i/2])
                        I=i/2
            N=N+1
END;
```

Time Complexity: ϴ(LogN)
Space Complexity:ϴ(1)

```
ALGORITHM PQDelete (A[],N)
BEGIN:
            x=A[i]
        A[i]=A[N]
        Adjust(A[],1,N-1)
       RETURN x
END;
```

Time Complexity: ϴ(logN)
Space Complexity:ϴ(1)

ALGORITHM Adjust(A[],i,N)
BEGIN:

```
            WHILE 2*i<=N DO
                        j=2*i
                        IF j+1<=N THEN
                                    IF  A[j+1]>A[j]
                                    j=j+1
                      IF A[j]>A[i] THEN
                                    Exchange(A[j],A[i])
                        ELSE
                                    BREAK
            i=j
```

END;
Time Complexity: ϴ(logN)
Space Complexity:ϴ(1)

Experiment 29

Object: Program for Hash Table Implementation for Basic Hash Function (Without collisions)

DOMAIN-String

```
ALGORITHM DivisionHash(Key, TS)
BEGIN:
        NearestPrime(TS)
        h=Key%TS
        RETURN h
END;
```

Time Complexity: θ(1)
Space Complexity:θ(1)

```
ALGORITHM MidsquareHash(Key, TS)
BEGIN:
        L=LengthOfKey(Key)
        n=key*key
        x=Ceil((2*L-TS)/2)
        n=n/10↑x
        h=n%10↑L
        RETURN h
END;
```

Time Complexity: θ(1)
Space Complexity:θ(1)

```
ALGORITHM FoldingHash(Key, TS)
BEGIN:
        L=LengthOfKey(Key)
        n=key
        Sum=0
        WHILE n>=0 DO
                r=n%TS
                Sum=Sum+r
                n=n/TS
        h=Sum%TS
        RETURN h
END;
```

Time Complexity: θ(1)
Space Complexity:θ(1)

Experiment 30

Object: Program for Hash Table Implementation for Collision Resolution using Linear Probing

ALGORITHM LinearProbing (T[ ], N, Key,h)

BEGIN:

```
    i=h
    DO
        IF T[i]==key THEN
            RETURN i
        ELSE
            IF T[i]==Blank THEN
                RETURN -1
            ELSE
                i=(i+1)%N
    WHILE(1)
```

END;

Time Complexity: θ(N)
Space Complexity:θ(1)

Experiment 31

Program for Hash Table Implementation for Collision Resolution using Quadratic Probing

ALGORITHM QuadraticProbing (T[ ], N, Key,h)

BEGIN:

```
    i=0
    x=h
    DO
        IF T[x]==key THEN
            RETURN x
        ELSE
            IF T[x]==Blank THEN
                RETURN -1
            ELSE
                x=(h+a*i+bi↑2)%N
                i=i+1
    WHILE(1)
```

END;

Program for Hash Table Implementation for Collision Resolution using Double Hashing/Rehashing

Time Complexity: θ(N)

Space Complexity:θ(1)

Experiment 32

ALGORITHM DoubleHashingProbe (T[ ], N, Key,h,h')

```
BEGIN:
        i=1
        x=h
        DO
                IF T[x]==key THEN
                        RETURN x
                ELSE
                        IF T[x]==Blank THEN
                                RETURN -1
                        ELSE
                                x=(h+i*h')%N
                                i=i+1
        WHILE(1)
END;
```

Time Complexity: θ(N)
Space Complexity:θ(1)

Experiment 33

Program for Hash Table Implementation for Collision Resolution using Chaining

Search

ALGORITHM HashCollisionResolutionChaining (SparseChain[ ], N, Key,h)
BEGIN:

```
        i=1
p=SparseChain[h]
WHILE p!=NULL DO
        IF p→data == key THEN
                RETURN p
        ELSE
                P=p→ Next
RETURN -1
```

END;

Storage

ALGORITHM HashCollisionResolutionChaining (SparseChain[ ], N, Key,h)
BEGIN:

```
p=SparseChain[h]
WHILE p→Next! = NULL DO
        p=p→ Next

q=GetNode(key)
p→Next=q
```

END;

Experiment 34

Finding Anagrams: There are two strings. Find out which characters should be deleted such that both strings contain the same characters (May be in dIfferent Order)

ALGORITHM Anagrams(str1,str2)

BEGIN:

```
        Counting[52] = {0}
        i=0
        WHILE str1[i]!='\0' DO
                IF str1[i]>='a' AND str1[i]<='z' THEN
                        Counting[str[i]-'a']++
                ELSE
                        IF str1[i]>='A' AND str1[i]<='Z' THEN
                        Counting[str[i]-'A'+26]++
                i++

        j=0
        WHILE str2[j]!='\0' DO
                IF str2[j]>='a' AND str2[j]<='z' THEN
                        IF(Counting[str2[j]-'a'] == 0)
                                WRITE(str2[j])
                IF str2[j]>='A' AND str2[j]<='Z' THEN
                        IF(Counting[str2[j]-'A'+26] == 0)
                                WRITE(str2[j])
                j++
```

END;

Time Complexity: θ(M+N)
Space Complexity:θ(C)

Experiment 35

There are some numbers in which some are appearing twice but one is not repeated. Find out the number which appears once.

ALGORITHM RepeatedElements(A[ ],N)
BEGIN:

```
        Maximum = - IntMax
        FOR i=0 TO N-1 DO
                IF A[i] < Maximum THEN
                        Maximum = A[i]

        C[Maximum] = {0}

        FOR i=0 TO N-1 DO
                C[A[i]]++

        FOR i=0 TO Maximum DO
                IF C[i] == 1 THEN
                        WRITE(C[i])
```

END;

Time Complexity: θ(Maximum+N)
Space Complexity:θ(Maximum)

Experiment 36

There are two arrays containing some elements. Find out what are the elements which are there in both the arrays what are not.

```
ALGORITHM RepeatedElements(A[ ], N, B[ ], M)
BEGIN:
        Maximum1 = - IntMax
        FOR i=0 TO N-1 DO
                IF A[i] < Maximum1 THEN
                        Maximum1 = A[i]

        CA[Maximum1] = {0}
        FOR i=0 TO N-1 DO
                CA[A[i]]++

        Maximum2 = - IntMax
        FOR i=0 TO M-1 DO
                IF B[i] < Maximum2 THEN
                        Maximum2 = B[i]

        CB[Maximum2] = {0}
        FOR i=0 TO N-1 DO
                CB[B[i]]++
        i=0

        WHILE i<= Maximum1 AND i<=Maximum2 DO
                IF CA[i] >0 AND CB[i]>0 THEN
                        WRITE(CA[i])
                i++

END;
```

Time Complexity: θ(M+N)
Space Complexity:θ(CA+CB) where CA is the Maximum element of A array and CB is the Maximum size of B array

Experiment 37

Find out the values of a,b,c,d (a,b,c,d<=1000) for which $a^3+b^3=c^3+d^3$.

```
Struct DAT
{
int a;
int b;
};

ALGORITHM DirectAddressTableApplication()
BEGIN:
        FOR i=0 TO 1000 DO
                FOR j=0 TO 1000 DO
                        Sum=i*i*i+j*j*j
                        DAT[Sum].a=i
                        DAT[Sum].b=j

        FOR     c=0 TO 1000 DO
                FOR d=0 TO 1000 DO
                        X=c*c*c+d*d*d
                        WRITE(DAT[X].a)
                        WRITE(DAT[X].b)
```

Time Complexity: $\theta(1000^2)$
Space Complexity:$\theta(1000^2)$

Experiment.38

Object-Write an ALGORITHM to find length of a string.
DOMAIN-String

```
ALGORITHM LengthOfString( String str[ ] )
BEGIN:
        i = 0
        WHILE  str[ i ] != '\0'
                i = i + 1
        RETURN i
END;
```

Time Complexity: $\theta(n)$
Space Complexity: $\theta(1)$

<u>**Experiment 39**</u>

Object-Write an ALGORITHM to reverse a given string.
DOMAIN-String

```
ALGORITHM ReverseOfString( String str 1[ ] )
BEGIN:
        String str 2[ ]
        L = LengthOf String( str 1[ ] )
        i = 0
        WHILE  L != '0'
                str 2[ i ]= str 2[ L -1]
                i = i + 1
                L = L – 1
        RETURN str 2
END;
```

Time Complexity: θ(L)
Space Complexity:θ(1)

<u>Experiment 40</u>

Object-Write an ALGORITHM to find IF the given string is palindrome or not.
DOMAIN-String

```
ALGORITHM StringPalindrome( String str[ ] )
BEGIN:
        L = Length Of String( str[ ] )
        i = 0
        WHILE  i != L/2
                IF  str[ i ] = str[ L – i ]
                        c = c +1
        IF c == L/2
                RETURN TRUE
        ELSE
                RETURN FALSE
END;
```

Time Complexity: θ(L)
Space Complexity: θ(1)

Experiment 41

Object-Write an ALGORITHM to find number of words in a paragraph.
DOMAIN-String

```
ALGORITHM Word Count( String str[ ] )
BEGIN:
        c = 0
        i = 0
        WHILE  str[ i ] !='\0'
                IF str[ i ] == "  "
                c = c + 1
        RETURN  c + 1
END;
```

Time Complexity: θ(L)
Space Complexity:θ(1)

Experiment-42

OBJECT-Write an ALGORITHM to convert all lowercase letters to uppercase letters and vice versa.

```
ALGORITHM    conversion(string  a[])
BEGIN:

  i=0;
  WHILE   a[i]!='\0'  DO

     IF  a[i]>='a'&&a[i]<='z'  THEN

        a[i]=a[i]-32;

     ELSE  IF  a[i]>='A'&&a[i]<='Z'  THEN

        a[i]=a[i]+32;

     i++;

END;
```

TIME COMPLEXITY - ϴ (n)
SPACE COMPLEXITY - ϴ (n)

Experiment-43

OBJECT-Write an ALGORITHM to find the given word is present in the sentence and at what location.

```
 ALGORITHM        wordcheck(string str[], string search[])
 BEGIN:

   count1 = 0, count2 = 0
   WHILE   str[count1] != '\0'   DO
            count1=count1+1
   WHILE  search[count2] != '\0'  DO
          count2=count2+1
   FOR  i = 0  TO  count1 - count2  DO

      FOR  j = i   TO   i + count2  DO

         flag = 1
         IF  str[j] != search[j - i]  THEN

            flag = 0
            break


      IF   flag == 1  THEN
         break

   IF  flag == 1
       WRITE("SEARCH SUCCESSFUL")
   ELSE
      WRITE("SEARCH UNSUCCESSFUL")

END;
```

Experiment-44

OBJECT-Write an ALGORITHM to sort names in a dictionary order.
ALGORITHM sortingofnames(string name[],int n)

```
BEGIN:
    FOR  i = 0  TO   n  DO

        strcpy(tname[i], name[i])


    FOR   i = 0  TO   n - 1   DO

        FOR  j = i + 1  TO  n  DO

            IF   strcmp(name[i],  name[j]>0)  THEN

                strcpy(temp, name[i])
                strcpy(name[i], name[j])
                strcpy(name[j], temp)


    RETURN  name

END;
```

Experiment No-45

OBJECT-Write an ALGORITHM for reversing all words in a string.

```
ALGORITHM    reversingwords(string  str[])
BEGIN:
   FOR  i = 0  TO  str[i] != '\0'  DO

      IF   str[i] == ' '  THEN

         str1[k][j]='\0'
         k=k+1
         j=0

      ELSE

         str1[k][j]=str[i]
         j=j+1

   }
   str1[k][j] = '\0'

   FOR   i = 0  TO   k   DO

      len = strlen(str1[i])
      FOR  j = 0, x = len - 1  TO   x  STEP+1,STEP-1   DO

         temp = str1[i][j]
         str1[i][j] = str1[i][x]
         str1[i][x] = temp

   RETURN  str
END;
```

Experiment 46

Write an ALGORITHM for Decimal to Binary Conversion.

```
ALGORITHM    Decimal to Binary(n)
BEGIN:
        Stack S
        Initialize(s)
        WHILE  n!=0 DO
                r=n%2
                PUSH(S,r)
                n=n/2
        WHILE ! Empty (s) DO
                X=Pop(s)
                Write(x)
END;
```

Time Complexity- θ(logN)
Space Complexity- θ(N)

Experiment 47

Write an ALGORITHM for Decimal to octal conversion.

```
ALGORITHM      Decimal to Octal(n)
BEGIN:
        Stack S
        Initialize(s)
        WHILE   n!=0 DO
                r=n%8
                PUSH(S,r)
                n=n/8
        WHILE  ! Empty (s) DO
                X=Pop(s)
                Write(x)
END;
```

Time Complexity- θ(logN)
Space Complexity- θ(N)

Experiment 48

Write an ALGORITHM for Decimal to hexadecimal Conversion.

```
ALGORITHM      Decimal to Hexadecimal(n)
BEGIN:
        Stack S
        Initialize(s)
        WHILE   n!=0 DO
                r=n%16
                PUSH(S,r)
                n=n/16
        WHILE  ! Empty (s) DO
                X=Pop(s)
              IF(x<10)   THEN
                WRITE  x
                ELSE
              IF(x==10)   THEN
                WRITE   A
              IF(x==11)   THEN
                WRITE   B
              IF(x==12)   THEN
                WRITE   C
              IF(x==13)   THEN
                WRITE   D
                  IF(x==14)   THEN
                WRITE   E
              IF(x<15)   THEN
                WRITE   F
END
```

Time Complexity- θ(logN)
Space Complexity- θ(N)

Experiment 49

Write an ALGORITHM for Decimal to any base conversion.

```
ALGORITHM      Decimal to AnyBase(n,b)
BEGIN:
          Stack S
          Initialize(s)
          WHILE   n!=0 DO
                    r=n%b
                    PUSH(S,r)
                    n=n/b
          WHILE  ! Empty (s) DO
                    X=Pop(s)
                    Write(x)
END;
```

Time Complexity- θ(logN)
Space Complexity- θ(N)

Experiment 50

Write An ALGORITHM for Stack Primitive Operations.

```
ALGORITHM           Initialize stack(Stack S)
Begin:
            S.TOP=-1
End;
```

Time Complexity- θ(1)
Space Complexity- θ(1)

```
ALGORITHM          Push(Stack S,key)
Begin:
          IF S.TOP==SIZE-1 THEN
                WRITE("Stack Overflows")
                EXIT (1)
          S.TOP++
           S.ITEM[S.TOP]=key
End;
```

Time Complexity- θ(N), Space Complexity- θ(1)

```
ALGORITHM          Empty (Stack S)
Begin:
          IF S.TOP==-1 THEN
               RETURN TRUE
          ELSE
               RETURN FALSE
End;
```

Time Complexity- θ(1) , Space Complexity- θ(1)

```
ALGORITHM          Pop  (Stack S,key)
  Begin:
          IF EMPTY(S) THEN
                WRITE ("Stack underflows")
                EXIT(1)
       X=S.ITEM[S.TOP]
          S.TOP- -
             RETURN X
End;
```

Time Complexity- θ(1), Space Complexity- θ(1)

```
ALGORITHM          STACKTOP (Stack S)
Begin:
          RETURN (S.ITEM[S.TOP])
End;
```

Time Complexity- θ(1), Space Complexity- θ(1)

EXPERIMENT 51

OBJECT: To write an ALGORITHM for Evaluation postfix expression.
DOMAIN: STACK

```
ALGORITHIM    POSTFIX EVALUATION (Postfix Expression)
BEGIN:
             STACK OpndStack
            Initialize (OpndStack)
    WHILE not end of input from postfix expression Do
           Symbol = Next character from postfix expression
       IF   Symbol is an operand THEN
             push  (OpndStack ,symbol)
      ELSE
              oprnd 2=POP (OpndStack)
             oprnd 1=POP(OpndStack)
             value=Result of applying symbol to oprnd1 and oprnd2
            push(OpndStack,value)
   Result = pop(OpndStack)
  RETURN Result
END;
```

TIME COMPLEXITY- ϴ(N)

SPACE COMPLEXITY- ϴ(N)

EXPERIMENT 52

OBJECT: To write an ALGORITHM for infix to postfix conversion.

DOMAIN: STACK

ALGORITHM Infix to postfix (Infix expression)

```
BEGIN:
        STACK  (Opstack)
        Initialize (Opstack)
      WHILE   not the end of input from Infix Expression DO
            Symbol = next symbol from Infix Expression
      IF Symbol is an operand THEN
            Add symbol to postfix expression
      ELSE
                   WHILE (! Empty  (Opstack)  &&  Prcd (Stack Top(Opstack) ,Symbol)
                         x=pop (Opstack)
                         Add x to postfix Expresssion

                   IF  Symbol = = ')' THEN
                         x = pop(Opstack)
                   ELSE
                   PUSH( Opstack ,Symbol)
      PUSH( Opstack ,Symbol)

      WHILE   ! Empty(Opstack) DO
            x= pop(Opstack)
            Add x to Postfix Expression
 RETURN Postfix Expression
END;
```

TIME COMPLEXITY- ϴ(N)

SPACE COMPLEXITY- ϴ(N)

<u>EXPERIMENT 53</u>

OBJECT: To write an ALGORITHM for infix to prefix conversion.

DOMAIN: STACK

```
ALGORITHM       Infix to postfix (Infix expression)
BEGIN:
        Reverse (Infix Expression)
        STACK  (Opstack)
        Initialize (Opstack)
        WHILE   not the end of Symbol from Infix Expression DO
                Symbol = next symbol from Infix Expression
                IF Symbol is an operand THEN
                        Add symbol to postfix expression
                ELSE
                        WHILE (! Empty  (Opstack)  && ! Prcd ( Symbol,Stack Top(Opstack) )
                                        x=pop (Opstack)
                                        Add x to postfix Expresssion
                                IF  Symbol = = ')' THEN
                                x = pop(Opstack)
                                ELSE
                                PUSH( Opstack ,Symbol)
                        PUSH( Opstack ,Symbol)

        WHILE   ! Empty(Opstack)  DO
                x= pop(Opstack)
                Add x to Postfix Expression
        RETURN  Reverse (Prefix Expression)
 END;
```

TIME COMPLEXITY- Ɵ(n2)

SPACE COMPLEXITY- Ɵ(n)

EXPERIMENT 54

OBJECT: To write an ALGORITHM for prefix evaluation.

DOMAIN: STACK

ALGORITHIM PREFIX EVALUATION (Prefix Expression)

```
BEGIN:
            Reverse  (Prefix Expression)
           STACK OpStack
          Initialize (OpStack)
   WHILE not end of input from prefix expression DO
         Symbol = Next character from prefix equation
      IF   Symbol is an operand THEN
           push  (OpStack ,symbol)
     ELSE
            oprnd 1=push(OpStack)
           oprnd 2=push(OpStack)
           value=Result of applying symbol to oprnd1 and oprnd2
          push(OpStack ,value)
   Result = pop(OpStack)
   RETURN Result
END;
```

TIME COMPLEXITY- Ө(N)

SPACE COMPLEXITY- Ө(N)

EXPERIMENT 55

OBJECT - Program to check the validity of Paranthesized Arithmetic Expression using Stack.

DOMAIN - Stack

ALGORITHM Valid Expression (String Exp[])

```
BEGIN:
          Valid=1
          Stack   S
          Initialize  S
     WHILE Exp[i] !='$' DO
          IF Exp[i] = = '(' THEN
          PUSH (S,Exp[i])
          ELSE
               IF Exp[i] = = '(' THEN
               IF (Empty (S))  THEN
                    Valid=0
                BREAK
               ELSE
                  Pop(S)
     i++
          IF valid = =0 THEN
          WRITE("Valid Expression")
          ELSE
          IF  Empty(S) THEN
             WRITE("Valid  Expression")
          ELSE
             WRITE("Invalid  Expression")
     END;
```

TIME COMPLEXITY-ϴ(n)

SPACE COMPLEXITY-ϴ(n)

Experiment. 57

OBJECT-Write an ALGORITHM to check IF given number is palindrome using stack.

DOMAIN -Stack

```
 ALGORITHM    palindrome check (str[])
Begin:
          i=0
          Stack S
          Initialize (s)
         WHILE   str[i] !='\0' DO
          PUSH (s , str[i])
          i++

          i=0
          WHILE  str[i]!='\0' DO
                    IF Str[i]==StackTOP(S) THEN
                       POP(s)
                    ELSE
                       Break;
          IF   Empty (s) THEN
                    WRITE ("Palindrome")
          ELSE
                    WRITE ("  Not  Palindrome")
End;
```

TIME COMPLEXITY - ϴ (n)

SPACE COMPLEXITY - ϴ (n)

Experiment.58

OBJECT - Write an ALGORITHM to reverse a string using Stack.

DOMAIN- Stack

```
ALGORITHM      String Reverse(String str[])
Begin:
        i=0
        Stack S
        Initialize (s)
        WHILE   str[i] !='\0' DO
        PUSH (s , str[i])
        i++
        WHILE   ! Empty(s) DO
         x=pop(s)
        WRITE(x)
 End;
```

TIME COMPLEXITY - ϴ(1)

SPACE COMPLEXITY - ϴ (1)

Experiment.59

Q. Write an ALGORITHM to calculate factorial of a number using recursion.

```
ALGORITHM FACTORIAL(a)
BEGIN :
IF a==0
        RETURN(1)
ELSE
        IF(a>0)
                RETURN(a*FACTORIAL(a-1))
END;
```

Time Complexity: Θ(n)
Space Complexity: Θ(n)

Experiment.60

Q. Write an ALGORITHM for tower of Hanoi for n disk.

```
ALGORITHM TOH(N,S,M,D)
BEGIN:
IF N==1 THEN
        Transfer disk from S to D
ELSE
        TOH(N-1,S,M,D)
        Transfer Disk From S to D
        TOH(N-1M,S,D)
End;
```

Time Complexity: Θ (2^n)
Space Complexity: Θ (n)

Experiment.61

Q. Write an ALGORITHM to calculate power of a given number using recursion.

```
ALGORITHM POWER(a,b)
BEGIN:
        IF b==0 THEN
                RETURN 1
         ELSE
                RETURN  a*POWER(a,b-1)
END;
```

Time Complexity: O(b)
Space Complexity: ϴ(b)

Experiment.62

Object: Write an ALGORITHM to calculate power of a given number using divide and Conquer

```
ALGORITHM POWER(a,b)
BEGIN:
          IF b == 0 THEN
                    RETURN 1
           ELSE
                    IF b%2 == 0 THEN
                              RETURN  POWER(a,b/2) * POWER(a,b/2)
                    ELSE
                              RETURN  a+ POWER(a,b/2) * POWER(a,b/2)

END;
```

Time Complexity: O(log b)
Space Complexity: Ө(log b)

Experiment.63

Q. Write an ALGORITHM to Calculate n terms of Fibonacci series using recursion.

```
ALGORITHM  Fibo(a)
BEGIN:
            IF a==1 THEN
                        RETURN 0
             ELSE
                        IF a==2 THEN
                                    RETURN 1
                         ELSE
                                    RETURN Fibo(a-1)+Fibo(a-2)
END;
```

Time Complexity: Ɵ (2^N)
Space Complexity: Ɵ(N)

Experiment.64

Q. Write an ALGORITHM to calculate HCF of 2 numbers.

```
ALGORITHM HCF(a,b)
BEGIN:
        IF a==b THEN
                RETURN  a
         ELSE IF a>b THEN
                RETURN HCF(a-b,b)
                 ELSE
                RETURN HCF (a,b-a)
END;
```

Time Complexity: O(log n)
Space Complexity: Θ(1)

Experiment.65

Q. Write an ALGORITHM to calculate reverse of a number using recursion.

```
ALGORITHM REV (a,len)
BEGIN:
        IF len ==1
                RETURN a
        ELSE
                RETURN((a%10)*pow(10,len-1))+REV(a/10,len-1)
END;
```

Time Complexity: Θ (log n)
Space Complexity: Θ (log n)

Experiment. 66

OBJECT: **Program for Array implementation of Linear Queue**

• **ALGORITHM INTIALIZE(QUEUE Q)**

```
BEGIN:
   Q.REAR=0
   Q.FRONT=1
END;
```

• **ALGORITHM ENQUEUE(QUEUE Q,key)**

```
BEGIN:
IF Q.REAR ==SIZE THEN
          write (Queue overflow)
          Exit 1
      Q.REAR=Q.REAR+1
      Q.item[Q.REAR]=key
END;
```

• **ALGORITHM DEQUEUE(QUEUE Q)**

```
BEGIN:
IF Q.REAR-Q.FRONT+1==0 THEN
           Exit(1)
x=Q.item[Q.FRONT]
 Q.FRONT =Q.FRONT +1
RETURN x
END;
```

• **ALGORITHM EMPTY(QUEUE Q)**

```
BEGIN:
          IF Q.REAR – Q.FRONT +1 ==  0 THEN
                    RETURN TRUE
          ELSE
                    RETURN FALSE
END;
```

For all the above ALGORITHM

Time Complexity: ϴ (1)

Space Complexity: ϴ (1)

Experiment.67

OBJECT: **Program for Array implementation of Circular Queue**

- **ALOGRITHM INTIALIZATION(QUEUE Q)**

```
BEGIN :
        CQ.REAR=Size-1
        CQ.FRONT=Size-1
END;
```

- **ALGORITHM ENQUEUE(CQUEUE CQ,KEY)**

```
BEGIN:
    IF (CQ.REAR+1)%SIZE==CQ.FRONT THEN
        write(Queue overflows)
        Exit(1)
    CQ.REAR=(CQ.REAR+1)%SIZE
    CQ.item[CQ.REAR]=key
END;
```

- **ALGORITHM DEQUEUE(CQUEUE CQ)**

```
BEGIN:
    IF CQ.REAR==CQ.FRONT
        write( queue overflow)
        Exit(1)
    CQ.FRONT=(CQ.FRONT+1)%Size
    x=CQ.item[CQ.FRONT]
    RETURN(x)
END;
```

- **ALGORITHM EMPTY(CQUEUE CQ)**

```
BEGIN:
    IF CQ.REAR = CQ.FRONT THEN
        RETURN TRUE
    ELSE
        RETURN FALSE
END;
```

For all the above ALGORITHM

Time Complexity: ϴ (1)

Space Complexity: ϴ (1)

Experiment.68

OBJECT: **Program for Array implementation of Double Ended Queue**

- **ALGORITHM INSREAR (DQUEUE DQ)**

```
BEGIN:

        IF(DQ.REAR==SIZE-1) THEN
            Write(queue overflow)
            Exit(1)

        ELSE
                REAR=REAR+1
                DQ.REAR=item

        IF(REAR=0) THEN:
                REAR=REAR+1

        IF(FRONT=0) THEN:
                FRONT= FRONT +1


END;
```

- **ALGORITHM INSFRONT(DQUEUE DQ)**

```
BEGIN:

        IF(FRONT<=1) THEN:
            Write(cannot add item at FRONT end)
            Exit(1)

        ELSE
                FRONT=FRONT-1;
                DQ.FRONT=item
END;
```

- **ALGORITHM DELFRONT(DQUEUE DQ)**

```
BEGIN:

        IF(FRONT=0) THEN:
          Write(queue underflow)
          Exit(1)

        ELSE
```

```
                    Item=DQ.FRONT
                    Write(item)

            IF(FRONT=REAR) THEN:
                    FRONT=0
                    REAR=0
            ELSE
                    FRONT=FRONT+!
    RETURN item
    END;
```

• **ALGORITHM DELREAR(DQUEUE DQ)**

```
    BEGIN:

            IF(REAR=0) THEN:
                Write(cannot delete value at REAR end)
                Exit(1)
            ELSE
                    Item=DQ.REAR
                    Write(item)

                    IF(FRONT=REAR) THEN:
                            FRONT=0
                            REAR=0
                    ELSE
                            REAR=REAR-1
            RETURN item


    END;
```

For all the above ALGORITHM
Time Complexity: ϴ (1)
Space Complexity: ϴ (1)

Experiment. 69

OBJECT: **Program for Array implementation of priority Queue**

- **ALGORITHM ARRAY INSERTION**(pq[],i,key,:*N)

```
BEGIN:
        For j=*N-1to i   STEP-1   Do
                pq[j+1]=pq[j]
        pq[i]=key
        *N=*N+1
END;
```

TIME COMPLEXITY: **Ω** (1), O(N)
SPACE COMPLEXITY: **Θ** (1)

- **ALGORITHM ARRAYDELETE**(pq[],*N,i)

```
BEGIN: y=pq[i]
        For j=i+1 to *N-1 Do
                pq[j-1]=pq[j]
        *N=*N-1
        RETURN y
END;
```

TIME COMPLEXITY: **Ω** (1), O(N)
SPACE COMPLEXITY: **Θ** (1)

- **ALGORITHM ARRAYINSERTION**(pq[],*N,key)

```
BEGIN:
         i=0
        WHILE i<=*N && key>pq[i]
             i=i+1
        Arrayinsertion(pq,i,key,N)
END;
```

- **ALGORITHM REMOVE** (pq[],*N)

```
BEGIN:
        x=Arraydelete(pq,N,1)
        RETURN x
END;
```

<u>Experiment 70</u>

<u>Object:</u> Program for 1-D array implementation of Upper Traingular Sparse Matrix **Sparse Matrix**

ALGORITHM UpperTriangularSparse (Key, i,j,A[ ],N)

BEGIN:

```
K = (i–1)*((2*N– (i–2))/2 + j – N + 1
A[K] = key
```

END;

Time Complexity: Ɵ (1)

Space Complexity: Ɵ (1)

Experiment 71

Object: Program for 1-D array implementation of Lower Traingular Sparse Matrix Sparse Matrix

ALGORITHM LowerTriangularSparse (Key, i,j,A[ ])

BEGIN:

K = i*(i–1)/2 + (j–1) + 1
A[K] = key

END;

Time Complexity: Θ (1)
Space Complexity: Θ (1)

Experiment 72

Object: Program for 1-D array implementation of Tridiagonal Sparse Matrix Sparse Matrix

ALGORITHM TridiagonalSparse (Key, i,j, A[ ])

BEGIN:

```
    K = 3 * (i – 2) + 2 + (j – i + 1 ) + 1
    A[K] = key
```

END;

Time Complexity: Θ (1)

Space Complexity: Θ (1)

Experiment 73

Program for Vector Representation of General Sparse Matrix   Sparse Matrix

```
Struct Sparse
{
int row;
int column;
int data;
};

Algorithm VectorSparse (S [ ], NoOfNonZeroEle)
BEGIN:
        FOR i=0 TO NoOfNonZeroEle – 1 DO
                READ(rowNo,columnNo,dataElement)
                S[i].row=rowNo
                S[i].column=columnNo
                S[i].data=dataElement
END;
```

Time Complexity: Θ (1) for each storage element

Space Complexity: Θ (1) for each storage element

Experiment 74

Object: Program For Linked List Implementation of General Sparse Matrix Sparse Matrix

```
Struct Node
{
int row;
int column;
int data;
struct node *Next;
};

Algorithm LinkedListSparse (NoOfNonZeroEle)
BEGIN:
        START=NULL
        P=START
        FOR i=0 TO NoOfNonZeroEle – 1 DO
                READ(rowNo,columnNo,dataElement)
                q=GetNode()
                q→row=rowNo
                q→column=columnNo
                q→data=dataElement
                q→Next=NULL
                p->Next=q
                p=q
END;
```

Time Complexity: Θ (1) for each storage element
Space Complexity: Θ (1) for each storage element

Experiment 75

Program for Addition of two sparse Matrices   Sparse Matrix

```
ALGORITHM AddSparseMarix(s1[ ],M,s2[ ],N)
BEGIN:
    s3[ ]
    i=1
    j=1
    k=1
    WHILE i<=M AND j<=N DO
        IF s1[i].row == s2[j].row THEN
            IF s1.column[i] == s2[j].column THEN
                s3[k].data=s1[i].data+s2[j].data
                s3[k].row=s1[i].row
                s3[k].column=s1[i].column
                i++     j++     k++
            ELSE
                IF s1.column[i] < s2[j].column THEN
                    i++
                ELSE
                    j++
        ELSE
            IF s1[i].row < s2[j].row THEN
                i++
            ELSE
                j++
    WHILE i<=M DO
        s3[k].data=s1[i].data
        s3[k].row=s1[i].row
        s3[k].column=s1[i].column
        i++     k++
    WHILE j<=N DO
        s3[k].data=s2[j].data
        s3[k].row=s2[j].row
        s3[k].column=s2[j].column
        j++     k++
    RETURN s3
END;
```

Experiment 76

OBJECT-Program for Linear Linked List primitive operations

DOMAIN-Linked List

ALGORITHM:

- **ALGORITHM:** **InsertBeginning(START,key)**

```
BEGIN:      p=getnode()
            p->info=key
            p->next=START
            START=p
END;
```

TIME COMPLEXITY: **Ɵ** (1)

SAPCE COMPLEXITY: **Ɵ** (1)

- **ALGORITHM:** **InsertAfter(p,key)**

```
BEGIN:      q=getnode()
            q->info=key
            q->next=p->next
            p->next=q
END;
```

TIME COMPLEXITY: **Ɵ** (1)

SAPCE COMPLEXITY:**Ɵ** (1)

- **ALGORITHM:** **InsertEnd(START,key)**

```
BEGIN:      IF START==NULL THEN
                InsertBeginning (START,key)  p=START

            ELSE
                P=START
                WHILE(p!=NULL) DO
                    q=getnode()
                    q->info=key
                    q->next=NULL
                    p->next=q
END;
```

TIME COMPLEXITY: **Ɵ** (N)

SAPCE COMPLEXITY:**Ɵ** (1)

- **ALGORITHM:** **DelBeg(START)**

```
BEGIN:      IF START==NULL THEN
                Write("void deletion")
                Exit(1)
            ELSE
                p=START
                START=p->next
            x=p->info
            RETURN x
```

END;

TIME COMPLEXITY: **ϴ** (1)

SAPCE COMPLEXITY:**ϴ** (1)

- **ALGORITHM: DelEnd(START)**

BEGIN:

```
p=START
q=NULL
WHILE(next(p)!=NULL)
        q=p
        p=p->next
q->next=NULL
x=p->info
freenode(p)
RETURN x
```

END;

TIME COMPLEXITY: **ϴ** (1)

SAPCE COMPLEXITY:**ϴ** (1)

Experiment 77

OBJECT-Program for pair wise swapping of elements of Linked List
DOMAIN-Linked List
ALGORITHM:
ALGORITHM **PairWiseSwap(START**)
BEGIN:

```
p=START
WHILE(p!=NULL&&p->next!=NULL) DO
    swap(p->info,p->next->info)
    p=p->next->next
    traverse(START)
```

END;
TIME COMPLEXITY:**ϴ** (N)
SPACE COMPLEXITY:**ϴ** (1)

Experiment 78

OBJECT-Program for printing the elements of Linked List in Reverse Order
DOMAIN-Linked List
ALGORITHM:
ALGORITHM ReverseTraversal(START)
BEGIN:

```
    IF p!=NULL THEN
        ReverseTraversal (p→next)
        WRITE (p→data)
```

END;

TIME COMPLEXITY:**ϴ** (N)
SPACE COMPLEXITY:**ϴ** (N)

Experiment 79

OBJECT-Program for reversing contents of Linear Linked List
DOMAIN: Linked List
ALGORITHM:

ALGORITHM **Reverse Linked LIST(START)**
BEGIN:

```
current=START
 previous=START->next
 WHILE(current!=NULL)
          q=current->next
          current->next=previous
           previous=current
           current=previous
           pevious=q
START->next=NULL
 RETURN START
```

End;

TIME COMPLEXITY: **ϴ** (N)
SPACE COMPLEXITY: **ϴ** (1)

Experiment 80

OBJECT- Program for concatenation of linear linked list.

DOMAIN- Linked LIst

ALGORITHM **Concatenate(START1 ,START2)**

BEGIN:

```
IF START1==null THEN
   RETURN (START2)
ELSE
          IF START2==null THEN
                    RETURN (START1)
          ELSE
                    p=START1
                    WHILE (p->next != null) do
                              p=p->next
p->next=START2
RETURN (START1)
```

END;

TIME COMPLEXITY: **Ө** (N): N is the length of Linked List 1
SPACE COMPLEXITY: **Ө** (1)

Experiment 81

OBJECT:Program for Creation of Ascending Order Linked List
DOMAIN:Linked List
ALGORITHM:

```
ALGORITHM INSERT(START,x)
BEGIN:
        P=NULL
         q = START
                WHILE( q->next != NULL and q->data <= data )
                        P=q
                        q=q->next
                IF p==NULL THEN
                        InsBeg(START,x)
                ELSE
                        InsAft(p, x)
END;
```

TIME COMPLEXITY: **Ω** (1), O(N)
SPACE COMPLEXITY: **Ɵ** (1)

Experiment 82:

Object: Program for merging two Linked Lists

DOMAIN:Linked List

ALGORITHM:

```
ALGORITHM Merge(list1,list2)
BEGIN:
        p=list1
        q=list2
        WHILE(p!=NULL&&q!=NULL) DO
                IF(info(p)<info(q)) THEN
                        insertend(list3,info(p))
                        p=next(p)
                ELSE
                        insertend(list3,info(q))
                        q=next(q)
        WHILE(p!=NULL) DO
                insertend(list3,info(p))
                p=next(p)
        WHILE(q!=NULL) DO
                insertend(list3,info(q))
                q=next(q)
RETURN list3
END;
```

Time complexity:O(M+N)
Space complexity:O(M+N)

Experiment. 83:

Object: Program for Union of two Linked Lists
DOMAIN:Linked List
ALGORITHM:

```
ALGORITHM Union(START1,START2)
BEGIN:
        START3=NULL
        p=START 1
        q=START2
        WHILE(p!=NULL&&q!=NULL) DO
                IF(info(p)==info(q)) THEN
                        p=next(p)
                        q=next(q)
                ELSE
                        IF(info(p)<info(q)) THEN
                                insertend(START3,info(p))
                                p=next(p)
                        ELSE
                                insertend(START3,info(p))
                                q=next(q)
        WHILE(p!=NULL) DO
                insertend(START3,info(p))
                p=next(p)
        WHILE(q!=NULL)
                insertend(START3,info(q))
                q=next(q)
        RETURN START3
END;
```

Time complexity:O(M+N)
Space complexity:O(M+N)

Experiment. 84

Object: ***Program for finding Intersection of two linked list (consider lists as sets)***
DOMAIN:Linked List

```
ALGORITHM Intersection(list1,list2)
BEGIN:
        list3=NULL
        p=list1
        q=list2
        WHILE(p!=NULL&&q!=NULL) DO
                IF(info(p) == info(q)) THEN
                        insend(list3,info(p))
                        p=next(p)
                        q=next(q)
                 ELSE
                        IF(info(p)<info(q))  THEN
                                p=next(p)
                        ELSE
                                q=next(q)
END:
```

Time Complexity:O(M+N)
Space Complexity:O(N): N is the length of first Linked List

Experiment. 85

Object: ***Program for finding difference of two linked list (consider lists as sets)***
DOMAIN:Linked List
This finds A – B (A is the first Linked List and B the Second one)

ALGORITHM Difference(list1,list2)
BEGIN:

```
    list3=NULL
    p=list1
    q=list2
    WHILE(p!=NULL&&q!=NULL) DO
        IF(info(p) == info(q)) THEN
            p=next(p)
            q=next(q)
        ELSE
            IF(info(p)<info(q))  THEN
                insend(list3,info(p))
                p=next(p)
            ELSE
                q=next(q)
    WHILE(p!=NULL) DO
        insertend(START3,info(p))
        p=next(p)

    RETURN list3
```

END:

Time Complexity:O(M+N)
Space Complexity:O(N): N is the length of first Linked List

Experiment.86

Object:program for sorting of Linked List.
DOMAIN:Linked List

```
ALGORITHM: SortingLinked List(START)
Begin:  sortingLinked List(START)
                p=START
                q=NULL
                temp=0
    WHILE(p->next!=q) DO
        IF(p->info>p->next->info)THEN
        temp=p->info
        p->info=p->next->info
        p->next->info=temp
        p=p->next
        q=p
END;
```

Space Complexity: $\Theta(1)$
Time complexity: $\Theta(n)$

Experiment. 87

Object: Program for splitting a Linked List.

DOMAIN:Linked List

```
ALGORITHM: split(START1)
   p=START1
   START2=NULL
   i=1,j=1
   WHILE(p->next!=NULL) DO
      i=i+1
      p=p->next
   p=START1
   WHILE(p!=NULL)
           IF(j==i/2) THEN
            START2=p->next
      p->next=NULL
     BREAK
     ELSE
            j=j+1
           p=p->next
```

TIME COMPLEXITY: **ϴ** (N)

SPACE COMPLEXITY: **ϴ** (1)

<u>**Experiment.88**</u>

Object: To Detect if there is any cycle in the linked list. (use two pointers, once moves at a speed of one node, other moves at a speed of two nodes. If they collide with each other it means there is a cycle.

DOMAIN: Linked List

ALGORITHM:

```
ALGORITHM: CycleDetection(START)
Begin:
        P=START
        Q=START
        WHILE(1) DO
                IF p→Next==NULL THEN
                        RETURN FALSE
                ELSE
                        P=p→Next
                IF q→Next==NULL THEN
                        RETURN FALSE
                ELSE
                        IF q→Next→Next==NULL THEN
                                RETURN FALSE
                        ELSE
                                q=q→Next→Next
                IF p==q THEN
                        RETURN TRUE
END;
```

Space Complexity: Θ(N)
Time complexity: Θ(1)

Experiment.89

Object: Program for polynomial addition using Linked List.
DOMAIN: Linked List

ALGORITHM:

```
ALGORITHM: PolynomialAdditionLinked List(poly1,poly2)
Begin:          poly3=NULL
                p=poly1
                q=poly2
                WHILE(p!=NULL AND q!=NULL) DO
                IF  p->Exp==q->Exp THEN
                InsEnd(poly3,p->coeffi+q->coeffi,p->Exp)
                p=p->next
                q=q->next
        ELSE
                IF  p->Exp>q->Exp THEN
                InsEnd(poly3,p->coeffi,p->Exp)
                p=p->next
        ELSE
                InsEnd(poly3,q->coeffi,q->Exp)
                q=q->next
        WHILE p!=NULL DO
                InsEnd(poly3,p->coeffi,p->Exp)
                p=p->next
        WHILE q!=NULL DO
                InsEnd(poly3,q->coeffi,p->Exp)
                q=q->next

        RETURN ploy3
END;
```

Space Complexity: ϴ(m+n)
Time complexity: ϴ(m+n)

Experiment 90

OBJECT:Program for primitive operations of Circular Linked List
DOMAIN:Linked List
ALGORITHM:

ALGORITHM InsertBeginning(START, key)

BEGIN:

p=GetNode()

p→Info=key

p→Next=START

START=p

END;

Time Complexity:ϴ(1)

The number executed statements are 4 in the above algorithm.

Space Complexity:ϴ(1)

An extra variable p is used. Also, the space is allocated in the memory to the new node.

ALGORITHM InsertAfter(p, key)

BEGIN:

q=GetNode()

q→ Info=key

q→Next=p→ Next

p→Next=q

END;

Time Complexity:ϴ(1)

The number executed statements are 4 in the above algorithm.

Space Complexity:ϴ(1)

An extra variable **q** is used. Also, the space is allocated in the memory to the new node.

ALGORITHMInsertEnd(START, key)

BEGIN:

p=START

```
    WHILE p!=NULL DO
        p=p→ Next

    q = GetNode()
    q→Info=key
    q→ Next=NULL
    p→ Next=q
END;
```

Time Complexity:ϴ(N)

The while loop runs until the end of the linked list is reached. Hence, for a linked list of N items, N+5 statements will be executed

Space Complexity:ϴ(1)

Extra variables used here are **p** and **q.** Also, the space is allocated in the memory to the new node

```
ALGORITHM DelBeg(START)
BEGIN:
    p=START
    START=START→Next
    x=p → Info
    FreeNode(p)
    RETURN x
END;
```

Time Complexity:ϴ(1)

The above algorithm executes four statements unconditionally

Space Complexity:ϴ(1)

Two extra variables, p & x are used

```
ALGORITHM DeleteEnd(START)
BEGIN:
    p=START
    q=NULL
```

```
    WHILE p→ Next!=NULL DO
        q=p
        p=p→ Next
    q→ Next=NULL
    x=p→ Info
    FreeNode(p)
    RETURN x
END;
```

Time Complexity:ϴ(N)

The while loop executes two statements repetitively until the last element is reached. Also, five more statements are executed outside the loop

Space Complexity:ϴ(1)

Three extra variables namely p, q & x are utilized

ALGORITHM DeleteAft(p)

```
BEGIN:
    IF p = = NULL OR p→ Next = = NULL THEN
        WRITE ("Void Deletion")
        EXIT(1)
    q=p→ Next
    x=q→ Info
    p→ Next=q→ Next
    FreeNode(q)
    RETURN x
END;
```

Time Complexity:ϴ(1)

Here, statements are executed

Space Complexity:ϴ(1)

Two extra variables q & x are used apart from the parameters passed

ALGORITHM Traverse(START)

BEGIN:

```
    p=START
    WHILE p!=NULL D
        WRITE p → Info
        p=p → Next
END;
```

Time Complexity:Θ(N)

It takes 2n statement executions by the while loop where n elements are present in the list. Also, one more statement is executed to initialize p

Space Complexity:Θ(1)

p is taken as an extra variable

Experiment 91

OBJECT:Program for Cocatenation of Circular Linked List
DOMAIN:Linked List
ALGORITHM:
ALGORITHM ConcatList(cList1, cList2)
BEGIN:

```
IF START1==null THEN
        RETURN (START2)
ELSE
        IF START2==null THEN
                RETURN (START1)
        ELSE
                p=START1->Next
                q=START2->Next
                START1->Next=q
                START2->Next=p
        RETURN (START2)
```

END;

Space Complexity: Θ(1)
Time complexity: Θ(1)

Experiment 92

OBJECT:Program for primitive operations of Doubly Linked List
DOMAIN:Linked List
ALGORITHM:

- **ALGORITHM InsertBeg(dSTART,key)**

```
BEGIN:
        p=getnode()
        info(p)=key
        left(p)=NULL
        right(p)=dSTART
        IF(dSTART!=NULL) THEN
                left(dSTART)=p
                dSTART=p
END;
```

TIME COMPLEXITY: **Θ** (1)
SPACE COMPLEXITY: **Θ** (1)

- **ALGORITHM Insertend(dSTART,key)**

```
BEGIN:
        p=dSTART
        q=getnode()
        info(q)=key
        IF(dSTART==NULL)THEN
                insertbeg(dSTART,key)
        ELSE
                WHILE(right(p)!=NULL)
                        p=right(p)
                        right(p)=q
                         right(q)=NULL
                         left(q)=p
END;
```

TIME COMPLEXITY: **Θ** (N)
SPACE COMPLEXITY: **Θ** (1)

- **ALGORITHM Insertleft(p,key)**

```
BEGIN:
        IF(left(p)==NULL) THEN
                insertbeg(p,key)
        ELSE
                 IF(left(p)!=NULL) THEN
                         q=getnode()
                                r=left(p)
                         info(q)=key
                         left(q)=r
                        right(r)=q
                        right(q)=p
                        left(p)=q
```

```
    ELSE
            insertbeg(p,key)
END;
```
TIME COMPLEXITY: **Θ** (1)
SPACE COMPLEXITY: **Θ** (1)

- **ALGORITHM Insertright(p,key)**

```
BEGIN:
        q=getnode()
        info(q)=key
        r=right(p)
        right(p)=q
        left(q)=p
        IF(right(p)!=NULL) THEN
                right(q)=r
                left(r)=q
        ELSE
                right(q)=NULL
END;
```
TIME COMPLEXITY: **Θ** (1)
SPACE COMPLEXITY: **Θ** (1)

- **ALGORITHM Delbeg(dSTART)**

```
BEGIN:
        IF(dSTART==NULL) THEN
                write("void deletion")
                exit(1)
        ELSE
                p=dSTART
                dSTART=right(dSTART)
                IF(right(p)!=NULL)THEN
                        left(dSTART)=NULL
                        x=info(p)
                freenode(p)
        RETURN x
```
TIME COMPLEXITY: **Θ** (1)
SPACE COMPLEXITY: **Θ** (1)

END;

- **ALGORITHM DelEnd(dSTART)**

```
BEGIN:
        IF(dSTART==NULL)
                write("void deletion")
                exit(1)
        ELSE
                p=dSTART
                q=NULL
                WHILE(right(p)!=NULL)DO
```

```
                    q=p
                    p=right(p)
                    IF(q==NULL) THEN
                dSTART=NULL
            ELSE
            right(q)=NULL
    x=info(p)
    freenode(p)
    RETURN x
```
END;
TIME COMPLEXITY: **Θ** (N)
SPACE COMPLEXITY: **Θ** (1)

- **ALGORITHM DelLeft(p)**

BEGIN:
```
    IF(p==NULL||left(p)==NULL) THEN
         write("void deletion")
          exit(1)
    ELSE
        q=left(p)
        r=left(q)
        right(r)=p
        right(p)=r
        x=info(q)
        freenode(q)
    RETURN x
```
END;
TIME COMPLEXITY: **Θ** (1)
SPACE COMPLEXITY: **Θ** (1)

- **ALGORITHM DelRight(p)**

BEGIN:
```
    IF(p==NULL||right(p)==NULL) THEN
         write("void deletion")
          exit(1)
    ELSE
        q=right(q)
        r=right(q)
        right(p)=r
        left(r)=p
        x=info(q)
        freenode(q)
    RETURN x
```
END;
TIME COMPLEXITY: **Θ** (1)
SPACE COMPLEXITY: **Θ** (1)

- **ALGORITHM Traverse(dSTART)**

```
BEGIN:
        p=dSTART
        WHILE(p!=NULL) DO
                write(info(p))
                p=right(p)
END;
```

TIME COMPLEXITY: **Ɵ** (1)

SPACE COMPLEXITY: **Ɵ** (1)

Experiment 93

OBJECT-Program for primitive operations of Circular Doubly Linked List
DOMAIN:Linked List

ALGORITHM:

- **ALGORITHM InsertEnd**

BEGIN:
```
    p=d.START
    q=getnode()
    IF(d.START==NULL) THEN
        insertbeg()
    ELSE
        right(q)=right(p)
        left(q)=p
        left(p)=q
        info(q)=key
        d.START=q
```
END;

TIME COMPLEXITY: **Ɵ** (1)
SPACE COMPLEXITY: **Ɵ** (1)

- **ALGORITHM DeleteBeginning**

BEGIN:
```
    IF(d.START=NULL) THEN
      write("void deletion")
    ELSE
      p=d.START
      q=right(p)
        left(p)=right(q)
       IF(right(q)!=q) THEN
            left(right(q))=p
            x=info(p)
        ELSE
            d.START=NULL
            x=info(p)
    RETURN x
    free q
```
END;

TIME COMPLEXITY: **Ɵ** (1)
SPACE COMPLEXITY: **Ɵ** (1)

- **ALGORITHM InsertLeft(p,key)**

BEGIN:
```
    IF(p==NULL) THEN

    ELSE
       IF(left(p)==NULL) THEN
          insertbeg()
```

```
                    ELSE
                                        q=getnode()
                                        left(q)=left(p)
                                        right(q)=p
                                        info(q)=key
                                        right(left(p))=q
```

END;

TIME COMPLEXITY: **Θ** (1)

SPACE COMPLEXITY: **Θ** (1)

- **ALGORITHM InsertRight(p,key)**

BEGIN:

```
          IF(p==NULL)
                    insertbeg()
          ELSE
                     q=getnode()
                      left(q)=p
                      right(q)=right(p)
                      left(right(p))=q
                              right(p)=q
```

END;

TIME COMPLEXITY: **Θ** (1)

SPACE COMPLEXITY: **Θ** (1)

- **ALGORITHM Traverse(cdSTART)**

BEGIN:

```
          p=cdSTART
          WHILE(p!=NULL) DO
                    write(info(p))
                    p=right(p)
```

END;

TIME COMPLEXITY: **Θ** (1)

SPACE COMPLEXITY: **Θ** (1)

Experiment 94

OBJECT:Program for linear linked list implementation of Linear Queue
DOMAIN:Linked List
ALGORITHM:

```
ALGORITHM  Initialize(FRONT,REAR)
BEGIN:
        REAR=NULL
        FRONT=NULL
END;
```
TIME COMPLEXITY: **Ɵ** (1)
SPACE COMPLEXITY: **Ɵ** (1)

```
ALGORITHM Empty(FRONT)
BEGIN:
        IF(FRONT==NULL) THEN
                RETURN TRUE
ELSE
        RETURN FALSE
END;
```
TIME COMPLEXITY: **Ɵ** (1)
SPACE COMPLEXITY: **Ɵ** (1)

```
ALGORITHM ENQUEUE(FRONT,REAR,x)
BEGIN:
        IF(REAR==NULL) THEN
                insertbeg(REAR,x)
                    FRONT=REAR
        ELSE
                insertsafter(REAR,x)
                    REAR=REAR(next)
 END;
```
TIME COMPLEXITY: **Ɵ** (1)
SPACE COMPLEXITY: **Ɵ** (1)

```
ALGORITHM Dequeue(FRONT,REAR)
BEGIN:
                IF(FRONT==NULL) THEN
                     write("void deletion")
                     exit(1)
                ELSE
                     x=delbeg(FRONT)
                     IF(FRONT==NULL)
                         REAR=NULL
                RETURN x
END;
```
TIME COMPLEXITY: **Ɵ** (1)

SPACE COMPLEXITY: **Ɵ** (1)

ALGORITHM Traverse(FRONT)
BEGIN:

```
        WHILE FRONT!=NULL
              write(FRONT(info) )
              FRONT=FRONT(next)
```

END;
TIME COMPLEXITY: **Ɵ** (1)
SPACE COMPLEXITY: **Ɵ** (1)

Experiment 95

OBJECT:Program for linear linked list implementation of Circular Queue
DOMAIN:Linked List
ALGORITHM:
ALGORITHM ENQUEUE(x)
BEGIN:

```
        q=getnode()
       IF(REAR==NULL) THEN
              q(next)=q
       ELSE
               q(next)=REAR(next)
              REAR(next)=q
        REAR=q
```

END;
TIME COMPLEXITY: **Θ** (1)
SPACE COMPLEXITY: **Θ** (1)

ALGORITHM Dequeue()
BEGIN:

```
              IF(REAR==NULL) THEN
                      write("void deletion")
                       exit(1)
              x=REAR.next.data
            IF REAR.next==REAR)
                REAR=NULL
            ELSE
               REAR.next=REAR.next.next
              RETURN x
```

END;
TIME COMPLEXITY: **Θ** (1)
SPACE COMPLEXITY: **Θ** (1)

Experiment 96

OBJECT:Program for linear linked list implementation of Priority Queue
DOMAIN:Linked List
ALGORITHM:

```
ALGORITHM INSERT(prn,x)
BEGIN:
        tmp->info = x
        tmp->prn = prn
        IF( FRONT == NULL or prn < FRONT->prn )
                tmp->next = FRONT
                FRONT = tmp
        ELSE
                  q = FRONT
                WHILE( q->next != NULL and q->next->prn <= prn )
                        q=q->next
                tmp->next = q->next
                q->next = tmp
END;
```

TIME COMPLEXITY: **Ω** (1), O(N)
SPACE COMPLEXITY: **Ө** (1)

```
ALGORITHM DEL()
BEGIN:
          IF(FRONT == NULL)
                write(Queue Underflow)
                   exit(1)
          ELSE
                tmp = FRONT;
                write(tmp->info);
                FRONT = FRONT->next;
                free(tmp);
END
```

TIME COMPLEXITY: **Ө** (1)
SPACE COMPLEXITY: **Ө** (1)

```
ALGORITHM DISPLAY()
BEGIN:
        ptr = FRONT;
        IF(FRONT == NULL)
                write(Queue is empty)
        ELSE
                  WHILE(ptr != NULL)
                        write(ptr->info)
                        ptr = ptr->next;
END;
```

TIME COMPLEXITY: **Ө** (N)

SPACE COMPLEXITY: **Ө** (1)

Experiment 97

OBJECT:Program for linked list implementation of stack
DOMAIN:Linked List

ALGORITHM:

ALGORITHM Initialize(FRONT,REAR)
BEGIN:
    top=NULL
END;
TIME COMPLEXITY: **Θ** (1)
SPACE COMPLEXITY: **Θ** (1)

ALGORITHM:
ALGORITHM Push(top,x)
BEGIN:

```
        insertbeg(top,x)
```

END;

ALGORITHM Pop(top)
BEGIN:

```
        IF(top==NULL) THEN
                write("underflow")
                 exit(1)
        ELSE
                x=delbeg(top)
        RETURN x
```

END;

ALGORITHM Empty(top)
BEGIN:

```
        IF(top==NULL) THEN
                RETURN TRUE
        ELSE
                RETURN FALSE
```

END;

ALGORITHM StackTop(top)
BEGIN:

```
        RETURN(info(top))
```

END;

Experiment 98

OBJECT:Program for recursive creation and traversal of Binary Tree and traversal
DOMAIN:Tree
ALGORITHM:

```
ALGORITHM PreorderTraversal(root)
BEGIN:
        IF root != NULL THEN
                WRITE(Data(root))
                PreorderTraversal(Left(root))
                PreorderTraversal(Right(root))
END;
```

TIME COMPLEXITY: **Θ** (N)
SPACE COMPLEXITY: **Θ** (LogN) for balanced trees

```
ALGORITHM PostorderTraversal(root)
BEGIN:
        IF root != NULL THEN
                PostorderTraversal(Left(root))
                PostorderTraversal(Right(root))
                WRITE(Data(root))
END;
```

TIME COMPLEXITY: **Θ** (N)
SPACE COMPLEXITY: **Θ** (LogN) for balanced trees

```
ALGORITHM InorderTraversal(root)
BEGIN:
        WHILE  root != NULL THEN
                InorderTraversal(Left(root))
                WRITE(Data(root))
                InorderTraversal(Right(root))
END;
```

TIME COMPLEXITY: **Θ** (N)
SPACE COMPLEXITY: **Θ** (LogN) for balanced trees

```
ALGORITHM CreateTree(tree)
BEGIN:
        WRITE("Whether Left of DATA(tree) exists (1/0)")
        READ(choice)
        IF (choice = = 1) THEN
                WRITE("Input the information of Left node")
                READ(x)
                p ← MakeNode(x)
                LEFT(tree)  ←  p
```

```
        CreateTree(p)

    WRITE("Whether Right of DATA(tree) exists (1/0)")
    READ(choice)

    IF (choice = = 1) THEN
        WRITE("Input the information of Right node")
        READ(x)
        p ← MakeNode(x)
        Right(tree)  ←  p
        CreateTree(p)
```

END;

Time Complexity: ϴ(N)

There are N elements for creating nodes.

Space Complexity: ϴ(N)

There are N nodes created

Experiment 99

Object: Program for creation of Binary Tree and finding its height
DOMAIN:Tree
ALGORITHM:

ALGORITHM HEIGHT(root)
BEGIN:

```
    IF root==NULL THEN
        RETURN 0
    ELSE
        IF Left(root)==NULL AND Right(root)==NULL
        RETURN 0
    ELSE
        RETURN 1+ Max(HEIGHT(Left(root)),Height(Right(root)))
```

END;

TIME COMPLEXITY: **Ɵ** (Log N) for balanced Trees. N for Skewed Trees
SPACE COMPLEXITY: **Ɵ** (1)

Experiment 100

Object: Program for finding count of nodes having 1 child (N_1 Nodes) in a Binary Tree
DOMAIN:Tree
ALGORITHM:

ALGORITHM CountOfN1(root)

BEGIN:

```
    IF tree = = NULL THEN
        RETURN 0
    ELSE
        IF LEFT(tree)  = =  NULL && Right(tree)  = =  NULL THEN
            RETURN 0
        ELSE
            IF LEFT(tree) != NULL && Right(tree) != NULL THEN
                RETURN CountOfN1(LEFT(tree))+CountOfN1(Right(tree));
            ELSE
                RETURN 1+CountOfN1(LEFT(tree))+CountOfN1(Right(tree))
```

Experiment 101

Object: Program for finding count of nodes having 2 child (N_1 Nodes) in a Binary Tree

DOMAIN:Tree

ALGORITHM:

```
ALGORITHM  CountOfN2(tree)
BEGIN:
    IF tree = = NULL THEN
        RETURN 0
    ELSE
        IF LEFT(tree)  = =  NULL && Right(tree)  = =  NULL THEN
            RETURN 0
        ELSE
            IF LEFT(tree) != NULL && Right(tree) != NULL THEN
                RETURN 1+ CountOfN1(LEFT(tree))+CountOfN1(Right(tree))
            ELSE
                RETURN CountOfN1(LEFT(tree))+CountOfN1(Right(tree))
```

Experiment 102

Object: Program for finding count of Leaf Nodes (N_0 Nodes) in a Binary Tree

DOMAIN:Tree

ALGORITHM:

Finding count of nodes having 0 children in Binary Tree

ALGORITHM CountOfN0(root)

BEGIN:

```
    IF tree = = NULL THEN
        RETURN 0
    ELSE
        IF LEFT(tree)  = =  NULL && Right(tree)  = =  NULL THEN
            RETURN 1
        ELSE
            RETURN CountOfN0(LEFT(tree))+CountOfN0(Right(tree));
```

Time Complexity: Θ(N)

Need to reach all the node once for computing height

Space Complexity: Ω(Log $_2$N), O(N)

If the tree is balanced, Call stack will have log $_2$n+1 entries to the maximum at any moment in case the tree is balanced. If the tree is skewed then the call stack can be grow up to n activation records

Experiment 103

Object: Program for Finding if the Binary Tree is Complete

DOMAIN:Tree

ALGORITHM:

ALGORITHM isComplete (root, index, number_nodes)

BEGIN:

IF tree = = NULL THEN

RETURN True;

// If index assigned to current node is more than

// number of nodes in tree, then tree is not complete

IF index >= number_nodes THEN

RETURN False;

// Recursive for Left and Right subtrees

RETURN (isComplete(LEFT(root), 2*index + 1, number_nodes) AND isComplete(Right(root), 2*index + 2, number_nodes))

END;

Time Complexity: $\Theta(N)$

Need to reach all the node once for computing height

Space Complexity: $\Omega(\log_2 N)$, O(N)

If the tree is balanced, Call stack will have $\log_2 n+1$ entries to the maximum at any moment in case the tree is balanced. If the tree is skewed then the call stack can be grow upto n activation records

Experiment 104

Object: Program for Level Order Traversal of Binary Tree
DOMAIN:Tree
ALGORITHM:

ALGORITHM LevelOrderTraversal(root)

BEGIN:

```
    Queue Q
    Initialize(Q)
    Enqueue(root)
    WHILE !Empty(Q) DO
        x ← DeQueue(Q)
        WRITE(Data(x))
        IF LEFT(x) !=NULL THEN
            EnQueue(x)
        IF Right(x) !=NULL THEN
            EnQueue(x)
```

END;

Time Complexity: ϴ(N)

Each node is inserted in the queue once and deleted from queue once. Total of 2*N operations of constant time.

Space Complexity: ϴ(N)

Queue size is N

Experiment 105

Object:Program for finding balancing factor of a node

DOMAIN:Tree

ALGORITHM BalanceFactor(N)

Finding Balancing factor of a node

LH represents the Left height and RH represents right height. LH – RH is the balance Factor of the given node.

ALGORITHM BalanceFactor(N)

BEGIN:

```
    IF Left(N)!=NULL
        LH ← 1+ h(Left(N))
    ELSE
        LH ← 0

    IF Right(N)!=NULL
            RH ← 1 + h(Right(N))
    ELSE
            RH ← 0

    RETURN(LH – RH)
```

END;

Time Complexity: ϴ(N)

Need to reach all the node once for computing height

Space Complexity: $\Omega(\log_2 N)$, O(N)

If the tree is balanced, Call stack will have $\log_2 n+1$ entries to the maximum at any moment in case the tree is balanced. If the tree is skewed then the call stack can be grow upto n activation records

Experiment 106

Object:Program for BST Insertion, traversal, Minimum, maximum and Successor operations

DOMAIN: BST

ALGORITHM:

ALGORITHM BST Minimum(Tree)

BEGIN:

```
    P = Tree
    WHILE Left(P) != NULL Do
        P = Left(P)
        RETURN P
```

END;

Time Complexity: Θ(LogN) if Tree is balanced

Space Complexity: Θ(1)

ALGORITHM BST Maximum(Tree)

BEGIN:

```
    P = Tree
    WHILE Right(P) != NULL Do
        P = Right(P)
        RETURN P
```

END;

Time Complexity: Θ(LogN) if Tree is balanced

Space Complexity: Θ(1)

ALGORITHM BSTSearch(Tree,key)

BEGIN:

```
    P = Tree
    WHILE P!=NULL
        IF key == Data(P) THEN
            RETURN P
        ELSE

            IF key &lt; Data(P) THEN
            P = Left(P)
        ELSE
            P = Right(P)
```

END;

Time Complexity: Θ(LogN) if Tree is balanced

Space Complexity: Θ(1)

ALGORITHM BSTInsert(Tree,key)
BEGIN:

```
    P = Tree
    Q = Null
    R = MakeNode(key)
    WHILE P!=NULL Do
        IF key &lt; Data(P)
            Q = P
            P = Left(P)
        ELSE
            Q = P
            P = Right(P)
        IF key &lt; Data(Q) THEN
            Left(Q) = R
            Father(R) = Q
        ELSE
            Right(Q) = R
            Father(R) = Q
```

END;

Time Complexity: Θ(LogN) if Tree is balanced
Space Complexity: Θ(1)

ALGORITHM BSTSuccesor(P)
BEGIN:

```
    IF Right(P) != NULL THEN
        q = BSTMinimum(Right(P))
        RETURN q
    ELSE
        q = Father(P)
    WHILE q!=NULL AND Right(q)==P Do
        p=q
        q=Father(q)
    RETURN q
```

END;

Time Complexity: Θ(LogN) if Tree is balanced
Space Complexity: Θ(1)

ALGORITHM BSTPredecessor(P)
BEGIN:

```
    IF Left(P) != NULL THEN
        q = BSTMinimum(Left(P))
        RETURN q
```

```
ELSE
        q = Father(P)
WHILE q!=NULL AND Left(q)==P Do
        p=q
        q=Father(q)

RETURN q
```

END;

Time Complexity: Θ(LogN) if Tree is balanced
Space Complexity: Θ(1)

Experiment 107

Object:Program for BST Deletion
DOMAIN: BST
ALGORITHM:
ALGORITHM BSTDelete(p)
BEGIN:

```
IF Left(p)==NULL AND RIGHT(q)==NULL THEN
        IF IsLeft(p) THEN
                Left(Father(p))=NULL
        ELSE
                Right(Father(p))=NULL
        X=Data(p)
        FreeNode(p)
        RETURN X
ELSE
        IF Left(p) == NULL THEN
                q=Right(p)
                IF IsLeft(p) THEN
                        Left(Father(p))=q
                        Father(q)=Father(p)
                ELSE
                        Right(Father(p))=q
                        Father(q)=Father(p)
        ELSE
                IF Right(p) == NULL THEN
                                q=Right(p)
                        IF IsLeft(p) THEN
                                Left(Father(p))=q
                                Father(q)=Father(p)
                        ELSE
                                Right(Father(p))=q
                                Father(q)=Father(p)
                        X=Data(p)
                        FreeNode(p)
                        RETURN X
                ELSE
                        q=BSTSuccessor(p)
                        y=BSTDelete(q)
                        x=Data(p)
                        Data(p)=y
                        RETURN
Right(Q) = R
Father(R) = Q
```

END;

Time Complexity: ϴ(LogN) if Tree is balanced
Space Complexity: ϴ(1)

Experiment 110

Object:Program for insertion in AVL Tree,Rotations in AVL Tree & traversal operations
DOMAIN:AVL Tree
ALGORITHM:

ALGORITHM RotateRight (x)
BEGIN:
y=x→Left
z=y→Right
y→Right =z
x→Left =z
RETURN y
END;

Time Complexity: ϴ(1)
Space Complexity: ϴ(1)

ALGORITHM RotateLeft(x)
BEGIN:
y=x→Right
z=y→Left
y→Left=z
x→Right =z
RETURN y
END;

Time Complexity: ϴ(1)
Space Complexity: ϴ(1)

ALGORITHM LL(T)

BEGIN:

T=RotateRight(T)

RETURN T

END;

Time Complexity: ϴ(1)

Space Complexity: ϴ(1)

ALGORITHM RR(T)

BEGIN:

T=RotateLeft(T)

```
            RETURN T
END;
```

Time Complexity: Θ(1)

Space Complexity: Θ(1)

ALGORITHM LR(T)

BEGIN:

```
            x= T→Left
            x=RotateLeft(x)
            T→Left=x
            T=RotateRight(T)
            RETURN T
END;
```

Time Complexity: Θ(1)

Space Complexity: Θ(1)

ALGORITHM RL(T)

BEGIN:

```
            x= T→Right
            x=RotateRight(x)
            T→Right =x
            T=RotateLeft(T)
            RETURN T
END;
```

Time Complexity: Θ(1)

Space Complexity: Θ(1)

ALGORITHM AVLInsertion(N, Key)

BEGIN:

```
      IF N = = NULL THEN
            N = MakeNode(Key)
```

```
    ELSE
        IF Key<N→DataTHEN
            N→Left = AVLInsertion(N→Left, Key)
            IF Balance(N)  = =  2 THEN
                IF Key<N→Left→Data
                    N = LL(N)
                ELSE
                    N = LR(N)
        ELSE
            N→Right = AVLInsertion(N→Right, Key)
            IF Balance(N)  = =  – 2 THEN
                IF Key > N→Left→Data
                    N = RR(N)
                ELSE
                    N = RL(N)

    N→ h=Height(N)
    RETURN N
END;
```

Time Complexity: Ω(Log 2N), O(N)

Space Complexity: Θ(1)

Experiment 111:

Object:Program for implementation of BFS

DOMAIN:Graph

ALGORITHM:

ALGORITHM BFS (G, s)

BEGIN:

QUEUE Q

Initialize (Q)

For all u ε V[G] DO

Π[u] ← Nil

Color[u] ← White

EnQueue (Q, s)

Color[s] ← Grey

d[s] ← 0

WHILE (!Empty(Q)) Do

u ← DeQueue (Q) Do

For each V ε Adj [u]

If Color[u] = = White

Color [u] ← Grey

EnQueue (Q, u)

d[v] ← d[u]+1

Π[u] ← u

Color[u] ← Black

WRITE (u)

END;

Time Complexity: ϴ(|V|+|E|)

Space Complexity: ϴ(|V|)

Experiment 112:

Object:Program for DFS on graph
DOMAIN:Graph
ALGORITHM:

ALGORITHM DFS (G)

BEGIN:

For all u ε V [G] Do

Color[u] ← White

Time ← 0

For all u ε V[u] Do

IF Color[u] ← White

DFS ← Visit (u)

END;

ALGORITHM DFS-VISIT (U)

BEGIN:

Color[u] ← Grey

S[u] ← Time+1

For all V ε Adj [U] Do

IF Color [V] ← White

Π[V] ← U

DFS ← Visit [V]

Color [U] ← Black

F[U] ← Time+1

Time ← Time+1

END;

Time Complexity: Ɵ(|V|+|E|)

Space Complexity: Ɵ(|V|)

Experiment 113

Object:Program for All pairs Shortest Path using Flloyd-Warshall ALGORITHM

DOMAIN:Graph

ALGORITHM:

ALGORITHM APSPFloydWarshall (W [ ] [ ], N)

BEGIN:

```
    FOR i ← 1 To N Do
        FOR i ← 1 To N DO
            IF W [i] [ j] = = 0
                IF i != j THEN
                    W [i] [j]= ∞
    FOR k ← 1 To N Do
        FOR i ← 1 To N DO
            FOR j ← 1 To N DO
                W [i] [j] ← Min (W [i] [j] , W [i] [k]+ W [k] [j])
```

END ;

Time Complexity: $\Theta(N^3)$

Space Complexity: $\Theta(1)$

Experiment 114

Object:Program for Dijikistra ALGORITHM
DOMAIN:Graph
ALGORITHM:

ALGORITHM SSSPDijkstra (G , W [ ] [ ], S)

BEGIN:

```
    Priority Queue PQ
    Initialize (PQ)
    For all U ∊ V [G] DO
        D [U] ← ∞
        Π[U] ← NIL
        PQ Insert (PQ , U)
        D [S] ← 0
    WHILE ! Empty (PQ) DO
        U ← ExtractMin(PQ)
        For All V ∊ Adj [U] AND V ∊ PQ
            IF d[V] <d[U] +W [U][V]
            d[V] ← d[U] +W [U][V]
            Π[V] ← U
```

END ;

Time Complexity: O(|E|+|V|Log |V|)

Space Complexity: ϴ(|V|)

Experiment 115

Object:Program for Warshall's ALGORITHM for Transitive Closure

DOMAIN:Graph

ALGORITHM TransitiveClosureWarshall (A[ ][ ], N)

BEGIN:

```
    FOR k ← 1 To N Do
        FOR J ← 1 To N DO
            FOR j ← 1 To N DO
                A [i] [j] ← A [i] [j] | (A [i] [k] & A [k] [j])
```

END ;

Time Complexity: ϴ(N^3)

Space Complexity: ϴ(1)

<u>Alternate Method</u>

ALGORITHM TransitiveClosure(A[ ][ ], N)

BEGIN:

```
    M[N][N] ← A
    B[N][N] ← A
    T[N][N] ← {0}
    For i ← 2 TO N DO
        B ← MatrixMultiply(B, A)
        M ← M+B
    FOR i ← 1 TO N DO
        FOR j ← 1 TO N DO
            IF M[i][j]!=0 THEN
                T[i][j] ← 1
    RETURN T
```

END;

Experiment 116

Object:Program for Prim's ALGORITHM for Minimal Spanning Tree

DOMAIN:Graph

ALGORITHM:

ALGORITHM MST Prims (G, W[ ][ ], R)

BEGIN:

```
    Priority Queue PQ
    For all U ϵ V[G] Do
        Key[U] ← ∞
        Π[U] ← NIL
        PQ Insert (PQ , U)
        Key [R] ← 0
    WHILE !Empty(PQ) Do
        U ← ExtarctMin (PQ)
        For all V ϵ Adj [U] AND V ϵ PQ DO
            IF W[U][V] < Key [V] DO
                Key [V] ← W [V][U]
                Π[U] ← U
```

END;

Time Complexity: O(|E|+|V|Log |V|)

Space Complexity: ϴ(|V|)

Experiment 117

Object: Program for Kruskal's ALGORITHM for Minimal Spanning Tree

DOMAIN:Graph

ALGORITHM:

ALGORITHM MST- Kruskal (G , W[ ][ ])

BEGIN:

    Sort all Edges In E According To their Weights
    Set of Edges of MST E'← ∅
    For all U ε V [G] DO
        MakeSet (U)
    For Each Edge(U , V) ε E [G] DO
        IF FindSet (U)! = FindSet (V)
            Union(U, V)
            Select(U, V) In E'
    RETURN E'

END ;

Time Complexity: O(|E|+|V|Log |V|)

Space Complexity: Ɵ(|V|)

Experiment 118

Object: Program for topological sorting of given graph

DOMAIN:Graph

ALGORITHM:

ALGORITHM Topological Sort (G , Indeg [ ])

BEGIN:

```
    Queue Q
    Initialize ( Q )
    FOR All U ε V [ G ]
        IF Indeg [ U]  = = 0
            Enqueue ( Q,  U)

    WHILE ! Empty (Q) DO
        U ← Dequeue ( Q )
        WRITE ( U )
    FOR All V ε Adj [U] DO
        IF Indeg [V]  = = 0
        Enqueue (Q,  U)
```

END;

Time Complexity: $\Theta(|V|+|E|)$

Space Complexity: $\Theta(|V|)$