DS Programs for Mid 1

1.Array Traversal

```c
//program for traversing an array

#include <stdio.h>

void traversearray(int n,int arr[]){
    printf("Enter elements");
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
}
void display(int n,int arr[30]){
    printf("Elements are: ")
    for(int i=0;i<n;i++){
        printf("%d",arr[i]);
    }
}

int main() {
    int n,arr[30];
    printf("Enter no of elements in array");
    scanf("%d",&n);
    traversearray(n, arr);
    display(n,arr);

}
```

2.Sum of array

```c
#include <stdio.h>

void readArray(int n,int arr[30]){
    for(int i = 0;i<n;i++){
        scanf("%d",&arr[i]);
    }
}

void display(int n,int arr[30]){
    for(int i = 0;i<n;i++){
        printf("%d",arr[i]);
```

```
    }
}

void sum(int n,int arr[30]){
    int sum=-0;
    for(int i =0;i<n;i++ ){
        sum = sum + arr[i];
    }
    printf("%d",sum);
}

int main(){
    int n,arr[30];
    puts("Enter no of elements");
    scanf("%d",&n);
    puts("Enter array elements: ");
    readArray(n,arr);
    puts("array elements are:\n");
    display(n,arr);
    puts("Sum is: ");
    sum(n,arr);
}
```

3.Sort array in ascending order

```
#include <stdio.h>

void swap(int *xp,int *yp){
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void readArray(int n,int arr[30]){
    puts("Enter array elements: ");
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
}

void displayNormalArr(int n, int arr[30]){
    puts("The array you entered is: \n");
    for(int i=0;i<n;i++){
        printf("%d",arr[i]);
```

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

void arrayAscending(int n,int arr[30]){
    int i,j,min;
    for(i=0;i<n-1;i++){
        min=i;
        for(j=i+1;j<n;j++){
            if(arr[j]<arr[min])
            min=j;
        }
        swap(&arr[min], &arr[i]);
    }
}

void displayAscending(int n, int arr[30]){
    puts("The sorted array is");
    for(int i=0;i<n;i++){
        printf("%d",arr[i]);
    }
}



int main(){
    int n,arr[30];
    puts("Enter the number of elements in array");
    scanf("%d",&n);
    readArray(n,arr);
    displayNormalArr(n,arr);
    arrayAscending(n,arr);
    displayAscending(n,arr);

}
```

4. Deleting an element from array

```
#include <stdio.h>

int *readArray(int n, int *arr){
    puts("Enter elements to be entered: ");
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]);
```

```
    }
    return arr;
}

void displayOld(int n, int *arr){
    puts("The entered array is: \n");
    for(int i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
}

int findEle(int n,int key, int arr[]){
    int i;
    for(i=0;i<n;i++){
        if(arr[i]==key){
            return i;
        }
    }
    return -1;
}

int *deleteEle(int n,int key,int arr[]){
    int pos = findEle(n,key,arr);
    if(pos==-1){
        puts("Element not found\n");
    }
    int i;
    for(i=pos;i<n-1;i++){
        arr[i]=arr[i+1];
    }
    return arr;
}

int main(){
    int n,arr[30],*war,key,*newarr;
    puts("Enter the size of array: ");
    scanf("%d",&n);
    war = readArray(n,arr);
    displayOld(n,war);
    puts("Enter element to be deleted: ");
    scanf("%d",&key);
    newarr=deleteEle(n,key,war);
    puts("The new array is: \n");
    for(int i;i<n-1;i++){
        printf("%d ",newarr[i]);
```

```
    }


}
```

5. Inserting an element into array

```c
#include <stdio.h>

void readArray(int n,int arr[30]){
    int i;
    puts("Enter array elements: \n");
    for(i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
}

void dispOrgArr(int n, int arr[30]){
    int i;
    puts("The array you entered is: ");
    for(i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
}

void insertEle(int n,int arr[30]){
    int ele,pos,i;
    puts("\nEnter the number that you want to insert: ");
    scanf("%d",&ele);
    puts("Enter the position: ");
    scanf("%d",&pos);

    n++;

    for(i=n-1;i>=pos;i--){
        arr[i]=arr[i-1];
    }

    arr[pos-1] = ele;

    puts("The new array is: ");
    for(i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
```

```
}

int main(){
    int n,arr[30];
    puts("Enter the size of array");
    scanf("%d",&n);
    readArray(n,arr);
    dispOrgArr(n,arr);
    insertEle(n,arr);
}
```

6. Linear Search

```
#include <stdio.h>

int main(){
  int n,num[30],i,key;
  printf("Enter size of array");
  scanf("%d",&n);
  printf("Enter array elements: ");
  for (i=0;i<n;i++){
      scanf("%d",&num[i]);
  }
  printf("Enter key element: ");
  scanf("%d",&key);
  for(i=0;i<n;i++){
      if (num[i]==key){
          printf("Key %d is found at %d position",key,i);
      }
      if(i==n){
          printf("Key not found");
      }
  }
}
```

7. Binary Search

```
#include <stdio.h>

int main(){
    int mid,a[10],n,i,low,high,key,flag=0;
    puts("Enter the size of array: ");
    scanf("%d",&n);
    puts("Enter elements into array with ascending order: ");
```

```
    for(i=0;i<n;i++){
        scanf("%d",&a[i]);
    }
    printf("Enter the element to search: ");
    scanf("%d",&key);
    low=0,high=n-1;
    while(low<=high){
        mid=(low+high)/2;
        if(key==a[mid]){
            flag=1;
            break;
        }
        else if(key<a[mid]){
            high=mid-1;
        }
        else{
            low=mid+1;
        }
    }
    if(flag==1){
        printf("Key is found at %d",mid+1);
    }
    else{
        printf("Key element not found in array\n");
    }
}
```

8. Sparse Matrix

```
#include <stdio.h>
int main(){
    int m,n,i,j,mat[30][30];
    printf("Enter the size of matrix: ");
    scanf("%d %d",&m,&n);
    printf("Enter elements of matrix\n: ");
    for(i=0;i<m;i++){
        for(j=0;j<n;j++){
            scanf("%d",&mat[i][j]);
        }
    }
    int sparse[10][3];
    int k=0;
    for(i=0;i<m;i++){
        for(j=0;j<n;j++){
```

```
            if(mat[i][j]!=0){
                sparse[k][0]=i;
                sparse[k][1]=j;
                sparse[k][2]=mat[i][j];
                k++;
            }
        }
    }
    printf("Sparse form: ");
    for(i=0;i<k;i++){
        for(j=0;j<3;j++){
            printf("%d\t",sparse[i][j]);
        }
        printf("\n");
    }
}
```

9. Creating 2d array using malloc

```
#include<stdio.h>
#include <stdlib.h>
int main(){
    int *parr,m,n,i;
    printf("Enter no of elements in array: ");
    scanf("%d",&n);
    parr=(int*)malloc(n*sizeof(int));

    printf("Enter the new no of elements in array: ");
    scanf("%d",&m);


    printf("Enter the array elements: \n");
    for(i=0;i<m;i++){
        scanf("%d",parr+i);
    }
    printf("The entered array elements are: ");
    for(i=0;i<m;i++){
        printf("%d ",*(parr+i));
    }
    free(parr);


}
```

10. Program to implement calloc and realloc

```c
#include<stdio.h>
#include <stdlib.h>
int main(){
    int *parr,*newparr,m,n,i;
    printf("Enter no of elements in array: ");
    scanf("%d",&n);
    parr=(int*)calloc(n,sizeof(int));
    if(parr==NULL){
        printf("Memory allocation failed");
        exit(0);
    }
    printf("Enter the new no of elements in array: ");
    scanf("%d",&m);
    newparr=(int*)realloc(parr,m*sizeof(int));

    printf("Enter the array elements: \n");
    for(i=0;i<m;i++){
        scanf("%d",newparr+i);
    }
    printf("The entered array elements are: ");
    for(i=0;i<m;i++){
        printf("%d ",*(newparr+i));
    }
    free(parr);
    free(newparr);

}
```

11. Stacks, all operations

```c
#include <stdio.h>
#include<stdlib.h>
#define SIZE 5
int top = -1;
int arr[SIZE];
void push();
void pop();
void display();

int main() {

    int choice;
```

```
    int ch = 1;
    while (ch == 1) {
        puts("Stack operations");
        puts("1.Push");
        puts("2.Pop");
        puts("3.Display");
        puts("4.Exit");
        puts("Enter your choice: ");
        scanf("%d", &choice);

    if (choice == 1) {
        push();
    } else if (choice == 2) {
        pop();
    } else if (choice == 3) {
        display();
    } else {
        exit(0);
    }

    }
}

void push(){
    int x;
    if(top==SIZE-1){
        printf("Overflow");
    }
    else{
        printf("Enter element to be added to stack: ");
        scanf("%d",&x);
        top=top+1;
        arr[top]=x;
    }
}

void pop(){
    if(top==-1){
        printf("Underflow");
    }
    else{
        printf("Popped element: %d",arr[top]);
        top=top-1;
    }
}
```

```
void display(){
    if(top==-1){
        printf("Underflow");

    }
    else{
        printf("Elements in the stack: \n");
        for(int i=top;i>=0;i--){
            printf("%d",arr[i]);
        }
    }
}
```

12. Infix to postfix conversion using stacks

```
#include<stdio.h>
#include<ctype.h>
int stack[20];
int top = -1;
void push(int x){
    stack[++top]=x;

}
int pop(){
    return stack[top--];
}
int main(){
    char exp[20];
    char *e;
    int n1,n2,n3,num;
    printf("Enter the expression: ");
    scanf("%s",exp);
    e=exp;
    while(*e!='\0'){
        if(isdigit(*e)){
            num=*e-48;
            printf("num == %d",num);
            push(num);
        }
        else{
            n1=pop();
            n2=pop();
            switch(*e){
```

```
                case'+': n3=n1+n2; break;
                case'-': n3=n2-n1; break;
                case'*': n3=n1*n2; break;
                case'/':n3=n2/n1; break;
            }
            push(n3);
        }
        e++;
    }
    printf("The result of expression %s = %d\n",exp,pop());
}
```

13. Tower of Hanoi recursion

```
#include <stdio.h>
void towerofhanoi(int,char,char,char);

int main(){
    int n;
    printf("Enter no of disks: ");
    scanf("%d",&n);
    towerofhanoi(n,'A','C','B');
}

void towerofhanoi(int n,char from,char to,char aux){
    if(n==1){
        printf("Move disk 1 from rod %c to rod %c\n",from,to);
        return;
    }
    towerofhanoi(n-1,from,aux,to);
    printf("Move disk %d from rod %c to rod %c\n",n,from,to);
    towerofhanoi(n-1,aux,to,from);
}
```

14. Fibonacci series recursion

```
#include <stdio.h>

int fibonacci(int);

int main() {
    int n;

    printf("Enter the number of terms\n");
```

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

    printf("Fibonacci Series: ");

    for (int i = 0; i < n; i++) {
        printf("%d ", fibonacci(i));
    }


}
int fibonacci(int n) {
    if(n == 0)
        return 0;
    else if(n == 1)
        return 1;
    else
        return (fibonacci(n-1) + fibonacci(n-2));
}
```

15. Factorial recursion

```
#include<stdio.h>
long int multiplyNumbers(int n);
int main() {
    int n;
    printf("Enter a positive integer: ");
    scanf("%d",&n);
    printf("Factorial of %d = %ld", n, multiplyNumbers(n));
    return 0;
}

long int multiplyNumbers(int n) {
    if (n>=1)
        return n*multiplyNumbers(n-1);
    else
        return 1;
}
```

16. Multi Stack

```
#include<stdio.h>
#define SIZEA 5
#define SIZEB 5
int topa=-1;
```

```
int topb=10;


void pusha(int *stack)
{
    int ele;

    if(topa>=(SIZEA-1))
    {   printf("\n\nStack OverFlow");
        return;
    }
    else
    {   printf("\n\nEnter The element To Push");
        scanf("%d",&ele);
        topa++;
        stack[topa]=ele;
    }}

void pushb(int *stack)
{
    int ele;

    if(topb<=(SIZEB))
    {
        printf("\n\nStack OverFlow");
        return;
    }
    else
    {
        printf("\n\nEnter The element To Push");
        scanf("%d",&ele);
        topa--;
        stack[topa]=ele;
    }
}

void popa(int *stack)
{   if(topa==-1)
    {
        printf("Stack A is Underflow");
        return;
    }
    else
    {
        printf("Item Popped from stack A is:%d\n",stack[topa]);
```

```
        topa--;
    }
}

void popb(int *stack)
{   if(topb==10)
    {printf("Stack B is Underflow");
        return;
    }
    else
    { printf("Item Popped from stack B is:%d\n",stack[topb]);
        topb++;
    }}

void displaya(int *stack)
{
    int i;
    if(topa==-1)
    {
        printf("Stack A is Empty");
        return;
    }
    else
    { for(i=topa;i>=0;i--)
        {printf("%d,",stack[i]);}
        printf("\n");
    }}

void displayb(int *stack)
{
    int i;
    if(topb==10)
    {printf("Stack Y is Empty");
        return;}
    else
    {for(i=topb;i<=9;i++)
        {
            printf("%d,",stack[i]);
        }
        printf("\n");
    }   }
/*End of display_y*/
/*Begin of main*/
int main()
```

```
{   int choice;
    char ch;
    int stack[SIZEA+SIZEB];
    do
    {   printf("1.Push A\n2.Push B\n");
        printf("3.Pop A\n4.Pop B\n");
        printf("5.Display A\n6.Display B\n");
        printf("7.Exit");
        printf("\nEnter Choice");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: pusha(stack);break;
            case 2: pushb(stack);break;

            case 3: popa(stack);break;
            case 4: popb(stack);break;

            case 5: displaya(stack);break;
            case 6: displayb(stack);break;
            case 7: break;
            default: printf("Wrong Option...");
        }
    }while(choice!=7);
}
```

17. Queue

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#define SIZE 5
void enqueue();
void dequeue();
void display();

int items[SIZE],front=-1,rear=-1;
int choice;

int main(){
    while(true) {
        puts("Queue Operations");
        puts("1.Enqueue");
        puts("2.Dequeue");
```

```
        puts("3.Display Queue");
        puts("4.Exit");
        puts("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                enqueue();
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
        }
    }
}
void enqueue(){
    if(rear==SIZE-1)
        printf("\nQueue is Full!!");
    else{
        printf("Enter a value: ");
        int value;
        scanf("%d",&value);
        if(front==-1)
            front=0;
        rear++;
        items[rear]=value;
        printf("Inserted element\n");
    }
}

void dequeue(){
    if(front==-1)
        printf("Queue is empty\n");
    else{
        printf("Deleted: %d\n",items[front]);
        front++;
        if(front>rear)
            front=rear=-1;
    }
}
void display(){
```

```
    if(rear==-1)
        printf("Queue is empty\n");
    else{
        int i;
        printf("Queue elements are:\n");
        for(i=front;i<=rear;i++){
            printf("%d ",items[i]);
        }
    }
    printf("\n");
}
```

18. Circular Queue

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

#define SIZE 5

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


int isFull() {
    if ((front == rear + 1) || (front == 0 && rear == SIZE - 1)) return 1;
    return 0;
}


int isEmpty() {
    if (front == -1) return 1;
    return 0;
}


void enqueue() {
    if (isFull())
        printf("\n Queue is full!! \n");
    else {
        int element;
        printf("Enter element: ");
        scanf("%d",&element);
        if (front == -1) front = 0;
        rear = (rear + 1) % SIZE;
```

```
        items[rear] = element;
        printf("\n Inserted -> %d", element);
    }
}

// Removing an element
int dequeue() {
    int element;
    if (isEmpty()) {
        printf("\n Queue is empty !! \n");
        return (-1);
    } else {
        element = items[front];
        if (front == rear) {
            front = -1;
            rear = -1;
        }
            // Q has only one element, so we reset the
            // queue after dequeing it. ?
        else {
            front = (front + 1) % SIZE;
        }
        printf("\n Deleted element -> %d \n", element);
        return (element);
    }
}

// Display the queue
void display() {
    int i;
    if (isEmpty())
        printf(" \n Empty Queue\n");
    else {
        printf("\n Front -> %d ", front);
        printf("\n Items -> ");
        for (i = front; i != rear; i = (i + 1) % SIZE) {
            printf("%d ", items[i]);
        }
        printf("%d ", items[i]);
        printf("\n Rear -> %d \n", rear);
    }
}

int main() {
    int choice;
```

```
    while(true) {
        puts("Circular Queue Operations: ");
        puts("1.Enqueue");
        puts("2.Dequeue");
        puts("3.Display");
        puts("Enter your choice: ");
        scanf("%d",&choice);
        switch(choice){
            case 1: enqueue(); break;
            case 2: dequeue(); break;
            case 3: display(); break;
            case 4: exit(0);

        }
    }

}
```

DS Programs for Mid 2

1. Singly Linked List with Head Node

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

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

Node *head ;

int insert(int);
int display();

int main(){
    int choice,newdata;
    while(true){
        puts("Singly Linked List: ");
        puts("1. Insert into Linked List");
        puts("2. Display Linked List");
        puts("3. Exit");
        puts("Enter your choice: ");

        scanf("%d",&choice);

        switch(choice){
            case 1:
                printf("Enter the data to be inserted: ");
                scanf("%d",&newdata);
                insert(newdata);
                break;

            case 2:
                display();
                break;

            case 3:
                exit(0);
        }
```

```
    }
}

int insert(int newdata){
    Node *temp;
    temp = (Node *)malloc(sizeof(Node));
    if(temp==NULL){
        printf("Memory allocation failed\n");
    }
    else{
        temp->data=newdata;
        temp->next=head;
        head=temp;
        printf("Node inserted\n");
    }
}

int display(){
    Node *temp;
    temp=head;
    if(temp==NULL){
        printf("Nothing to print\n");
    }
    else{
        printf("Singly Linked list is: ");
        while(temp!=NULL){
            printf("%d->",temp->data);
            temp=temp->next;
        }
        printf("NULL\n");

    }
}
```

2. Circular Linked List with Head

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

typedef struct node{
    int data;
    struct node *next;
}Node;
```

```c
Node *head ;

int insert(int);
int display();

int main(){
    int choice,newdata;
    while(true){
        puts("Circular Linked List: ");
        puts("1. Insert into Linked List");
        puts("2. Display Linked List");
        puts("3. Exit");
        puts("Enter your choice: ");

        scanf("%d",&choice);

        switch(choice){
            case 1:
                printf("Enter the data to be inserted: ");
                scanf("%d",&newdata);
                insert(newdata);
                break;

            case 2:
                display();
                break;

            case 3:
                exit(0);
        }
    }
}

int insert(int newdata){
    Node *temp,*ptr;
    ptr = (Node *)malloc(sizeof(Node));
    if(ptr==NULL){
        printf("Memory allocation failed\n");
    }
    else {
        ptr->data=newdata;
        if (head == NULL) {
            head=ptr;
            ptr->next=head;
```

```
        } else {
            temp=head;
            while(temp->next!=head)
                temp=temp->next;

            ptr->next = head;
            temp->next = ptr;
            head = ptr;


        }
        printf("Node inserted\n");
    }
}

int display(){
    Node *ptr;
    ptr=head;
    if(ptr==NULL){
        printf("Nothing to print\n");
    }
    else{
        printf("Circular Linked list is: ");
        while(ptr->next!=head){
            printf("%d->",ptr->data);
            ptr=ptr->next;
        }
        printf("%d->",ptr->data);
        printf("NULL\n");

    }
}
```

3. Doubly Linked List with Head

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

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

```c
Node *head ;

int insert(int);
int display();

int main(){
    int choice,newdata;
    while(true){
        puts("Doubly Linked List: ");
        puts("1. Insert into Linked List");
        puts("2. Display Linked List");
        puts("3. Exit");
        puts("Enter your choice: ");

        scanf("%d",&choice);

        switch(choice){
            case 1:
                printf("Enter the data to be inserted: ");
                scanf("%d",&newdata);
                insert(newdata);
                break;

            case 2:
                display();
                break;

            case 3:
                exit(0);
        }
    }
}

int insert(int newdata){
    Node *temp,*ptr;
    ptr = (Node *)malloc(sizeof(Node));
    if(ptr==NULL){
        printf("Memory allocation failed\n");
    }
    else {

        if (head == NULL) {
            ptr->next=NULL;
            ptr->prev=NULL;
```

```
            ptr->data=newdata;
            head=ptr;

        } else {
           ptr->data=newdata;
           ptr->prev=NULL;
           ptr->next=head;
           head->prev=ptr;
           head=ptr;

        }
        printf("Node inserted\n");
    }
}

int display(){
    Node *ptr;
    ptr=head;
    if(ptr==NULL){
        printf("Nothing to print\n");
    }
    else{
        printf("Doubly Linked list is: ");
        while(ptr!=NULL){
            printf("%d->",ptr->data);
            ptr=ptr->next;
        }

        printf("NULL\n");

    }
}
```

4. Circular Doubly Linked List with Head Node

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

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

```
Node *head ;

int insert(int);
int display();

int main(){
    int choice,newdata;
    while(true){
        puts("Circular Doubly Linked List: ");
        puts("1. Insert into Linked List");
        puts("2. Display Linked List");
        puts("3. Exit");
        puts("Enter your choice: ");

        scanf("%d",&choice);

        switch(choice){
            case 1:
                printf("Enter the data to be inserted: ");
                scanf("%d",&newdata);
                insert(newdata);
                break;

            case 2:
                display();
                break;

            case 3:
                exit(0);
        }
    }
}

int insert(int newdata){
    Node *temp,*ptr;
    ptr = (Node *)malloc(sizeof(Node));
    if(ptr==NULL){
        printf("Memory allocation failed\n");
    }
    else {
        ptr->data=newdata;
        if (head == NULL) {
            head=ptr;
            ptr->next=head;
```

```
            ptr->prev=head;



        } else {
          temp=head;
          while(temp->next!=head){
              temp=temp->next;
          }
          temp->next=ptr;
          ptr->prev=temp;
          head->prev=ptr;
          ptr->next=head;
          head=ptr;
        }
        printf("Node inserted\n");
    }
}

int display(){
    Node *ptr;
    ptr=head;
    if(ptr==NULL){
        printf("Nothing to print\n");
    }
    else{
        printf("Circular Doubly Linked list is: ");
        while(ptr->next!=head){
            printf("%d->",ptr->data);
            ptr=ptr->next;
        }
        printf("%d->",ptr->data);
        printf("NULL\n");

    }
}
```

5. Singly Linked List – All Operations

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

typedef struct node{
```

```
    int data;
    struct node *next;
}Node;

Node *head ;

int insert_first(int);
int insert_last(int);
int insert_random(int,int);
int delete_first();
int delete_last();
int delete_random();
int display();

int main(){
    int choice,newdata,key;
    while(true){
        puts("Singly Linked List: ");
        puts("1. Insert at beginning");
        puts("2. Insert at end");
        puts("3. Insert at specific position");
        puts("4. Delete First Node");
        puts("5. Delete Last Node");
        puts("6. Display Linked List");
        puts("7. Exit");
        puts("Enter your choice: ");

        scanf("%d",&choice);

        switch(choice){
            case 1:
                printf("Enter the data to be inserted: ");
                scanf("%d",&newdata);
                insert_first(newdata);
                break;

            case 2:
                printf("Enter the data to be inserted: ");
                scanf("%d",&newdata);
                insert_last(newdata);
                break;

            case 3:
                printf("Enter the data to be inserted: ");
                scanf("%d",&newdata);
```

```
                printf("Enter location to be inserted: ");
                scanf("%d",&key);
                insert_random(newdata,key);
                break;

            case 4:
                delete_first();
                break;

            case 5:
                delete_last();
                break;

            case 6:
                display();
                break;

            case 7:
                exit(0);
        }
    }
}

int insert_first(int newdata){
    Node *temp;
    temp = (Node *)malloc(sizeof(Node));
    if(temp==NULL){
        printf("Memory allocation failed\n");
    }
    else{
        temp->data=newdata;
        temp->next=head;
        head=temp;
        printf("Node inserted\n");
    }
}

int insert_last(int newdata){
    Node *ptr,*temp;
    ptr=(Node *)malloc(sizeof(Node));


    if(ptr==NULL){
        printf("Memory allocation failed\n");
    }
```

```
    ptr->data=newdata;
    ptr->next=NULL;
    if(head==NULL){
        head=ptr;
    }
    else {
        temp = head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = ptr;
    }
    printf("Node inserted\n");
}

int insert_random(int newdata,int key){
    Node *temp,*ptr;
    int i;
    ptr=(Node *)malloc(sizeof(Node));
    if(ptr==NULL){
        printf("Mem allocation failed lol");
    }
    ptr->data=newdata;
    ptr->next=NULL;
    if(head==NULL){
        if(key==0){
            head=ptr;
        }
    }
    else{

        temp=head;
        for(i=0;i<key;i++){
            temp=temp->next;
            if(temp==NULL){
                printf("Cant insert\n");
            }

        }
        ptr->next=temp->next;
        temp->next=ptr;
        printf("Node inserted\n");
    }
}
```

```c
int delete_first(){
    Node *ptr;
    if(head==NULL){
        printf("List is empty\n");
    }
    else{
        ptr=head;
        head=ptr->next;
        free(ptr);
        printf("First node deleted\n");
    }
}

int delete_last(){
    Node *ptr,*ptr1;
    if(head==NULL){
        printf("List is empty\n");
    }
    else if(head->next==NULL){
        head=NULL;
        free(head);
        printf("Node deleted\n");
    }
    else{
        ptr=head;
        while(ptr->next!=NULL){
            ptr1=ptr;
            ptr=ptr->next;
        }
        ptr1->next=NULL;
        free(ptr);
        printf("Deleted node from last\n");
    }
}


int display(){
    Node *temp;
    temp=head;
    if(temp==NULL){
        printf("Nothing to print\n");
    }
    else{
        printf("Singly Linked list is: ");
        while(temp!=NULL){
```

```
            printf("%d->",temp->data);
            temp=temp->next;
        }
        printf("NULL\n");

    }
}
```

6. Circular Linked List – All Operations

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

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

Node *head ;

int insert_first(int);
int insert_last(int);
int delete_first();
int delete_last();
int search(int);
int display();

int main(){
    int choice,newdata,key;
    while(true){
        puts("Circular Linked List: ");
        puts("1. Insert at Beginning");
        puts("2. Insert at Last");
        puts("3. Delete First");
        puts("4. Delete Last");
        puts("5. Search Element");
        puts("6. Display Linked List");
        puts("7. Exit");
        puts("Enter your choice: ");

        scanf("%d",&choice);

        switch(choice){
```

```
            case 1:
                printf("Enter the data to be inserted: ");
                scanf("%d",&newdata);
                insert_first(newdata);
                break;

            case 2:
                printf("Enter the data to be inserted: ");
                scanf("%d",&newdata);
                insert_last(newdata);
                break;

            case 3:
                delete_first();
                break;

            case 4:
                delete_last();
                break;

            case 5:
                printf("Enter item to search: ");
                scanf("%d",&key);
                search(key);
                break;

            case 6:
                display();
                break;

            case 7:
                exit(0);
        }
    }
}

int insert_first(int newdata){
    Node *temp,*ptr;
    ptr = (Node *)malloc(sizeof(Node));
    if(ptr==NULL){
        printf("Memory allocation failed\n");
    }
    else {
        ptr->data=newdata;
        if (head == NULL) {
```

```
            head=ptr;
            ptr->next=head;
        } else {
            temp=head;
            while(temp->next!=head)
                temp=temp->next;

            ptr->next = head;
            temp->next = ptr;
            head = ptr;

        }
        printf("Node inserted\n");
    }
}

int insert_last(int newdata){
    Node *ptr,*temp;
    ptr=(Node *)malloc(sizeof(Node));
    if(ptr==NULL){
        printf("Mem allocation failed\n");
    }
    else{
        ptr->data=newdata;
        if(head==NULL){
            head=ptr;
            ptr->next=head;
        }
        else{
            temp=head;
            while(temp->next!=head){
                temp=temp->next;
            }
            temp->next=ptr;
            ptr->next=head;
        }
        printf("Node inserted\n");
    }
}

int delete_first(){
    if(head == NULL){
        printf("List is empty\n");
        return -1;
    }
```

```
    else{
        Node *temp = head;
        while(temp->next != head){
            temp = temp->next;
        }
        Node *ptr = head;
        temp->next = ptr->next;
        head = ptr->next;
        free(ptr);
        printf("First node deleted\n");
        return 0;
    }
}

int delete_last(){
    if(head == NULL){
        printf("List is empty\n");
        return -1;
    }
    else{
        Node *temp = head;
        Node *prev = NULL;
        while(temp->next != head){
            prev = temp;
            temp = temp->next;
        }
        prev->next = temp->next;
        free(temp);
        printf("Last node deleted\n");
        return 0;
    }
}

int search(int key){
    if(head == NULL){
        printf("List is empty\n");
        return -1;
    }
    else{
        Node *temp = head;
        int index = 0;
        do{
            if(temp->data == key){
                printf("Element found at index %d\n", index);
                return index;
```

```
            }
            temp = temp->next;
            index++;
        }while(temp != head);
        printf("Element not found\n");
        return -1;
    }
}

int display(){
    Node *ptr;
    ptr=head;
    if(ptr==NULL){
        printf("Nothing to print\n");
    }
    else{
        printf("Circular Linked list is: ");
        while(ptr->next!=head){
            printf("%d->",ptr->data);
            ptr=ptr->next;
        }
        printf("%d->",ptr->data);
        printf("NULL\n");

    }
}
```

7. Doubly Linked List – All Operations

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

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

Node *head ;

int insert_first(int);
int insert_last(int);
```

```c
int insert_random(int,int);
int delete_first();
int delete_last();
int search();
int display();

int main(){
    int choice,newdata,loc,key;
    while(true){
        puts("Doubly Linked List: ");
        puts("1. Insert first");
        puts("2. Insert last");
        puts("3. Insert specified");
        puts("4. Delete First");
        puts("5. Delete Last");
        puts("6. Search");
        puts("7. Display Linked List");
        puts("8. Exit");
        puts("Enter your choice: ");

        scanf("%d",&choice);

        switch(choice){
            case 1:
                printf("Enter the data to be inserted: ");
                scanf("%d",&newdata);
                insert_first(newdata);
                break;

            case 2:
                printf("Enter the data to be inserted: ");
                scanf("%d",&newdata);
                insert_last(newdata);
                break;

            case 3:
                printf("Enter the data to be inserted: ");
                scanf("%d",&newdata);
                printf("Enter location: ");
                scanf("%d",&loc);
                insert_random(newdata,loc);

            case 4:
                delete_first();
                break;
```

```
            case 5:
                delete_last();
                break;

            case 6:
                printf("Enter key: ");
                scanf("%d",&key);
                search(key);
                break;

            case 7:
                display();
                break;

            case 8:
                exit(0);
        }
    }
}

int insert_first(int newdata){
    Node *temp,*ptr;
    ptr = (Node *)malloc(sizeof(Node));
    if(ptr==NULL){
        printf("Memory allocation failed\n");
    }
    else {

        if (head == NULL) {
            ptr->next=NULL;
            ptr->prev=NULL;
            ptr->data=newdata;
            head=ptr;

        } else {
            ptr->data=newdata;
            ptr->prev=NULL;
            ptr->next=head;
            head->prev=ptr;
            head=ptr;

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

```
}

int insert_last(int newdata){
    Node *ptr,*temp;
    ptr=(Node *)malloc(sizeof(Node));
    if(ptr==NULL){
        printf("Mem allocation failed\n");
    }
    else{
        ptr->data=newdata;
        if(head==NULL){
            ptr->next=NULL;
            ptr->prev=NULL;
            head=ptr;
        }
        else{
            temp=head;
            while(temp->next!=NULL){
                temp=temp->next;
            }
            temp->next=ptr;
            ptr->prev=temp;
            ptr->next=NULL;
        }
    printf("Node inserted");
    }
}

int insert_random(int newdata, int loc){
    Node *ptr,*temp;
    int i;
    ptr=(Node *)malloc(sizeof(Node));
    if(ptr==NULL){
        printf("Mem allocation failed");
    }
    else{
        temp=head;
        for(i=0;i<loc;i++){
            temp=temp->next;
            if(temp==NULL){
                printf("less elements");
            }
        }
        ptr->data=newdata;
        ptr->next=temp->next;
```

```
        ptr->prev = temp;
        temp->next=ptr;
        temp->next->prev=ptr;
        printf("Node inserted\n");
    }
}

int delete_first(){
    Node *ptr;
    if(head==NULL){
        printf("List empty\n");
    }
    else if(head->next==NULL){
        head=NULL;
        free(head);
        printf("Node deleted\n");
    }
    else{
        ptr = head;
        head = head -> next;
        head-> prev = NULL;
        free(ptr);
        printf("Node deleted\n");
    }
}

int delete_last(){
    Node *ptr;
    if(head==NULL){
        printf("No elements\n");
    }
    else if(head->next==NULL){
        head=NULL;
        free(head);
        printf("Node deleted\n");
    }
    else{
        ptr=head;
        if(ptr->next!=NULL){
            ptr=ptr->next;
        }
        ptr->prev->next=NULL;
        free(ptr);
        printf("Node deleted\n");
    }
```

```
}

int search(int key){
    Node *ptr;
    int i=0,flag;
    ptr=head;
    if(ptr==NULL){
        printf("Empty list\n");
    }
    else{
        while(ptr!=NULL){
            if(ptr->data==key){
                printf("Item found at %d",i+1);
                flag=0;
                break;
            }
            else{
                flag=1;
            }
            i++;
            ptr=ptr->next;
        }
        if(flag==1){
            printf("Item not found\n");
        }
    }
}

int display(){
    Node *ptr;
    ptr=head;
    if(ptr==NULL){
        printf("Nothing to print\n");
    }
    else{
        printf("Doubly Linked list is: ");
        while(ptr!=NULL){
            printf("%d->",ptr->data);
            ptr=ptr->next;
        }

        printf("NULL\n");

    }
}
```

8. Circular Doubly Linked List – All Operations

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

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

Node *head ;

int insert_first(int);
int insert_last(int);
int insert_random(int,int);
int delete_first();
int delete_last();
int search(int);
int display();

int main(){
    int choice,newdata,loc,key;
    while(true){
        puts("Circular Doubly Linked List: ");
        puts("1. Insert First");
        puts("2. Insert last");
        puts("3. Insert anywhere");
        puts("4. Delete first");
        puts("5. Delete last");
        puts("6. Search");
        puts("7. Display Linked List");
        puts("8. Exit");
        puts("Enter your choice: ");

        scanf("%d",&choice);

        switch(choice){
            case 1:
                printf("Enter the data to be inserted: ");
                scanf("%d",&newdata);
```

```
                insert_first(newdata);
                break;

            case 2:
                printf("Enter the data to be inserted: ");
                scanf("%d",&newdata);
                insert_last(newdata);
                break;

            case 3:
                printf("Enter the data to be inserted: ");
                scanf("%d",&newdata);
                printf("Enter position: ");
                scanf("%d",&loc);
                insert_random(newdata,loc);
                break;

            case 4:
                delete_first();
                break;

            case 5:
                delete_last();
                break;

            case 6:
                printf("Enter location to search: ");
                scanf("%d",&key);
                search(key);
                break;

            case 7:
                display();
                break;

            case 8:
                exit(0);
        }
    }
}

int insert_first(int newdata){
    Node *temp,*ptr;
    ptr = (Node *)malloc(sizeof(Node));
    if(ptr==NULL){
```

```
        printf("Memory allocation failed\n");
    }
    else {
        ptr->data=newdata;
        if (head == NULL) {
            head=ptr;
            ptr->next=head;
            ptr->prev=head;


        } else {
            temp=head;
            while(temp->next!=head){
                temp=temp->next;
            }
            temp->next=ptr;
            ptr->prev=temp;
            head->prev=ptr;
            ptr->next=head;
            head=ptr;
        }
        printf("Node inserted\n");
    }
}

int insert_last(int newdata){
    Node *ptr,*temp;
    ptr=(Node *)malloc(sizeof(Node));

    if(ptr==NULL){
        printf("Mem allocation failed\n");
    }
    else{
        ptr->data=newdata;
        if(head==NULL){
            head=ptr;
            ptr->next=head;
            ptr->prev=head;
        }
        else{
            temp=head;
            while(temp->next!=head){
                temp=temp->next;
            }
            temp->next=ptr;
```

```
            ptr->prev=temp;
            head->prev=ptr;
            ptr->next=head;
        }
    }
    printf("Node inserted\n");
}

int insert_random(int newdata,int loc){
    Node *ptr,*temp;
    int i;
    ptr=(Node *)malloc(sizeof(Node));
    if(ptr==NULL){
        printf("Mem allocation failed\n");

    }
    else{
        temp=head;
        for(i=0;i<loc;i++){
            temp=temp->next;
            if(temp==NULL) {
                printf("Less no of elements\n");
            }
        }
        ptr->data=newdata;
        ptr->next=temp->next;
        ptr->prev=temp;
        temp->next=ptr;
        temp->next->prev=ptr;
        printf("Node inserted\n");
    }
}

int delete_first(){
    Node *ptr;
    if(head==NULL){
        printf("No elements in list\n");
    }
    else if(head->next==NULL){
        ptr=head;
        head=NULL;
        free(ptr);
        printf("Node deleted\n");
    }
    else{
```

```
        ptr=head;
        head=head->next;
        head->prev=ptr->prev;
        ptr->prev->next=head;
        free(ptr);
        printf("Node deleted\n");
    }
}

int delete_last(){
    Node *ptr, *prev;
    if(head==NULL){
        printf("No elements in list\n");
        return -1;
    }
    else if(head->next==NULL){
        ptr = head;
        head=NULL;
        free(ptr);
        printf("Node deleted\n");

    }
    else{
        ptr = head;
        while(ptr->next != head){
            ptr = ptr->next;
        }
        prev = ptr->prev;
        prev->next = head;
        head->prev = prev;
        free(ptr);
        printf("Node deleted\n");

    }
}

int search(int key){
    Node *ptr;
    int index = 0;
    if(head==NULL){
        printf("No elements in list\n");
        return -1;
    }
    else{
        ptr = head;
```

```
        do{
            if(ptr->data == key){
                printf("Element found at index %d\n", index);
                return index;
            }
            ptr = ptr->next;
            index++;
        }while(ptr != head);
        printf("Element not found\n");
        return -1;
    }
}

int display(){
    Node *ptr;
    ptr=head;
    if(ptr==NULL){
        printf("Nothing to print\n");
    }
    else{
        printf("Circular Doubly Linked list is: ");
        while(ptr->next!=head){
            printf("%d->",ptr->data);
            ptr=ptr->next;
        }
        printf("%d->",ptr->data);
        printf("NULL\n");

    }
}
```

9. XOR Linked List

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int data;
    struct node* npx;
};
struct node* XOR(struct node *a, struct node *b) {
    return (struct node*) ((unsigned int) (a) ^ (unsigned int) (b));
}

void insert(struct node **head_ref, int data) {
```

```c
    struct node *new_node = (struct node *) malloc(sizeof(struct node));
    new_node->data = data;


    new_node->npx = XOR(*head_ref, NULL);


    if (*head_ref != NULL) {
        struct node* next = XOR((*head_ref)->npx, NULL);
        (*head_ref)->npx = XOR(new_node, next);
    }
    *head_ref = new_node;
}


void printList(struct node *head) {
    struct node *curr = head;
    struct node *prev = NULL;
    struct node *next;
    printf("Following are the nodes of Linked List: \n");
    while (curr != NULL) {

        printf("%d ", curr->data);
        next = XOR(prev, curr->npx);

        prev = curr;
        curr = next;
    }
}

int main() {

    struct node *head = NULL;

    insert(&head, 10);
    insert(&head, 20);
    insert(&head, 30);
    insert(&head, 40);

    printList(head);
    return (0);
}
```

10.Skip List

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

#define MAX_LEVEL 6

const float P = 0.5;

struct Node {
    int key;
    struct Node* forward[MAX_LEVEL + 1];
};

struct SkipList {
    int level;
    struct Node *header;
};

struct Node* createNode(int level, int key) {
    struct Node *n = (struct Node*)malloc(sizeof(struct Node));
    n->key = key;
    for (int i = 0; i <= level; i++) {
        n->forward[i] = NULL;
    }
    return n;
}

struct SkipList* createSkipList() {
    struct SkipList *sl = (struct SkipList*)malloc(sizeof(struct SkipList));
    sl->level = 0;
    sl->header = createNode(MAX_LEVEL, 0);
    return sl;
}

int randomLevel() {
    int level = 0;
    while (rand() < P*RAND_MAX && level < MAX_LEVEL)
        level++;
    return level;
}

void insert(struct SkipList* sl, int key) {
    struct Node *update[MAX_LEVEL + 1];
    struct Node *x = sl->header;
```

```
    for (int i = sl->level; i >= 0; i--) {
        while (x->forward[i] != NULL && x->forward[i]->key < key)
            x = x->forward[i];
        update[i] = x;
    }
    x = x->forward[0];
    if (x == NULL || x->key != key) {
        int level = randomLevel();
        if (level > sl->level) {
            for (int i = sl->level + 1; i <= level; i++)
                update[i] = sl->header;
            sl->level = level;
        }
        x = createNode(level, key);
        for (int i = 0; i <= level; i++) {
            x->forward[i] = update[i]->forward[i];
            update[i]->forward[i] = x;
        }
    }
}

struct Node* search(struct SkipList* sl, int key) {
    struct Node *x = sl->header;
    for (int i = sl->level; i >= 0; i--) {
        while (x->forward[i] != NULL && x->forward[i]->key < key)
            x = x->forward[i];
    }
    x = x->forward[0];
    if (x != NULL && x->key == key)
        return x;
    else
        return NULL;
}

void display(struct SkipList* sl) {
    printf("\n*****Skip List*****\n");
    for (int i = 0; i <= sl->level; i++) {
        struct Node *node = sl->header->forward[i];
        printf("Level %d: ", i);
        while (node != NULL) {
            printf("%d ", node->key);
            node = node->forward[i];
        }
        printf("\n");
    }
```

```
}

int main() {
    struct SkipList* sl = createSkipList();
    int choice, key;
    while (1) {
        printf("\n1.Insert\n2.Search\n3.Display\n4.Exit\nEnter your choice: ");
        scanf("%d", &choice);
        switch(choice) {
            case 1:
                printf("Enter key to insert: ");
                scanf("%d", &key);
                insert(sl, key);
                break;
            case 2:
                printf("Enter key to search: ");
                scanf("%d", &key);
                if (search(sl, key) == NULL)
                    printf("Key %d not found.\n", key);
                else
                    printf("Key %d found.\n", key);
                break;
            case 3:
                display(sl);
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice.\n");
        }
    }
    return 0;
}
```

11.Stacks using Linked List

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

typedef struct node{
    int data;
    struct node *ptr;
}Node;
```

```
Node *top=NULL,*top1,*temp;

void push(int);
void pop();
int peek();
void display();

int count =0;

int main(){
    int choice, num,e;
    top=NULL; // Initialize top to NULL here
    while(true){
        puts("1. Push");
        puts("2. Pop");
        puts("3.Peek");
        puts("4.Display");
        puts("5.Exit");
        puts("Enter your choice: ");
        scanf("%d",&choice);
        switch(choice){
            case 1:
                printf("Enter data: ");
                scanf("%d",&num);
                push(num);
                break;

            case 2:
                pop();
                break;

            case 3:
                if(top==NULL)
                    printf("Underflow\n");
                else{
                    printf("Top element: %d",peek());
                }
                break;

            case 4:
                display();
                break;

            case 5:
```

```
                exit(0);
        }
    }
}

void push(int newdata){
    temp=(Node *)malloc(sizeof(Node));
    temp->ptr=top;
    temp->data=newdata;
    top=temp;
    count++;
}

void pop(){
    top1=top;
    if(top1==NULL){
        printf("Error with popping\n");
    }
    else{
        top=top1->ptr;
        printf("Deleted value= %d\n",top1->data);
        free(top1);
        count--;
    }
}

int peek(){
    if(top!=NULL)
        return top->data;
    else
        return -1;
}

void display(){
    top1=top;
    if(top1==NULL){
        printf("Underflow\n");
    }
    else{
        while(top1!=NULL){
            printf("%d ",top1->data);
            top1=top1->ptr;
        }
        printf("\n");
    }
```

```
}
```

12.Queue using Linked List

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

#define MAX_LEVEL 3

struct Node {
    int key;
    struct Node* forward[MAX_LEVEL+1];
};

struct SkipList {
    struct Node *header;
    int level;
};

struct Node* createNode(int key) {
    struct Node* newNode = (struct Node*) malloc(sizeof(struct Node));
    newNode->key = key;
    for(int i = 0; i <= MAX_LEVEL; i++) {
        newNode->forward[i] = NULL;
    }
    return newNode;
}

struct SkipList* createSkipList() {
    struct SkipList* list = (struct SkipList*) malloc(sizeof(struct SkipList));
    list->header = createNode(-1);
    list->level = 0;
    return list;
}

void insert(struct SkipList* list, int key) {
    struct Node* update[MAX_LEVEL+1];
    struct Node* current = list->header;

    for(int i = list->level; i >= 0; i--) {
        while(current->forward[i] != NULL && current->forward[i]->key < key) {
            current = current->forward[i];
        }
        update[i] = current;
```

```
    }

    current = current->forward[0];

    if(current == NULL || current->key != key) {
        int rlevel = rand() % (MAX_LEVEL+1);

        if(rlevel > list->level) {
            for(int i = list->level+1; i <= rlevel; i++) {
                update[i] = list->header;
            }
            list->level = rlevel;
        }

        struct Node* newNode = createNode(key);

        for(int i = 0; i <= rlevel; i++) {
            newNode->forward[i] = update[i]->forward[i];
            update[i]->forward[i] = newNode;
        }
    }
}

void display(struct SkipList* list) {
    printf("\n*****Skip List*****\n");
    for(int i = 0; i <= list->level; i++) {
        struct Node* node = list->header->forward[i];
        printf("Level %d: ", i);
        while(node != NULL) {
            printf("%d ", node->key);
            node = node->forward[i];
        }
        printf("\n");
    }
}

int main() {
    struct SkipList* list = createSkipList();
    int choice, key;

    while(1) {
        printf("\n1.Insert\n2.Display\n3.Exit\nEnter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
```

```
            case 1:
                printf("\nEnter the key to insert: ");
                scanf("%d", &key);
                insert(list, key);
                break;
            case 2:
                display(list);
                break;
            case 3:
                exit(0);
            default:
                printf("\nInvalid choice!");
        }
    }


}
```

13.Bubble Sort

```
#include <stdio.h>

int main(){
    int arr[100],n,c,d,swap;
    printf("Enter no of elements: ");
    scanf("%d",&n);
    printf("Enter %d integers",n);
    for(c=0;c<n;c++){
        scanf("%d",&arr[c]);
    }
    for(c=0;c<n-1;c++){
        for(d=0;d<n-c-1;d++){
            if(arr[d]>arr[d+1]){
                swap = arr[d];
                arr[d]=arr[d+1];
                arr[d+1]=swap;

            }
        }
    }
    printf("Sorted List in ascending order: ");
    for(c=0;c<n;c++)
        printf("%d\n",arr[c]);
}
```

14.Selection Sort

```c
#include <stdio.h>

int main(){
    int arr[10],n,i,j,pos,swap;

    printf("Enter number of elements: ");
    scanf("%d",&n);
    printf("Enter %d Elements: ",n);
    for(i=0;i<n;i++){
        scanf("%d",&arr[i]);
    }
    for(i=0;i<(n-1);i++){
        pos=i;
        for(j=i+1;j<n;j++){
            if(arr[pos]>arr[j])
                pos=j;
        }
        if (pos!=i){
            swap=arr[i];
            arr[i]=arr[pos];
            arr[pos]=swap;
        }
    }
    printf("Sorted List is: ");
    for(i=0;i<n;i++){
        printf("%d\t",arr[i]);
    }
}
```

15.Insertion Sort

```c
#include <stdio.h>

void insertionSort(int arr[],int n){
    int i,key,j;
    for(i=1;i<n;i++){
        key=arr[i];
        j=i-1;
        while(j>=0&&arr[j]>key){
            arr[j+1]=arr[j];
            j=j-1;
        }
```

```
        arr[j+1]=key;
    }
}

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

int main(){
    int arr[10],n,i;
    printf("Enter size: ");
    scanf("%d",&n);
    printf("Enter elements: ");
    for(i=0;i<n;i++)
        scanf("%d",&arr[i]);
    insertionSort(arr,n);
    printf("Sorted array is: ");
    arrayPrinter(arr,n);
}
```

16.Shell Sort

```
#include <stdio.h>

int shell(int a[],int n){
    int inter,i,temp,j;
    for(inter=n/2;inter>0;inter/=2){
        for(i=inter;i<n;i+=1){
            temp=a[i];
            for(j=i;j>=inter&&a[j-inter]>temp;j-=inter)
                a[j]=a[j-inter];
            a[j]=temp;
        }
    }
}

int printArr(int a[],int n){
    int i;
    for(i=0;i<n;i++)
        printf("%d ",a[i]);
```

```
}

int main(){
    int a[10],n,i;
    printf("Enter size of array: ");
    scanf("%d",&n);
    printf("Enter values: ");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    shell(a,n);
    printf("Sorted List is: ");
    printArr(a,n);
}
```

17.Radix Sort

```
#include <stdio.h>

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

void countSort(int a[],int n,int place){
    int out[n+1],i;
    int count[10]={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--){
        out[count[(a[i]/place)%10]-1]=a[i];
        count[(a[i]/place)%10]--;
    }
    for(i=0;i<n;i++)
        a[i]=out[i];
}

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

```c
    for(int place = 1; max/place>0;place*=10)
        countSort(a,n,place);
}

void printArray(int a[],int n){
    int i;
    for(i=0;i<n;i++)
        printf("%d ",a[i]);
}

int main(){
    int i,a[30],n;
    printf("Enter no of elements: ");
    scanf("%d",&n);
    printf("Enter elements: ");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    radixSort(a,n);
    printf("Sorted Array is: ");
    printArray(a,n);
}
```