1. Search an element in a two dimensional array

Algorithm

```
1. start
2. accept rows and columns
3. accept elements into matrix m
4. accept element e to be searched for in the matrix
5. set i=0,found=false
6. repeat while i < rows
    1. set j=0
    2. repeat while j < columns
        1. if e = m[i][j] , then print found at (i,j) and set flag=true
        2. set j = j + 1
    3. set i = i + 1
7. if found = false, then print not found
8. stop
```

Source Code

```c
#include <stdio.h>

int main()
{
    int x, y, flag = 0;
    printf("\n To search an element from 2D array : \n");

    printf("\n Enter n.o rows in array : ");
    scanf("%d", &x);

    printf("\n Enter n.o cols in array : ");
    scanf("%d", &y);

    int arr[x][y], i, j, search;

    for (i = 0; i < x; i++) {
        printf("enter row %d : \n", i + 1);
        for (j = 0; j < y; j++) {
            scanf("%d", &arr[i][j]);
        }
    }

    printf("\n Enter element to be searched : ");
    scanf("%d", &search);

    for (i = 0; i < x; i++)
    {
        for (j = 0; j < y; j++)
        {
            if (arr[i][j] == search)
            {
                flag = 1;
                printf("\n\n %d Found at position (%d,%d) ", search, i + 1, j + 1);
            }
        }
    }
    if (!flag)
        printf("\n\n Not found ");
    return 0;
}
```

2. Binary Search (Iterative)

Algorithm

```
1. start
2. accept sorted list of number arr and length of list , len
3. accept element e to be searched for in the list
4. set beg=0,end=len
5. repeat while beg <= len
    1. set mid = (beg + end)/2
    2. if arr[mid]= e , then print found at mid , break
    3. else if arr[mid] > e , then set end = mid - 1
    4. else , set beg = mid + 1
6. if beg <= len , print not found
7. stop
```

Sorting algorithm (Bubble Sort)

```
1. start
2. accept list of numbers , arr and length of list , len
3. set i = 0
4. repeat while i < len
    1. set j=0
    2. repeat while j < len - i - 1
        1. if arr[j] > arr[j+1] , then swap arr[j] and arr[j+1]
        2. j = j + 1
    3. i = i + 1
5. stop
```

Source Code

```c
#include <stdio.h>

int main()
{
    int x, y, flag = 0;
    printf("\n To search an element from 2D array : \n");

    printf("\n Enter n.o rows in array : ");
    scanf("%d", &x);

    printf("\n Enter n.o cols in array : ");
    scanf("%d", &y);

    int arr[x][y], i, j, search;

    for (i = 0; i < x; i++) {
        printf("enter row %d : \n", i + 1);
        for (j = 0; j < y; j++) {
            scanf("%d", &arr[i][j]);
        }
    }

    printf("\n Enter element to be searched : ");
    scanf("%d", &search);
    for (i = 0; i < x; i++)
    {
        for (j = 0; j < y; j++)
        {
            if (arr[i][j] == search)
            {
                flag = 1;
                printf("\n\n %d Found at position (%d,%d) ", search, i + 1, j + 1);
            }
        }
    }
    if (!flag)
        printf("\n\n Not found ");
    return 0;
}
```

3. Implementation of singly linked list

Algorithm

Insertion at index

```
1. start
2. accept a node , newNode and pos
3. if start = NULL , then set start = newNode
4. if pos = 0, do
    1. newNode->next = start
    2. start = newNode
5. else , do
    1. set cur = 0
    2. set i = start
    3. repeat while cur < pos - 1 and i != NULL
        1. i = i->next
    4. if i->next = NULL, then set i->next = newNode
    5. else , do
        1. newNode->next = i->next
        2. i->next = newNode
    6. if pos = 0, then set start = newNode
6. stop
```

Deletion at index

```
1. start
2. accept index
3. if start = NULL, stop
4. set cur = 0 , i = start , temp = NULL
5. repeat while cur < pos - 1 and i != NULL
    1. i = i->next
6. if pos != 0 , do
    1. temp = i->next
    2. i->next = (i->next)->next
7. else ,
    1. temp = i
    2. start = i->next
8. stop
```

Searching in list

```
1. start
2. accept number n to be searched for in the list
3. set loc = 0
4. if start = NULL, stop
5. set i = start
6. repeat while i != NULL
    1. if n = i->data, do
        1. print found at loc
        2. stop
7. print not found
8. stop
```

Source Code

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

struct NODE
{
    int num;
    struct NODE * next;
}*start=NULL;

typedef struct NODE NODE;
int count=0;

NODE* create_node(int num){
    NODE *newNode = calloc(1,sizeof(NODE));
    newNode->num = num;
    newNode->next = NULL;
    count++;
    return newNode;
}
/* inserts a node exactly at index , i.e , insertion after index-1
if the index is greater than the number of nodes in the list , the new node is appeneded to the list
prepend to list : insert at index 0
append to list : inset at index (count+1) or INFINITY
    */
void insert_at(int num,int pos)
{
    NODE *newNode = create_node(num);

    if(start==NULL){
        start=newNode;
        return;
    }
    int cur=0;
    NODE *i;
    for(i=start;cur<pos-1 && i->next ;i=i->next,cur++)
        if(!(i->next)){
            i->next=newNode;
            return;
        }
    newNode->next=i->next;
    i->next=newNode;

    if(pos==0)
        start=newNode;
}
// deletes the node at index
bool delete_at(int pos)
{
    if(!start)
        return false;
    int cur=0;
    NODE *i=start,*temp=NULL;
    for(;cur<pos-1 &&i;i=i->next,cur++);
    if(pos){
        temp=i->next;
        i->next=(i->next)->next;
    }
    else{
        temp=i;
        start=i->next;
    }

    free(temp);
    count--;
    return true;
}

// to display node
void display()
{
    for(NODE *i=start;i;i=i->next)
        printf(" %d", i->num);
}

// searching
int searching(int search)
{
    int loc=0;
    if(!start)
        return -1;
    for(NODE *i=start;i!=NULL;i=i->next,loc++)
        if(i->num==search)
            return loc;
    return -1;
}

// To free list
void freelist()
{
    NODE *temp=NULL;
    for(NODE*i=start;i!=NULL;i=i->next){
        free(temp);
        temp=i;
    }
    free(temp);
```

```
}

int main()
{
    start = NULL;
    int choice;
    int num,x;

    printf("1. Append\n2. Prepend\n3. Insert at index\n4. Delete from index \n5. Search in list
↪   \n6. Display list \n0. Exit");
    do
    {
        printf("\nEnter Choice: ");
        scanf("%d", &choice);

        switch(choice)
        {
            case 1:{
                       printf("\nEnter number: ");
                       scanf("%d", &num);
                       insert_at(num,count+1);
                       break;
                   }
            case 2:{
                       printf("\nEnter number: ");
                       scanf("%d", &num);
                       insert_at(num,0);
                       break;
                   }
            case 3:{
                       int pos;
                       printf("\nEnter number: ");
                       scanf("%d", &num);
                       printf("Enter position: ");
                       scanf("%d", &pos);
                       insert_at(num,pos);
                       break;
                   }
            case 4:{
                       int pos;
                       printf("\nEnter position: ");
                       scanf("%d", &pos);
                       if(!delete_at(pos))
                           printf("\nList Empty!");
                       break;
                   }
            case 5:{
                       printf("\n Enter element to search : ");
                       scanf("%d",&x);
                       searching(x);
                       break;
                   }
            case 6:{
                       display();
                       break;
                   }
        }

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

4. Implementation of Sparse Matrix

Algorithm

check if matrix is sparse

```
1. start
2. accept elements to 2D array arr of m rows and n columns
3. set i <-0,val <-0
4. repeat while i < m
    1. set j = 0
    2. repeat while j < n
        1. if arr[i][j] != 0, then val = val +1
        2. j = j +1
    3. i = i + 1
5. if val >= m*n/2, the print Not sparse
6. else , print sparse
7. stop
```

compression algorithm (let arr[m][n] be the entered sparse matrix of row m and column n, and trip[k][3] be an array to store the compressed values)

```
1. start
2. accept sparse matrix arr of m rows and n columns
3. set i = 0,j = 0,k = 0
4. repeat while i < m
    1. j = 0
    2. repeat while j < n
        1. if arr[m][n] != 0, then do
            1. set trip[k][0] = i, trip[k][1] = j, trip[k][3] = arr[i][j]
            2. set k = k+1
    3. j = j+1
    4. i = i+1
5. stop
```

Source Code

```
#include<stdio.h>

int main()
{
    int x,y,i,j;
    int arr[10][10];
    printf("\nTo respresent sparse matrix ");
    printf("\nEnter no of rows in matrix : ");
    scanf("%d",&x);
    printf("\nEnter no of columns in matrix : ");
    scanf("%d",&y);
    printf("\nEnter matrix elements ");
    for(i=0;i<x;i++) // to accept the matrix from user
    {
        printf("\n Enter row %d : " ,i);
        for(j=0;j<y;j++)
            scanf("%d",&arr[i][j]);
    }

    int val=0;

    for(i=0;i<x;i++)   // to search the no.of ones in the matrix
        for(j=0;j<y;j++)
            if(arr[i][j])
                val++;

    if ( val >= x\*y/2) //If not sparse then exit.
    {
        printf("\n Not a sparse mtrix");
        return 0;
    }

    int trip[10][3];
    int k=0;
    for(i=0;i<x;i++) //to store the row , coloumn and value in triplete form
    {
        for(j=0;j<y;j++)
        {
            if(arr[i][j])
            {
                trip[k][0]=i;
                trip[k][1]=j;
                trip[k][2]=arr[i][j];
                k++;
            }
        }
    }

    printf("The triplete form with row no,column no and value is : \n");
    for(i=0;i<val;i++)
        printf("%d %d %d \n",trip[i][0],trip[i][1],trip[i][2]);

    // size of sparse matrix :
    int a=x*y*sizeof(int);
    //size of triplete array :
    int b=val*3*sizeof(int);
    printf("\n size of sparse matrix = %d ",a);
    printf("\n size of compresed array = %d ",b);
    return 0;
}
```

5. Binary Search (Recursive)

Algorithm

recursive searching algorithm

1. start
2. accept a sorted list of numbers into arr
3. input lower, upper and search
4. if lower < upper
    1. print Not found

```
5. set x = (lower+upper)/2
6. if arr[x] = search
    1. return Found
7. if search < arr[x]
    1. binary_search(arr,lower,x-1,search)
8. else if search > arr[x]
    1. binary_search(arr,x+1,upper,search)
9. stop
```

sorting algorithm

```
1. start
2. accept list of numbers , arr and length of list , len
3. set i = 0
4. repeat while i < len
    1. set j=0
    2. repeat while j < len - i - 1
        1. if arr[j] > arr[j+1] , then swap arr[i] and arr[j+1]
        2. j = j + 1
    3. i = i + 1
5. stop
```

Source Code

```c
#include<stdio.h>

// bubble sort
void sorting(int arr[],int x)
{ int temp;
    for(int i=0;i<x;i++)
    {
        for(int j=0;j<x-i-1;j++)
        {
            if(arr[j]>arr[j+1])
            {
                temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
            }
        }
    }
}

// recursive binary search
void searching(int arr[],int lower,int upper,int search)
{
    int x;

    if(lower>upper) {
        printf("\n Not found.");
        return;
    }

    x=(lower+upper)/2;
    if(arr[x]==search)
    {
        printf("\n Found at %d",x+1);
        return;
    }
    if(search<arr[x])
        searching(arr,lower,x-1,search);
    else if(search>arr[x])
        searching(arr,x+1,upper,search);
}

int main()
{
    int x,i,search;
    printf("Recursive binary search !\n \n Enter size of array : ");
    scanf("%d",&x);

    int arr[x];
    printf("\n Enter array elements : ");
    for(i=0;i<x;i++)
        scanf("%d",&arr[i]);

    sorting(arr,x);
    printf("\n The sorted array is : ");
    for(i=0;i<x;i++)
        printf("%d ",arr[i]);

    printf("\n Enter the element to search : ");
    scanf("%d",&search);

    searching(arr,0,x,search);

    return 0;
}
```