Aim:

To insert an element at a given position in an array.

Algorithm:

1. Read the number of elements n and the array.
2. Read the position and value to insert.
3. Shift elements from the end to the right until the position.
4. Insert the new value at the given position.
5. Print the updated array.

Program

```
#include <stdio.h>

Int main() {

   Int arr[100], n, i, pos, val;

   // Input number of elements

   Printf("Enter number of elements: ");

   Scanf("%d", &n);

   // Input array elements

   Printf("Enter elements:\n");

   For(i = 0; i < n; i++)

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

   // Input position and value to insert

   Printf("Enter position to insert (1 to %d): ", n + 1);

   Scanf("%d", &pos);

   Printf("Enter value to insert: ");

   Scanf("%d", &val);

   // Check for valid position

   If(pos < 1 || pos > n + 1) {
```

```
    Printf("Invalid position\n");
    Return 0;
  }
 // Shift elements to make space
  For(i = n; i >= pos; i--)
    Arr[i] = arr[i – 1];
  // Insert value
  Arr[pos – 1] = val;
  N++;
  // Print updated array
  printf("Array after insertion:\n");
  for(i = 0; i < n; i++)
    Printf("%d ", arr[i]);
  printf t("\n");
  return 0;
}
```

Aim:
To determine whether a given matrix is a **sparse matrix.**

Algorithm

1. Start
2. Input number of rows `m` and columns `n`
3. Input the matrix elements
4. Initialize `zero_count` to 0
5. For each element in the matrix:
    - If it is 0, increment `zero_count`
6. If `zero_count > (m × n) / 2`, then:
    - Print **"Matrix is sparse"**
    - Else print **"Matrix is not sparse"**

7. End

Program

```
#include <stdio.h>

int main() {

  int mat[10][10], m, n, i, j, zero_count = 0;

  printf("Enter number of rows and columns: ");

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


  printf("Enter elements:\n");

  forr(i = 0; i < m; i++) {

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

      Scanf("%d", &mat[i][j]);

    }

  }

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

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

      If(mat[i][j] == 0)

        Zero_count++;

    }

  }

  If(zero_count > (m * n) / 2)

    Printf("Matrix is sparse\n");

  else
```

```
    Printf("Matrix is not sparse\n");

  return 0;

}
```

Aim:

To create and display a simple singly linked list using an array.

Algorithm:

1. Start
2. Read the number of nodes n
3. Repeat for i = 1 to n:
   Read the value for the node.
   Create a new node with the value.
   If it is the first node, set it as head.
   Else, link it to the previous node.
4. Traverse the list from head and print each node's data.
5. End

Program

```
#include <stdio.h>

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

int main() {
  int n, i;
  struct Node nodes[100];  // Maximum 100 nodes
  struct Node *head = NULL, *temp = NULL;

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

```
    if(n > 100) {
        printf("Too many nodes (max 100 allowed)\n");
        return 1;
    }

    for(i = 0; i < n; i++) {
        printf("Enter value for node %d: ", i + 1);
        scanf("%d", &nodes[i].data);
        nodes[i].next = (i < n - 1) ? &nodes[i + 1] : NULL;
    }

    head = &nodes[0];

    printf("Linked list elements:\n");
    temp = head;
    while(temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }

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

Aim:

To implement stack operations using array

Algorithm: Stack using Array

1. Start
2. Initialize top = -1 and define a stack array.
3. Push operation

   If top == MAX – 1, display "Overflow".

   Else, increment top and insert the value.

4. Pop operation:

If top == -1, display “Underflow”.

Else, print and remove the top element.

5. Display operation

If top == -1, print “Stack is empty”.

Else, print elements from top to 0.

6. End

Program

```
#include <stdio.h>
int main() {
    int stack[100], top = -1;
    // Push operation
    if(top < 99) stack[++top] = 10;
    if(top < 99) stack[++top] = 20;
    if(top < 99) stack[++top] = 30;
    // Display stack
    if(top == -1) printf("Empty\n");
    else {
        for(int i = top; i >= 0; i--) printf("%d ", stack[i]);
        printf("\n");
    }
    // Pop operation
    if(top >= 0) printf("Popped: %d\n", stack[top--]);
    else printf("Underflow\n");
    // Display stack again
    if(top == -1) printf("Empty\n");
    else {
```

```
        for(int i = top; i >= 0; i--) printf("%d ", stack[i]);
        printf("\n");
    }
    return 0;
}
```

Aim:

To implement stack operations using linked list

Algorithm:

1. Start
2. Initialize top = NULL.
3. Push operation:

   Create a new node.

   Set its data.

   Point its next to current top.

   Update top to the new node.

4. Pop operation:

   If top is NULL, print “Underflow”.

   Else, print and delete the top node.

5. Display operation:

   Traverse from top and print each node’s data.

6. End

Program

```
#include <stdio.h>
struct Node {
    int data;
    struct Node* next;
```

```
};

int main() {

  struct Node n1 = {10, NULL}, n2 = {20, &n1}, n3 = {30, &n2};

  struct Node* top = &n3;

  for (struct Node* t = top; t; t = t->next) printf("%d ", t->data);

  printf("\n");

  if (top) {

    printf("Popped: %d\n", top->data);

    top = top->next;

  }

  for (struct Node* t = top; t; t = t->next) printf("%d ", t->data);

  printf("\n");

  return 0;

}
```

<u>Aim</u>:

To implement queue operations using an array

<u>Algorithm</u>:

1. Start
2. Initialize front = -1 and rear = -1.
3. Enqueue Operation:

If rear == SIZE – 1, print “Overflow”

Else:

If front == -1, set front = 0.

Insert value at ++rear.

4. Dequeue Operation:

If front == -1 or front > rear, print “Underflow”.

Else:

Print and remove the element at front++.

If front > rear, reset both to -1.

5. Display Operation:

If front == -1, print “Queue is Empty”.

Else print elements from front to rear.

6. End

Program

```
#include <stdio.h>

int q[100], f = -1, r = -1;

void enqueue(int v) {

  if (r == 99) printf("Overflow\n");

  else {

    if (f == -1) f = 0;

    q[++r] = v;

  }

}

void dequeue() {

  if (f == -1 || f > r) printf("Underflow\n");

  else {

    printf("Dequeued: %d\n", q[f++]);

    if (f > r) f = r = -1;

  }

}

void display() {

  if (f == -1) printf("Empty\n");
```

```
    else {
        for (int i = f; i <= r; i++) printf("%d ", q[i]);
        printf("\n");
    }
}
int main() {
    enqueue(10); enqueue(20); enqueue(30);
    display();
    dequeue();
    display();
}
```

Aim:

To implement a queue using a static linked list without dynamic memory allocation.

Algorithm:

1. Start
2. Create nodes with values and manually link them (static allocation)
3. Set front to first node, rear to last node
4. Display all nodes from front to rear
5. Dequeue: Move front to the next node
6. Display again from new front
7. End

Program

```
#include <stdio.h>
struct Node {
    int data;
    struct Node* next;
};
int main() {
```

```
    struct Node n1 = {10, NULL}, n2 = {20, NULL}, n3 = {30, NULL};
    struct Node *front = &n1, *rear = &n3;
    n1.next = &n2; n2.next = &n3;
    // Display
    for (struct Node* t = front; t; t = t->next) printf("%d ", t->data);
    printf("\n");
    // Dequeue
    printf("Dequeued: %d\n", front->data);
    front = front->next;
    // Display again
    for (struct Node* t = front; t; t = t->next) printf("%d ", t->data);
    printf("\n");
    return 0;
}
```

Aim:

To search an element in a sorted array using binary search.

Algorithm:

1. Read the array size n and elements (in sorted order).
2. Read the element to search key.
3. Set low = 0, high = n – 1.
4. While low <= high:

   Find mid = (low + high) / 2.

   If key == arr[mid], print found and stop.

   If key < arr[mid], set high = mid – 1.

   Else, set low = mid + 1.

5. If not found, print not found.

Program

```c
#include <stdio.h>

int main() {

   int a[100], n, k, l = 0, h, m;

   printf("Size & Elements: ");

   scanf("%d", &n);

   for(int i = 0; i < n; i++) scanf("%d", &a[i]);

   printf("Search: ");

   scanf("%d", &k);

   h = n - 1;

   while(l <= h) {

      m = (l + h) / 2;

      if(a[m] == k) { printf("Found at %d\n", m+1); return 0; }

      else if(a[m] < k) l = m + 1;

      else h = m - 1;

   }

   printf("Not found\n");

}
```

Aim:

To sort an array using Bubble Sort.

Algorithm:

1. Input number of elements and the array.
2. Repeat for each element (n-1 times):

   a. Compare each pair of adjacent elements.

   b. Swap if they are in the wrong order.

3. Display the sorted array.

Program

```
#include <stdio.h>

int main() {

  int a[100], n, i, j, t;

  scanf("%d", &n);

  for(i = 0; i < n; i++) scanf("%d", &a[i]);

  for(i = 0; i < n-1; i++)

    for(j = 0; j < n-i-1; j++)

      if(a[j] > a[j+1]) {

        t = a[j];

        a[j] = a[j+1];

        a[j+1] = t;

      }

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

return 0;

}
```

Aim:

To sort elements of an array using Quick Sort.

Algorithm:

1. Start
2. Input number of elements and the array.
3. Choose a pivot element.
4. Partition array:

   Move smaller elements to the left

   Larger elements to the right of pivot

5. Recursively apply steps 3–4 on left and right sub-arrays.
6. Display the sorted array.

7. End

Program

```
#include <stdio.h>
void quickSort(int a[], int low, int high) {
  if (low < high) {
    int i = low, j = high, pivot = a[low], t;
    while (i < j) {
      while (a[i] <= pivot && i < high) i++;
      while (a[j] > pivot) j--;
      if (i < j) t = a[i], a[i] = a[j], a[j] = t;
    }
    t = a[low], a[low] = a[j], a[j] = t;
    quickSort(a, low, j - 1);
    quickSort(a, j + 1, high);
  }
}
int main() {
  int a[100], n;
  printf("Enter no. of elements: ");
  scanf("%d", &n);
  for(int i = 0; i < n; i++) scanf("%d", &a[i]);
  quickSort(a, 0, n - 1);
  for(int i = 0; i < n; i++) printf("%d ", a[i]);
return 0;
}
```