1] Programs to implement Linear Search operations on matrix.

```
#include <stdio.h>

#define MAX_ROWS 10
#define MAX_COLS 10

void linearSearchMatrix(int rows, int cols, int matrix[MAX_ROWS][MAX_COLS], int
target) {     int found = 0;     for (int i = 0; i < rows; i++) {         for (int j
= 0; j < cols; j++) {             if (matrix[i][j] == target) {
printf("Element %d found at position (%d, %d)\n", target, i, j);
found = 1;                 break;
          }         }
if (found) {
break;
      }     }     if (!found) {         printf("Element %d not
found in the matrix\n", target);
   }
}

int main() {
int rows, cols;

   printf("Enter the number of rows and columns of the matrix (e.g., 3 3): ");
   if (scanf("%d %d", &rows, &cols) != 2 || rows <= 0 || cols <= 0 || rows > MAX_ROWS ||
cols > MAX_COLS) {
      printf("Invalid input for matrix size.\n");
return 1;
   }
```

```
    int matrix[MAX_ROWS][MAX_COLS];

    printf("Enter the elements of the matrix:\n");     for
(int i = 0; i < rows; i++) {         for (int j = 0; j < cols;
j++) {             if (scanf("%d", &matrix[i][j]) != 1) {
printf("Invalid input for matrix element.\n");
return 1;
            }
        }
    }

    int target;

    printf("Enter the element to search: ");     if
(scanf("%d", &target) != 1) {         printf("Invalid
input for the target element.\n");         return 1;
    }

    linearSearchMatrix(rows, cols, matrix, target);

    return 0;
}
```

2] Programs to implement Binary Search operations on matrix.

```
#include <stdio.h>

#define MAX_ROWS 10
#define MAX_COLS 10
```

```
int binarySearch(int arr[], int size, int target) {
int low = 0;     int high = size - 1;

   while (low <= high) {         int
mid = low + (high - low) / 2;

       if (arr[mid] == target) {
return mid;
       } else if (arr[mid] < target) {
low = mid + 1;
       } else {
high = mid - 1;
       }
   }

   return -1;
}

void flattenMatrix(int rows, int cols, int matrix[MAX_ROWS][MAX_COLS], int
arr[MAX_ROWS * MAX_COLS]) {
   int index = 0;     for (int i = 0; i
< rows; i++) {         for (int j = 0; j
< cols; j++) {            arr[index++]
= matrix[i][j];         }

   }
}
```

```
void binarySearchMatrix(int rows, int cols, int matrix[MAX_ROWS][MAX_COLS], int
target) {
   int flattened[MAX_ROWS * MAX_COLS];
   flattenMatrix(rows, cols, matrix, flattened);

   int result = binarySearch(flattened, rows * cols, target);

   if (result != -1) {         int row = result / cols;         int col = result %
cols;         printf("Element %d found at position (%d, %d)\n", target,
row, col);
   } else {         printf("Element %d not found in the
matrix\n", target);
   }
}

int main() {
int rows, cols;

   printf("Enter the number of rows and columns of the matrix (e.g., 3 3): ");
   if (scanf("%d %d", &rows, &cols) != 2 || rows <= 0 || cols <= 0 || rows > MAX_ROWS ||
cols > MAX_COLS) {
      printf("Invalid input for matrix size.\n");
return 1;
   }

   int matrix[MAX_ROWS][MAX_COLS];

   printf("Enter the elements of the sorted matrix:\n");
for (int i = 0; i < rows; i++) {         for (int j = 0; j <
```

```
cols; j++) {            if (scanf("%d", &matrix[i][j]) !=
1) {                printf("Invalid input for matrix
element.\n");               return 1;
          }
       }
   }

   int target;

   printf("Enter the element to search: ");     if
(scanf("%d", &target) != 1) {         printf("Invalid
input for the target element.\n");         return 1;
   }

   binarySearchMatrix(rows, cols, matrix, target);

   return 0;
}
```

3.Programs to implement Sentinel Search operations on matrix.

```
#include <stdio.h>

#define MAX_ROWS 10
#define MAX_COLS 10

void sentinelSearchMatrix(int rows, int cols, int matrix[MAX_ROWS][MAX_COLS], int
```

```
target) {     for (int i = 0; i <
rows; i++) {
matrix[i][cols] = target;
    }

    int i = 0, j = 0;

    while (matrix[i][j] != target) {
if (j == cols) {             i++;
j = 0;         } else {             j++;
        }
    }

    if (i < rows) {         printf("Element %d found at position (%d,
%d)\n", target, i, j);
    } else {         printf("Element %d not found in the
matrix\n", target);
    }
}

int main() {
int rows, cols;

    printf("Enter the number of rows and columns of the matrix (e.g., 3 3): ");
    if (scanf("%d %d", &rows, &cols) != 2 || rows <= 0 || cols <= 0 || rows > MAX_ROWS ||
cols > MAX_COLS) {
        printf("Invalid input for matrix size.\n");
return 1;
    }
```

```
    int matrix[MAX_ROWS][MAX_COLS];

    printf("Enter the elements of the matrix:\n");     for
(int i = 0; i < rows; i++) {         for (int j = 0; j < cols;
j++) {             if (scanf("%d", &matrix[i][j]) != 1) {
printf("Invalid input for matrix element.\n");
return 1;
            }
        }
    }

    int target;

    printf("Enter the element to search: ");     if
(scanf("%d", &target) != 1) {         printf("Invalid
input for the target element.\n");         return 1;
    }

    sentinelSearchMatrix(rows, cols, matrix, target);

    return 0;
}
```

4).Programs to implement Fibonacci Search operations on matrix.

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

```
#define MAX_ROWS 10
#define MAX_COLS 10

int fibonacciSearch(int arr[], int size, int target) {
int fibM2 = 0;     int fibM1 = 1;     int fib =
fibM1 + fibM2;

   while (fib < size) {
fibM2 = fibM1;
fibM1 = fib;         fib =
fibM1 + fibM2;
   }

   int offset = -1;

   while (fib > 1) {         int i = (offset + fibM2 < size - 1) ?
offset + fibM2 : size - 1;

      if (arr[i] == target) {
return i;
      } else if (arr[i] < target) {
fib = fibM1;            fibM1 =
fibM2;            fibM2 = fib -
fibM1;            offset = i;
      } else {            fib =
fibM2;            fibM1 -=
fibM2;            fibM2 = fib -
fibM1;
```

```
        }
    }

    if (fibM1 && arr[offset + 1] == target) {
return offset + 1;
    }

    return -1;
}

void flattenMatrix(int rows, int cols, int matrix[MAX_ROWS][MAX_COLS], int
arr[MAX_ROWS * MAX_COLS]) {
    int index = 0;     for (int i = 0; i
< rows; i++) {         for (int j = 0; j
< cols; j++) {             arr[index++]
= matrix[i][j];
        }
    }
}

void fibonacciSearchMatrix(int rows, int cols, int matrix[MAX_ROWS][MAX_COLS], int
target) {
    int flattened[MAX_ROWS * MAX_COLS];
    flattenMatrix(rows, cols, matrix, flattened);

    int result = fibonacciSearch(flattened, rows * cols, target);

    if (result != -1) {         int row = result / cols;         int col = result %
cols;         printf("Element %d found at position (%d, %d)\n", target,
row, col);
```

```
    } else {        printf("Element %d not found in the
matrix\n", target);
    }
}

int main() {
int rows, cols;

    printf("Enter the number of rows and columns of the matrix (e.g., 3 3): ");
    if (scanf("%d %d", &rows, &cols) != 2 || rows <= 0 || cols <= 0 || rows > MAX_ROWS ||
cols > MAX_COLS) {
        printf("Invalid input for matrix size.\n");
return 1;
    }

    int matrix[MAX_ROWS][MAX_COLS];

    printf("Enter the elements of the matrix:\n");     for
(int i = 0; i < rows; i++) {        for (int j = 0; j < cols;
j++) {            if (scanf("%d", &matrix[i][j]) != 1) {
printf("Invalid input for matrix element.\n");
return 1;
            }
        }
    }

    int target;
```

```
    printf("Enter the element to search: ");     if
(scanf("%d", &target) != 1) {         printf("Invalid
input for the target element.\n");         return 1;
    }

    fibonacciSearchMatrix(rows, cols, matrix, target);

    return 0;
}
```

5] Programs to implement Bubble sorting techniques to sort an array of 0s, 1s and 2s an Array.

```
#include <stdio.h>

void bubbleSort(int arr[], int size) {
for (int i = 0; i < size - 1; i++) {
for (int j = 0; j < size - i - 1; j++) {
if (arr[j] > arr[j + 1]) {                int
temp = arr[j];                arr[j] = arr[j +
1];                arr[j + 1] = temp;
            }
        }
    }
}

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

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

int main() {
    int size;

    printf("Enter the size of the array: ");     if
(scanf("%d", &size) != 1 || size <= 0) {
printf("Invalid input for array size.\n");
return 1;
    }

    int arr[size];

    printf("Enter the elements of the array (0s, 1s, and 2s only):\n");     for (int i
= 0; i < size; i++) {         if (scanf("%d", &arr[i]) != 1 || (arr[i] != 0 && arr[i] !=
1 && arr[i] != 2)) {             printf("Invalid input for array element. Please enter
0, 1, or 2.\n");             return 1;
        }
    }

    bubbleSort(arr, size);

    printf("Sorted array: ");
printArray(arr, size);     return
0;
}
```

6] Programs to implement Merge sorting techniques to sort an array of 0s, 1s and 2s an Array.

```
#include <stdio.h>

void merge(int arr[], int left, int mid, int right) {
int n1 = mid - left + 1;     int n2 = right - mid;

   int leftArr[n1], rightArr[n2];

   for (int i = 0; i < n1; i++)
leftArr[i] = arr[left + i];     for (int
j = 0; j < n2; j++)        rightArr[j]
= arr[mid + 1 + j];

   int i = 0, j = 0, k = left;

   while (i < n1 && j < n2) {
if (leftArr[i] <= rightArr[j]) {
arr[k] = leftArr[i];            i++;
} else {            arr[k] =
rightArr[j];            j++;        }
k++;
   }

   while (i < n1) {
arr[k] = leftArr[i];
i++;

       k++;
```

```
    }

    while (j < n2) {
arr[k] = rightArr[j];
j++;        k++;
    }
}

void mergeSort(int arr[], int left, int right) {
if (left < right) {         int mid = left + (right
- left) / 2;         mergeSort(arr, left, mid);
mergeSort(arr, mid + 1, right);
merge(arr, left, mid, right);
    }
}

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

int main() {
    int size;

    printf("Enter the size of the array: ");
```

```
    if (scanf("%d", &size) != 1 || size <= 0) {
printf("Invalid input for array size.\n");          return
1;
    }

    int arr[size];

    printf("Enter the elements of the array (0s, 1s, and 2s only):\n");      for (int i
= 0; i < size; i++) {          if (scanf("%d", &arr[i]) != 1 || (arr[i] != 0 && arr[i] !=
1 && arr[i] != 2)) {              printf("Invalid input for array element. Please enter
0, 1, or 2.\n");              return 1;
        }
    }

    mergeSort(arr, 0, size - 1);

    printf("Sorted array: ");
printArray(arr, size);

    return 0;
}
```

7] Programs to implement Selection sorting techniques to sort an array of 0s, 1s and 2s an Array.

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

```
void selectionSort(int arr[], int size) {
for (int i = 0; i < size - 1; i++) {
int minIndex = i;
      for (int j = i + 1; j < size; j++) {
if (arr[j] < arr[minIndex]) {
minIndex = j;
          }
       }

       int temp = arr[minIndex];
arr[minIndex] = arr[i];         arr[i]
= temp;
   }
}

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

int main() {
   int size;

   printf("Enter the size of the array: ");     if
(scanf("%d", &size) != 1 || size <= 0) {
```

```
printf("Invalid input for array size.\n");
return 1;
    }

    int arr[size];

    printf("Enter the elements of the array (0s, 1s, and 2s only):\n");      for (int i
= 0; i < size; i++) {         if (scanf("%d", &arr[i]) != 1 || (arr[i] != 0 && arr[i] !=
1 && arr[i] != 2)) {             printf("Invalid input for array element. Please enter
0, 1, or 2.\n");             return 1;
        }
    }

    selectionSort(arr, size);

    printf("Sorted array: ");
printArray(arr, size);

    return 0;
}
```

8] Programs to implement Quick sorting techniques to sort an array of 0s, 1s and 2s an Array.

```
#include <stdio.h>

void partition(int arr[], int low, int high, int *left, int *right) {
int pivot = arr[low];
```

```
    *left = low + 1;
    *right = high;

    while (*left <= *right) {         while (*left <=
high && arr[*left] <= pivot)
            (*left)++;

        while (*right > low && arr[*right] > pivot)
            (*right)--;

        if (*left < *right) {
int temp = arr[*left];
arr[*left] = arr[*right];
arr[*right] = temp;
        }
    }

    int temp = arr[low];
arr[low] = arr[*right];
arr[*right] = temp;
}

void quickSort(int arr[], int low, int high) {
if (low < high) {         int left, right;
partition(arr, low, high, &left, &right);
quickSort(arr, low, right - 1);
quickSort(arr, right + 1, high);
    }
}
```

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

int main() {
    int size;

    printf("Enter the size of the array: ");     if
(scanf("%d", &size) != 1 || size <= 0) {
printf("Invalid input for array size.\n");
return 1;
    }

    int arr[size];

    printf("Enter the elements of the array (0s, 1s, and 2s only):\n");     for (int i
= 0; i < size; i++) {         if (scanf("%d", &arr[i]) != 1 || (arr[i] != 0 && arr[i] !=
1 && arr[i] != 2)) {             printf("Invalid input for array element. Please enter
0, 1, or 2.\n");             return 1;
        }
    }

    quickSort(arr, 0, size - 1);
```

```
    printf("Sorted array: ");
printArray(arr, size);

    return 0;
}
```

9] Programs to implement stack using array

```
#include <stdio.h>

#define MAX_SIZE 10

struct Stack {     int
arr[MAX_SIZE];
    int top;
};

void initialize(struct Stack *stack) {     stack-
>top = -1;
}

int isEmpty(struct Stack *stack) {
return stack->top == -1;
}

int isFull(struct Stack *stack) {
return stack->top == MAX_SIZE - 1;
}
```

```
void push(struct Stack *stack, int value) {     if (isFull(stack)) {
printf("Stack Overflow: Cannot push element %d, stack is full.\n", value);
   } else {         stack->arr[++stack->top] = value;
printf("Pushed %d onto the stack.\n", value);
   }
}

int pop(struct Stack *stack) {     if (isEmpty(stack)) {
printf("Stack Underflow: Cannot pop from an empty stack.\n");
       return -1;     } else {         int poppedValue = stack-
>arr[stack->top--];         printf("Popped %d from the
stack.\n", poppedValue);         return poppedValue;
   }
}

void display(struct Stack *stack) {
if (isEmpty(stack)) {
printf("Stack is empty.\n");
   } else {         printf("Stack elements:
");         for (int i = 0; i <= stack->top;
i++) {             printf("%d ", stack-
>arr[i]);
       }
printf("\n");
   }
}
```

```
int main() {     struct
Stack stack;
initialize(&stack);

   push(&stack, 10);
push(&stack, 20);     push(&stack,
30);

   display(&stack);

   pop(&stack);
pop(&stack);

   display(&stack);

   push(&stack, 40);
push(&stack, 50);

   display(&stack);

   return 0;
}
```

10] Programs to implement stack using linked list.

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

struct Node {
```

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

struct Stack {
struct Node* top;
};

void initialize(struct Stack* stack) {     stack-
>top = NULL;
}

int isEmpty(struct Stack* stack) {
return stack->top == NULL;
}

struct Node* createNode(int data) {     struct Node* newNode = (struct
Node*)malloc(sizeof(struct Node));     if (newNode == NULL) {
printf("Memory allocation failed.\n");         exit(EXIT_FAILURE);
   }
   newNode->data   =   data;
newNode->next   =   NULL;
return newNode;
}

void push(struct Stack* stack, int value) {
struct Node* newNode = createNode(value);
newNode->next = stack->top;     stack->top =
```

```
newNode;     printf("Pushed %d onto the
stack.\n", value);
}

int pop(struct Stack* stack) {     if (isEmpty(stack)) {
printf("Stack Underflow: Cannot pop from an empty stack.\n");
        return -1;
    } else {         struct Node* temp = stack->top;         int
poppedValue = temp->data;         stack->top = temp-
>next;         free(temp);         printf("Popped %d from the
stack.\n", poppedValue);         return poppedValue;
    }
}

void display(struct Stack* stack) {
if (isEmpty(stack)) {
printf("Stack is empty.\n");
    } else {         printf("Stack elements:
");         struct Node* current = stack-
>top;         while (current != NULL) {
printf("%d ", current->data);
current = current->next;
        }
printf("\n");
    }
}
```

```
void freeStack(struct Stack* stack) {
while (!isEmpty(stack)) {
pop(stack);
   }
}

int main() {     struct
Stack stack;
initialize(&stack);

         push(&stack, 10);
        push(&stack, 20);
        push(&stack, 30);

   display(&stack);

   pop(&stack);
pop(&stack);

   display(&stack);

         push(&stack, 40);
        push(&stack, 50);

   display(&stack);

   freeStack(&stack);
```

```
    return 0; }
```

11] Programs to implement queue using array.

```
#include <stdio.h>

#define MAX_SIZE 10

struct Queue {     int
arr[MAX_SIZE];
    int front, rear;
};

void initialize(struct Queue* queue) {
queue->front = -1;     queue->rear = -
1;
}

int isEmpty(struct Queue* queue) {
return queue->front == -1;
}

int isFull(struct Queue* queue) {     return (queue->rear + 1)
% MAX_SIZE == queue->front;
}
```

```
void enqueue(struct Queue* queue, int value) {     if (isFull(queue)) {
printf("Queue Overflow: Cannot enqueue element %d, queue is full.\n", value);
   } else {         if
(isEmpty(queue)) {
queue->front = 0;
       }
       queue->rear = (queue->rear + 1) % MAX_SIZE;
queue->arr[queue->rear] = value;         printf("Enqueued
%d into the queue.\n", value);
   }
}

int dequeue(struct Queue* queue) {     if (isEmpty(queue)) {
printf("Queue Underflow: Cannot dequeue from an empty queue.\n");
       return -1;     } else {         int dequeuedValue =
queue->arr[queue->front];         if (queue->front ==
queue->rear) {             queue->front = -1;
queue->rear = -1;
       } else {
           queue->front = (queue->front + 1) % MAX_SIZE;
       }
       printf("Dequeued %d from the queue.\n", dequeuedValue);
return dequeuedValue;
   }
}
```

```
void display(struct Queue* queue) {
if (isEmpty(queue)) {
printf("Queue is empty.\n");
   } else {         printf("Queue
elements: ");         int i = queue-
>front;         do {
            printf("%d ", queue->arr[i]);
i = (i + 1) % MAX_SIZE;
        } while (i != (queue->rear + 1) % MAX_SIZE);
        printf("\n");
   }
}

int main() {     struct
Queue queue;
initialize(&queue);

   enqueue(&queue, 10);
enqueue(&queue, 20);     enqueue(&queue,
30);

   display(&queue);

   dequeue(&queue);
dequeue(&queue);

   display(&queue);
```

```
    enqueue(&queue, 40);
enqueue(&queue, 50);

    display(&queue);

    return 0;
}
```

12] Programs to implement queue using linked list.

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

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

struct Queue {
struct Node* front;
struct Node* rear;
};

void initialize(struct Queue* queue) {
queue->front = NULL;     queue->rear
= NULL;
}

int isEmpty(struct Queue* queue) {
return queue->front == NULL;
```

```
}

struct Node* createNode(int data) {     struct Node* newNode = (struct
Node*)malloc(sizeof(struct Node));     if (newNode == NULL) {
printf("Memory allocation failed.\n");         exit(EXIT_FAILURE);
    }
    newNode->data = data;
    newNode->next = NULL;
return newNode;
}

void enqueue(struct Queue* queue, int value) {
struct Node* newNode = createNode(value);
if (isEmpty(queue)) {         queue->front =
newNode;         queue->rear = newNode;
    } else {         queue->rear->next =
newNode;         queue->rear =
newNode;
    }
    printf("Enqueued %d into the queue.\n", value);
}

int dequeue(struct Queue* queue) {     if (isEmpty(queue)) {
printf("Queue Underflow: Cannot dequeue from an empty queue.\n");
        return -1;     } else {         struct Node* temp = queue-
>front;         int dequeuedValue = temp->data;         queue-
>front = temp->next;         free(temp);         printf("Dequeued
```

```
%d from the queue.\n", dequeuedValue);        return
dequeuedValue;
    }
}

void display(struct Queue* queue) {
if (isEmpty(queue)) {
printf("Queue is empty.\n");
    } else {        printf("Queue elements:
");        struct Node* current = queue-
>front;        while (current != NULL) {
printf("%d ", current->data);
current = current->next;
        }
printf("\n");
    }
}

void freeQueue(struct Queue* queue) {
while (!isEmpty(queue)) {
dequeue(queue);
    }
}

int main() {     struct
Queue queue;
initialize(&queue);
```

```
    enqueue(&queue, 10);
enqueue(&queue, 20);     enqueue(&queue,
30);

    display(&queue);

    dequeue(&queue);
dequeue(&queue);

    display(&queue);

    enqueue(&queue, 40);
enqueue(&queue, 50);

    display(&queue);

    freeQueue(&queue);

    return 0;
}
```

13] Programs to implement create, add, and delete operations on linked list

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

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

```
};

struct Node* createNode(int data) {     struct Node* newNode = (struct
Node*)malloc(sizeof(struct Node));     if (newNode == NULL) {
printf("Memory allocation failed.\n");         exit(EXIT_FAILURE);
    }
   newNode->data   =   data;
newNode->next   =   NULL;
return newNode;
}

void addAtEnd(struct Node** head, int data) {
struct Node* newNode = createNode(data);     if
(*head == NULL) {
        *head = newNode;     } else {
struct Node* current = *head;
while (current->next != NULL) {
current = current->next;
        }
        current->next = newNode;
   }
}

void addAtBeginning(struct Node** head, int data) {
struct Node* newNode = createNode(data);     newNode-
>next = *head;
   *head = newNode;
}
```

```
void deleteNode(struct Node** head, int value) {
if (*head == NULL) {         printf("Linked list is
empty.\n");         return;
    }

    struct Node* current = *head;
struct Node* prev = NULL;

    while (current != NULL && current->data != value) {
prev = current;         current = current->next;
    }

    if (current == NULL) {         printf("Node with
value %d not found.\n", value);
        return;
    }

    if (prev == NULL) {
        *head = current->next;
    } else {         prev->next =
current->next;
    }

    free(current);     printf("Node with value %d
deleted.\n", value);
}
```

```
void display(struct Node* head) {
printf("Linked list: ");     while
(head != NULL) {
printf("%d ", head->data);
head = head->next;
   }
   printf("\n");
}

void freeLinkedList(struct Node** head) {
while (*head != NULL) {         struct
Node* temp = *head;         *head =
(*head)->next;         free(temp);
   }
}

int main() {     struct Node*
linkedList = NULL;

   addAtEnd(&linkedList, 10);
addAtEnd(&linkedList, 20);     addAtEnd(&linkedList,
30);

   display(linkedList);

   addAtBeginning(&linkedList, 5);
addAtEnd(&linkedList, 40);
```

```
    display(linkedList);


    deleteNode(&linkedList, 20);

deleteNode(&linkedList, 5);


    display(linkedList);     freeLinkedList(&linkedList);


    return 0; }
```

14] Programs to implement BFS.

```
#include<stdio.h>
#define size 20

int q[size],front=-1,rear=-1,a[size][size], vis[size];
int delete();
void add(int item);
void bfs(int s,int n);

int main()
{
  int n,i,s,ch,j;
  printf("Enter the Number of Vertices: ");
  scanf("%d",&n);
  printf("Enter graph data in matrix form:    \n");
  for(i=1;i<=n;i++)
  {
  for(j=1;j<=n;j++)
  {   scanf("%d",&a[i][j]);
  }
  }
  printf("Enter the Source Vertex :");
  scanf("%d",&s);
  printf("\nThe nodes visited in BFS as :\n");
  bfs(s,n);
  return 0;
}//end of main()


void bfs(int s,int n)
{
    int p,i;
    add(s);
```

```
    vis[s]=1;
    p=delete();
    if(p!=0)
    printf(" %d",p);
    while(p!=0)
    {
    for(i=1;i<=n;i++)
    if((a[p][i]!=0)&&(vis[i]==0))
    {
    add(i);
    vis[i]=1;
    }
    p=delete();
    if(p!=0)
    printf(" %d ",p);
    }
    for(i=1;i<=n;i++)
    if(vis[i]==0)
    bfs(i,n);
}//end of bfs()


void add(int item)
{
    if(rear==size-1)
    printf("QUEUE FULL");
    else
    {
    if(rear==-1)
    {
    q[++rear]=item;
    front++;
    }
    else
    q[++rear]=item;
    }
}//End of add()

int delete()
{
    int k;
    if((front>rear)||(front==-1))
    return(0);
    else
    {
    k=q[front++];
    return(k);
    }
}//end of delete()
```

15] Programs to implement DFS.

```
#include<stdio.h>
#define size 20

int top=-1,a[size][size],vis[size],stack[size];

void dfs(int s,int n);
void push(int item);
int pop();

int main()
{
  int n,i,s,ch,j;
  printf("Enter the number of vertices: ");
  scanf("%d",&n);

  printf("Enter graph data in matrix form:    \n");
  for(i=1;i<=n;i++)
  {
  for(j=1;j<=n;j++)
  {
  scanf("%d",&a[i][j]);
  }
  }
  printf("Enter the Source Vertex:");
  scanf("%d",&s);
  printf("\nThe nodes visited in DFS as :\n");
  dfs(s,n);
  return 0;
}//End of main()

void dfs(int s,int n)
{
int i,k;
push(s);
vis[s]=1;
k=pop();
if(k!=0)
printf(" %d ",k);
while(k!=0)
{
for(i=1;i<=n;i++)
if((a[k][i]!=0)&&(vis[i]==0))
{
push(i);
vis[i]=1;
}
k=pop();
if(k!=0)
```

```
printf(" %d ",k);
}
for(i=1;i<=n;i++)
if(vis[i]==0)
dfs(i,n);
}//End of dfs()

void push(int item)
{
  if(top==size-1)
  printf("Stack overflow ");
  else
  stack[++top]=item;
}//End of push()

int pop()
{
  int k;
  if(top==-1)
  return(0);
  else
  {
  k=stack[top--];
  return(k);
  }
}//End of pop()
```

16] Programs to implement Memory efficient linked list.

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

// Define a structure for a memory-efficient node
struct Node {
    unsigned int data : 30; // 30 bits for data
    bool isEnd : 1;         // 1 bit to indicate the end of the list
    struct Node* next;      // Pointer to the next node
};

// Function to create a new node
struct Node* createNode(unsigned int data, bool isEnd) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode != NULL) {
        newNode->data = data;
```

```
        newNode->isEnd = isEnd;
        newNode->next = NULL;
    }
    return newNode;
}

// Function to insert a new node at the end of the list
void insert(struct Node** head, unsigned int data) {
    struct Node* newNode = createNode(data, false);
    if (*head == NULL) {
        *head = newNode;
    } else {
        struct Node* temp = *head;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

// Function to display the linked list
void display(struct Node* head) {
    printf("Linked List: ");
    while (head != NULL) {
        printf("%u -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

// Function to free memory allocated for the linked list
void freeLinkedList(struct Node* head) {
    struct Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}

int main() {
    struct Node* head = NULL;

    // Insert elements into the memory-efficient linked list
    insert(&head, 10);
    insert(&head, 20);
    insert(&head, 30);
    insert(&head, 40);

    // Display the linked list
```

```
    display(head);

    // Free memory allocated for the linked list
    freeLinkedList(head);

    return 0;
}
```

17] Programs to implement Quick sorting techniques to sort an array of 0s, 1s and 2s an Array.

```
#include <stdio.h>

// Function to swap two elements
void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

// Function to sort an array of 0s, 1s, and 2s using quicksort
void sortColors(int* nums, int numsSize) {
    int low = 0, mid = 0, high = numsSize - 1;

    while (mid <= high) {
        switch (nums[mid]) {
            case 0:
                swap(&nums[low], &nums[mid]);
                low++;
                mid++;
                break;
            case 1:
                mid++;
                break;
            case 2:
                swap(&nums[mid], &nums[high]);
                high--;
                break;
        }
    }
}

// Function to print the elements of an array
void printArray(int* nums, int numsSize) {
    printf("Sorted Array: ");
    for (int i = 0; i < numsSize; i++) {
        printf("%d ", nums[i]);
    }
    printf("\n");
}
```

```
int main() {
    int nums[] = {2, 0, 1, 2, 1, 0};
    int numsSize = sizeof(nums) / sizeof(nums[0]);

    printf("Original Array: ");
    for (int i = 0; i < numsSize; i++) {
        printf("%d ", nums[i]);
    }
    printf("\n");

    sortColors(nums, numsSize);
    printArray(nums, numsSize);

    return 0;
}
```