1.Sparse Matrix

```
#include <stdio.h>


void SparseMatrix(int row, int col, int matrix[row][col]) {
  int size = 0;


  for (int i = 0; i < row; i++)
    for (int j = 0; j < col; j++)
      if (matrix[i][j] != 0) size++;


  int sparseMatrix[size][3];
  int k = 0;
  for (int i = 0; i < row; i++) {
    for (int j = 0; j < col; j++) {
      if (matrix[i][j] != 0) {
        sparseMatrix[k][0] = i;
        sparseMatrix[k][1] = j;
        sparseMatrix[k][2] = matrix[i][j];
        k++;
      }
    }
  }


  printf("Satir\tSutun\tDeger\n");
  for (int i = 0; i < size; i++) {
    printf("%d\t%d\t%d\n", sparseMatrix[i][0], sparseMatrix[i][1], sparseMatrix[i][2]);
  }
}
```

2.Hanaoi Kuleleri

```
 void hanoi(int n, char from_rod, char to_rod, char a_rod) {
    if (n == 1) {
        printf("\n Disk 1 şuradan: %c şuraya taşındı: %c", from_rod, to_rod);
        return;
    }
    hanoi(n - 1, from_rod, a_rod, to_rod);
    printf("\n Disk %d şuradan: %c şuraya taşındı: %c", n, from_rod, to_rod);
    hanoi(n - 1, a_rod, to_rod, from_rod);
}
```

3. Shunting Yard (Infix -> Postfix)

```
int precedence(char x) {
    if (x == '+' || x == '-') return 1;
    if (x == '*' || x == '/') return 2;
    return 0;
}

void infixToPostfix(char* exp) {
    char stack[100];
    int top = -1;
    char result[100];
    int k = 0;

    for (int i = 0; exp[i]; i++) {
        if (isalnum(exp[i]))
            result[k++] = exp[i];
        else if (exp[i] == '(')
        stack[++top] = exp[i];
      else if (exp[i] == ')') {
        while (top != -1 && stack[top] != '(')
          result[k++] = stack[top--];
        top--;
      } else {
        while (top != -1 && precedence(stack[top]) >= precedence(exp[i]))
          result[k++] = stack[top--];
        stack[++top] = exp[i];
      }
    }
    while (top != -1) result[k++] = stack[top--];
    result[k] = '\0';
    printf("Postfix: %s\n", result);
}
```

4. Max Heap (Heapify)

```
void swap(int *a, int *b) { int temp = *a; *a = *b; *b = temp; }

void heapify(int arr[], int n, int i) { int largest = i; int left = 2 * i + 1; int right = 2 * i + 2;

if (left < n && arr[left] > arr[largest])
  largest = left;
if (right < n && arr[right] > arr[largest])
  largest = right;

if (largest != i) {
  swap(&arr[i], &arr[largest]);
  heapify(arr, n, largest);
}

}
```

5. Doubly Linked List (Çift Yönlü Bağlı Liste)

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

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

void push(struct Node** head_ref, int new_data) { struct Node* new_node = (struct Node*)malloc(sizeof(struct Node)); new_node->data = new_data; new_node->next = (*head_ref); new_node->prev = NULL; if ((*head_ref) != NULL) (*head_ref)->prev = new_node; (*head_ref) = new_node; }

void deleteNode(struct Node** head_ref, struct Node* del) { if (*head_ref == NULL || del == NULL) return; if (*head_ref == del) *head_ref = del->next; if (del->next != NULL) del->next->prev = del->prev; if (del->prev != NULL) del->prev->next = del->next; free(del); }

void printList(struct Node* node) { while (node != NULL) { printf("%d ", node->data); node = node->next; }

}
```

6. Çembersel (Circular) Linked List İşlemleri

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

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

void insert(struct Node** head_ref, int data) { struct Node* ptr1 = (struct Node*)malloc(sizeof(struct Node)); struct Node* temp = *head_ref; ptr1->data = data; ptr1->next = *head_ref;

if (*head_ref != NULL) {
   while (temp->next != *head_ref) temp = temp->next;
   temp->next = ptr1;
} else {
   ptr1->next = ptr1;
   *head_ref = ptr1;
}


}

void printCircular(struct Node* head) { struct Node* temp = head; if (head != NULL) { do { printf("%d ", temp->data); temp = temp->next; } while (temp != head); } }
```

7. Array Ekleme ve Silme

```
#include <stdio.h>


void deleteElement(int arr[], int *n, int pos) {

    if (pos >= *n || pos < 0) return;

    for (int i = pos; i < *n - 1; i++)

        arr[i] = arr[i + 1];

    (*n)--;

}


void insertElement(int arr[], int *n, int x, int pos) {

    for (int i = *n - 1; i >= pos; i--)

        arr[i + 1] = arr[i];

    arr[pos] = x;

    (*n)++;

}
```

8. Array Graph Cluster Çıkarma

```
#include <stdio.h>

#define V 5
```

```
void DFS_Cluster(int M[][V], int row, int visited[]) {
  visited[row] = 1;
  printf("%d ", row);
  for (int i = 0; i < V; i++) {
    if (M[row][i] == 1 && !visited[i]) {
      DFS_Cluster(M, i, visited);
    }
  }
}

void findClusters(int M[][V]) {
  int visited[V] = {0};
  int clusterCount = 0;
  for (int i = 0; i < V; i++) {
    if (!visited[i]) {
      printf("\nCluster %d: ", ++clusterCount);
      DFS_Cluster(M, i, visited);
    }
  }
}
```

9. DFS ve BFS Uygulaması

```
#include <stdio.h>
#define V 5
```

```
void BFS(int graph[V][V], int startNode) {
  int visited[V] = {0};
  int queue[V], front = 0, rear = 0;

  visited[startNode] = 1;
  queue[rear++] = startNode;

  while (front < rear) {
    int current = queue[front++];
    printf("%d ", current);

    for (int i = 0; i < V; i++) {
      if (graph[current][i] == 1 && !visited[i]) {
        visited[i] = 1;
        queue[rear++] = i;
      }
    }
  }
}
```

10. Tree Creation Using Array

```
#include <stdio.h> #include <stdlib.h>
struct TreeNode { int data; struct TreeNode *left, *right; };
struct TreeNode* newNode(int data) { struct TreeNode* node = (struct
TreeNode*)malloc(sizeof(struct TreeNode)); node->data = data; node->left = node-
>right = NULL; return node; }

struct TreeNode* createTree(int arr[], int i, int n) { struct TreeNode* root = NULL; if (i < n)
{ root = newNode(arr[i]); root->left = createTree(arr, 2 * i + 1, n); root->right =
createTree(arr, 2 * i + 2, n); } return root; }
```