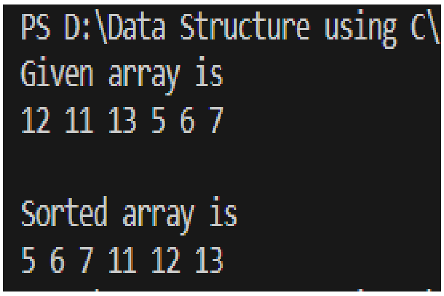
Write a program to demonstrate Merge Sort:

```
#include <stdio.h>


void merge(int arr[], int l, int m, int r) {
   int n1 = m - l + 1;
   int n2 = r - m;


   int L[n1], R[n2];


   for (int i = 0; i < n1; i++)
      L[i] = arr[l + i];
   for (int j = 0; j < n2; j++)
      R[j] = arr[m + 1 + j];


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


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


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


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

void mergeSort(int arr[], int l, int r) {
   if (l < r) {
       int m = l + (r - l) / 2;

       mergeSort(arr, l, m);
       mergeSort(arr, m + 1, r);

       merge(arr, l, m, r);
   }
}

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

int main() {
   int arr[] = {12, 11, 13, 5, 6, 7};
   int arr_size = sizeof(arr) / sizeof(arr[0]);

   printf("Given array is \n");
   printArray(arr, arr_size);

   mergeSort(arr, 0, arr_size - 1);

   printf("\nSorted array is \n");
   printArray(arr, arr_size);
   return 0; }
```

Output

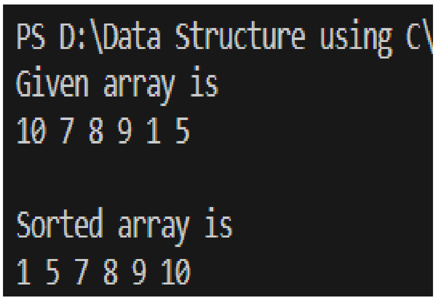


```
PS D:\Data Structure using C\
Given array is
12 11 13 5 6 7

Sorted array is
5 6 7 11 12 13
```

Write a program to demonstrate Quick Sort:

```
#include <stdio.h>


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


// Function to partition the array and return the pivot index
int partition(int arr[], int low, int high) {
    int pivot = arr[high]; // Pivot element
    int i = (low - 1); // Index of smaller element


    for (int j = low; j <= high - 1; j++) {
        // If the current element is smaller than or equal to the pivot
        if (arr[j] <= pivot) {
            i++; // Increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}


// Function to implement QuickSort
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        // pi is the partitioning index, arr[pi] is now at the right place
        int pi = partition(arr, low, high);
        // Separately sort elements before and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

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

// Main function to test the QuickSort algorithm
int main() {
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, n);

    quickSort(arr, 0, n - 1);

    printf("\nSorted array is \n");
    printArray(arr, n);
    return 0;
}
```

Output

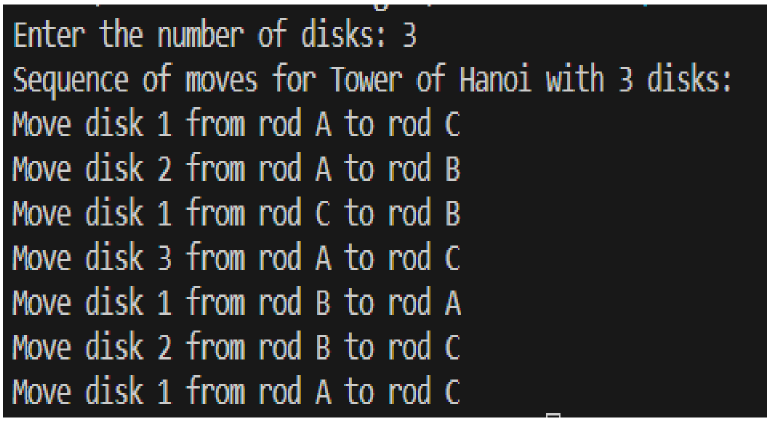


```
PS D:\Data Structure using C\
Given array is
10 7 8 9 1 5

Sorted array is
1 5 7 8 9 10
```

Write a program to demonstrate Tower of Hanoi:

```
#include <stdio.h>


void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod) {
    if (n == 1) {
        printf("Move disk 1 from rod %c to rod %c\n", from_rod, to_rod);
        return;
    }
    towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
    printf("Move disk %d from rod %c to rod %c\n", n, from_rod, to_rod);
    towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
}


int main() {
    int num_disks;


    printf("Enter the number of disks: ");
    scanf("%d", &num_disks);


    printf("Sequence of moves for Tower of Hanoi with %d disks:\n", num_disks);
    towerOfHanoi(num_disks, 'A', 'C', 'B');


    return 0;
}
```

Output



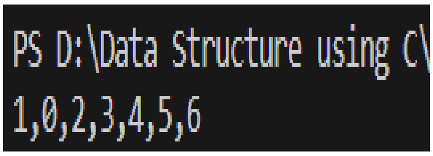
```
Enter the number of disks: 3
Sequence of moves for Tower of Hanoi with 3 disks:
Move disk 1 from rod A to rod C
Move disk 2 from rod A to rod B
Move disk 1 from rod C to rod B
Move disk 3 from rod A to rod C
Move disk 1 from rod B to rod A
Move disk 2 from rod B to rod C
Move disk 1 from rod A to rod C
```

Write a C program to demonstrate graph traversal by Breadth-First Search:

```
#include<stdio.h>

#include<stdlib.h>


struct queue {

   int size;

   int f;

   int r;

   int* arr;

};


int isEmpty(struct queue *q){

   if(q->r == q->f){

      return 1;

   }

   return 0;

}


int isFull(struct queue *q){

   if(q->r == q->size - 1){

      return 1;

   }

   return 0;

}


void enqueue(struct queue *q, int val){

   if(isFull(q)){

      printf("This Queue is full\n");

   }

   else{

      q->r++;
```

```
        q->arr[q->r] = val;
    }
}

int dequeue(struct queue *q){
    int a = -1;
    if(isEmpty(q)){
        printf("This Queue is empty\n");
    }
    else{
        q->f++;
        a = q->arr[q->f];
    }
    return a;
}

int main(){
    struct queue q;
    q.size = 400;
    q.f = q.r = 0;
    q.arr = (int*) malloc(q.size * sizeof(int));

    int node;
    int i = 1;
    int visited[7] = {0,0,0,0,0,0,0};
    int a [7][7] = {
        {0,1,1,1,0,0,0},
        {1,0,1,0,0,0,0},
        {1,1,0,1,1,0,0},
        {1,0,1,0,1,0,0},
        {0,0,1,1,0,1,1},
        {0,0,0,0,1,0,0},
        {0,0,0,0,1,0,0}
    };
    printf("%d", i);
    visited[i] = 1;
    enqueue(&q, i);

    while (!isEmpty(&q))
    {
        int node = dequeue(&q);
        for (int j = 0; j < 7; j++)
        {
            if(a[node][j] == 1 && visited[j] == 0){
                printf(",%d", j);
                visited[j] = 1;
                enqueue(&q, j);
            }
        }
    }
    return 0;
}
```

Output



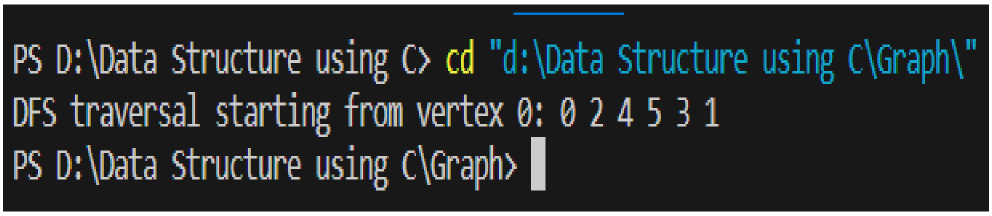
```
PS D:\Data Structure using C\
1,0,2,3,4,5,6
```

Write a C program to demonstrate graph traversal by Depth-First Search:

```
#include <stdio.h>

#include <stdlib.h>


struct Node {

    int data;

    struct Node* next;

};


struct Graph {

    int numVertices;

    struct Node** adjacencyList;

    int* visited;

};


struct Node* createNode(int data) {

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));

    newNode->data = data;

    newNode->next = NULL;

    return newNode;

}


struct Graph* createGraph(int vertices) {

    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));

    graph->numVertices = vertices;


    graph->adjacencyList = (struct Node**)malloc(vertices * sizeof(struct Node*));

    graph->visited = (int*)malloc(vertices * sizeof(int));


    for (int i = 0; i < vertices; i++) {

        graph->adjacencyList[i] = NULL;
        graph->visited[i] = 0;
    }

    return graph;
}

void addEdge(struct Graph* graph, int src, int dest) {
    // Add edge from src to dest
    struct Node* newNode = createNode(dest);
    newNode->next = graph->adjacencyList[src];
    graph->adjacencyList[src] = newNode;

    // For undirected graph, add edge from dest to src as well
    newNode = createNode(src);
    newNode->next = graph->adjacencyList[dest];
    graph->adjacencyList[dest] = newNode;
}

void DFS(struct Graph* graph, int vertex) {
    struct Node* temp = graph->adjacencyList[vertex];
    graph->visited[vertex] = 1;
    printf("%d ", vertex);

    while (temp != NULL) {
        int adjVertex = temp->data;
        if (graph->visited[adjVertex] == 0) {
            DFS(graph, adjVertex);
        }
        temp = temp->next;
    }
}
```

```
int main() {
    int numVertices = 6; // Change this value to match the number of vertices in your graph
    struct Graph* graph = createGraph(numVertices);

    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 2, 3);
    addEdge(graph, 2, 4);
    addEdge(graph, 3, 4);
    addEdge(graph, 3, 5);
    addEdge(graph, 4, 5);

    printf("DFS traversal starting from vertex 0: ");
    DFS(graph, 0);

    return 0;
}
```

Output



```
PS D:\Data Structure using C> cd "d:\Data Structure using C\Graph\"
DFS traversal starting from vertex 0: 0 2 4 5 3 1
PS D:\Data Structure using C\Graph>
```