**1.Write a program in C to read n number of values in an array and display them in reverse order.**

**The values store into the array are : 2 5 7**

**The values store into the array in reverse are : 7 5 2**

#include <stdio.h>

int main() {

int n;

printf("Enter the number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter %d values:\n", n);

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

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

}

printf("Values in reverse order:\n");

for (int i = n - 1; i >= 0; i--) {

printf("%d ", arr[i]);

}

printf("\n");

return 0;

}

**2. Implement a C Program for AVL tree and perform Insertion and Deletion of Nodes**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int key;

struct Node \*left;

struct Node \*right;

int height;

} Node;

int height(Node \*node) {

if (node == NULL)

return 0;

return node->height;

}

int max(int a, int b) {

return (a > b) ? a : b;

}

Node \*newNode(int key) {

Node \*node = (Node \*)malloc(sizeof(Node));

node->key = key;

node->left = NULL;

node->right = NULL;

node->height = 1;

return node;

}

Node \*rightRotate(Node \*y) {

Node \*x = y->left;

Node \*T2 = x->right;

x->right = y;

y->left = T2;

y->height = max(height(y->left), height(y->right)) + 1;

x->height = max(height(x->left), height(x->right)) + 1;

return x;

}

Node \*leftRotate(Node \*x) {

Node \*y = x->right;

Node \*T2 = y->left;

y->left = x;

x->right = T2;

x->height = max(height(x->left), height(x->right)) + 1;

y->height = max(height(y->left), height(y->right)) + 1;

return y;

}

int getBalance(Node \*node) {

if (node == NULL)

return 0;

return height(node->left) - height(node->right);

}

Node \*insert(Node \*node, int key) {

if (node == NULL)

return newNode(key);

if (key < node->key)

node->left = insert(node->left, key);

else if (key > node->key)

node->right = insert(node->right, key);

else

return node;

node->height = 1 + max(height(node->left), height(node->right));

int balance = getBalance(node);

if (balance > 1 && key < node->left->key)

return rightRotate(node);

if (balance < -1 && key > node->right->key)

return leftRotate(node);

if (balance > 1 && key > node->left->key) {

node->left = leftRotate(node->left);

return rightRotate(node);

}

if (balance < -1 && key < node->right->key) {

node->right = rightRotate(node->right);

return leftRotate(node);

}

return node;

}

Node\* minValueNode(Node \*node) {

Node\* current = node;

while (current && current->left != NULL)

current = current->left;

return current;

}

Node \*deleteNode(Node \*root, int key) {

if (root == NULL)

return root;

if (key < root->key)

root->left = deleteNode(root->left, key);

else if (key > root->key)

root->right = deleteNode(root->right, key);

else {

if (root->left == NULL) {

Node \*temp = root->right;

free(root);

return temp;

} else if (root->right == NULL) {

Node \*temp = root->left;

free(root);

return temp;

}

Node \*temp = minValueNode(root->right);

root->key = temp->key;

root->right = deleteNode(root->right, temp->key);

}

if (root == NULL)

return root;

root->height = 1 + max(height(root->left), height(root->right));

int balance = getBalance(root);

if (balance > 1 && getBalance(root->left) >= 0)

return rightRotate(root);

if (balance > 1 && getBalance(root->left) < 0) {

root->left = leftRotate(root->left);

return rightRotate(root);

}

if (balance < -1 && getBalance(root->right) <= 0)

return leftRotate(root);

if (balance < -1 && getBalance(root->right) > 0) {

root->right = rightRotate(root->right);

return leftRotate(root);

}

return root;

}

void PrintTree(Node\* root, int space) {

int count = 10;

if (root == NULL)

return;

space += count;

PrintTree(root->right, space);

printf("\n");

for (int i = count; i < space; i++)

printf(" ");

printf("%d\n", root->key);

PrintTree(root->left, space);

}

int main() {

Node\* root = NULL;

root = insert(root, 10);

root = insert(root, 20);

root = insert(root, 30);

root = insert(root, 40);

root = insert(root, 50);

root = insert(root, 25);

printf("Tree structure:\n");

PrintTree(root, 0);

root = deleteNode(root, 40);

printf("\nTree structure after deletion:\n");

PrintTree(root, 0);

return 0;

}

**3.Implement a C Program to Check for a valid String**

#include <stdio.h>

int isValidString(const char \*str) {

if (str[0] == '\0') {

return 0;

}

for (int i = 0; str[i] != '\0'; i++) {

char ch = str[i];

if ((ch < '0' || ch > '9') && (ch < 'A' || ch > 'Z') && (ch < 'a' || ch > 'z')) {

return 0;

}

}

return 1;

}

int main() {

char str[100];

printf("Enter a string: ");

scanf("%99s", str);

if (isValidString(str)) {

printf("The string is valid.\n");

} else {

printf("The string is not valid.\n");

}

return 0;

}

**4. Implement a C Program whether it is a Valid stack**

**Input: pushed = { 1, 2, 3, 4, 5 }, popped = { 4, 5, 3, 2, 1 }**

**Output: True**

#include <stdio.h>

#define MAX 100

int validateStackSequences(int pushed[], int pushedSize, int popped[], int poppedSize) {

int stack[MAX];

int top = -1;

int popIndex = 0;

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

stack[++top] = pushed[i];

while (top >= 0 && stack[top] == popped[popIndex]) {

top--;

popIndex++;

}

}

return top == -1 ? 1 : 0;

}

int main() {

int pushed[] = {1, 2, 3, 4, 5};

int popped[] = {4, 5, 3, 2, 1};

int pushedSize = 5;

int poppedSize = 5;

if (validateStackSequences(pushed, pushedSize, popped, poppedSize))

printf("Output: True\n");

else

printf("Output: False\n");

return 0;

}

**5. Implement a C Program to Merge two Arrays**

**Input:**

**arr1 = [1, 2, 3, 4, 5]**

**arr2 = [6, 7, 8, 9, 10]**

**Output:**

**arr3 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]**

#include <stdio.h>

void mergeArrays(int arr1[], int size1, int arr2[], int size2, int arr3[]) {

int i, j;

for (i = 0; i < size1; i++) {

arr3[i] = arr1[i];

}

for (j = 0; j < size2; j++) {

arr3[i + j] = arr2[j];

}

}

void printArray(int arr[], int size) {

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

printf("%d ", arr[i]);

}

printf("\n");

}

int main() {

int arr1[] = {1, 2, 3, 4, 5};

int arr2[] = {6, 7, 8, 9, 10};

int size1 = sizeof(arr1) / sizeof(arr1[0]);

int size2 = sizeof(arr2) / sizeof(arr2[0]);

int size3 = size1 + size2;

int arr3[size3];

mergeArrays(arr1, size1, arr2, size2, arr3);

printf("Merged array:\n");

printArray(arr3, size3);

return 0;

}

**6. Implement a C Program for Graph to Identify shortest path**

**Input :**

**Enter number of nodes: 4**

**Enter weight of all the paths in adjacency matrix form**

**0 10 30 100**

**10 0 10 90**

**30 10 0 30**

**100 90 30 0**

**Enter the source: 1**

**Enter the target: 4**

**1 to 2 to 3 to 4**

**Output: shortest path is 50**

**PROGRAM:**

#include<stdio.h>

#include <limits.h>

#define MAX 100

#define INF INT\_MAX

void dijkstra(int graph[MAX][MAX], int n, int src, int target) {

int dist[MAX], prev[MAX], visited[MAX];

int i, j;

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

dist[i] = INF;

prev[i] = -1;

visited[i] = 0;

}

dist[src] = 0;

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

int u = -1;

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

if (!visited[j] && (u == -1 || dist[j] < dist[u])) {

u = j;

}

}

visited[u] = 1;

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

if (graph[u][j] && dist[u] != INF && dist[u] + graph[u][j] < dist[j]) {

dist[j] = dist[u] + graph[u][j];

prev[j] = u;

}

}

}

if (dist[target] == INF) {

printf("No path exists from %d to %d\n", src + 1, target + 1);

return;

}

printf("Shortest path is %d\n", dist[target]);

printf("Path: ");

int path[MAX], pathIndex = 0;

for (i = target; i != -1; i = prev[i]) {

path[pathIndex++] = i;

}

for (i = pathIndex - 1; i > 0; i--) {

printf("%d to ", path[i] + 1);

}

printf("%d\n", path[0] + 1);

}

int main() {

int n, i, j;

int graph[MAX][MAX];

printf("Enter number of nodes: ");

scanf("%d", &n);

printf("Enter weight of all the paths in adjacency matrix form:\n");

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

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

scanf("%d", &graph[i][j]);

}

}

int src, target;

printf("Enter the source: ");

scanf("%d", &src);

printf("Enter the target: ");

scanf("%d", &target)

src--;

target--;

dijkstra(graph, n, src, target);

return 0;

}

**7.Write a program in C to count the total number of duplicate elements in an array.**

#include <stdio.h>

int countDuplicates(int arr[], int size) {

int count = 0;

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

int isDuplicate = 0;

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

if (arr[i] == arr[j]) {

isDuplicate = 1;

break;

}

}

if (isDuplicate) {

int isCounted = 0;

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

if (arr[i] == arr[k]) {

isCounted = 1;

break;

}

}

if (!isCounted) {

count++;

}

}

}

return count;

}

int main() {

int arr[] = {1, 2, 3, 2, 4, 5, 5, 6, 7, 7, 8};

int size = sizeof(arr) / sizeof(arr[0]);

int totalDuplicates = countDuplicates(arr, size);

printf("Total number of duplicate elements: %d\n", totalDuplicates);

return 0;

}

**8. Implement a C Program Traveling Salesman Problem to Identify shortest path**

**Given a set of cities and distances between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.**

**PROGRAM:**

#include <stdio.h>

#include <limits.h>

#define MAX 10

#define INF 1000000

int n;

int dist[MAX][MAX];

int visited[MAX];

int minPath = INF;

int path[MAX];

void tsp(int start, int current, int count, int cost) {

if (count == n && dist[current][start]) {

if (cost + dist[current][start] < minPath) {

minPath = cost + dist[current][start];

}

return;

}

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

if (!visited[i] && dist[current][i]) {

visited[i] = 1;

tsp(start, i, count + 1, cost + dist[current][i]);

visited[i] = 0;

}

}

}

int main() {

printf("Enter number of cities: ");

scanf("%d", &n);

printf("Enter distance matrix:\n");

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

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

scanf("%d", &dist[i][j]);

}

}

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

visited[i] = 0;

}

visited[0] = 1;

tsp(0, 0, 1, 0);

printf("Minimum cost: %d\n", minPath);

return 0;

}

**9. Implement a C Program for Merging of list**

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* next;

} Node;

Node\* createNode(int data) {

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

newNode->data = data;

newNode->next = NULL;

return newNode;

}

Node\* mergeLists(Node\* list1, Node\* list2) {

Node\* mergedHead = NULL;

Node\* mergedTail = NULL;

while (list1 != NULL && list2 != NULL) {

Node\* temp = NULL;

if (list1->data < list2->data) {

temp = createNode(list1->data);

list1 = list1->next;

} else {

temp = createNode(list2->data);

list2 = list2->next;

}

if (mergedHead == NULL) {

mergedHead = temp;

mergedTail = temp;

} else {

mergedTail->next = temp;

mergedTail = temp;

}

}

while (list1 != NULL) {

Node\* temp = createNode(list1->data);

if (mergedHead == NULL) {

mergedHead = temp;

mergedTail = temp;

} else {

mergedTail->next = temp;

mergedTail = temp;

}

list1 = list1->next;

}

while (list2 != NULL) {

Node\* temp = createNode(list2->data);

if (mergedHead == NULL) {

mergedHead = temp;

mergedTail = temp;

} else {

mergedTail->next = temp;

mergedTail = temp;

}

list2 = list2->next;

}

return mergedHead;

}

void printList(Node\* head) {

Node\* current = head;

while (current != NULL) {

printf("%d ", current->data);

current = current->next;

}

printf("\n");

}

int main() {

Node\* list1 = NULL;

Node\* list2 = NULL;

Node\* mergedList = NULL;

int n1, n2, data;

printf("Enter number of elements in the first list: ");

scanf("%d", &n1);

printf("Enter elements of the first list:\n");

for (int i = 0; i < n1; i++) {

scanf("%d", &data);

Node\* newNode = createNode(data);

newNode->next = list1;

list1 = newNode;

}

printf("Enter number of elements in the second list: ");

scanf("%d", &n2);

printf("Enter elements of the second list:\n");

for (int i = 0; i < n2; i++) {

scanf("%d", &data);

Node\* newNode = createNode(data);

newNode->next = list2;

list2 = newNode;

}

mergedList = mergeLists(list1, list2);

printf("Merged list:\n");

printList(mergedList);

return 0;

}

**10. Implement a C Program for Binary search tree - search for a element, min element and Max element**

**PROGRAM:**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* left;

struct Node\* right;

} Node;

Node\* createNode(int data) {

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

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

Node\* insert(Node\* root, int data) {

if (root == NULL) {

return createNode(data);

}

if (data < root->data) {

root->left = insert(root->left, data);

} else {

root->right = insert(root->right, data);

}

return root;

}

Node\* search(Node\* root, int data) {

if (root == NULL || root->data == data) {

return root;

}

if (data < root->data) {

return search(root->left, data);

} else {

return search(root->right, data);

}

}

Node\* findMin(Node\* root) {

while (root->left != NULL) {

root = root->left;

}

return root;

}

Node\* findMax(Node\* root) {

while (root->right != NULL) {

root = root->right;

}

return root;

}

void printTree(Node\* root) {

if (root != NULL) {

printTree(root->left);

printf("%d ", root->data);

printTree(root->right);

}

}

int main() {

Node\* root = NULL;

int n, data;

printf("Enter number of elements: ");

scanf("%d", &n);

printf("Enter elements:\n");

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

scanf("%d", &data);

root = insert(root, data);

}

printf("Enter element to search: ");

scanf("%d", &data);

Node\* result = search(root, data);

if (result != NULL) {

printf("Element %d found in the tree.\n", data);

} else {

printf("Element %d not found in the tree.\n", data);

}

Node\* minNode = findMin(root);

Node\* maxNode = findMax(root);

if (minNode != NULL) {

printf("Minimum element: %d\n", minNode->data);

} else {

printf("Tree is empty.\n");

}

if (maxNode != NULL) {

printf("Maximum element: %d\n", maxNode->data);

} else {

printf("Tree is empty.\n");

}

return 0;

}

**11. Implement a C Program Given an array of reg nos need to search for particular reg no**

**PROGRAM:**

#include <stdio.h>

int searchRegNo(int arr[], int size, int regNo) {

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

if (arr[i] == regNo) {

return i;

}

}

return -1;

}

int main() {

int regNos[] = {101, 202, 303, 404, 505};

int size = sizeof(regNos) / sizeof(regNos[0]);

int regNoToSearch = 303;

int result = searchRegNo(regNos, size, regNoToSearch);

if (result != -1) {

printf("Registration number %d found at index %d.\n", regNoToSearch, result);

} else {

printf("Registration number %d not found.\n", regNoToSearch);

}

return 0;

}

**12.Implement a C Program for Haystack. There are two strings needle and haystack (or hay). You need to check if all the characters in the needle are present in haystack or not. If yes then return True (1) or False (0)**

**PROGRAM:**

#include <stdio.h>

#include <string.h>

int isPresent(char \*needle, char \*haystack) {

int needleLen = strlen(needle);

int haystackLen = strlen(haystack);

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

int found = 0;

for(int j = 0; j < haystackLen; j++) {

if(needle[i] == haystack[j]) {

found = 1;

break;

}

}

if(!found) {

return 0;

}

}

return 1;

}

int main() {

char needle[] = "abc";

char haystack[] = "aabbccdde";

if(isPresent(needle, haystack)) {

printf("True\n");

} else {

printf("False\n");

}

return 0;

}

**13. Write a program in C to count the frequency of each element of an array.**

**Test Data :**

**Input the number of elements to be stored in the array :3**

**Input 3 elements in the array :**

**element - 0 : 25**

**element - 1 : 12**

**element - 2 : 43**

**Expected Output :**

**The frequency of all elements of an array :**

**25 occurs 1 times**

1. **ccurs 1 times**

**43 occurs 1 times.**

**PROGRAM:**

#include <stdio.h>

int main() {

int n;

printf("Input the number of elements to be stored in the array: ");

scanf("%d", &n);

int arr[n], freq[n];

printf("Input %d elements in the array:\n", n);

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

printf("element - %d: ", i);

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

freq[i] = -1;

}

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

int count = 1;

for (int j = i + 1; j < n; j++) {

if (arr[i] == arr[j]) {

count++;

freq[j] = 0;

}

}

if (freq[i] != 0) {

freq[i] = count;

}

}

printf("The frequency of all elements of an array:\n");

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

if (freq[i] != 0) {

printf("%d occurs %d times\n", arr[i], freq[i]);

}

}

return 0;

}

**14. Implement a C Program for Given Graph convert array and print minimum edges (Prim’s Algorithm)**

**PROGRAM:**

#include <stdio.h>

#include <limits.h>

#include <stdbool.h>

#define V 5

int minKey(int key[], bool mstSet[]) {

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++) {

if (mstSet[v] == false && key[v] < min) {

min = key[v];

min\_index = v;

}

}

return min\_index;

}

void printMST(int parent[], int n, int graph[V][V]) {

printf("Edge \tWeight\n");

for (int i = 1; i < V; i++) {

printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);

}

}

void primMST(int graph[V][V]) {

int parent[V];

int key[V];

bool mstSet[V];

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

key[i] = INT\_MAX;

mstSet[i] = false;

}

key[0] = 0;

parent[0] = -1;

for (int count = 0; count < V - 1; count++) {

int u = minKey(key, mstSet);

mstSet[u] = true;

for (int v = 0; v < V; v++) {

if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v]) {

parent[v] = u;

key[v] = graph[u][v];

}

}

}

printMST(parent, V, graph);

}

int main() {

int graph[V][V] = {

{0, 2, 0, 6, 0},

{2, 0, 3, 8, 5},

{0, 3, 0, 0, 7},

{6, 8, 0, 0, 9},

{0, 5, 7, 9, 0}

};

primMST(graph);

return 0;

}

**15. Write a program in C to separate odd and even integers into separate arrays.**

**Test Data :**

**Input the number of elements to be stored in the array :5**

**Input 5 elements in the array :**

**element - 0 : 25**

**element - 1 : 47**

**element - 2 : 42**

**element - 3 : 56**

**element - 4 : 32**

**Expected Output :**

**The Even elements are : 42 56 32**

**The Odd elements are : 25 47**

**PROGRAMS:**

#include <stdio.h>

int main() {

int n;

printf("Input the number of elements to be stored in the array: ");

scanf("%d", &n);

int arr[n], even[n], odd[n];

int evenCount = 0, oddCount = 0;

printf("Input %d elements in the array:\n", n);

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

printf("element - %d: ", i);

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

}

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

if (arr[i] % 2 == 0) {

even[evenCount++] = arr[i];

} else {

odd[oddCount++] = arr[i];

}

}

printf("Even elements:\n");

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

printf("%d ", even[i]);

}

printf("\n");

printf("Odd elements:\n");

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

printf("%d ", odd[i]);

}

printf("\n");

return 0;

}

**16.** **Implement a C Program for Given Graph - Print valid path (BFS or DFS)**

**PROGRAM:**

#include <stdio.h>

#include <stdbool.h>

#define V 5

void printPath(int parent[], int j) {

if (parent[j] == -1) {

printf("%d ", j);

return;

}

printPath(parent, parent[j]);

printf("%d ", j);

}

bool BFS(int graph[V][V], int start, int end) {

bool visited[V] = {false};

int queue[V], parent[V];

int front = 0, rear = 0;

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

parent[i] = -1;

}

visited[start] = true;

queue[rear++] = start;

while (front < rear) {

int current = queue[front++];

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

if (graph[current][i] && !visited[i]) {

visited[i] = true;

parent[i] = current;

queue[rear++] = i;

if (i == end) {

printPath(parent, end);

return true;

}

}

}

}

return false;

}

int main() {

int graph[V][V] = {

{0, 1, 1, 0, 0},

{1, 0, 0, 1, 0},

{1, 0, 0, 1, 1},

{0, 1, 1, 0, 1},

{0, 0, 1, 1, 0}

};

int start = 0, end = 4;

if (!BFS(graph, start, end)) {

printf("No path found\n");

}

return 0;

}

**17. Implement a C Program sum of Fibonacci Series using recursion**

**Input : n = 1**

**Output : 1**

**Input : n = 9**

**Output : 34**

**Input : n = 10**

**Output : 55**

**PROGRAM:**

#include <stdio.h>

int fibonacci(int n) {

if (n <= 1) {

return n;

}

return fibonacci(n - 1) + fibonacci(n - 2);

}

int sumFibonacci(int n) {

if (n == 0) {

return 0;

}

return fibonacci(n) + sumFibonacci(n - 1);

}

int main() {

int n;

printf("Enter the number of terms: ");

scanf("%d", &n);

int sum = sumFibonacci(n - 1);

printf("Sum of Fibonacci series up to %d terms: %d\n", n, sum);

return 0;

}

**18.** **Implement a C Program to perform heap sort**

**PROGRAM:**

#include <stdio.h>

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);

}

}

void heapSort(int arr[], int n) {

for (int i = n / 2 - 1; i >= 0; i--) {

heapify(arr, n, i);

}

for (int i = n - 1; i > 0; i--) {

swap(&arr[0], &arr[i]);

heapify(arr, i, 0);

}

}

void printArray(int arr[], int n) {

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

printf("%d ", arr[i]);

}

printf("\n");

}

int main() {

int arr[] = {12, 11, 13, 5, 6, 7};

int n = sizeof(arr) / sizeof(arr[0]);

heapSort(arr, n);

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}

**19.** **Implement a C Program for Finding factorial of a number using recursion**

**Enter a positive integer: 6**

**Factorial of 6 = 720**

**PROGRAM:**

#include <stdio.h>

int factorial(int n) {

if (n == 0) {

return 1;

}

return n \* factorial(n - 1);

}

int main() {

int n;

printf("Enter a number: ");

scanf("%d", &n);

int result = factorial(n);

printf("Factorial of %d is %d\n", n, result);

return 0;

}

**20.** **Implement a C Program to perform quick sort**

**How many elements are u going to enter?: 10**

**Enter 10 elements: 2 3 5 7 1 9 3 8 0 4**

**Order of Sorted elements: 0 1 2 3 3 4 5 7 8 9**

**PROGRAM:**

#include <stdio.h>

int partition(int arr[], int low, int high) {

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j < high; j++) {

if (arr[j] < pivot) {

i++;

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

int temp = arr[i + 1];

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

arr[high] = temp;

return (i + 1);

}

void quickSort(int arr[], int low, int high) {

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void printArray(int arr[], int n) {

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

printf("%d ", arr[i]);

}

printf("\n");

}

int main() {

int arr[] = {10, 7, 8, 9, 1, 5};

int n = sizeof(arr) / sizeof(arr[0]);

quickSort(arr, 0, n - 1);

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}