LAB PROGRAMS: (SHARMILA.N – 192311330)

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

**Test Data :**

**Input the number of elements to store in the array :3**

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

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

PROGRAM:

#include <stdio.h>

int main() {

int n;

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

scanf("%d", &n);

int arr[n];

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

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

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

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

}

printf("The values stored into the array are:\n");

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

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

}

printf("\n");

printf("The values stored into the array in reverse are:\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**

PROGRAM:

#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->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) || (root->right == NULL)) {

Node \*temp = root->left ? root->left : root->right;

if (temp == NULL) {

temp = root;

root = NULL;

} else

\*root = \*temp;

free(temp);

} else {

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 inOrder(Node \*root) {

if (root != NULL) {

inOrder(root->left);

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

inOrder(root->right);

}

}

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("Inorder traversal of the AVL tree:\n");

inOrder(root);

printf("\n");

root = deleteNode(root, 30);

printf("Inorder traversal after deletion:\n");

inOrder(root);

printf("\n");

return 0;

}

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

PROGRAM:

#include <stdio.h>

#include <string.h>

int isAlpha(char c) {

return (c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z');

}

int isValidString(const char \*str) {

if (str == NULL || strlen(str) == 0)

return 0;

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

if (!isAlpha(str[i]))

return 0;

}

return 1;

}

int main() {

char str[100];

printf("Enter a string: ");

fgets(str, sizeof(str), stdin);

size\_t len = strlen(str);

if (len > 0 && str[len - 1] == '\n') {

str[len - 1] = '\0';

}

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**

PROGRAM:

#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]**

PROGRAM:

#include <stdio.h>

#define MAX 100

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++] = arr2[j];

}

}

int main() {

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

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

int arr3[MAX];

int size1 = 5;

int size2 = 5;

int size3 = size1 + size2;

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

printf("Merged array: ");

for (int i = 0; i < size3; i++) {

printf("%d ", arr3[i]);

}

printf("\n");

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

void dijkstra(int graph[MAX][MAX], int numNodes, int start, int end) {

int dist[MAX];

int visited[MAX];

int path[MAX];

int i, j, min, nextNode;

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

dist[i] = INT\_MAX;

visited[i] = 0;

path[i] = -1;

}

dist[start - 1] = 0;

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

min = INT\_MAX;

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

if (!visited[j] && dist[j] < min) {

min = dist[j];

nextNode = j;

}

}

visited[nextNode] = 1;

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

if (!visited[j] && graph[nextNode][j] && dist[nextNode] != INT\_MAX &&

dist[nextNode] + graph[nextNode][j] < dist[j]) {

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

path[j] = nextNode;

}

}

}

printf("Shortest path is %d\n", dist[end - 1]);

printf("Path: ");

int node = end - 1;

int stack[MAX];

int top = -1;

while (node != -1) {

stack[++top] = node;

node = path[node];

}

while (top >= 0) {

printf("%d ", stack[top] + 1);

top--;

}

printf("\n");

}

int main() {

int graph[MAX][MAX];

int numNodes;

int source, target;

printf("Enter number of nodes\n");

scanf("%d", &numNodes);

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

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

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

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

}

}

printf("Enter the source\n");

scanf("%d", &source);

printf("Enter the target\n");

scanf("%d", &target);

dijkstra(graph, numNodes, source, target);

return 0;

}

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

**Test Data :**

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

**Input 3 elements in the array :**

**element - 0 : 5**

**element - 1 : 1**

**element - 2 : 1**

**Expected Output :**

**Total number of duplicate elements found in the array is : 1**

PROGRAM:

#include <stdio.h>

#define MAX 100

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

int count = 0;

int visited[MAX] = {0};

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

if (visited[i])

continue;

int isDuplicate = 0;

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

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

isDuplicate = 1;

visited[j] = 1;

}

}

if (isDuplicate) {

count++;

}

}

return count;

}

int main() {

int arr[MAX];

int size;

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

scanf("%d", &size);

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

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

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

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

}

int duplicateCount = countDuplicates(arr, size);

printf("Total number of duplicate elements found in the array is : %d\n", duplicateCount);

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>

int tsp(int currentCity, int count, int cost, int startCity, int n, int dist[][n], int visited[]) {

if (count == n && dist[currentCity][startCity]) {

return cost + dist[currentCity][startCity];

}

int minCost = INT\_MAX;

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

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

visited[i] = 1;

int newCost = tsp(i, count + 1, cost + dist[currentCity][i], startCity, n, dist, visited);

minCost = (newCost < minCost) ? newCost : minCost;

visited[i] = 0;

}

}

return minCost;

}

int main() {

int n;

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

scanf("%d", &n);

int dist[n][n];

int visited[n];

printf("Enter the 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;

int result = tsp(0, 1, 0, 0, n, dist, visited);

printf("The shortest path has length: %d\n", result);

return 0;

}

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

PROGRAM:

#include <stdio.h>

void merge(int arr1[], int n1, int arr2[], int n2, int merged[]) {

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

while (i < n1 && j < n2) {

if (arr1[i] <= arr2[j])

merged[k++] = arr1[i++];

else

merged[k++] = arr2[j++];

}

while (i < n1)

merged[k++] = arr1[i++];

while (j < n2)

merged[k++] = arr2[j++];

}

int main() {

int arr1[] = {1, 3, 5, 7};

int n1 = sizeof(arr1) / sizeof(arr1[0]);

int arr2[] = {2, 4, 6, 8};

int n2 = sizeof(arr2) / sizeof(arr2[0]);

int merged[n1 + n2];

merge(arr1, n1, arr2, n2, merged);

printf("Merged array: ");

for (int i = 0; i < n1 + n2; i++) {

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

}

printf("\n");

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>

struct Node {

int data;

struct Node\* left;

struct Node\* right;

};

struct Node\* createNode(int data) {

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

newNode->data = data;

newNode->left = newNode->right = NULL;

return newNode;

}

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

if (root == NULL) {

return createNode(data);

}

if (data < root->data) {

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

} else if (data > root->data) {

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

}

return root;

}

struct Node\* search(struct Node\* root, int key) {

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

return root;

}

if (key < root->data) {

return search(root->left, key);

}

return search(root->right, key);

}

struct Node\* findMin(struct Node\* root) {

struct Node\* current = root;

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

current = current->left;

}

return current;

}

struct Node\* findMax(struct Node\* root) {

struct Node\* current = root;

while (current && current->right != NULL) {

current = current->right;

}

return current;

}

void inorder(struct Node\* root) {

if (root != NULL) {

inorder(root->left);

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

inorder(root->right);

}

}

int main() {

struct Node\* root = NULL;

root = insert(root, 50);

insert(root, 30);

insert(root, 20);

insert(root, 40);

insert(root, 70);

insert(root, 60);

insert(root, 80);

printf("Inorder traversal: ");

inorder(root);

printf("\n");

int key = 40;

struct Node\* searchResult = search(root, key);

if (searchResult != NULL) {

printf("Element %d found in the BST.\n", key);

} else {

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

}

struct Node\* minNode = findMin(root);

if (minNode != NULL) {

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

}

struct Node\* maxNode = findMax(root);

if (maxNode != NULL) {

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

}

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[] = {1001, 1002, 1003, 1004, 1005};

int size = 5;

int regNoToSearch = 1003;

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

if (index != -1) {

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

} 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>

int isNeedleInHaystack(char needle[], char haystack[]) {

int hash[256] = {0};

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

hash[(unsigned char)haystack[i]]++;

}

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

if (hash[(unsigned char)needle[i]] == 0) {

return 0;

}

}

return 1;

}

int main() {

char needle[] = "abc";

char haystack[] = "aabbcc";

if (isNeedleInHaystack(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 ,12 occurs 1 times,43 occurs 1 times.**

PROGRAM:

#include <stdio.h>

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

int freq[size];

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

freq[i] = -1;

}

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

int count = 1;

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

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

count++;

freq[j] = 0;

}

}

if (freq[i] != 0) {

freq[i] = count;

}

}

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

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

if (freq[i] != 0) {

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

}

}

}

int main() {

int n;

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

scanf("%d", &n);

int arr[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]);

}

countFrequency(arr, n);

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>

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

int min = INT\_MAX, min\_index;

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

if (mstSet[v] == 0 && key[v] < min) {

min = key[v], min\_index = v;

}

}

return min\_index;

}

void printMST(int parent[], int graph[][20], int vertices) {

printf("Edge \tWeight\n");

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

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

}

}

void primMST(int graph[][20], int vertices) {

int parent[20];

int key[20];

int mstSet[20];

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

key[i] = INT\_MAX;

mstSet[i] = 0;

}

key[0] = 0;

parent[0] = -1;

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

int u = minKey(key, mstSet, vertices);

mstSet[u] = 1;

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

if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v]) {

parent[v] = u;

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

}

}

}

printMST(parent, graph, vertices);

}

int main() {

int vertices;

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

scanf("%d", &vertices);

int graph[20][20];

printf("Enter the adjacency matrix of the graph:\n");

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

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

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

}

}

primMST(graph, vertices);

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**

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], 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("The Even elements are:\n");

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

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

}

printf("\n");

printf("The Odd elements are:\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>

#define MAX\_VERTICES 100

void bfs(int graph[MAX\_VERTICES][MAX\_VERTICES], int start, int numVertices) {

int queue[MAX\_VERTICES], front = 0, rear = 0;

int visited[MAX\_VERTICES] = {0};

queue[rear++] = start;

visited[start] = 1;

printf("BFS Path:\n");

while (front < rear) {

int vertex = queue[front++];

printf("%d ", vertex);

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

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

queue[rear++] = i;

visited[i] = 1;

}

}

}

printf("\n");

}

void dfs(int graph[MAX\_VERTICES][MAX\_VERTICES], int vertex, int visited[], int numVertices) {

visited[vertex] = 1;

printf("%d ", vertex);

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

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

dfs(graph, i, visited, numVertices);

}

}

}

int main() {

int numVertices;

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

scanf("%d", &numVertices);

int graph[MAX\_VERTICES][MAX\_VERTICES] = {0};

int startVertex;

printf("Enter the adjacency matrix:\n");

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

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

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

}

}

printf("Enter the starting vertex for BFS and DFS: ");

scanf("%d", &startVertex);

printf("Performing BFS:\n");

bfs(graph, startVertex, numVertices);

int visited[MAX\_VERTICES] = {0};

printf("Performing DFS:\n");

dfs(graph, startVertex, visited, numVertices);

return 0;

}

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

**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 main() {

int n;

printf("Input n = ");

scanf("%d", &n);

if (n <= 0) {

printf("Sum of Fibonacci series up to %d terms is 0\n", n);

} else {

int result = fibonacci(n);

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

}

return 0;

}

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

PROGRAM:

#include <stdio.h>

void swap(int \*x, int \*y) {

int temp = \*x;

\*x = \*y;

\*y = 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 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 n = 6;

printf("Original array:\n");

printArray(arr, n);

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 || n == 1) {

return 1;

}

return n \* factorial(n - 1);

}

int main() {

int n;

printf("Enter a positive integer: ");

scanf("%d", &n);

if (n < 0) {

printf("Factorial of a negative number is not defined.\n");

} else {

printf("Factorial of %d = %d\n", n, factorial(n));

}

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>

void swap(int \*x, int \*y) {

int temp = \*x;

\*x = \*y;

\*y = temp;

}

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++;

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

}

}

swap(&arr[i + 1], &arr[high]);

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 size) {

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

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

}

printf("\n");

}

int main() {

int n;

printf("How many elements are you going to enter?: ");

scanf("%d", &n);

int arr[n];

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

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

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

}

quickSort(arr, 0, n - 1);

printf("Order of Sorted elements:\n");

printArray(arr, n);

return 0;

}