**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**

#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.**

#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**.

#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**

#include <stdio.h>

#include <stdlib.h>

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

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

while (i < size1 && j < size2) {

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

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

} else {

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

}

}

while (i < size1) {

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

}

while (j < size2) {

merged[k++] = 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, 3, 5, 7};

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

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

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

int merged[size1 + size2];

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

printf("Merged Array: ");

printArray(merged, size1 + size2);

return 0;

}

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

#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 key) {

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

return root;

}

if (key < root->data) {

return search(root->left, key);

} else {

return search(root->right, key);

}

}

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 inorder(Node\* root) {

if (root != NULL) {

inorder(root->left);

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

inorder(root->right);

}

}

int main() {

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("In-order Traversal: ");

inorder(root);

printf("\n");

int key = 40;

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

if (result != NULL) {

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

} else {

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

}

Node\* minNode = findMin(root);

Node\* maxNode = findMax(root);

printf("Minimum element: %d\n

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

#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, 102, 103, 104, 105};

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

int searchNo;

printf("Enter registration number to search: ");

scanf("%d", &searchNo);

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

if (result != -1) {

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

} else {

printf("Registration number %d not found in the array.\n", searchNo);

}

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)**

#include <stdio.h>

#include <stdbool.h>

#include <string.h>

bool allCharsPresent(const char\* needle, const char\* haystack) {

int haystackLength = strlen(haystack);

int needleLength = strlen(needle);

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

bool found = false;

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

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

found = true;

break;

}

}

if (!found) {

return false;

}

}

return true;

}

int main() {

char needle[100];

char haystack[100];

printf("Enter the needle string: ");

fgets(needle, sizeof(needle), stdin);

needle[strcspn(needle, "\n")] = '\0'; // Remove newline character

printf("Enter the haystack string: ");

fgets(haystack, sizeof(haystack), stdin);

haystack[strcspn(haystack, "\n")] = '\0'; // Remove newline character

if (allCharsPresent(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.**

**Array=[25,12,43]**

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

1. **occurs 1 times**

**12 occurs 1 times**

**43 occurs 1 times.**

#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];

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

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

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

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

}

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

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

int count = 1;

int alreadyCounted = 0;

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

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

alreadyCounted = 1;

break;

}

}

if (!alreadyCounted) {

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

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

count++;

}

}

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

}

}

return 0;

}

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

#include <stdio.h>

#include <limits.h>

#define V 5

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

int min = INT\_MAX, min\_index;

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

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

min = key[v];

min\_index = v;

}

}

return min\_index;

}

void printMST(int parent[], 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];

int mstSet[V];

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

key[i] = INT\_MAX;

mstSet[i] = 0;

}

key[0] = 0;

parent[0] = -1;

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

int u = minKey(key, mstSet);

mstSet[u] = 1;

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

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

parent[v] = u;

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

}

}

}

printMST(parent, 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 :**

**Array=[42,25,56,36,47]**

**Expected Output :**

**The Even elements are :**

**42 56 32**

**The Odd elements are :**

**25 47**

#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];

int even[n];

int 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)**

#include <stdio.h>

#include <stdlib.h>

#define V 5 // Number of vertices in the graph

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

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

printf("%d ", j);

return;

}

printPath(parent, parent[j]);

printf("%d ", j);

}

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

int visited[V] = {0};

int queue[V], front = 0, rear = 0;

int parent[V];

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

parent[i] = -1;

}

visited[start] = 1;

queue[rear++] = start;

while (front < rear) {

int u = queue[front++];

if (u == end) {

printf("Path from %d to %d using BFS:\n", start, end);

printPath(parent, end);

printf("\n");

return;

}

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

if (graph[u][v] && !visited[v]) {

visited[v] = 1;

parent[v] = u;

queue[rear++] = v;

}

}

}

printf("No path found using BFS.\n");

}

void DFSUtil(int graph[V][V], int v, int visited[], int parent[], int end) {

visited[v] = 1;

if (v == end) {

printf("Path from %d to %d using DFS:\n", parent[v], end);

printPath(parent, end);

printf("\n");

exit(0);

}

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

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

parent[i] = v;

DFSUtil(graph, i, visited, parent, end);

}

}

}

void DFS(int graph[V][V], int start, int end) {

int visited[V] = {0};

int parent[V];

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

parent[i] = -1;

}

DFSUtil(graph, start, visited, parent, end);

printf("No path found using DFS.\n");

}

int main() {

int graph[V][V] = {

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

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

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

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

{0, 1, 1, 1, 0}

};

int start, end;

printf("Enter start vertex (0 to %d): ", V-1);

scanf("%d", &start);

printf("Enter end vertex (0 to %d): ", V-1);

scanf("%d", &end);

if (start < 0 || start >= V || end < 0 || end >= V) {

printf("Invalid vertex.\n");

return 1;

}

printf("Choose the traversal method (1 for BFS, 2 for DFS): ");

int choice;

scanf("%d", &choice);

switch (choice) {

case 1:

BFS(graph, start, end);

break;

case 2:

DFS(graph, start, end);

break;

default:

printf("Invalid choice.\n");

break;

}

return 0;

}

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

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

}

int main() {

int terms;

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

scanf("%d", &terms);

if (terms <= 0) {

printf("Number of terms must be positive.\n");

return 1;

}

int sum = sumFibonacci(terms);

printf("The sum of the first %d terms of the Fibonacci series is: %d\n", terms, sum);

return 0;

}

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

#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 = sizeof(arr) / sizeof(arr[0]);

printf("Unsorted 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**

#include <stdio.h>

int factorial(int n) {

if (n == 0) {

return 1;

} else {

return n \* factorial(n - 1);

}

}

int main() {

int number;

printf("Enter a positive integer: ");

scanf("%d", &number);

if (number < 0) {

printf("Please enter a non-negative integer.\n");

return 1;

}

int result = factorial(number);

printf("Factorial of %d = %d\n", number, 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**

#include <stdio.h>

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

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; 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) {

// Partitioning index

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

// Recursively sort elements before and after partition

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");

}

// Main function

int main() {

int n;

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

scanf("%d", &n);

int arr[n];

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

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

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

quickSort(arr, 0, n - 1);

printf("Order of Sorted elements: ");

printArray(arr, n);

return 0;

}

**21.Implement a C Program for an Array to sort in ascending order**

#include <stdio.h>

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

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

for (int j = 0; j < n - 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 n;

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

scanf("%d", &n);

int arr[n];

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

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

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

bubbleSort(arr, n);

printf("Array in ascending order: ");

printArray(arr, n);

return 0;

}

**22.Implement a C Program to Print no of nodes in the given linked list**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int data;

struct Node\* next;

};

int countNodes(struct Node\* head) {

int count = 0;

struct Node\* current = head;

while (current != NULL) {

count++;

current = current->next;

}

return count;

}

void insertEnd(struct Node\*\* head\_ref, int new\_data) {

struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));

struct Node\* last = \*head\_ref;

new\_node->data = new\_data;

new\_node->next = NULL;

if (\*head\_ref == NULL) {

\*head\_ref = new\_node;

return;

}

while (last->next != NULL) {

last = last->next;

}

last->next = new\_node;

}

void printList(struct Node\* node) {

while (node != NULL) {

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

node = node->next;

}

printf("NULL\n");

}

int main() {

struct Node\* head = NULL;

insertEnd(&head, 10);

insertEnd(&head, 20);

insertEnd(&head, 30);

insertEnd(&head, 40);

printf("Linked List: ");

printList(head);

int num\_nodes = countNodes(head);

printf("Number of nodes in the linked list: %d\n", num\_nodes);

return 0;

}

**23.A Implement a C Program to perform search for sorted elements**

#include <stdio.h>

int binarySearch(int arr[], int size, int target) {

int left = 0;

int right = size - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (arr[mid] == target) {

return mid;

} else if (arr[mid] < target) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return -1;

}

int main() {

int n, target, result;

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

scanf("%d", &n);

int arr[n];

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

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

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

}

printf("Enter the element to search: ");

scanf("%d", &target);

result = binarySearch(arr, n, target);

if (result != -1) {

printf("Element found at index %d\n", result);

} else {

printf("Element not found\n");

}

return 0;

}

**24.Implement a C Program to Print the index of repeated characters given in an array**

#include <stdio.h>

#define MAX 256

void printRepeatedIndices(char arr[], int size) {

int visited[MAX] = {0};

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

if (visited[(unsigned char)arr[i]] == 1) {

printf("Character '%c' is repeated at index %d\n", arr[i], i);

} else {

visited[(unsigned char)arr[i]] = 1;

}

}

}

int main() {

int n;

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

scanf("%d", &n);

char arr[n];

printf("Enter %d characters: ", n);

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

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

}

printRepeatedIndices(arr, n);

return 0;

}

**25.Implement a C Program for given set of Array elements - display 5th iterated element**

#include <stdio.h>

int main() {

int n;

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

scanf("%d", &n);

if (n < 5) {

printf("Array does not have enough elements.\n");

return 1;

}

int arr[n];

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

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

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

}

printf("The 5th iterated element is: %d\n", arr[4]);

return 0;

}