Q1) Write a program in C to implement Linear Search

Code

#include <stdio.h>

int linearSearch(int arr[], int n, int key) {

    for (int i = 0; i < n; i++) {

        if (arr[i] == key) {

            return i;

        }

    }

    return -1;

}

int main() {

    int n, key;

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

    scanf("%d", &n);

    int arr[n];

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

    for (int i = 0; i < n; i++) {

        scanf("%d", &arr[i]);

    }

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

    scanf("%d", &key);

    int result = linearSearch(arr, n, key);

    if (result == -1) {

        printf("Element not found in the array.\n");

    } else {

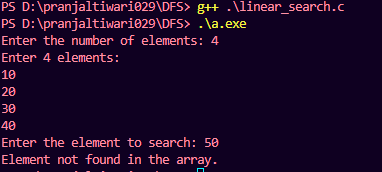
        printf("Element found at index %d.\n", result);

    }

    return 0;

}

Output



Q2) Write a program in C to implement Binary Search

#include <stdio.h>

int binarySearch(int arr[], int n, int key) {

    int low = 0, high = n - 1;

    while (low <= high) {

        int mid = low + (high - low) / 2;

        if (arr[mid] == key)

            return mid;

        else if (arr[mid] < key)

            low = mid + 1;

        else

            high = mid - 1;

    }

    return -1;

}

int main() {

    int n, key;

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

    scanf("%d", &n);

    int arr[n];

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

    for (int i = 0; i < n; i++) {

        scanf("%d", &arr[i]);

    }

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

    scanf("%d", &key);

    int result = binarySearch(arr, n, key);

    if (result == -1) {

        printf("Element not found in the array.\n");

    } else {

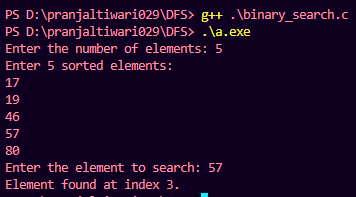
        printf("Element found at index %d.\n", result);

    }

    return 0;

}

OUTPUT



Q3) Write a program in C to implement Matrix Multiplication on two Matrices of 3x3.

Code

#include <stdio.h>

int main() {

    int a[3][3], b[3][3], result[3][3];

    int i, j, k;

    printf("Enter elements of first 3x3 matrix:\n");

    for (i = 0; i < 3; i++) {

        for (j = 0; j < 3; j++) {

            scanf("%d", &a[i][j]);

        }

    }

    printf("Enter elements of second 3x3 matrix:\n");

    for (i = 0; i < 3; i++) {

        for (j = 0; j < 3; j++) {

            scanf("%d", &b[i][j]);

        }

    }

    for (i = 0; i < 3; i++) {

        for (j = 0; j < 3; j++) {

            result[i][j] = 0;

        }

    }

    for (i = 0; i < 3; i++) {

        for (j = 0; j < 3; j++) {

            for (k = 0; k < 3; k++) {

                result[i][j] += a[i][k] \* b[k][j];

            }

        }

    }

    printf("Resultant matrix after multiplication:\n");

    for (i = 0; i < 3; i++) {

        for (j = 0; j < 3; j++) {

            printf("%d\t", result[i][j]);

        }

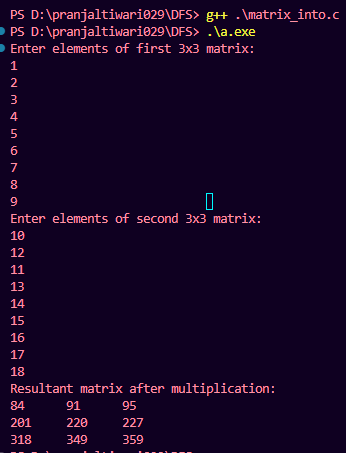
        printf("\n");

    }

    return 0;

}

OUTPUT



Q4) Write a program in C to implement Selection Sort

Code

#include <stdio.h>

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

    int i, j, min\_idx, temp;

    for (i = 0; i < n-1; i++) {

        min\_idx = i; *// Assume the current element is the minimum*

*// Find the index of the minimum element in the remaining array*

        for (j = i+1; j < n; j++) {

            if (arr[j] < arr[min\_idx]) {

                min\_idx = j;

            }

        }

*// Swap the found minimum element with the first element*

        temp = arr[min\_idx];

        arr[min\_idx] = arr[i];

        arr[i] = temp;

    }

}

int main() {

    int n, i;

    printf("Enter number of elements: ");

    scanf("%d", &n);

    int arr[n];

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

    for (i = 0; i < n; i++) {

        scanf("%d", &arr[i]);

    }

    selectionSort(arr, n);

    printf("Sorted array:\n");

    for (i = 0; i < n; i++) {

        printf("%d ", arr[i]);

    }

    printf("\n");

    return 0;

}

OUTPUT

A screenshot of a computer

AI-generated content may be incorrect.

Q5) Write a program in C to implement insertion sort

#include <stdio.h>

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

    int i, key, j;

    for (i = 1; i < n; i++) {

        key = arr[i]; *// current element to be inserted*

        j = i - 1;

*// Move elements of arr[0..i-1] that are greater than key*

*// to one position ahead of their current position*

        while (j >= 0 && arr[j] > key) {

            arr[j + 1] = arr[j];

            j = j - 1;

        }

        arr[j + 1] = key; *// Insert the key at correct position*

    }

}

int main() {

    int n, i;

    printf("Enter number of elements: ");

    scanf("%d", &n);

    int arr[n];

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

    for (i = 0; i < n; i++) {

        scanf("%d", &arr[i]);

    }

    insertionSort(arr, n);

    printf("Sorted array:\n");

    for (i = 0; i < n; i++) {

        printf("%d ", arr[i]);

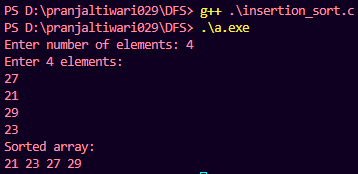
    }

    printf("\n");

    return 0;

}

OUTPUT



Q6) Write a program in C to implement Bubble Sort

Code

#include <stdio.h>

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

    int i, j, temp;

    for (i = 0; i < n-1; i++) {

*// Last i elements are already in place*

        for (j = 0; j < n-i-1; j++) {

            if (arr[j] > arr[j+1]) {

*// Swap arr[j] and arr[j+1]*

                temp = arr[j];

                arr[j] = arr[j+1];

                arr[j+1] = temp;

            }

        }

    }

}

int main() {

    int n, i;

    printf("Enter number of elements: ");

    scanf("%d", &n);

    int arr[n];

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

    for (i = 0; i < n; i++) {

        scanf("%d", &arr[i]);

    }

    bubbleSort(arr, n);

    printf("Sorted array:\n");

    for (i = 0; i < n; i++) {

        printf("%d ", arr[i]);

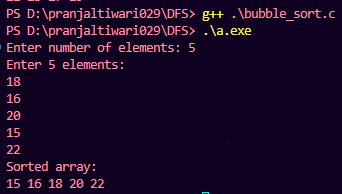
    }

    printf("\n");

    return 0;

}

Output



Q7) Write a program in C to implement Merge Sort

Code

#include <stdio.h>

*// Merge two subarrays of arr[]*

void merge(int arr[], int l, int m, int r) {

    int i, j, k;

    int n1 = m - l + 1;

    int n2 = r - m;

*// Create temporary arrays*

    int L[n1], R[n2];

*// Copy data to temp arrays L[] and R[]*

    for (i = 0; i < n1; i++)

        L[i] = arr[l + i];

    for (j = 0; j < n2; j++)

        R[j] = arr[m + 1 + j];

*// Merge the temp arrays back into arr[l..r]*

    i = 0; *// Initial index of first subarray*

    j = 0; *// Initial index of second subarray*

    k = l; *// Initial index of merged subarray*

    while (i < n1 && j < n2) {

        if (L[i] <= R[j]) {

            arr[k] = L[i];

            i++;

        } else {

            arr[k] = R[j];

            j++;

        }

        k++;

    }

*// Copy remaining elements of L[], if any*

    while (i < n1) {

        arr[k] = L[i];

        i++;

        k++;

    }

*// Copy remaining elements of R[], if any*

    while (j < n2) {

        arr[k] = R[j];

        j++;

        k++;

    }

}

*// l is for left index and r is right index*

void mergeSort(int arr[], int l, int r) {

    if (l < r) {

        int m = l + (r - l) / 2; *// Find the middle point*

        mergeSort(arr, l, m); *// Sort first half*

        mergeSort(arr, m + 1, r); *// Sort second half*

        merge(arr, l, m, r); *// Merge the sorted halves*

    }

}

int main() {

    int n;

    printf("Enter number of elements: ");

    scanf("%d", &n);

    int arr[n];

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

    for (int i = 0; i < n; i++) {

        scanf("%d", &arr[i]);

    }

    mergeSort(arr, 0, n - 1);

    printf("Sorted array:\n");

    for (int i = 0; i < n; i++) {

        printf("%d ", arr[i]);

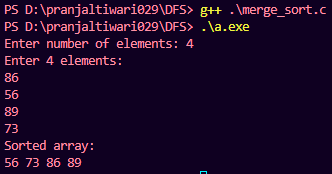
    }

    printf("\n");

    return 0;

}

OUTPUT



Q8) Write a program in C to implement quick sort

Code

#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]; *// choosing the last element as pivot*

    int i = (low - 1); *// index of smaller element*

    for (int j = low; j <= high - 1; j++) {

        if (arr[j] < pivot) {

            i++;

            swap(&arr[i], &arr[j]);

        }

    }

    swap(&arr[i + 1], &arr[high]);

    return (i + 1); *// return the partition point*

}

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

    if (low < high) {

        int pi = partition(arr, low, high); *// pi is partitioning index*

        quickSort(arr, low, pi - 1); *// sort elements before partition*

        quickSort(arr, pi + 1, high); *// sort elements after partition*

    }

}

int main() {

    int n;

    printf("Enter number of elements: ");

    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("Sorted array:\n");

    for (int i = 0; i < n; i++) {

        printf("%d ", arr[i]);

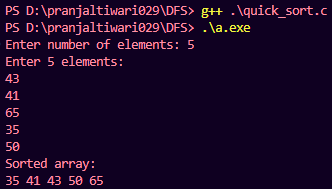
    }

    printf("\n");

    return 0;

}

Output



Q9) Write a program in C to implement Count Sort.

Code

#include <stdio.h>

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

    int i;

*// Find the maximum element in the array*

    int max = arr[0];

    for (i = 1; i < n; i++) {

        if (arr[i] > max)

            max = arr[i];

    }

*// Create a count array to store count of individual elements*

    int count[max + 1];

*// Initialize count array with 0*

    for (i = 0; i <= max; i++) {

        count[i] = 0;

    }

*// Store the count of each element*

    for (i = 0; i < n; i++) {

        count[arr[i]]++;

    }

*// Modify the original array using the count array*

    int index = 0;

    for (i = 0; i <= max; i++) {

        while (count[i] > 0) {

            arr[index++] = i;

            count[i]--;

        }

    }

}

int main() {

    int n, i;

    printf("Enter number of elements: ");

    scanf("%d", &n);

    int arr[n];

    printf("Enter %d non-negative integers:\n", n);

    for (i = 0; i < n; i++) {

        scanf("%d", &arr[i]);

    }

    countSort(arr, n);

    printf("Sorted array:\n");

    for (i = 0; i < n; i++) {

        printf("%d ", arr[i]);

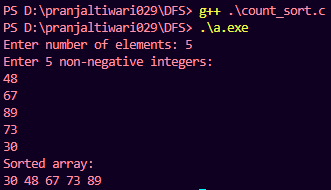
    }

    printf("\n");

    return 0;

}

Output



Q10) Write a program in C to implement Radix Sort.

Code

#include <stdio.h>

*// A utility function to get the maximum value in arr[]*

int getMax(int arr[], int n) {

    int max = arr[0];

    for (int i = 1; i < n; i++)

        if (arr[i] > max)

            max = arr[i];

    return max;

}

*// A function to do counting sort based on a specific digit (exp is 1, 10, 100, etc.)*

void countSort(int arr[], int n, int exp) {

    int output[n]; *// output array*

    int count[10] = {0};

*// Store count of occurrences*

    for (int i = 0; i < n; i++)

        count[(arr[i] / exp) % 10]++;

*// Change count[i] so that it contains actual position*

    for (int i = 1; i < 10; i++)

        count[i] += count[i - 1];

*// Build the output array*

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

        int digit = (arr[i] / exp) % 10;

        output[count[digit] - 1] = arr[i];

        count[digit]--;

    }

*// Copy the output array back to arr[]*

    for (int i = 0; i < n; i++)

        arr[i] = output[i];

}

*// The main function to that sorts arr[] using Radix Sort*

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

    int max = getMax(arr, n);

*// Do counting sort for every digit (exp = 1, 10, 100, ...)*

    for (int exp = 1; max / exp > 0; exp \*= 10)

        countSort(arr, n, exp);

}

int main() {

    int n;

    printf("Enter number of elements: ");

    scanf("%d", &n);

    int arr[n];

    printf("Enter %d non-negative integers:\n", n);

    for (int i = 0; i < n; i++) {

        scanf("%d", &arr[i]);

    }

    radixSort(arr, n);

    printf("Sorted array:\n");

    for (int i = 0; i < n; i++) {

        printf("%d ", arr[i]);

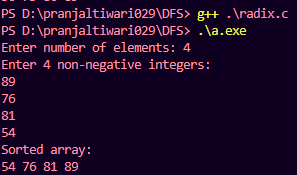
    }

    printf("\n");

    return 0;

}

Output



Q11) Write a program in C to implement binary trees operations and traversals .

Code

#include <stdio.h>

#include <stdlib.h>

*// Define the structure for tree node*

typedef struct Node {

    int data;

    struct Node\* left;

    struct Node\* right;

} Node;

*// Define queue structure for level order operations*

typedef struct QueueNode {

    Node\* treeNode;

    struct QueueNode\* next;

} QueueNode;

typedef struct Queue {

    QueueNode \*front, \*rear;

} Queue;

*// Function to create a new tree node*

Node\* createNode(int data) {

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

    newNode->data = data;

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

    return newNode;

}

*// ----------------- Queue Functions -----------------*

Queue\* createQueue() {

    Queue\* q = (Queue\*)malloc(sizeof(Queue));

    q->front = q->rear = NULL;

    return q;

}

void enqueue(Queue\* q, Node\* node) {

    QueueNode\* temp = (QueueNode\*)malloc(sizeof(QueueNode));

    temp->treeNode = node;

    temp->next = NULL;

    if (q->rear == NULL) {

        q->front = q->rear = temp;

        return;

    }

    q->rear->next = temp;

    q->rear = temp;

}

Node\* dequeue(Queue\* q) {

    if (q->front == NULL)

        return NULL;

    QueueNode\* temp = q->front;

    Node\* node = temp->treeNode;

    q->front = q->front->next;

    if (q->front == NULL)

        q->rear = NULL;

    free(temp);

    return node;

}

int isQueueEmpty(Queue\* q) {

    return q->front == NULL;

}

*// ----------------- Build Tree using Level Order -----------------*

Node\* buildTree(int arr[], int n) {

    if (n == 0)

        return NULL;

    Node\* root = createNode(arr[0]);

    Queue\* q = createQueue();

    enqueue(q, root);

    int i = 1;

    while (i < n) {

        Node\* temp = dequeue(q);

        if (i < n) {

            temp->left = createNode(arr[i++]);

            enqueue(q, temp->left);

        }

        if (i < n) {

            temp->right = createNode(arr[i++]);

            enqueue(q, temp->right);

        }

    }

    return root;

}

*// ----------------- Recursive Traversals -----------------*

void preorderRecursive(Node\* root) {

    if (root == NULL)

        return;

    printf("%d ", root->data);

    preorderRecursive(root->left);

    preorderRecursive(root->right);

}

void inorderRecursive(Node\* root) {

    if (root == NULL)

        return;

    inorderRecursive(root->left);

    printf("%d ", root->data);

    inorderRecursive(root->right);

}

void postorderRecursive(Node\* root) {

    if (root == NULL)

        return;

    postorderRecursive(root->left);

    postorderRecursive(root->right);

    printf("%d ", root->data);

}

*// ----------------- Level-order Traversal -----------------*

void levelOrderTraversal(Node\* root) {

    if (root == NULL)

        return;

    Queue\* q = createQueue();

    enqueue(q, root);

    while (!isQueueEmpty(q)) {

        Node\* temp = dequeue(q);

        printf("%d ", temp->data);

        if (temp->left)

            enqueue(q, temp->left);

        if (temp->right)

            enqueue(q, temp->right);

    }

}

*// ----------------- Main Function -----------------*

int main() {

    int arr[] = {1, 2, 3, 4, 5, 6}; *// Sample input*

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

    Node\* root = buildTree(arr, n);

    printf("\nRecursive Preorder Traversal: ");

    preorderRecursive(root);

    printf("\nRecursive Inorder Traversal: ");

    inorderRecursive(root);

    printf("\nRecursive Postorder Traversal: ");

    postorderRecursive(root);

    printf("\n\nLevel-order Traversal: ");

    levelOrderTraversal(root);

    printf("\n");

    return 0;

}

OUTPUT  
A screenshot of a computer code

AI-generated content may be incorrect.

Q12) Write a program in C to implement binary search trees

Code

#include <stdio.h>

#include <stdlib.h>

*// Node structure*

typedef struct Node {

    int data;

    struct Node\* left;

    struct Node\* right;

} Node;

*// Function to create a new node*

Node\* createNode(int data) {

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

    newNode->data = data;

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

    return newNode;

}

*// Insert into BST*

Node\* insert(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;

}

*// Inorder Traversal*

void inorderTraversal(Node\* root) {

    if (root == NULL)

        return;

    inorderTraversal(root->left);

    printf("%d ", root->data);

    inorderTraversal(root->right);

}

*// Search in BST*

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

    if (root == NULL)

        return 0;

    if (root->data == key)

        return 1;

    else if (key < root->data)

        return search(root->left, key);

    else

        return search(root->right, key);

}

*// Find minimum value node*

Node\* findMin(Node\* root) {

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

        root = root->left;

    return root;

}

*// Delete a node from BST*

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

    if (root == NULL)

        return NULL;

    if (key < root->data)

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

    else if (key > root->data)

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

    else {

*// Node found*

        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 with two children*

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

        root->data = temp->data;

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

    }

    return root;

}

*// Main function*

int main() {

    Node\* root = NULL;

*// Insert elements*

    int elements[] = {56, 33, 71, 20, 49, 63, 88};

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

    for (int i = 0; i < n; i++) {

        root = insert(root, elements[i]);

    }

    printf("Inorder Traversal after insertions: ");

    inorderTraversal(root);

    printf("\n");

*// Delete nodes:*

    int toDelete[] = {20, 33, 49};

    for (int i = 0; i < 3; i++) {

        root = deleteNode(root, toDelete[i]);

    }

    printf("\nInorder Traversal after deletions: ");

    inorderTraversal(root);

    printf("\n");

    int toSearch[] = {71, 88};

    for (int i = 0; i < 2; i++) {

        if (search(root, toSearch[i]))

            printf("\nSearch %d: Found", toSearch[i]);

        else

            printf("\nSearch %d: Not Found", toSearch[i]);

    }

    printf("\n");

    return 0;

}

Output

A computer screen with text and numbers

AI-generated content may be incorrect.

Q13) Write a program in C to implement avl trees

Code

#include <stdio.h>

#include <stdlib.h>

*// AVL Tree Node*

typedef struct Node {

    int key;

    struct Node\* left;

    struct Node\* right;

    int height;

} Node;

*// Utility to get max*

int max(int a, int b) {

    return (a > b) ? a : b;

}

*// Get height of node*

int height(Node\* N) {

    if (N == NULL)

        return 0;

    return N->height;

}

*// Create a new node*

Node\* newNode(int key) {

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

    node->key = key;

    node->left = node->right = NULL;

    node->height = 1; *// New node is initially at height 1*

    return node;

}

*// Right rotate*

Node\* rightRotate(Node\* y) {

    Node\* x = y->left;

    Node\* T2 = x->right;

*// Perform rotation*

    x->right = y;

    y->left = T2;

*// Update heights*

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

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

    return x;

}

*// Left rotate*

Node\* leftRotate(Node\* x) {

    Node\* y = x->right;

    Node\* T2 = y->left;

*// Perform rotation*

    y->left = x;

    x->right = T2;

*// Update heights*

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

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

    return y;

}

*// Get balance factor*

int getBalance(Node\* N) {

    if (N == NULL)

        return 0;

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

}

*// Insert into AVL Tree*

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; *// Equal keys not allowed*

*// Update height*

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

    int balance = getBalance(node);

*// If unbalanced, there are 4 cases:*

*// Left Left Case*

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

        return rightRotate(node);

*// Right Right Case*

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

        return leftRotate(node);

*// Left Right Case*

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

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

        return rightRotate(node);

    }

*// Right Left Case*

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

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

        return leftRotate(node);

    }

    return node;

}

*// Find minimum node*

Node\* minValueNode(Node\* node) {

    Node\* current = node;

    while (current->left != NULL)

        current = current->left;

    return current;

}

*// Delete from AVL Tree*

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);

*// Left Left*

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

        return rightRotate(root);

*// Left Right*

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

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

        return rightRotate(root);

    }

*// Right Right*

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

        return leftRotate(root);

*// Right Left*

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

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

        return leftRotate(root);

    }

    return root;

}

*// Print Level Order Traversal*

void printLevelOrder(Node\* root) {

    if (root == NULL)

        return;

    Node\* queue[100];

    int front = 0, rear = 0;

    queue[rear++] = root;

    while (front < rear) {

        Node\* current = queue[front++];

        printf("%d ", current->key);

        if (current->left != NULL)

            queue[rear++] = current->left;

        if (current->right != NULL)

            queue[rear++] = current->right;

    }

}

int main() {

    Node\* root = NULL;

    int insert\_elements[] = {10, 20, 30, 40, 50, 25};

    int n = sizeof(insert\_elements) / sizeof(insert\_elements[0]);

    printf("Level-order after insertions:\n");

    for (int i = 0; i < n; i++) {

        root = insert(root, insert\_elements[i]);

        printLevelOrder(root);

        printf("\n");

    }

    printf("\nDeleting 40...\n");

    root = deleteNode(root, 40);

    printf("Level-order after deletion:\n");

    printLevelOrder(root);

    printf("\n");

    return 0;

}

Output

A screenshot of a computer

AI-generated content may be incorrect.

Q14) Write a program in C to implement Tree Sort using BST.

Code

#include <stdio.h>

#include <stdlib.h>

*// Define a BST node*

typedef struct Node {

    int key;

    struct Node\* left;

    struct Node\* right;

} Node;

*// Create a new node*

Node\* newNode(int key) {

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

    node->key = key;

    node->left = node->right = NULL;

    return node;

}

*// Insert into BST*

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

    if (root == NULL)

        return newNode(key);

    if (key < root->key)

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

    else

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

    return root;

}

*// Inorder Traversal (prints sorted elements)*

void inorderTraversal(Node\* root) {

    if (root != NULL) {

        inorderTraversal(root->left);

        printf("%d ", root->key);

        inorderTraversal(root->right);

    }

}

int main() {

    int elements[] = {5, 3, 7, 2, 8, 4};

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

    Node\* root = NULL;

*// Insert elements into BST*

    for (int i = 0; i < n; i++) {

        root = insert(root, elements[i]);

    }

    printf("Sorted: ");

    inorderTraversal(root);

    printf("\n");

    return 0;

}

Output

A blue background with red and white text

AI-generated content may be incorrect.

Q16) Write a program in C to implement heap sort.  
Code

#include <stdio.h>

*// Function to heapify a subtree rooted at index i in a max heap*

void heapify(int arr[], int n, int i) {

    int largest = i; *// Initialize largest as root*

    int left = 2 \* i + 1; *// Left child index*

    int right = 2 \* i + 2; *// Right child index*

*// If left child is larger than root*

    if (left < n && arr[left] > arr[largest]) {

        largest = left;

    }

*// If right child is larger than the largest so far*

    if (right < n && arr[right] > arr[largest]) {

        largest = right;

    }

*// If largest is not root*

    if (largest != i) {

*// Swap root and largest*

        int temp = arr[i];

        arr[i] = arr[largest];

        arr[largest] = temp;

*// Recursively heapify the affected subtree*

        heapify(arr, n, largest);

    }

}

*// Function to perform heap sort*

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

*// Build a max heap (rearrange array)*

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

        heapify(arr, n, i);

    }

*// One by one extract elements from the heap*

    for (int i = n - 1; i > 0; i--) {

*// Swap current root with the end element*

        int temp = arr[0];

        arr[0] = arr[i];

        arr[i] = temp;

*// Call heapify on the reduced heap*

        heapify(arr, i, 0);

    }

}

*// Function to print the array*

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

    for (int i = 0; i < n; i++) {

        printf("%d ", arr[i]);

    }

    printf("\n");

}

int main() {

    int arr[] = {3, 19, 1, 14, 8, 7};

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

    printf("Input: ");

    printArray(arr, n);

    heapSort(arr, n);

    printf("Output: ");

    printArray(arr, n);

    return 0;

}

Output

A screenshot of a computer

AI-generated content may be incorrect.

Q18) Write a program in C to implement Graph Traversals & Connected Components using Adjacency Matrix

Code

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

int adj[MAX][MAX]; *// Adjacency matrix*

int visited[MAX]; *// Visited array*

int queue[MAX]; *// Queue for BFS*

int front = -1, rear = -1;

*// Function to insert into queue*

void enqueue(int v) {

    if (rear == MAX - 1)

        return;

    if (front == -1)

        front = 0;

    queue[++rear] = v;

}

*// Function to delete from queue*

int dequeue() {

    if (front == -1 || front > rear)

        return -1;

    return queue[front++];

}

*// BFS Traversal*

void bfs(int start, int V) {

    for (int i = 0; i < V; i++)

        visited[i] = 0;

    enqueue(start);

    visited[start] = 1;

    printf("BFS Traversal: ");

    while (front <= rear) {

        int current = dequeue();

        printf("%d ", current);

        for (int i = 0; i < V; i++) {

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

                enqueue(i);

                visited[i] = 1;

            }

        }

    }

    printf("\n");

}

*// DFS Traversal (Recursive)*

void dfs\_util(int v, int V) {

    visited[v] = 1;

    printf("%d ", v);

    for (int i = 0; i < V; i++) {

        if (adj[v][i] && !visited[i])

            dfs\_util(i, V);

    }

}

void dfs(int start, int V) {

    for (int i = 0; i < V; i++)

        visited[i] = 0;

    printf("DFS Traversal: ");

    dfs\_util(start, V);

    printf("\n");

}

*// Function to find connected components*

void find\_connected\_components(int V) {

    for (int i = 0; i < V; i++)

        visited[i] = 0;

    int count = 0;

    printf("Connected Components:\n");

    for (int i = 0; i < V; i++) {

        if (!visited[i]) {

            count++;

            printf("Component %d: ", count);

            dfs\_util(i, V);

            printf("\n");

        }

    }

    printf("Total Connected Components: %d\n", count);

}

int main() {

    int V, E;

    printf("Enter number of vertices and edges: ");

    scanf("%d %d", &V, &E);

*// Initialize adjacency matrix*

    for (int i = 0; i < V; i++)

        for (int j = 0; j < V; j++)

            adj[i][j] = 0;

    printf("Enter edges (pairs of vertices):\n");

    for (int i = 0; i < E; i++) {

        int u, v;

        scanf("%d %d", &u, &v);

        adj[u][v] = 1;

        adj[v][u] = 1; *// Since undirected*

    }

    printf("Enter starting vertex for traversal: ");

    int start;

    scanf("%d", &start);

*// Print adjacency matrix*

    printf("\nAdjacency Matrix:\n");

    for (int i = 0; i < V; i++) {

        for (int j = 0; j < V; j++) {

            printf("%d ", adj[i][j]);

        }

        printf("\n");

    }

    printf("\n");

    bfs(start, V);

    dfs(start, V);

    printf("\n");

    find\_connected\_components(V);

    return 0;

}

Output

A screenshot of a computer program

AI-generated content may be incorrect.

Q19) Write a program in C to implement Krushkal’s Algorithm

Code

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

*// Structure to represent an edge*

typedef struct {

    int u, v, w;

} Edge;

int adj[MAX][MAX];

int parent[MAX];

*// Find function for Union-Find*

int find(int i) {

    if (parent[i] == i)

        return i;

    return parent[i] = find(parent[i]); *// Path compression*

}

*// Union function for Union-Find*

void union\_set(int u, int v) {

    int pu = find(u);

    int pv = find(v);

    if (pu != pv)

        parent[pu] = pv;

}

*// Compare function for qsort*

int compare(const void \*a, const void \*b) {

    Edge \*e1 = (Edge \*)a;

    Edge \*e2 = (Edge \*)b;

    return e1->w - e2->w;

}

int main() {

    int V, E;

    printf("Enter number of vertices and edges: ");

    scanf("%d %d", &V, &E);

*// Initialize adjacency matrix*

    for (int i = 0; i < V; i++)

        for (int j = 0; j < V; j++)

            adj[i][j] = 0;

    Edge edges[E];

    printf("Enter edges (u v w):\n");

    for (int i = 0; i < E; i++) {

        int u, v, w;

        scanf("%d %d %d", &u, &v, &w);

        adj[u][v] = w;

        adj[v][u] = w; *// Since undirected*

        edges[i].u = u;

        edges[i].v = v;

        edges[i].w = w;

    }

*// Print adjacency matrix*

    printf("\nAdjacency Matrix:\n");

    for (int i = 0; i < V; i++) {

        for (int j = 0; j < V; j++) {

            printf("%d ", adj[i][j]);

        }

        printf("\n");

    }

*// Sort edges by weight*

    qsort(edges, E, sizeof(Edge), compare);

*// Initialize Union-Find structure*

    for (int i = 0; i < V; i++)

        parent[i] = i;

    printf("\nEdges in MST:\n");

    int total\_weight = 0;

    for (int i = 0; i < E; i++) {

        int u = edges[i].u;

        int v = edges[i].v;

        int w = edges[i].w;

        if (find(u) != find(v)) {

            printf("%d - %d : %d\n", u, v, w);

            total\_weight += w;

            union\_set(u, v);

        }

    }

    printf("\nTotal weight of MST: %d\n", total\_weight);

    return 0;

}

Output

A screenshot of a computer screen

AI-generated content may be incorrect.

Q20) Write a program in C to implement Prim's Algorithm for MST

Code

#include <stdio.h>

#include <limits.h>

#define MAX 100

#define INF 99999

int adj[MAX][MAX];

int visited[MAX];

int main() {

    int V, E;

    printf("Enter number of vertices and edges: ");

    scanf("%d %d", &V, &E);

*// Initialize adjacency matrix*

    for (int i = 0; i < V; i++) {

        for (int j = 0; j < V; j++) {

            if (i == j)

                adj[i][j] = 0;

            else

                adj[i][j] = INF;

        }

    }

    printf("Enter edges (u v w):\n");

    for (int i = 0; i < E; i++) {

        int u, v, w;

        scanf("%d %d %d", &u, &v, &w);

        adj[u][v] = w;

        adj[v][u] = w; *// Undirected graph*

    }

    int start;

    printf("Enter starting vertex: ");

    scanf("%d", &start);

*// Print adjacency matrix*

    printf("\nAdjacency Matrix:\n");

    for (int i = 0; i < V; i++) {

        for (int j = 0; j < V; j++) {

            if (adj[i][j] == INF)

                printf("INF ");

            else

                printf("%d ", adj[i][j]);

        }

        printf("\n");

    }

*// Initialize visited array*

    for (int i = 0; i < V; i++)

        visited[i] = 0;

    visited[start] = 1;

    int edges\_accepted = 0;

    int total\_weight = 0;

    printf("\nEdges in MST:\n");

    while (edges\_accepted < V - 1) {

        int min = INF;

        int u = -1, v = -1;

        for (int i = 0; i < V; i++) {

            if (visited[i]) {

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

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

                        min = adj[i][j];

                        u = i;

                        v = j;

                    }

                }

            }

        }

        if (u != -1 && v != -1) {

            printf("%d - %d : %d\n", u, v, adj[u][v]);

            total\_weight += adj[u][v];

            visited[v] = 1;

            edges\_accepted++;

        }

    }

    printf("\nTotal weight of MST: %d\n", total\_weight);

    return 0;

}

Output

A screenshot of a computer program

AI-generated content may be incorrect.

Q22) Write a program in C to implement Floyd-Warshall Algorithm (All-pairs shortest path)

Code

#include <stdio.h>

#define MAX 100

#define INF 99999

int dist[MAX][MAX];

int next[MAX][MAX]; *// For path reconstruction*

void printPath(int u, int v) {

    if (next[u][v] == -1) {

        printf("No path");

        return;

    }

    printf("%d", u);

    while (u != v) {

        u = next[u][v];

        printf(" -> %d", u);

    }

}

int main() {

    int V, E;

    printf("Enter number of vertices and edges: ");

    scanf("%d %d", &V, &E);

*// Initialize distance and next matrices*

    for (int i = 0; i < V; i++) {

        for (int j = 0; j < V; j++) {

            if (i == j)

                dist[i][j] = 0;

            else

                dist[i][j] = INF;

            next[i][j] = -1;

        }

    }

    printf("Enter edges (u v w):\n");

    for (int i = 0; i < E; i++) {

        int u, v, w;

        scanf("%d %d %d", &u, &v, &w);

        dist[u][v] = w;

        next[u][v] = v;

    }

*// Print initial adjacency matrix*

    printf("\nInitial Adjacency Matrix:\n");

    for (int i = 0; i < V; i++) {

        for (int j = 0; j < V; j++) {

            if (dist[i][j] == INF)

                printf("INF ");

            else

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

        }

        printf("\n");

    }

*// Floyd-Warshall algorithm*

    for (int k = 0; k < V; k++) {

        for (int i = 0; i < V; i++) {

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

                if (dist[i][k] != INF && dist[k][j] != INF &&

                    dist[i][k] + dist[k][j] < dist[i][j]) {

                    dist[i][j] = dist[i][k] + dist[k][j];

                    next[i][j] = next[i][k];

                }

            }

        }

    }

*// Print final distance matrix*

    printf("\nFinal Distance Matrix (Shortest Paths):\n");

    for (int i = 0; i < V; i++) {

        for (int j = 0; j < V; j++) {

            if (dist[i][j] == INF)

                printf("INF ");

            else

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

        }

        printf("\n");

    }

*// Sample path reconstruction*

    int u, v;

    printf("\nEnter two vertices to reconstruct path (u v): ");

    scanf("%d %d", &u, &v);

    printf("Path from %d to %d: ", u, v);

    printPath(u, v);

    printf("\n");

    return 0;

}

Output

A screenshot of a computer program

AI-generated content may be incorrect.

Q23) Write a program in C to implement Topological Sort & Shortest Path in DAG

Code

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

#define INF 99999

int adj[MAX][MAX];

int visited[MAX];

int stack[MAX];

int top = -1;

*// Push to stack*

void push(int v) {

    stack[++top] = v;

}

*// Pop from stack*

int pop() {

    return stack[top--];

}

*// Topological Sort (DFS based)*

void dfs(int v, int V) {

    visited[v] = 1;

    for (int i = 0; i < V; i++) {

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

            dfs(i, V);

        }

    }

    push(v);

}

int main() {

    int V, E;

    printf("Enter number of vertices and edges: ");

    scanf("%d %d", &V, &E);

*// Initialize adjacency matrix*

    for (int i = 0; i < V; i++)

        for (int j = 0; j < V; j++)

            adj[i][j] = 0;

    printf("Enter edges (u v):\n");

    for (int i = 0; i < E; i++) {

        int u, v;

        scanf("%d %d", &u, &v);

        adj[u][v] = 1; *// Weight is 1*

    }

    int source;

    printf("Enter source vertex: ");

    scanf("%d", &source);

*// Print adjacency matrix*

    printf("\nAdjacency Matrix:\n");

    for (int i = 0; i < V; i++) {

        for (int j = 0; j < V; j++) {

            printf("%d ", adj[i][j]);

        }

        printf("\n");

    }

*// Initialize visited array*

    for (int i = 0; i < V; i++)

        visited[i] = 0;

*// Perform Topological Sort*

    for (int i = 0; i < V; i++) {

        if (!visited[i])

            dfs(i, V);

    }

    printf("\nTopological Order: ");

    for (int i = top; i >= 0; i--) {

        printf("%d ", stack[i]);

    }

    printf("\n");

*// Initialize distances*

    int dist[MAX];

    for (int i = 0; i < V; i++)

        dist[i] = INF;

    dist[source] = 0;

*// Shortest Path using Topological Order*

    while (top != -1) {

        int u = pop();

        if (dist[u] != INF) {

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

                if (adj[u][v]) {

                    if (dist[u] + 1 < dist[v]) {

                        dist[v] = dist[u] + 1;

                    }

                }

            }

        }

    }

    printf("\nShortest distances from source %d:\n", source);

    for (int i = 0; i < V; i++) {

        if (dist[i] == INF)

            printf("%d -> INF\n", i);

        else

            printf("%d -> %d\n", i, dist[i]);

    }

    return 0;

}

Output

A screenshot of a computer program

AI-generated content may be incorrect.

Q24) Write a program in C to implement Hash Table with Chaining

Code

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 100

typedef struct Node {

    int key;

    struct Node\* next;

} Node;

Node\* hashTable[MAX];

int table\_size;

*// Hash function*

int hash(int key) {

    return key % table\_size;

}

*// Insert key*

void insert(int key) {

    int index = hash(key);

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

    newNode->key = key;

    newNode->next = hashTable[index];

    hashTable[index] = newNode;

}

*// Search key*

void search(int key) {

    int index = hash(key);

    Node\* temp = hashTable[index];

    while (temp != NULL) {

        if (temp->key == key) {

            printf("Found\n");

            return;

        }

        temp = temp->next;

    }

    printf("Not Found\n");

}

*// Delete key*

void deleteKey(int key) {

    int index = hash(key);

    Node\* temp = hashTable[index];

    Node\* prev = NULL;

    while (temp != NULL) {

        if (temp->key == key) {

            if (prev == NULL) {

                hashTable[index] = temp->next;

            } else {

                prev->next = temp->next;

            }

            free(temp);

            printf("Deleted\n");

            return;

        }

        prev = temp;

        temp = temp->next;

    }

    printf("Key not found\n");

}

*// Display hash table*

void display() {

    for (int i = 0; i < table\_size; i++) {

        printf("[%d]: ", i);

        Node\* temp = hashTable[i];

        while (temp != NULL) {

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

            temp = temp->next;

        }

        printf("NULL\n");

    }

}

int main() {

    printf("Enter size of hash table: ");

    scanf("%d", &table\_size);

*// Initialize table*

    for (int i = 0; i < table\_size; i++) {

        hashTable[i] = NULL;

    }

    char command[20];

    int key;

    printf("Enter commands (insert <key>, search <key>, delete <key>, display, exit):\n");

    while (1) {

        scanf("%s", command);

        if (strcmp(command, "insert") == 0) {

            scanf("%d", &key);

            insert(key);

        } else if (strcmp(command, "search") == 0) {

            scanf("%d", &key);

            search(key);

        } else if (strcmp(command, "delete") == 0) {

            scanf("%d", &key);

            deleteKey(key);

        } else if (strcmp(command, "display") == 0) {

            display();

        } else if (strcmp(command, "exit") == 0) {

            break;

        } else {

            printf("Invalid command\n");

        }

    }

    return 0;

}

Output

A screenshot of a computer program

AI-generated content may be incorrect.

Q25) Write a program in C to implement Hash Table with Open Addressing (Linear Probing)

Code

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 100

#define EMPTY -1

#define DELETED -2

int table[MAX];

int table\_size;

*// Hash function*

int hash(int key) {

    return key % table\_size;

}

*// Insert key*

void insert(int key) {

    int index = hash(key);

    int original\_index = index;

    int i = 0;

    while (table[index] != EMPTY && table[index] != DELETED) {

        i++;

        index = (original\_index + i) % table\_size;

        if (i == table\_size) {

            printf("Hash Table is full! Cannot insert.\n");

            return;

        }

    }

    table[index] = key;

}

*// Search key*

void search(int key) {

    int index = hash(key);

    int original\_index = index;

    int i = 0;

    while (table[index] != EMPTY) {

        if (table[index] == key) {

            printf("Found at index %d\n", index);

            return;

        }

        i++;

        index = (original\_index + i) % table\_size;

        if (i == table\_size) {

            break;

        }

    }

    printf("Not Found\n");

}

*// Delete key*

void deleteKey(int key) {

    int index = hash(key);

    int original\_index = index;

    int i = 0;

    while (table[index] != EMPTY) {

        if (table[index] == key) {

            table[index] = DELETED;

            printf("Deleted\n");

            return;

        }

        i++;

        index = (original\_index + i) % table\_size;

        if (i == table\_size) {

            break;

        }

    }

    printf("Key not found\n");

}

*// Display hash table*

void display() {

    for (int i = 0; i < table\_size; i++) {

        if (table[i] == EMPTY) {

            printf("[%d]: EMPTY\n", i);

        } else if (table[i] == DELETED) {

            printf("[%d]: DELETED\n", i);

        } else {

            printf("[%d]: %d\n", i, table[i]);

        }

    }

}

int main() {

    printf("Enter size of hash table: ");

    scanf("%d", &table\_size);

*// Initialize table*

    for (int i = 0; i < table\_size; i++) {

        table[i] = EMPTY;

    }

    char command[20];

    int key;

    printf("Enter commands (insert <key>, search <key>, delete <key>, display, exit):\n");

    while (1) {

        scanf("%s", command);

        if (strcmp(command, "insert") == 0) {

            scanf("%d", &key);

            insert(key);

        } else if (strcmp(command, "search") == 0) {

            scanf("%d", &key);

            search(key);

        } else if (strcmp(command, "delete") == 0) {

            scanf("%d", &key);

            deleteKey(key);

        } else if (strcmp(command, "display") == 0) {

            display();

        } else if (strcmp(command, "exit") == 0) {

            break;

        } else {

            printf("Invalid command\n");

        }

    }

    return 0;

}

Output

A screenshot of a computer screen

AI-generated content may be incorrect.

Q26) Write a program in C to implement Sequential File Handling in C (Student Records)

Code

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct {

    int roll\_no;

    char name[50];

    float marks;

} Student;

void createFile() {

    FILE \*fp;

    Student s;

    int n;

    fp = fopen("students.txt", "w");

    if (fp == NULL) {

        printf("Error creating file!\n");

        return;

    }

    printf("Enter number of students: ");

    scanf("%d", &n);

    for (int i = 0; i < n; i++) {

        printf("Enter details for student %d\n", i + 1);

        printf("Roll No: ");

        scanf("%d", &s.roll\_no);

        printf("Name: ");

        scanf(" %[^\n]", s.name); *// to read full line including spaces*

        printf("Marks: ");

        scanf("%f", &s.marks);

        fwrite(&s, sizeof(Student), 1, fp);

    }

    fclose(fp);

    printf("File created successfully!\n");

}

void displayRecords() {

    FILE \*fp;

    Student s;

    fp = fopen("students.txt", "r");

    if (fp == NULL) {

        printf("Error opening file!\n");

        return;

    }

    printf("\nStudent Records:\n");

    printf("Roll No\tName\t\tMarks\n");

    printf("------------------------------------\n");

    while (fread(&s, sizeof(Student), 1, fp)) {

        printf("%d\t%-15s%.2f\n", s.roll\_no, s.name, s.marks);

    }

    fclose(fp);

}

void searchRecord(int roll\_no) {

    FILE \*fp;

    Student s;

    int found = 0;

    fp = fopen("students.txt", "r");

    if (fp == NULL) {

        printf("Error opening file!\n");

        return;

    }

    while (fread(&s, sizeof(Student), 1, fp)) {

        if (s.roll\_no == roll\_no) {

            printf("\nRecord Found:\n");

            printf("Roll No: %d\n", s.roll\_no);

            printf("Name: %s\n", s.name);

            printf("Marks: %.2f\n", s.marks);

            found = 1;

            break;

        }

    }

    if (!found) {

        printf("\nRecord with Roll No %d not found.\n", roll\_no);

    }

    fclose(fp);

}

int main() {

    int choice, roll\_no;

    while (1) {

        printf("\n--- Menu ---\n");

        printf("1. Create File\n");

        printf("2. Display Records\n");

        printf("3. Search Record by Roll No\n");

        printf("4. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                createFile();

                break;

            case 2:

                displayRecords();

                break;

            case 3:

                printf("Enter Roll No to search: ");

                scanf("%d", &roll\_no);

                searchRecord(roll\_no);

                break;

            case 4:

                exit(0);

            default:

                printf("Invalid choice. Try again.\n");

        }

    }

    return 0;

}

Output

A screenshot of a computer program

AI-generated content may be incorrect.

Q27) Write a program in C to implement Binary File Operations in C (Seek/Read/Write)

Code

#include <stdio.h>

#include <stdlib.h>

void createFile(const char \*filename) {

    FILE \*fp;

    int n, num;

    fp = fopen(filename, "wb");

    if (fp == NULL) {

        printf("Error creating file!\n");

        exit(1);

    }

    printf("Enter number of integers: ");

    scanf("%d", &n);

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

    for (int i = 0; i < n; i++) {

        scanf("%d", &num);

        fwrite(&num, sizeof(int), 1, fp);

    }

    fclose(fp);

    printf("File created successfully!\n");

}

void modifyValue(const char \*filename) {

    FILE \*fp;

    int index, new\_value;

    long offset;

    fp = fopen(filename, "rb+"); *// open for read and write*

    if (fp == NULL) {

        printf("Error opening file!\n");

        exit(1);

    }

    printf("Enter index to modify (starting from 0): ");

    scanf("%d", &index);

    printf("Enter new value: ");

    scanf("%d", &new\_value);

    offset = index \* sizeof(int);

    if (fseek(fp, offset, SEEK\_SET) != 0) {

        printf("Error seeking to position!\n");

        fclose(fp);

        return;

    }

    fwrite(&new\_value, sizeof(int), 1, fp);

    printf("Value modified successfully!\n");

    fclose(fp);

}

void displayFile(const char \*filename) {

    FILE \*fp;

    int num;

    fp = fopen(filename, "rb");

    if (fp == NULL) {

        printf("Error opening file!\n");

        exit(1);

    }

    printf("\nContents of the file:\n");

    while (fread(&num, sizeof(int), 1, fp)) {

        printf("%d ", num);

    }

    printf("\n");

    fclose(fp);

}

int main() {

    const char \*filename = "data.bin";

    int choice;

    while (1) {

        printf("\n--- Menu ---\n");

        printf("1. Create File\n");

        printf("2. Modify Value\n");

        printf("3. Display File\n");

        printf("4. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                createFile(filename);

                break;

            case 2:

                modifyValue(filename);

                break;

            case 3:

                displayFile(filename);

                break;

            case 4:

                exit(0);

            default:

                printf("Invalid choice. Try again.\n");

        }

    }

    return 0;

}

Output

A screenshot of a computer program

AI-generated content may be incorrect.

Q28) Write a program in C to implement Two-Way Merge for External Sorting

Code

#include <stdio.h>

#include <stdlib.h>

void mergeFiles(const char \*file1, const char \*file2, const char \*mergedFile) {

    FILE \*f1, \*f2, \*fout;

    int num1, num2;

    int end1 = 0, end2 = 0;

    f1 = fopen(file1, "r");

    f2 = fopen(file2, "r");

    fout = fopen(mergedFile, "w");

    if (f1 == NULL || f2 == NULL || fout == NULL) {

        printf("Error opening files!\n");

        exit(1);

    }

*// Read first numbers from both files*

    if (fscanf(f1, "%d", &num1) != 1) end1 = 1;

    if (fscanf(f2, "%d", &num2) != 1) end2 = 1;

*// Merge process*

    while (!end1 && !end2) {

        if (num1 <= num2) {

            fprintf(fout, "%d ", num1);

            if (fscanf(f1, "%d", &num1) != 1) end1 = 1;

        } else {

            fprintf(fout, "%d ", num2);

            if (fscanf(f2, "%d", &num2) != 1) end2 = 1;

        }

    }

*// Write remaining numbers from file1*

    while (!end1) {

        fprintf(fout, "%d ", num1);

        if (fscanf(f1, "%d", &num1) != 1) end1 = 1;

    }

*// Write remaining numbers from file2*

    while (!end2) {

        fprintf(fout, "%d ", num2);

        if (fscanf(f2, "%d", &num2) != 1) end2 = 1;

    }

    printf("Files merged successfully into '%s'!\n", mergedFile);

    fclose(f1);

    fclose(f2);

    fclose(fout);

}

void displayFile(const char \*filename) {

    FILE \*fp;

    int num;

    fp = fopen(filename, "r");

    if (fp == NULL) {

        printf("Error opening file %s\n", filename);

        return;

    }

    printf("Contents of %s:\n", filename);

    while (fscanf(fp, "%d", &num) == 1) {

        printf("%d ", num);

    }

    printf("\n");

    fclose(fp);

}

int main() {

    const char \*file1 = "file1.txt";

    const char \*file2 = "file2.txt";

    const char \*mergedFile = "merged.txt";

    mergeFiles(file1, file2, mergedFile);

    displayFile(mergedFile);

    return 0;

}

Output

A computer screen shot of a computer code

AI-generated content may be incorrect.

Q29) Write a program in C to implement Natural Merge Sort on Files

Code

#include <stdio.h>

#include <stdlib.h>

#define MAX 1000

void splitIntoRuns(const char \*inputFile, const char \*run1, const char \*run2) {

    FILE \*in = fopen(inputFile, "r");

    FILE \*r1 = fopen(run1, "w");

    FILE \*r2 = fopen(run2, "w");

    int prev, curr;

    int toggle = 0;

    if (!in || !r1 || !r2) {

        printf("Error opening files!\n");

        exit(1);

    }

    if (fscanf(in, "%d", &prev) != 1) {

        fclose(in);

        fclose(r1);

        fclose(r2);

        return;

    }

    fprintf(r1, "%d ", prev);

    while (fscanf(in, "%d", &curr) == 1) {

        if (curr < prev) {

            toggle = 1 - toggle; *// Switch files*

        }

        if (toggle == 0)

            fprintf(r1, "%d ", curr);

        else

            fprintf(r2, "%d ", curr);

        prev = curr;

    }

    fclose(in);

    fclose(r1);

    fclose(r2);

}

int isSingleRun(const char \*fileName) {

    FILE \*fp = fopen(fileName, "r");

    int prev, curr;

    if (!fp) {

        printf("Error opening file %s\n", fileName);

        exit(1);

    }

    if (fscanf(fp, "%d", &prev) != 1) {

        fclose(fp);

        return 1; *// Empty file considered as sorted*

    }

    while (fscanf(fp, "%d", &curr) == 1) {

        if (curr < prev) {

            fclose(fp);

            return 0;

        }

        prev = curr;

    }

    fclose(fp);

    return 1;

}

void mergeRuns(const char \*run1, const char \*run2, const char \*outputFile) {

    FILE \*r1 = fopen(run1, "r");

    FILE \*r2 = fopen(run2, "r");

    FILE \*out = fopen(outputFile, "w");

    int num1, num2;

    int end1 = 0, end2 = 0;

    if (!r1 || !r2 || !out) {

        printf("Error opening files for merging!\n");

        exit(1);

    }

    if (fscanf(r1, "%d", &num1) != 1) end1 = 1;

    if (fscanf(r2, "%d", &num2) != 1) end2 = 1;

    while (!end1 && !end2) {

        if (num1 <= num2) {

            fprintf(out, "%d ", num1);

            if (fscanf(r1, "%d", &num1) != 1) end1 = 1;

        } else {

            fprintf(out, "%d ", num2);

            if (fscanf(r2, "%d", &num2) != 1) end2 = 1;

        }

    }

    while (!end1) {

        fprintf(out, "%d ", num1);

        if (fscanf(r1, "%d", &num1) != 1) end1 = 1;

    }

    while (!end2) {

        fprintf(out, "%d ", num2);

        if (fscanf(r2, "%d", &num2) != 1) end2 = 1;

    }

    fclose(r1);

    fclose(r2);

    fclose(out);

}

void displayFile(const char \*filename) {

    FILE \*fp = fopen(filename, "r");

    int num;

    if (!fp) {

        printf("Error opening file %s\n", filename);

        return;

    }

    printf("Contents of %s:\n", filename);

    while (fscanf(fp, "%d", &num) == 1) {

        printf("%d ", num);

    }

    printf("\n");

    fclose(fp);

}

int main() {

    const char \*inputFile = "input.txt";

    const char \*run1 = "run1.txt";

    const char \*run2 = "run2.txt";

    const char \*outputFile = "sorted.txt";

    while (!isSingleRun(inputFile)) {

        splitIntoRuns(inputFile, run1, run2);

        mergeRuns(run1, run2, outputFile);

*// Update input for next pass*

        FILE \*src = fopen(outputFile, "r");

        FILE \*dst = fopen(inputFile, "w");

        int num;

        while (fscanf(src, "%d", &num) == 1) {

            fprintf(dst, "%d ", num);

        }

        fclose(src);

        fclose(dst);

    }

    printf("Sorting completed!\n");

    displayFile(outputFile);

    return 0;

}

Output

A screenshot of a computer

AI-generated content may be incorrect.